

# Service Differentiation Through End-to-end Rate Control in Low Bandwidth Wireless Packet Networks

Thyagarajan Nandagopal Tae Eun Kim Prasun Sinha Vaduvur Bharghavan  
Coordinated Science Laboratory  
University of Illinois at Urbana-Champaign  
Email: {thyagu, tkim, prasun, bharghav}@timely.crhc.uiuc.edu

**Abstract**— With the increased commercial deployment of wireless networks, the issue of service differentiation among flows in a wireless/wireline network is becoming important. While many wireline solutions for service differentiation depend on end-to-end rate control upon packet loss, this approach does not work well for wireless networks because packet losses do not necessarily indicate congestion in the network in the wireless domain.

In a related work, we presented WTCP, a reliable transport protocol that is designed to operate efficiently and fairly over wireless wide area networks. The rate control algorithm in WTCP is based on the use of the ratio of the inter-packet separation at the receiver and the sender as the metric for detecting and reacting to congestion. In this paper, we generalize the rate control mechanisms of WTCP to support both reliable and unreliable flows, and to provide service differentiation among flows.

We present preliminary results using the ns2 simulator to show that our rate control algorithm provides service differentiation for low bandwidth wireless networks and scales well with the variations in the link loss characteristics.

**Keywords**— Service Differentiation, End-to-End Rate Control, Low Bandwidth Wireless Network

## I. INTRODUCTION

The increasing commercial deployment of wide-area wireless access networks (WWANs) has brought the issue of *service differentiation* in wireless network environments to the fore. Currently available WWANs have limited and expensive network bandwidth  $O(1 - 100)$ Kbps, and this problem is not expected to be alleviated significantly in the immediate future. On the other hand, low-end portable computing devices are increasingly coming equipped with wireless connectivity and are enabled with communication-intensive applications. Consequently, resolving the contention for the limited wireless channel bandwidth is now being considered to be a major issue in the design and deployment of future wireless access networks. Furthermore, since allocation of wireless bandwidth is expensive (compared to wireline resources), there is a strong case for providing *service differentiation* in this environment (i.e. users belonging to a higher “service class” must receive better service).

In a previous work, we have designed the WTCP wireless transport protocol for WWANs [14]. WTCP assumes a proxy-based model of communication, wherein a dedicated proxy in the wireline backbone network interfaces with the client and the server in a typical client-server communication, and behaves like the server with respect to the client and the client with respect to the server. This model is quite common for the operating environment under consideration. Within this model, WTCP provides an optimized transport protocol for reliable data delivery and effective rate control between the client and the proxy over a heterogeneous wireline/wireless connection that traverses a backbone network segment and a wireless access network seg-

ment. In this paper, we present an end-to-end rate control algorithm that extends the rate control algorithm of WTCP using the following two key extensions.

- We adapt the well known linear increase-multiplicative decrease (LIMD) rate control algorithm in order to provide *service differentiation* through rate control. In our approach, each flow belongs to a service class, which has an associated “rate weight”. The goal of our rate control algorithm is to achieve “weighted max-min fair” rate allocation [2], wherein contending flows that traverse the same bottleneck link are allocated rates in proportion to their rate weights.
- We correct several design flaws in the original WTCP rate control algorithm that limited its applicability to very low bandwidth links  $O(10)$  Kbps). As a result of the improved design, we are able to push the applicability of our rate control algorithm to heterogeneous wireline/wireless networks where *per-flow rates* can go up to approximately 100 Kbps. Thus, the target environment for this work is a heterogeneous network that consists of an Internet backbone and a low bandwidth wireless access network with per-flow connection capacities of approximately 100 Kbps or less.

In providing service differentiation, we chose to focus on “relative service differentiation” rather than provide “absolute guarantees” for obvious reasons. Relative service differentiation is gaining popularity as a QoS service model of choice even in the Internet because it can be achieved using comparatively simple and low overhead mechanisms. In the context of wireless networks, it is the only viable option because absolute QoS metrics are very hard to provide in such environments due to the lossy nature of the wireless channel and the mobility of users.

The problem is that many wireline solutions for service differentiation [10] do not work well over wireless networks because they assume negligible non-congestion loss or lossless delivery of network feedback to end hosts upon congestion. Our challenge is to achieve service differentiation by adapting well known end-to-end techniques for rate control while addressing the unique issues introduced by the wireless environment. The key features of our approach are the following:

1. We use end-to-end “weighted max-min fair rate control” based on the LIMD paradigm as the way of achieving service differentiation, as we have mentioned above. We further modify LIMD to perform graded multiplicative decrease to handle congestion. The rate is throttled gently upon detecting incipient congestion, but aggressively upon detecting severe congestion (or congestion-induced packet loss).
2. We use the ratio of the receiver’s and sender’s inter-packet separation as the primary metric for (incipient) congestion de-

tection rather than packet loss or explicit notifications from network routers. This approach is very useful to deal with random packet loss in the wireless channel, but we will see later in the paper that this is also the reason why our solution is not applicable for larger flow rates beyond 100Kbps.

3. We perform the rate control algorithm at the receiver rather than the sender. Consequently, the rate computation is affected predominantly by the state of the forward path and not of the reverse path, and is useful to deal with asymmetric forward and reverse paths.

While the starting point for our rate control algorithm is our previous work in WTCP, our approach has commonalities with several contemporary approaches. We try to achieve service differentiation through weighted rate control similar to [7]. We use inter-packet separation rather than packet loss to compute the progression of the connection similar to [3], [14]. We perform graded multiplicative decrease similar to [11].

The rest of the paper is organized as follows. Section II describes the network and service model of this work. Section III presents the key algorithmic principles for achieving service differentiation through rate control. Section IV evaluates the performance of the algorithm using simulations. Section V discusses the limitations of this work in terms of the applicable operating environment. Section VI briefly presents related work and summarizes the salient features of our algorithm.

## II. MODEL AND BACKGROUND

In this section, we first describe the channel model and service model, and then briefly discuss the LIMD rate control algorithm on which our work is based.

### A. Channel Model and Service Model

Our target environment is a heterogeneous wireless/wireline packet data network that consists of a wireline backbone network and a low bandwidth wireless access network. The wireless network characteristics may include low bandwidth, high and varying packet latency, non-congestion related packet loss that may be random, bursty and location-dependent, and asymmetric uplink and downlink channels. While the wireless link is almost always the bottleneck in the target environment, one important characteristic of the wireline backbone is that the delays perceived by packets may be highly variable. We will see the impact of this characteristic on the rate control algorithm in Section V.

Our challenge is to design algorithms for service differentiation and rate control in this target environment. Further, because of the autonomous nature in which wireless networks are deployed and the heterogeneity in the wireless technologies, we must work under the following constraints: (a) there are no special mechanisms to achieve service differentiation at the base station, (b) wireline routers are unaware of wireless hops further downstream, and perform no special actions to support flows with a wireless last hop, and (c) we will only use end-to-end mechanisms to achieve rate control and service differentiation.

Given the highly dynamic nature of the wireless network, absolute rate guarantees are very hard to provide, since the resources of the wireless channel keep changing continually. Hence, in this paper we focus on achieving relative service differentiation via *weighted max-min fairness*, which we define below. Let us first start with defining max-min fairness [2]. Consider a set of flows  $\{1...n\}$  and a rate allocation vector for the

flows  $R = [r_1...r_n]$  such that  $r_i \leq r_j$  if  $i < j$  without loss of generality. Then  $R$  is a max-min fair rate allocation if it is impossible to increase the rate allocation  $r_i$  without causing the decrease of some  $r_j \leq r_i$ . Weighted max-min fairness is a simple extension of max-min fairness. Let each flow  $i$  have a corresponding weight  $w_i$ . Then a rate allocation vector  $R$  is weighted max-min fair if it is impossible to increase the rate allocation of  $r_i$  without causing the decrease of some  $r_j$  such that  $\frac{r_i}{w_j} \leq \frac{r_j}{w_i}$ . In terms of the service model, each flow belongs to a service class, and each service class has an associated rate weight. A flow is assigned the rate weight associated with its class.

### B. Linear Increase/Multiplicative Decrease

In wireline networks, the linear increase/multiplicative decrease (LIMD) algorithm is predominantly used for rate control [6]. LIMD is usually implemented as a sender-based rate control algorithm that is based on end-to-end loss feedback. Let us consider a framework in which periodic feedback is used, and let  $r_i$  denote the sending rate maintained by the sender during epoch  $i$ ,  $f_i$  denote the loss feedback from the receiver to the sender at the end of epoch  $i$  that specifies the fraction of packet loss in the epoch, and  $\alpha$  and  $\beta$  denote constants. LIMD works as follows:

- The sender increases the transmission rate by a constant factor if no packet was lost in the last epoch.  
 $r_{i+1} \leftarrow r_i + \alpha$ , if  $f = 0$ , where  $\alpha$  is typically set to 1 packet/sec.
- The sender decreases the transmission rate by a multiplicative factor if one or more packets were lost in the last epoch.  
 $r_{i+1} \leftarrow \max\{r_i(1 - \beta), \alpha\}$ , if  $f > 0$ , where  $\beta$  is typically set to 0.5.

The above algorithm, or some variant thereof, is used almost universally in wireline rate control and window based algorithms today including TCP. It turns out that none of these variants is particularly well suited for wireless networks, for reasons described below.

1. *Packet Loss*: In wireline networks, an overwhelmingly large fraction of packet losses is caused due to buffer overflow; hence packet loss reliably indicates congestion. Thus, LIMD's reaction of aggressively throttling sending rate in response to packet loss is appropriate for this case. However, in wireless networks, losses may stem from several non-congestion related causes such as random channel error, blackouts, misrouting (e.g. due to mobility), etc. In response, we need a rate control algorithm that either distinguishes congestion loss from non-congestion loss, or does not use losses as the primary metric for rate control.

2. *Low Bandwidth and High Delay*: LIMD increases sending rate by an additive constant when it does not detect congestion (or equivalently, no packet loss). Thus, even when incipient congestion is building up queues in the network, LIMD continues to increase sending rate (unless the routers themselves drop packets using some RED-like mechanism [9]). In low bandwidth and high delay networks such as outdoor wireless networks (e.g. CDPD [1]), this can become a problem because of two reasons: (a) the increase phase itself may be the cause of congestion, particularly in cases where the delay bandwidth product is small and an increase of rate by 1 packet/second represents a significant percentage increase in rate, and (b) for window-based congestion control algorithms, increase in the congestion window causes packets to be queued at the low bandwidth bottleneck link and causes large variations in estimated round trip time, which further skews the delay-bandwidth product estimation and possibly loss detection [14].

3. *Asymmetric Channel*: Most LIMD implementations are sender-based. Thus the loss of feedback or acknowledgements can distort the sender's perception of the connection, even if the forward path does not experience any loss. In wireless packet cellular networks, uplink and downlink channels can have significantly different characteristics. For the downlink channel, the base station can perform centralized scheduling and is not power-constrained in transmission, while for the uplink channel, channel access is decentralized and mobile hosts may be power constrained. Typically, downlink channels have better quality than uplink channels, and this fact can be exploited for the common case of bulk data transfer on the downlink (from static servers to mobile clients).

### III. RATE CONTROL AND SERVICE DIFFERENTIATION

While an initial reading of the problems described in the previous section seems to indicate that LIMD is not appropriate for the target environment, most of the problems we raised are not fundamental to the basic LIMD algorithm, but to the implementation framework of LIMD. In fact, we use LIMD as the building block for our weighted fair rate control algorithm, but we tailor its implementation to address the issues of the wireless network in three important ways: (a) the metric for triggering rate increase or decrease is the *ratio of the receiver to sender inter-packet separation* rather than packet losses, (b) the *granularity of rate decrease is based on a simple history of rate evolution in the recent past*, and (c) the *granularity of rate increase is weighted* according to the rate weight of the flow. In concert, these three modifications enable us to achieve efficient rate control as well as service differentiation in a wireless network environment. We now motivate the key aspects of our algorithm, and then present the detailed rate control algorithm.

#### A. Key Aspects

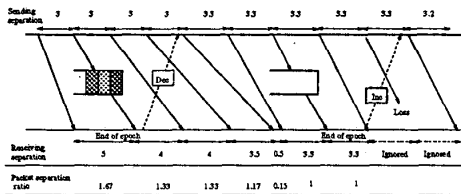


Fig. 1. Rate control using receiving to sending packet separation ratio

Queues build up in the initial stage since the sending rate is greater than the receiving rate. Hence, at the end of the first epoch, the ratio is greater than 1 and the receiver sends a *decrease* feedback to the sender. The sender reduces its rate, thus relieving congestion in the link. As a result, the queues become empty. At the end of the second epoch, the ratio of the sending rate to the receiving rate is 1, indicating that there is no congestion in the network. Hence the receiver sends an *increase* feedback to probe the network for extra bandwidth.

1. *Interpacket Delay as the Metric for Rate Control*: We start with a very simple observation: congestion builds up when the aggregate sending rate exceeds the link capacity. In such a scenario, the sending rate is greater than the receiving rate for at least one flow. Thus, if the receiver measures the ratio of the sender rate to receiver rate and this ratio exceeds a threshold, then we predict that the sender must throttle its rate. Unfortunately, the receiver cannot measure rate accurately merely by counting packets in a given time window because of possible packet losses that are not related to congestion. On the other

hand, if the sender sends packets at a fixed rate and spreads out these packets evenly over time (as opposed to self-clocking packet transmissions using CACKs [15], which may cause bunching of packets), then we can use the inverse of the sender's inter-packet separation as the sending rate and we can measure the received inter-packet separation for consecutive packets at the receiver (packets have monotonically increasing congestion control sequence numbers in addition to the byte sequence number used for reliability), the inverse of which is the receiver's rate. The ratio of the receiver's inter-packet separation to the sender's inter-packet separation is equal to the ratio of the rate at the sender to the rate observed at the receiver, and can serve as the metric for rate control at the receiver, even in the presence of channel error. This idea is illustrated in Figure 1. The advantages of using this metric are twofold: first, it is not affected by random packet loss; and second, we can detect and react to incipient congestion, thereby remaining in the congestion avoidance phase most of the time. In [3], the authors use this approach to distinguish between congestion losses and random losses, but they do not use this for congestion control. In WTCP [14], we use this approach to perform congestion control. The disadvantage is that the sender must now explicitly clock packet transmissions rather than using acknowledgements for self-clocking. More importantly, this metric is susceptible to large delay variations that packets observe as they traverse through the network. These variations do not affect the correct operation of the rate control algorithm for low bandwidth networks. Hence, we will not consider their effect in this section. We will revisit this issue in Section V.

The main issue is: when do we increase the rate of a flow (to probe the connection for additional capacity) and when do we throttle its rate? In our algorithm, we maintain a running average of the ratio of receiver to sender inter-packet separation, and update this average at the end of each epoch. Recall that the feedback is epoch-based. The running average is used to determine if the rate has to be increased or decreased for the next epoch as follows. Consider a single flow traversing a link of capacity,  $C$ . Let the rate of the flow be denoted by  $r$ . When  $r < C$ , the link can support the rate  $r$ , and hence the ratio of the sending to receiving rate is 1. Thus, the receiver can increase the sending rate to probe the network for additional bandwidth. Note that the maximum value of  $r$  that the link can support is  $C$ . If the rate is increased beyond this value such that  $r = C + \alpha$  (where  $\alpha$  is the constant of increase in LIMD), packets will start getting queued in the network as the receiving rate is bounded by the link capacity, namely  $C$ . At this point, the ratio of the sending rate to the receiving rate is therefore  $(C + \alpha)/C$ , which translates to  $1 + \alpha * receive\_separation$ . As the flow is now sending at a rate higher than the receiver can receive at, the sender needs to slow down its rate. Thus,  $(1 + \alpha * receive\_separation)$  defines the threshold for increase or decrease. We explain below how this mechanism can be extended to support weighted rate fairness.

Note that, unlike WTCP, the threshold for determining increase or decrease is not based on static design parameters; rather, it is a self adjusting threshold that is less sensitive for lower rate flows.

2. *Using History for Graded Multiplicative Decrease*: Since we start to throttle the sending rate as soon as the ratio of the receiver to sender inter-packet separation exceeds the threshold, we react quickly to incipient congestion, and rarely get to the stage where packets must be dropped at routers due to conges-

tion. Since we expect to be in the congestion avoidance phase most of the time, we can afford to throttle more gently upon incipient congestion (rather than the traditional 50% in TCP). However, as congestion builds up, we must throttle rate more aggressively in order to quickly account for the built-up congestion and prevent packet losses from occurring. Essentially, this motivates having a simple notion of “history” of rate control and a simple gradation for rate throttling, where we progressively throttle rate more severely for back-to-back decrease phases. In our algorithm, we do this by maintaining a *decrease constant*  $d$  that is reset to 1 whenever we encounter the increase phase, and is doubled for each decrease phase. During the decrease phase, we throttle the rate to a fraction  $(1 - \delta)$ , where  $\delta = d * \beta$ , with  $d$  acting as the graded multiplicative decrease factor that exponentially increases the granularity of decrease for successive decrease phases (i.e. if the previous decrease phase was not sufficient to throttle the rate adequately). As a result of this approach, incipient congestion is handled by a gentle decrease of the transmission rate, but severe congestion is handled by an aggressive decrease in transmission rate. In a related work, we have explored this algorithm in more detail [11].

**3. Using Weighted Increase to Achieve Service Differentiation:** Thus far, we have focused on rate control rather than service differentiation. The key question is: how can we adapt this basic approach to achieve service differentiation? One obvious approach is to use *weighted increase in the increase phase*. In other words, we increase the rate in proportion to the rate weight of the flow. Using arguments similar to the ones made above, the threshold for increase or decrease becomes  $(1 + \alpha w * receive\_separation)$ . The equations for increase and decrease become

$$r \leftarrow r + \alpha w, \text{ for increase and} \\ r \leftarrow r(1 - \beta), \text{ for decrease}$$

For a small range of rate weights, this approach works well, and converges to the fairness line as shown in Figure 2(a) However, as the range of rate weights becomes large, flows with larger weights increase their rates by larger steps, and this may cause congestion, particularly for low bandwidth outdoor wireless networks.

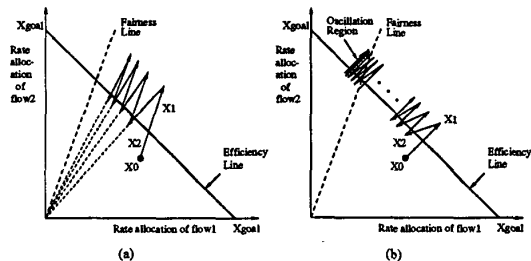


Fig. 2. Service Differentiation using (a) Weighted Increase and (b) Weighted Decrease

There are two flows. Flow 1 has a rate weight of 1 and Flow 2 has a rate weight of 2. Consider the point  $X_0$ . In (a) the flows perform weighted increase to reach point  $X_1$ . Then in the decrease phase, both flows cut back by the same percentage to fall back to  $X_2$ . The system progressively moves to the fairness line. In (b) the increase amount from  $X_0$  to  $X_1$  is equal for both flows. While decreasing from  $X_1$  to  $X_2$ , however, Flow 1 and Flow 2 reduce in inverse proportion of their rate weights. Thus, Flow 1 reduces more than Flow 2. It can be shown that the system oscillates in a very small band around the fairness line.

This problem can be solved by modifying the decrease phase

rather than the increase phase of the LIMD algorithm, as shown in Figure 2(b) Specifically, if we set the *decrease fraction to be inversely proportional to the rate weight* rather than setting the increase in rate to be directly proportional to the rate weight, then we will still converge approximately to the same result, as illustrated in the figure. The increase/decrease equations now are

$$r \leftarrow r + \alpha, \text{ for increase} \\ r \leftarrow r(1 - \beta/w), \text{ for decrease}$$

The difference is that flows with larger rate weights decrease less aggressively as opposed to increasing more aggressively. Note that substituting  $w = 1$  makes both the algorithms the same as the WTCP rate control algorithm, which was not designed for service differentiation. A detailed proof of convergence and the different weighted fairness properties of these algorithms under the assumptions of uniform feedback and of constant packet dropping probability are presented in [12].

#### 4. Distinguishing Congestion Loss from Random Loss

As we have discussed earlier, a major point of this paper is that we do not use packet losses as the metric for detecting congestion - in fact, our goal is to detect and react to incipient congestion - in fact, our goal is to detect and react to incipient congestion so that packets are never lost due to buffer overflow at intermediate routers. However, in spite of our early congestion detection and response approach, it is still possible for flows to experience congestion losses. Our algorithm must detect such losses and throttle sending rate aggressively. In other words, when the receiver observes packet losses, it must predict the cause of the losses and react appropriately. How can a receiver distinguish random losses from congestion losses? We have considered two possible heuristics:

(a) *Using average per-packet separation:* In this heuristic, the receiver initially predicts that all losses are non-congestion losses. Consider that packets  $i$  and  $j$  were received ( $j > i$ ), but packets  $i + 1 \dots j - 1$  were all lost. In this case the receiver computes the average inter-packet separation for each of the lost packets equal to  $per\_pktsep \leftarrow \frac{recv\_time_j - recv\_time_i}{j - i}$ . If the value of  $per\_pktsep$  is close to the measured inter-packet separation at the receiver (i.e. within the band [average - 2\*mean deviation, average + 2\*mean deviation]), then the receiver predicts that the losses were all random losses. Otherwise, the receiver predicts that there was at least one congestion loss, and throttles the sending rate by half.

(b) *Using ratio of the receiver to sender inter-packet separation:* In this heuristic, the receiver maintains the ratio of the receiver to sender inter-packet separation as before. When there is a packet loss, if the ratio is below the “congestion detection threshold”, then the receiver predicts random loss, otherwise it predicts congestion loss and throttles the sending rate by half. This approach is similar to the heuristic in [3].

Neither of these heuristics is accurate. In the first case, it is possible to mistakenly predict some congestion losses as random losses, while in the second case, it is possible to mistakenly predict random losses as congestion losses. Since our response to congestion losses is to throttle the sending rate aggressively, and since congestion losses are not expected to happen often due to early congestion avoidance, we believe that the first heuristic is better.

In summary, the basic tenets of our approach are that it is rate based with receiver based rate computations, and that it uses rate weights for increase/decrease phase to achieve weighted rate allocations. The sender sends packets according to a recommended rate by the receiver, and enforces an inter-packet separation

```

1  process_data_packet()
2  if (packet is out of sequence)
3    process_packet_loss()
4  else
5    current_pktsep ← current_time - prev_pkt_time
6    if (sender has started using the latest computed rate)
7      ratio ← current_pktsep / hdr→snd_packetsep
8      aggregate_ratio ← aggregate_ratio + ratio
9      num_samples ← num_samples + 1
10   prev_pkt_time ← current_time
11
12   update_timer_wakeup()
13   if (there were no samples in last epoch)
14     return
15   savg_ratio ← aggregate_ratio/num_samples
16   lavg_ratio ← exponential average of savg_ratio over last 6 epochs
17     with weight of 0.125 for new sample
18   num_samples ← 0
19   aggregate_ratio ← 0
20   if (lavg_ratio < 1 + savg_ratio * send_packet_sep * α * rate_weight)
21     /* savg_ratio * send_packet_sep measures average receive_packet_sep */
22     rate ← rate + α * rate_weight /* Weighted increase */
23     δ ← 0.1 /* reset decrease variable */
24   else
25     rate ← rate × (1 - δ) /* Decrease rate by δ% */
26     δ ← max(2δ, 0.5)

```

Fig. 3. Algorithm for Rate Control at receiver

ration that is the inverse of the rate. The sender advertises the inter-packet separation in each packet, and assigns monotonically increasing congestion control sequence numbers for packet transmissions. Each time the receiver receives a packet in sequence, it collects a sample of the receiver to sender inter-packet separation ratio. At the end of each epoch, the receiver updates the running average of this ratio, checks to see if it must increase or decrease the rate, adjusts the rate accordingly, and then advertises the updated value back to the sender in acknowledgements during the current epoch. The frequency of the receiver acknowledgement can be controlled by the sender depending on the characteristics of the reverse path - in the worst case, every packet can be acknowledged. Loss of the receiver feedback only delays the sender's adjusting to the new recommended rate, but does not change the computed rate value. Similarly, random loss of data packets does not lead to aggressive throttling of rate, but impacts the algorithm in only one way - there are fewer samples of the receiver to sender inter-packet ratio (thus with very large random losses, the algorithm does not work well because there are not enough samples). With this framework in mind, we will now present the details of the rate control algorithm.

### B. The Rate Control Algorithm

The pseudo-code for the rate control algorithm is shown in Figure 3. The rate control algorithm has two events associated with it: the arrival of a packet at the receiver, and a periodic timer based update.

When a packet arrives at the receiver, the receiver checks to see if the packet is in sequence. If the packet is out of sequence (Line 1), the receiver assumes that the intermediate packets have been lost and attempts to identify the reason for the losses, as the rate needs to be reduced if there is network congestion. For congestion losses, it reduces the rate by half, or else, it treats the losses as random losses and does not react to it (Line 2). If the packet has been received in sequence, the receiver computes the separation between this packet and the previous packet (Line 4). If the packet has the updated rate, then the ratio of the receiving to sending separation is computed, and the aggregate of ratio over all samples is used to compute the short term and long term

averages at the end of an update epoch (Lines 6-8).

Rate updates occur at the end of every update epoch. The short term average of the ratio of receiver to sender inter-packet separation is computed based on the samples obtained in the epoch (Line 12). A long term average is used to maintain limited history of ratios and to smooth out the effect of brief network dynamics. It is maintained as an exponential running average over the last six short term averages (Line 13).

Compared to WTCP, there are subtle but crucial changes in the computation of the ratio of inter-packet separations at the receiver and sender. We use an epoch's worth of samples to compute the short term average of the inter-packet separation at the receiver. Further, we limit the history computation for the long term average to a fixed number of epochs rather than using exponential averaging. Consequently, the algorithm is more stable and less susceptible to delay variations experienced by packets.

If the average ratio is less than the threshold (Line 16), then it implies that the network was able to support the previous rate. Hence, the rate control algorithm tries to do probing by increasing the rate. The rate increases for flows are in proportion to their rate weights (Line 17). A flow with a higher  $w$  will increase more than a flow with a lower  $w$ . If a flow enters the increase phase, then decrease variable is reset to 10% (Line 18).

If the average ratio is greater than the threshold, then the receiver reduces the rate using the decrease variable  $\delta$  (Line 20). The initial value of  $\delta$  is 10%. However, if the average ratio is greater than the threshold for consecutive epochs, it implies that the network resources have decreased or that the flow might have to give some room to another flow. Hence, the rate control algorithm doubles the value of  $\delta$ , subject to a maximum of 50% (Line 21).

## IV. SIMULATIONS

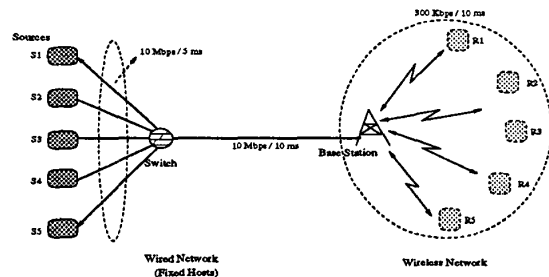


Fig. 4. Network topology

In this section, we evaluate the performance of our rate control algorithm using the *ns-2* simulator. We present four sets of tests here. The simulation topology for the first three sets is shown in Figure 4. This topology is a simplistic representation of typical low bandwidth wireless networks, wherein mobile hosts contend for access to a base station which connects the mobile hosts to the fixed network. The base station controls the downlink; hence, there is no contention on the downlink. The wireless segment is the bottleneck link in this topology.

There are five flows, with fixed hosts acting as sources, and the mobile hosts acting as the receivers. In the first set of tests, we illustrate that inter-packet separation based rate control achieves fairness, even in the presence of channel error. The second set of tests focuses on service differentiation through rate

weights. The third set of tests demonstrate the scalability of the rate control algorithm.

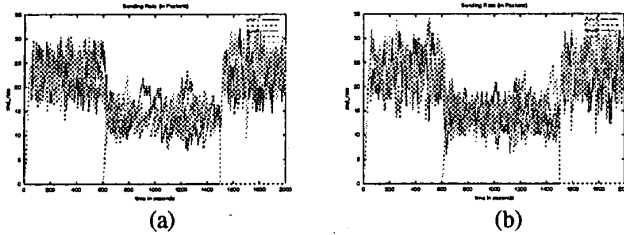


Fig. 5. All flows have equal rate weights: Sending rate (a) with no channel error and (b) with 1 % channel error

In all simulations, the packet size is 512 bytes, the epoch size at the receiver is 350 msec, and the bottleneck link buffer size is 50 packets. The constant increase factor for each *increase* feedback from the receiver is 0.5 packets/second without rate weight and  $(0.1 * rate\_weight)$  packets/second with rate weight. For each *decrease* notification, the rate is reduced by 10% with history, i.e. for successive decreases, the rate reduction is 10% at first, then doubled to 20% and so on, subject to a maximum of 50%. The channel error is exponentially distributed and the mean rate is set to 1% unless mentioned otherwise. The durations of all the simulations are 2000 seconds.

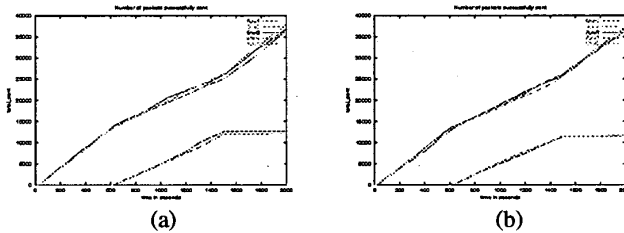


Fig. 6. All flows have equal rate weights: Total throughput (a) with no channel error and (b) with 1 % channel error

### A. Rate Control

The first set of results demonstrates the fairness of the rate control algorithm.

Flows 1 to 5 have equal rate weights. Flows 3, 4 and 5 last the entire duration of the simulation, while flows 1 and 2 begin at  $t = 600$  seconds and end at  $t = 1500$  seconds. The wireless channel capacity is 300 Kbps which is equivalent to 73 packets/sec. Figures 5 (a) and 6 (a) show the sending rate and the achieved throughput when there is no channel error. Figure 5 (a) shows that the sending rates of flows 3, 4 and 5 are 24 packets per second on an average up to 600 seconds, fall to 14 packets per second from 600 seconds to 1500 seconds due to the introduction of flows 1 and 2, and increase back to 24 packets per second after 1500 seconds when flows 1 and 2 leave the network. Flows 1 and 2 achieve the fair sending rate of 14 packets per second during their life time. Even though it appears that the rates of individual flows vary significantly within a band, Figure 6 (a) shows that the rate control algorithm achieves fairness.

From the results, we can see that the system utilization is above 90%. The rate control algorithm always operates in the

congestion avoidance region, thereby avoiding losses due to the introduction of new flows.

We introduce a channel error of 1% in the wireless segment to see how our rate control algorithm perform in the presence of channel error. The results are presented in Figures 5 (b) and 6 (b). Since we are operating in the congestion avoidance phase, all losses are typically random losses that occur due to channel error. Due to these random losses, jitter is introduced in inter-packet separation at the receivers. As a result, the variation in the sending rate increases slightly in Figure 5 (b). However, we still obtain fairness and high utilization in the presence of channel error as shown in Figure 6 (b).

### B. Service Differentiation

We now illustrate the service differentiation achieved using our rate control algorithm. There are five flows in this example with each flow  $i$  having a rate weight of  $i$ . Thus, the expected throughput of flows should be in the ratio 1:2:3:4:5. Flows 1 and 2 join the system at time  $t = 600$  seconds and leave the system at time  $t = 1500$  seconds, while flows 3, 4 and 5 last for the entire duration of the simulation. The round-trip times for all the flows are the same and the flows do not perceive any wireless channel error.

The obtained throughput is shown in Figure 7 (a). There is no observed packet loss. The startup is linear and since we do not use a non-linear increase mechanism like slow-start, the convergence speed is slow. The average rates between periods when flows join and leave is presented in Table 1.

From the table, it can be seen clearly that we are able to achieve service differentiation in proportion to the rate weights of the flows. The deviation from the  $W/i$  for flows 1 and 2 during 601-1500 seconds is due to their startup behavior. However, from the figure, we observe that flows 1 and 2 obtain their weighted fair share once they come out of the startup phase.

### C. Scalability

In this set of tests, we show the performance of our rate control algorithm in various network environments. In the first test, we show that the rate control algorithm preserves fairness when all flows have different round trip times. The second test shows how fairness is affected by different channel error rates.

#### C.1 Round Trip Times

In this example, we consider the same scenario as in Section IV-B. However, each flow has different RTTs as in Table II.

The results are presented in Figure 7 (b). We can see that fairness and the throughput are still maintained even with different round-trip times among flows.

	Flow 1	Flow 2	Flow 3	Flow 4	Flow 5
RTT (ms)	220	270	370	520	820

TABLE II  
ROUND TRIP TIMES FOR EXAMPLE IV-C.1

We introduce channel error of 1% in this scenario and present the obtained throughput in Figure 7 (c). As shown in the figure, the channel error does not affect the fairness of the system, and all flows are able to attain weighed service differentiation.

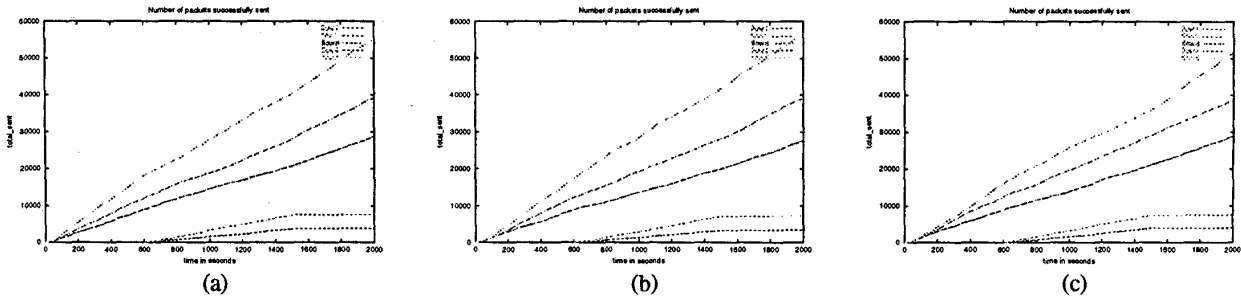


Fig. 7. Service Differentiation: Flow  $i$  has a rate weight of  $i$   
 (a) Equal RTTs, (b) Different RTTs without channel error and (c) Different RTTs with channel error of 1%

Time	0 - 600			601 - 1500			1501 - 2000		
	$W_{ideal}$	$W_{actual}$	$W/i$	$W_{ideal}$	$W_{actual}$	$W/i$	$W_{ideal}$	$W_{actual}$	$W/i$
Flow 1				4.9	4.2	4.2			
Flow 2				9.7	8.6	4.3			
Flow 3	19	16	5.3	14.6	13.8	4.6	19	16.4	5.3
Flow 4	24	22	5.5	19.5	18.3	4.6	24	22	5.5
Flow 5	30	29	5.8	24.3	25.1	5.0	30	30	6

TABLE I  
 SERVICE DIFFERENTIATION: FAIR SHARE AND OBTAINED THROUGHPUT

Flow  $i$  has a rate weight of  $i$ . The average rate of each flow is represented by  $W$ .

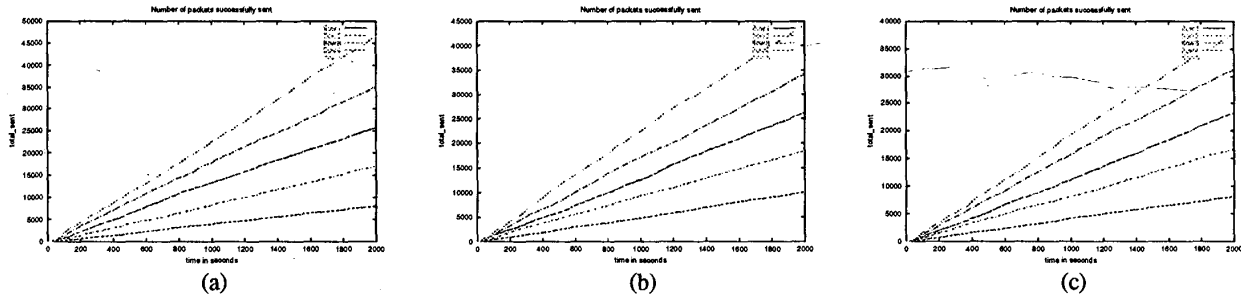


Fig. 9. Service Differentiation for different error rates: Total throughput  
 (a) 1%, (b) 3%, and (c) 5% channel error

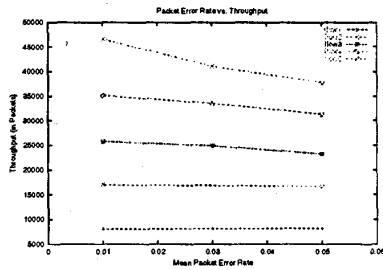


Fig. 8. Total throughput vs. Channel error rate

## C.2 Error rates

Channel error is an important cause for performance degradation in wireless networks. In this test, we vary the mean channel error rates from 1% to 5% and evaluate the performance of our rate control algorithm. The channel error is exponentially distributed.

We simulated five flows with equal round trip times and with flow  $i$  having a rate weight of  $i$ . All flows last for the entire duration of the simulation. In Figures 9 (a), (b) and (c) we present the throughput for the five flows with error rates of 1%, 3% and 5% respectively. We plot the final throughput against error rate in Figure 8. We observe from the graphs that the service differentiation does not happen in exact ratio of the flows rate weights with high channel error rates. Flows with higher rate weights suffer more from channel error since they lose more packets thereby losing more samples for measurement in a single epoch. Thus, when they decrease, they perform successive multiple decreases, thereby affecting their throughput.

This is because the rate control algorithm does not throttle aggressively upon random losses and thereby is able to mask the effects of channel error.

### D. Heterogeneous Networks

Thus far, we have illustrated the features of our algorithm using a very simplistic scenario. We now introduce a slightly more complex scenario that consists of a multihop network with background traffic created by other TCP flows.

#### D.1 Wireline/wireless networks

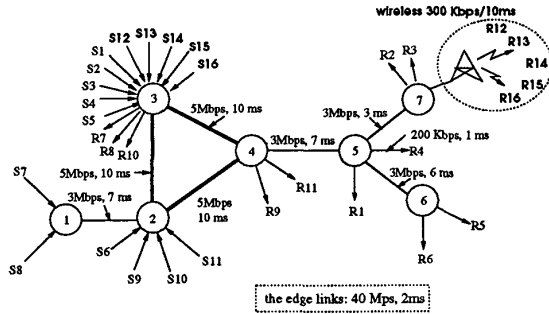


Fig. 10. Multihop network topology

In the multihop topology, as shown in Figure 10, the wireline component consists of nodes 1 through 7 and the wireless component consists of a base station with 5 nodes in its cell. The wired links have bandwidths in the range of 3 Mbps to 5 Mbps with link delays not exceeding 10 ms. S1 through S11 represent 11 TCP sources with corresponding receivers represented by R1 through R11. The sources for service-differentiated flows are S12 through S16 at the node 3, with the corresponding receivers R12 through R16 at the mobile hosts. The bottleneck link for the service-differentiated flows is the wireless channel of 300Kbps shared by five flows. The 3Mbps link acts as a bottleneck for the 6 TCP flows and is also shared by the five service-differentiated flows.

We observe that even in a heterogeneous multihop network with background TCP traffic, our algorithm achieves fairness as shown in Figure 11. Note that the per-flow share of a flow is of the order of 60 Kbps (300 Kbps shared by five flows in the wireless cell).

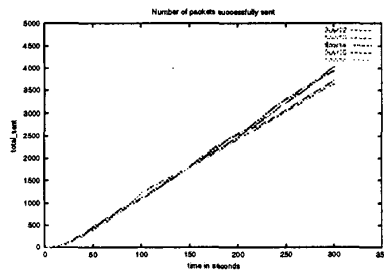


Fig. 11. Total throughput in the multihop network

#### E. Multiple wireless networks

In this section, we show how close the rate control algorithm achieves weighted rate fairness in a cluster of heterogeneous networks.

We use the topology shown in Figure 12. There are three wireless networks with different characteristics. The first wire-

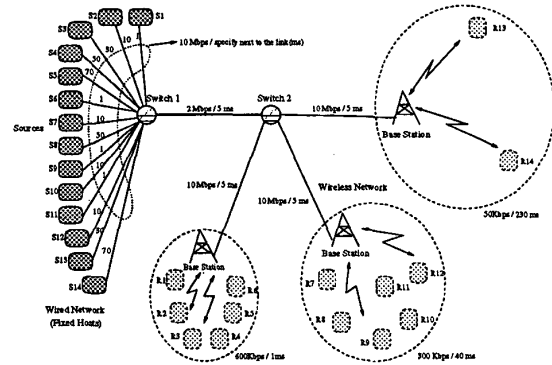


Fig. 12. Heterogeneous wireless network topology

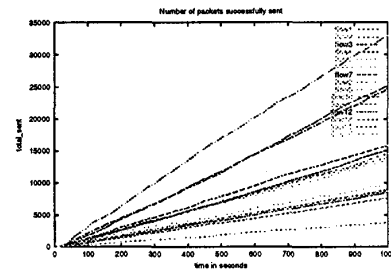


Fig. 13. Total throughput in heterogeneous networks

less network has a low bandwidth of 50Kbps and a channel delay of 230 msec, and represents wide area networks, the second and third represent a campus wireless access network with a bandwidth of 300Kbps and a channel delay of 40msec, and with a higher bandwidth of 600Kbps and a delay of 20 msec, respectively. The channel error rate for all the network links is 1%. Fourteen TCP flows pass  $switch_1 - switch_2$  as background traffic. Flows 1 to 6 pass through the high bandwidth wireless network, flows 7 to 12 pass through the medium bandwidth wireless network, and flows 13 and 15 pass through the low bandwidth wireless network. Thus, flows 7 to 15 have the last hop wireless channel as their bottleneck and flows 1 to 6 have the intermediate link  $switch_1 - switch_2$  as their bottleneck. Flows 1, 3, 7, 10, and 13 have a weight of 2, flows 2, 5, 6, 8, 9, 11, 12 and 14 have a weight of 1, and flow 4 has a weight of 3.

The result is shown in Figure 13. Since the throughput of flows 7 to 15 is limited by the last-hop wireless channel of capacity, they achieve an average sending rate as shown in Table III. The remaining bandwidth of link  $switch_1 - switch_2$  is shared by the background TCP flows and by flows 1 to 6 in proportion to their weights.

### V. LIMITATIONS

In previous sections, we have presented a rate control algorithm and showed through simple simulations that it can achieve service differentiation as well as efficient rate control in heterogeneous wireline/wireless networks. Now let us step back and consider the basic assumptions of our work, and see under what conditions these assumptions are valid.

The fundamental premise of the rate control algorithm is that interpacket separation can be used to derive a valid metric for

<i>flow<sub>id</sub></i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ideal fair share	32	16	32	48	16	16	19	9	9	19	9	9	8.2	4.1
actual receiving rate	24.6	13.1	25.1	33	14	13.5	15.1	8.7	8.9	15.8	8.6	8.5	7.5	3.8

TABLE III  
MAX-MIN FAIRNESS FAIR SHARE AND OBTAINED THROUGHPUT

determining incipient congestion. Now let us consider a flow that traverses the Internet on the backbone segment of its path. It is well known that packets in the Internet experience variable delays, and the queueing delays may vary significantly even for successive packets. Obviously, this observation affects the computation of the average receive packet separation.

Recall that we trigger the decrease or increase phase of our rate control algorithm depending on whether the following condition is satisfied or not:

$$\frac{\text{receive\_separation}}{\text{send\_separation}} \geq (1 + \alpha w * \text{receive\_separation})$$

Note that when the bandwidth of the wireless link is very low, then the value of *receive\_separation* is very large. For example, on a 8Kbps channel, it takes 1 second to transmit a 1KB packet (i.e. *receive\_separation* = 1 sec). For large values of *receive\_separation*, delay variations of the order of 10-100ms that are observed by packets in the network will not significantly affect the behavior of the rate control algorithm. This is because our rate control algorithm has the feature that the distortion in the  $\frac{\text{receive\_separation}}{\text{send\_separation}}$  ratio must be of the order of the *receive\_separation* in order to affect the correct operation of the rate control algorithm. However, at higher data rates, *receive\_separation* becomes much smaller and is thus susceptible to distortions induced by the delay variations experienced by packets. The key question therefore is: "at what flow rates do the distortions due to delay variations in the network become significant?" In the rest of this section, we will present some very simple approximations in order to develop a coarse understanding about the range of applicability of our rate control algorithm.

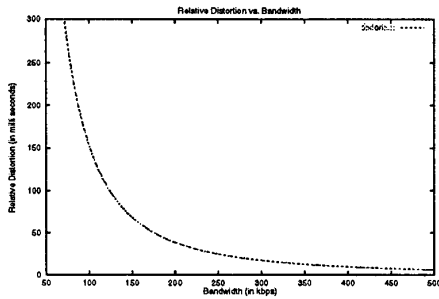


Fig. 14. Plot of tolerable relative distortion vs. flow rate

Let  $s_i$  be the send time at the sender of packet  $i$  of a flow,  $r_i$  be the receive time at the receiver of packet  $i$ ,  $p$  be the propagation delay along the path of the flow,  $q_i$  be the aggregate queueing delay experienced by packet  $i$ , and  $k$  be the clock skew between the receiver and the sender. Then the delay experience by packet  $i$  is

$$r_i - s_i = p + q_i - k$$

For simplicity, let us assume that  $p$  is invariant for a flow and  $c$  is invariant for a sender-receiver pair. Let us consider a single epoch within which there is no rate adaptation. Then,

$$s_i = s_0 + i \cdot s_{sep}$$

where  $s_{sep}$  is the interpacket separation at the sender.

$$r_i - r_0 = s_i - s_0 + q_i - q_0 = i \cdot s_{sep} + (q_i - q_0)$$

Let

$$r_i - r_0 = i \cdot r_{sep}$$

where  $r_{sep}$  is the average interpacket separation at the receiver. Then,

$$i \cdot (r_{sep} - s_{sep}) = q_i - q_0$$

Now consider the ideal case wherein the flow traverses through a dedicated pipe of capacity  $C$ . Let the weight of the flow  $w = 1$ , and the increase constant  $\alpha = 1$ . Then, at the point when the rate control algorithm transitions from the increase phase to the decrease phase, the following equalities hold.

$$r_{sep} = \frac{1}{C} \quad s_{sep} = \frac{1}{C+1}$$

$$q_i = q_0 + i \left( \frac{1}{C} - \frac{1}{C+1} \right)$$

In the ideal case, the queueing delay experienced by the packets in a flow satisfies the above relationship. Unfortunately, for typical flows traversing the Internet, we cannot accurately specify the value of  $q_i$  merely by knowing the values of  $q_0$ ,  $s_{sep}$ , and  $r_{sep}$ . Thus, we need to develop an understanding of the average distortion in the receive packet separation that we can tolerate without breaking the rate control algorithm. Let this average distortion be  $D$ . Then, when the sender and receiver rates are equal (i.e.  $s_{sep} = r_{sep}$ ), the distortion must be such that

$$\frac{r_{sep} + D}{s_{sep}} \leq 1 + r_{sep}$$

Thus,  $D \leq (s_{sep})^2$ . In other words, the average distortion must be upper bounded by the square of the value of the sender packet separation in seconds. The average distortion in an epoch is given by  $\frac{r_n - r_0 - n \cdot r_{sep}}{n}$ . Since we are averaging over a window of  $n$  samples, the distortions within the window do not impact the computed average distortion at the end of the epoch, and only the "snapshots" of the queue sizes for the  $0^{th}$  and the  $n^{th}$  packets impact the computed value of the distortion. Thus, increasing the sample size can help to reduce the effect of distortion. If we average of  $n$  samples, the "relative distortion" between  $q_0$  and  $q_n$  must be equal to  $nD$  (to produce an average distortion of  $D$ ). Further, since we compute the running average of  $r_{sep}$  and allocate a weight of 0.25 to the computed

average in the current epoch, the relative distortion in the current window must be  $4nD$  to introduce an average distortion of  $D$  at the end of one epoch (assuming that we start the epoch with no distortion). Figure 14 shows the relative distortion that the rate control algorithm can tolerate and still operate correctly for a variety of bandwidths (for  $n = 6$ ). From this figure, we see that for a per-flow rate of 100Kbps, the tolerable relative distortion is approximately 150 ms. If the network introduces larger relative distortions, then the sustainable per-flow rate will reduce correspondingly. In the discussion above, it is important to bear two things in mind: (a) the requirement of  $D \leq (s_{sep})^2$  is conservative and we may be able to tolerate larger distortions than considered above, and (b) our analysis is very coarse and is provided more as an intuition than a rigorous treatment of the acceptable operation range of the algorithm.

## VI. RELATED WORK AND SUMMARY

We place our work in the context of related work in two areas: rate control for wireline and wireless networks, and service differentiation.

The seminal work of Chiu and Jain in [6] showed that the LIMD rate control paradigm is the only robust approach for achieving convergence to fairness from any arbitrary starting configuration. It has thus formed the basis of most currently deployed rate control algorithms that are deployed today, including the different flavors of TCP. Recently, rate control algorithms for supporting multimedia flows in the Internet have also used the LIMD approach. As we mentioned in Section II, there are two aspects to LIMD: (a) the increase and decrease policies, and (b) determining when to increase or decrease (depending on whether congestion is predicted or not). Most wireline rate control algorithms use packet losses as the primary indication of congestion, and execute the rate control algorithm at the sender. As we noted in Section 2, neither of these decisions is appropriate for wireless networks. Kunniyur and Srikant showed in [13] that for loss based LIMD algorithms to work well in wireless networks, the sender must be able to exactly distinguish between random losses and congestion losses. Unfortunately, Baiz and Vaidya showed in [3] that most currently proposed “loss predictors” (i.e. metrics that are used to distinguish random losses from congestion losses) such as those based on the congestion detection metrics in [4], do not work well in practice. These results further motivate our approach of not using packet losses as the primary metric for congestion detection. In WTCP [14], as in this work, we use the ratio of the receiver to sender inter-packet separation in order to determine whether to initiate the increase phase or the decrease phase of the LIMD algorithm. Baiz and Vaidya also use this metric in [3], but only to predict if the loss is congestion-related or not. We believe that detecting and reacting to incipient congestion is one of the powerful mechanisms in our approach. Even in wireline networks, algorithms such as TCP Vegas have tried to detect and react to incipient congestion and reduce or eliminate packet loss [4]. However, the specific mechanisms of TCP Vegas are prone to unfairness and are not applicable for low bandwidth wireless networks (because round trip time is highly affected by packet bursts in TCP Vegas), as shown in [14].

While rate control has been extensively studied for over two decades, service differentiation is a relatively new topic. Recently, there have been several papers that have discussed the idea of providing relative service differentiation in terms of rate and delay within the context of the IETF Diffserv approach [10]

End-to-end algorithms to achieve service differentiation have necessarily focused only on rate rather than delay because they do not assume any network-level support [16]. Our work is similar in spirit to these approaches, because all of the approaches use some notion of weighted rate fairness that can be achieved by modifying the LIMD algorithm. Specifically, the weighted increase algorithm that we have discussed in this paper is a restricted case (congestion avoidance only) of the “schizophrenic TCP” approach proposed in [7]. Both approaches suffer from the restriction that they do not work well for a large range of weights. This problem is overcome in the weighted decrease algorithm that we briefly mentioned in Section III.

In summary, we have proposed a mechanism for service differentiation using end-to-end rate control based on four key principles: (a) we use a receiver-based weighted rate control variant of LIMD to achieve service differentiation in rate, (b) we use the ratio of receiver to sender inter-packet separation as the metric to detect congestion rather than packet loss or explicit congestion notification from the network, (c) we use a graded decrease algorithm that reacts gently to incipient congestion and aggressively to persistent congestion, and (d) we use a weighted increase algorithm (and an equivalent weighted decrease algorithm) to achieve weighted rate fairness. Our algorithm is primarily targeted towards low bandwidth networks with per flow rates of 100Kbps or less. Our current work focuses on developing better loss predictors and achieving a more sophisticated service model for service differentiation.

## REFERENCES

- [1] J. Agosta and T. Russle. *CDPD: Cellular Digital Packet Data Standards and Technology*. McGraw Hill, New York, NY, 1997.
- [2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1997.
- [3] S. Baiz and N. H. Vaidya. Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver. In *Proceedings of IEEE ASSET*, Dallas, Texas, March 1999.
- [4] L. Brakmo and L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, October 1995.
- [5] H. Chaskar, T. V. Lakshman, U. Madhow. TCP Over Wireless with Link Level Error Control: Analysis and Design Methodology, [http://tesla.csl.uiuc.edu/madhow/publications/tcp\\_wireless.ps](http://tesla.csl.uiuc.edu/madhow/publications/tcp_wireless.ps).
- [6] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, vol. 17, no. 1, June 1989.
- [7] J. Crowcroft and P. Oechslin. Weighted Proportionally Fair Differentiated Service TCP. *ACM Computer Communications Review*, vol. 28, no. 3, July 1998.
- [8] D. Eckhardt and P. Steenkiste. Improving Wireless LAN Performance via Adaptive Local Error Control. *Proceedings of IEEE International Conference on Network Protocols*, Austin, Texas, October 1998.
- [9] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transaction on Networking*, vol. 1, no. 4, December 1995.
- [10] IETF Working Groups: Transport Area-Differentiated Services (diffserv). <http://www.ietf.org/html.charters/diffserv-charter.html>.
- [11] T. Kim, S. Lu, and V. Bharghavan. Improving Congestion Control Performance Through Loss Differentiation. *Proceedings of IEEE International Conference on Computer and Communication Networks*, Boston, Massachusetts, October 1999.
- [12] T. Kim, S. Lu, and V. Bharghavan. Loss Proportional Decrease Based Congestion Control in the Future Internet. *TIMELY Research Report*, July 1999.
- [13] S. Kunniyur and R. Srikant. Fairness of Congestion Avoidance Schemes in Heterogeneous Networks. *Proceedings of the International Teletraffic Congress*, Edinburgh, Scotland, June 1999.
- [14] P. Sinha, N. Venkitaraman, R. Sivakumar and V. Bharghavan. WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks. In *Proceedings of ACM Mobicom*, Seattle, Washington, August 1999.
- [15] W. Stevens. *TCP/IP Illustrated*, Volume 1. Addison Wesley, 1994.
- [16] Z. Wang. User-share Differentiation (USD) Scalable Bandwidth Allocation for Differentiated Services. *Internet Draft*, November 1997.