Improving TCP Smoothness by Synchronized and Measurement-based Congestion Avoidance

Chi Zhang
College of Computer Science
Northeastern University
Boston, MA 20115
czhang@ccs.neu.edu

Vassilis Tsaoussidis
College of Computer Science
Northeastern University
Boston, MA 20115
vassilis@ccs.neu.edu

ABSTRACT
In this paper, we observe that although multiplicative decrease is necessary to accomplish fairness in congestion control, it does not inevitably sacrifice system throughput, as long as the system operates between the knee and the cliff, according to an equation. However, even when the system throughput is relatively stable, end users of real-time applications do not necessarily experience a smooth traffic, mainly caused by the unsynchronized window adjustments due to random congestion indications. We analyzed and evaluated the negative impact of random window adjustments on smoothness, short-term fairness, and even long-term fairness measured by a novel fairness metric defined in this paper. Therefore, we propose an experimental congestion avoidance mechanism to improve TCP smoothness for media-streaming applications. The mechanism relies on a fine-grained RTT estimation to measure the network condition, and coordinates the upward and backward window adjustments to abolish the damage of unsynchronized window control. Congestive packet drops are reduced by a new control parameter γ, and the bottleneck queue length can also be controlled in an end-to-end way. Simulation results confirm that the new mechanism enhance significantly the smoothness and fairness, without a cost of responsiveness. In fact, by enabling a new parameter δ, the responsiveness can be even enhanced when the bandwidth is under-utilized.

Categories and Subject Descriptors
C.2.6 [**Computer-Communication Networks**]: Internetworking – *Standards (e.g. TCP/IP).*

General Terms
Design, Performance.

Keywords
Congestion Control, Fairness, Smoothness, Responsiveness, AIMD, Real-time Applications, TCP-friendly Protocols.

NOSSDAV'03, June 1-3, 2003, Monterey, California, USA.

1. INTRODUCTION
Transmission control of standard TCP [1] is based on the Additive Increase / Multiplicative Decrease (AIMD) window adjustment strategy [3] that exploits available bandwidth, avoids persistent congestion, and achieves system fairness. Traditional AIMD is a somewhat "blind" mechanism, in the sense that the congestion window increases steadily until the occurrence of congestion, which necessitates the subsequent error recovery. That is, congestion control mechanism itself is the natural cause of congestion, and congestion can be detected only by packet drops.

The window adjustments of TCP Vegas [2] take a congestion-avoidance approach. Vegas defines a BaseRTT to be the minimum of all measured RTTs, and ExpectedRate to be the ratio of congestion window to BaseRTT. The sender measures the ActualRate based on the sample RTTs. If the difference between the ExpectedRate and the ActualRate is below a threshold α, the congestion window increases linearly during the next RTT; if the difference exceeds another threshold β, Vegas decreases the congestion window linearly during the next RTT. According to [2], TCP Vegas achieves better transmission rates than standard TCP. However, [8] shows that Vegas can not guarantee fairness.

While TCP congestion control is basically appropriate for bulk data transfers, some real-time applications such as media-streaming find the standard multiplicative decrease by a factor of 2 upon congestion to be unnecessarily severe, as it can cause data-rate oscillations and even transmission gaps [6]. TCP-friendly protocols therefore have been proposed with two fundamental goals: (i) to achieve smooth backward adjustments; this is done by increasing the window decrease ratio during congestion, and (ii) to compete fairly with TCP flows; this is approached by reducing the window increase step according to a steady-state TCP throughput equation [11]. That is, TCP friendly protocols favor *smoothness* by using a gentle backward adjustment upon congestion, at the cost of lesser *responsiveness* - through moderated upward adjustments.

TCP Friendly Rate Control (TFRC) is an equation-based TCP-Friendly congestion control protocol for unicast applications [6]. The sender explicitly adjusts its sending rate as a function of the measured rate of loss events, to compete fairly with TCP. A loss event consists of one or more packet drops within a single round-trip time. The receiver calculates the loss event rate and reports feedback to the sender. The benefit of TFRC is its "gentle" rate regression upon congestion.

GAIMD [14] is a parameterized TCP-friendly protocol. It generalizes AIMD congestion control by parameterizing the additive increase value α and multiplicative decrease ratio β[1]. For the family of TCP(α, β) protocols, authors of [14] derive a simple relationship between α and β to be friendly to standard TCP (α = 1, β = 1/2). Based on experiments, they propose a β = 0.875 as the appropriate ratio for smooth backward adjustment, and a moderated increase value α = 0.31 to achieve TCP friendliness. In [15], we investigated the interrelation of TCP smoothness and responsiveness by studying the impact of network conditions on the friendliness-oriented α/β tradeoff. We confirmed experimentally that, in general, smoothness and responsiveness constitute a tradeoff; however, we uncover undesirable dynamics of the protocols when the network or flow characteristics do not follow a prescribed and static behavior. For example, we showed that moderated upward adjustments (as a result of the tradeoff for smoothness) confine the protocol's capability to exploit resources that become available rapidly, where responsiveness is the dominant factor. Also, smooth backward adjustments of existing flows embarrass the fair and efficient growth of new incoming flows.

Therefore, the challenge does not lie in simply achieving smooth congestion control, but rather in providing smoothness along with bandwidth efficiency, fairness, responsiveness, as well as controlled queue length. Responsiveness to variations of bandwidth availability facilitates quick adjustments for rate-adaptive real-time applications in a dynamic environment. Queuing delay is also a concern here, because it affects the subjective performance of delay-sensitive applications. Furthermore, we realize that previous theoretical work on congestion control [3, 9, 11] ignores some basic factors which are essential to the understanding of smoothness. For example, [3] [11] (and hence the TCP-friendly protocols based on [11]) overlook the role of bottleneck queue in the dynamics of congestion control (see section 2.1). Analyses based on fluid model [9] do not discuss the impact of unsynchronized multiplicative decrease on the performance smoothness and system fairness (see section 2.2).

In this paper, we provide an intuitive interpretation to the dynamics of congestion control. We first extend the AIMD analysis model presented in [3], by taking into account the role of the bottleneck queue. We observe that although multiplicative decrease is necessary to accomplish fairness, it does not necessarily sacrifice system throughput, as long as the system operates between the knee and the cliff defined in [3]. We derive an analytical expression of the knee and the cliff. Based on that, we provide an equation for adaptively setting an efficient window decrease ratio. We further emphasize that in a real system multiplicative window decreases are often unsynchronized among competing TCP flows, due to random congestive drops. We analyze the negative impact of random multiplicative decrease on the short-term fairness, and hence the throughput smoothness experienced by end users. We argue that the major obstacle for achieving smoothness is the unsynchronized window adjustments. We also reveal that random window decreases can undermine even long-term fairness, if measured by worst-case fairness, a new metric proposed to provide a tight bound on fairness.

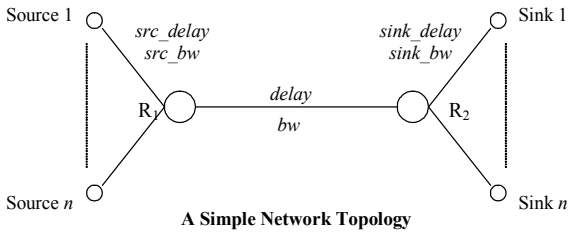
[1] The notation of AIMD parameters matches coincidentally the threshold's notation in TCP Vegas.

Based on these observations, we propose an experimental congestion avoidance mechanism to improve the sending-rate smoothness for real-time applications, within the framework of bandwidth efficiency and fairness. The mechanism relies on a fine-grained RTT estimation to measure the network condition, and coordinates the upward and backward window adjustments to abolish the damage of unsynchronized window control on throughput smoothness. It introduces new control parameters adaptable to the current network condition. Congestive packet drops can be avoided by defining an adaptive parameter γ, which determines the window decreasing ratio when the level of contention exceeds a threshold that indicates an upcoming congestion. Therefore damaging transmission gaps due to packet losses are reduced, and queuing delays can also be controlled in an end-to-end approach. Our simulation results verify that the new mechanism enhances significantly the smoothness and fairness, without compromising the responsiveness. In fact, by enabling a new parameter δ, the responsiveness can be even improved when bandwidth underutilization is detected. Notably, our mechanism can be easily adapted and incorporated into unreliable transports for real-time applications.

The paper is organized as follows: In section 2 we discuss the dynamics of congestion control and provide some observations on improving the smoothness of TCP 'sending rate'. In section 3, we describe and justify an experimental congestion avoidance mechanism. Simulation results are presented in section 4 and our conclusion is summarized in section 5.

2. DISCUSSIONS ON TCP SMOOTHNESS

2.1 The Dynamics of Congestion Control



A Simple Network Topology

First we extend the analysis model of [3] by taking into account the role of bottleneck queue. Consider a simple network topology shown above, in which the link bandwidth and propagation delay are labeled. In our scheme, *n* TCP flows share a bottleneck link with capacity of *bw*, and the round trip propagation delay is $RTT_0 = 2 * (src\_delay + delay + sink\_delay)$. Since our focus in this subsection is on the overall system behavior, we define the aggregated congestion window size at time *t* as:

$$cwnd(t) = \sum_{i=1}^{n} cwnd_i(t) \quad (1)$$

where $cwnd_i(t)$ is the window size of the i^{th} flow. Consequently, the system throughput at time *t* can be given by the following equation:

$$throughput(t) = \frac{cwnd(t)}{RTT(t)} = \frac{cwnd(t)}{RTT_0 + qdelay(t)} \quad (2)$$

where $qdelay(t)$ is the queuing delay at the bottleneck router R_1. As can be seen from (2), the throughput is not only a function of

the congestion window, but also a function of the queuing delay, which was not incorporated into the analyses in [3, 11].

Assume all flows are in the additive increase stage. First consider the case where *cwnd(t)* is below the point *knee* [3]:

$$cwnd_{knee} = RTT_0 \cdot bw \quad (3)$$

Then there is no *steady* queue build-up[2] in R_1 (i.e. *RTT(t)* = RTT_0), and according to (2), the throughput grows in proportion to *cwnd*. The bottleneck capacity is not fully utilized until *cwnd* increases to $cwnd_{knee}$.

If *cwnd(t)* increases further beyond $cwnd_{knee}$, however, the system displays different dynamics. The bottleneck queue starts to build up, after the bottleneck capacity is saturated. Rewrite *cwnd(t)* as:

$$cwnd(t) = cwnd_{knee} + \Delta w(t) \quad (\Delta w(t) > 0) \quad (4)$$

Since the bottleneck link can transmit at most $cwnd_{knee}$ packets in one RTT_0 (see (3)), $\Delta w(t)$ packets will linger in the queue. Hence the steady queuing delay at the bottleneck will be:

$$qdelay(t) = \Delta w(t) / bw \quad (5)$$

Intuitively, the system throughput is bounded by the physical capacity *bw*, in spite of the increase of *cwnd(t)* beyond the knee, because *qdelay(t)* in the denominator of (2) grows as well. This is confirmed by the following computation:

$$\begin{aligned} throughput(t) &= \frac{cwnd_{knee} + \Delta w(t)}{RTT_0 + qdelay(t)} \\ &= \frac{RTT_0 \cdot bw + qdelay(t) \cdot bw}{RTT_0 + qdelay(t)} \\ &= bw \end{aligned} \quad (6)$$

The system dynamics can be continuously described by equations (4) – (6), until the queue length $\Delta w(t)$ reaches the maximum buffer size, i.e. when *cwnd* touches the point *cliff*[3]

$$cwnd_{cliff} = (RTT_0 + \max qdelay) \cdot bw \quad (7)$$

TCP senders then multiplicatively decrease their congestion window, after packet losses due to buffer overflow are detected.

The computation of (6) demonstrates that increasing *cwnd* beyond the knee does not enhance further the system throughput, but only results in increasing queuing delay. However, our analysis also indicates that some queue build-up is inevitable, in order to provide fairness-oriented AIMD algorithm an operating scope where the system throughput fully exploits the bottleneck bandwidth. More precisely, although multiplicative decrease is necessary to accomplish fairness dynamically [3, 15], it does not necessarily mean that the throughput will be sacrificed, as long as the system operates between the knee and the cliff. In order to prevent the system from operating below the knee where bandwidth is under-utilized, and meanwhile maintain adequate AIMD oscillation (which affects the speed to converge to fairness [15]), an efficient window decreasing ratio could be

$$\beta = \frac{cwnd_{knee}}{cwnd_{cliff}} = \frac{1}{1+k} \quad (8)$$

$$where\ k = \frac{\max qdelay}{RTT_0} = \frac{\max qdelay \cdot bw}{RTT_0 \cdot bw} = \frac{BufferSize}{RTT_0 \cdot bw}$$

When the bottleneck buffer size equals to delay-bandwidth product, $k = 1$ and $\beta = 0.5$. Equation (8) corroborates that an efficient window decrease ratio depends on the network settings. It calls for a measurement-based congestion control scheme that can adapt the control parameters to the network condition.

2.2 Observations on Improving TCP Traffic Smoothness.

Assume the system operates between the knee and the cliff in equilibrium, where the overall system throughput is kept at maximum and therefore relatively stable. Does this mean that each end user will observe a smooth throughput? Before we go one step further, an important intricacy need to be examined. The analysis of [3] assumed a synchronized model, meaning that all flows synchronously adjust backward upon congestion indication. However, our simulation experience confirms the early findings [4] that global synchronization rarely happens even with drop tail buffer. That is, in a real system, packet losses do not occur to all flows when the bottleneck buffer overflows. Some flows experiencing early packet drops reduce their sending windows quickly, which might bring about partial queue draining. This could leave sufficient space for additive increase afterwards, and hence the remaining flows keep growing. Due to this *partial* backward adjustment upon congestion, from the system perspective, the multiplicative decrease ratio of the aggregated window is higher than the ratio β an individual flow adopts. The selection of which flows to drop is random by nature. With active queue management, such as RED [7], random congestion indications are explicitly performed.

The detailed analysis on the impact of unsynchronized and random multiplicative decrease on system fairness and the queue length is out of the scope of this paper. The work based on fluid model [9], which takes into account both the random multiplicative decrease and the role of bottleneck buffer, shows that the system can still converges to fairness. However, our simulation results (see section 4) reveal that it takes very long time to converge to the long-term fairness (measured at a timescale of connection time), when evaluated by a new fairness metric proposed in section 4.2.

We are particularly interested in the impact of random multiplicative decrease on the smoothness observed by end users. Authors in [13] show that smoothness is directly related to the short-term fairness (measured at a time scale of several RTTs). We now give our intuitive explanation based on the notion of random multiplicative decrease. Assume an adequate level of long-term fairness is achieved. Similar to (2), the throughput of the i^{th} flow at time *t* is:

$$throughput_i(t) = \frac{cwnd_i(t)}{RTT_0 + qdelay(t)} \quad (9)$$

Since *qdelay(t)* is common to all flows, the throughput distribution among flows at time *t* depends mainly on the difference in $cwnd_i(t)$. Obviously, unsynchronized multiplicative decrease degrades the short-term fairness, due to random congestive drops that permit some flows to grow beyond their fair shares, at the cost of the other flows forced to decrease, in a short period of time. Assuming an adequate level of long-term fairness is achieved (due to the inherent characteristic of randomness in selecting which flows to drop), flows consuming extra bandwidth at one time period must pay back the credit to the flows consuming less bandwidth, at some other time period. As a result

[2] There could be *temporary* queue build-up in this scenario, due to the traffic burstiness. This is neglected to simplify our analysis.

[3] The intuitive concept of knee and cliff was first introduced in [3]. Here we give an analytical expression.

of the long-term fairness accomplished without short-term guarantee, individual end users are unavoidably subject to throughput oscillations, even though the overall system throughput is smooth. Therefore, we argue that the major obstacle for achieving smoothness is the unsynchronized and random window adjustments.

If the system follows the synchronized model presented in [3], however, window upward and downward adjustments are coordinated after the system reaches equilibrium. Therefore short-term fairness is not damaged. Given the condition of a stable system throughput, the bandwidth allocated to each end user will be also smooth along the time. From the perspective of an individual flow, multiplicative decrease of $cwnd_i(t)$ in equation (9) does not necessarily affect the throughput, if $qdelay(t)$ in the denominator decreases correspondingly because of synchronized backward adjustments of all flows. Therefore, smoothness can be achieved along with bandwidth efficiency and fairness. Traditional wisdom might argue that global synchronization is more likely to cause the system to operate below the knee where bandwidth is under-utilized. However, as shown by equation (8), this can be avoided by setting the multiplicative decrease ratio adaptively.

3. AN EXPERIMENTAL PROTOCOL

Based on the observations in section 2, in this section we propose a synchronized and measurement-based congestion avoidance mechanism to improve the smoothness of TCP sending-rate for real-time applications, within the framework of bandwidth efficiency and fairness, and without compromising responsiveness to variations of bandwidth availability.

3.1 The Congestion Avoidance Mechanism

The sender measures the fine-grained RTT (see section 3.2 for implementation details). It records the minimum RTT and the maximum RTT perceived. The queuing delay can be derived by deducting the minimum RTT (which corresponds to the round-trip propagation delay) from the current RTT measured. The slow start mechanism of TCP is not modified, to allow the queue length to fully grow to overflow during initialization, so that the maximum RTT with the highest queuing delay could be observed before the congestion avoidance stage takes over.

In the congestion avoidance stage, the additive increase speed (α = 1) of standard TCP is untouched, and the sender halves the congestion window upon a packet loss (β = 0.5). However, the standard congestion control is complemented with the following congestion avoidance mechanism. Upon the detection of the following condition:

$$\frac{qdelay(t)}{\max qdelay} = \frac{RTT(t) - \min RTT}{\max RTT - \min RTT} \geq Th_{upper} \qquad (10)$$

where the threshold Th_{upper} is experimentally set to be 0.5, the congestion window is decreased after one RTT, with window decrease ratio γ set to be:

$$\begin{aligned} \gamma &= \frac{cwnd_{knee}}{cwnd(t)} = \frac{bw \cdot \min RTT}{bw \cdot RTT(t)} \\ &= \frac{\min RTT}{\min RTT + Th_{upper} \cdot \max qdelay} \\ &= \frac{\min RTT}{Th_{upper} \cdot \max RTT + (1 - Th_{upper}) \min RTT} \\ &= \frac{1}{1 + Th_{upper} \cdot k} \end{aligned} \qquad (11)$$

where k is defined in Equation (8). Note that γ-based multiplicative decrease is carried out one RTT after the condition of (10) is detected, in order to assure that no sender adjusts backward before the condition is observed by all the other senders. Equation (11) bears some similarities with equation (8). Both of them assume a synchronized model, and intend to prevent the system from operating below the knee. However, equation (8) presumes that drop-tail buffer synchronously feeds back congestion indications (i.e. packet drops) to all flows, which is not true in reality. In contrast, backward adjustments based on (11) is triggered by a threshold on the queuing delay, which can be synchronously detected by all senders. From the perspective of congestion control, γ determines the window decreasing ratio when the level of contention exceeds a threshold that indicates an upcoming congestion. Since γ-based multiplicative decrease is applied when the system is half way between the knee and the cliff, γ is higher than β.

Another optional control parameter δ is introduced, in order to enhance the additive increase speed when the network is under-utilized. More specifically, when the following condition persists:

$$\frac{qdelay(t)}{\max qdelay} = \frac{RTT(t) - \min RTT}{\max RTT - \min RTT} \leq Th_{lower} \qquad (12)$$

the queue is relatively close to empty and the bandwidth is possibly under-utilized. The additive increase adopts a faster speed: δ = 2. The threshold Th_{lower} is experimentally set to be 0.1. Once the threshold is exceeded, δ hands over the control to α.

3.2 Fine-grained RTT Measurement

In our fine-grained RTT measurement, the sender records the system clock each time a packet is sent. When the corresponding ACK is returned, the sender reads the clock again and computes the RTT. As in the coarse-grained RTT estimation in standard TCP, fine-grained RTT measurement is taken only for packets that have been sent just once.

However, there are two interfering factors to be canceled: (i) TCP delayed acknowledgement is widely deployed. The receiver transmits an ACK for every second data packets received. If a second data packet is not received within a given timeout, an ACK is transmitted. In the latter case, the RTT measured at the sender incorporates the ACK delay time at the receiver. (ii) TCP acknowledgement is *cumulative* ACK. When a packet is dropped due to congestion, duplicate ACKs received at the sender can not be used for RTT computation, although duplicative ACKs are triggered by the data packets that do get through the bottleneck link and provide valuable information on the maximum queuing delay when the bottleneck buffer overflows.

Our solution is attaching to the cumulative ACK the sequence number of the data packet that triggers this ACK, as a TCP header option. Therefore RTT computation can be conducted even with

duplicate ACKs: use the attached sequence number to compute the RTT for the data packet that triggers the duplicative ACK. When the ACK is triggered by a timeout on ACK delay, the receiver does not attach the option to the ACK, and RTT computation is skipped at the sender.

3.3 Mechanism Justification and Comparison

The protocol introduces a number of novel features:

(a) It is measurement-based. It relies on a fine-grained RTT estimation to measure the network condition, beyond the traditional congestion indication - packet drops. Therefore, sophisticated methods are enabled.

(b) It follows a synchronized model. The upward and downward window adjustments of all flows in the system are coordinated, so that the system can escape the damage of unsynchronized and random window adjustments on short-term fairness and throughput smoothness. Notably the smoothness is accomplished without compromising responsiveness to the variations of bandwidth availability in a dynamic environment. The responsiveness is maintained through the unchanged additive increase rate. In fact, by enabling the parameter δ, the responsiveness can be even improved when bandwidth underutilization is detected.

(c) It avoids congestion by scheduling backward window adjustments well before the occurrence of congestion and consequent packet drops. Congestion avoidance prior to congestion reduces packet drops, which cause the damaging transmission gaps, and timeout events that disturb the system stability.

(d) Its control parameters are adaptive. Not only can the multiplicative decrease ratio be tuned (through γ) to prevent the system from operating below the knee, but also the additive increase speed can be enhanced (through δ) when the bandwidth is under-utilized.

(e) It's an end-to-end solution that controls the average queue length through the thresholds in equations (10) and (11), without the deployment of active queue management in routers.

Above all, these features do not violate the established framework of fairness-oriented AIMD. Notably, TCP Vegas also has features (a) (c) and (e). However, its fine-grained RTT measurements do not take into account the impact of delayed ACK. More importantly, its congestion avoidance mechanism follows an Additive Increase and Additive Decrease (AIAD) scheme, which does not guarantee fairness [3]. Window adjustment strategy of Vegas can't be categorized as synchronized or unsynchronized, because the design philosophy of Vegas is to maintain the status quo if each flow has a couple of extra packets in the buffer and the bottleneck bandwidth is fully utilized [2]. This lack of concern for fairness is verified by experiments in [8], and amplified by our new fairness metric shown in section 4.3.1.

4. EVALUATION AND ANALYSIS

4.1 Testing Plan and Methodology

The protocol was evaluated on ns-2 network simulator [10]. Simulations were conducted over the network topology shown in section 2.1. By default, the propagation delay is 30ms for the bottleneck link and 5ms for the access links. Simulations with diverse propagation delays were also conducted. The bottleneck buffer size at R_1 was set to be the product of the round-trip propagation delay and the bottleneck capacity, which varied from 10Mbps to 100Mbps in simulations. The deployment of RED at bottleneck router R_1 was also tested, in order to investigate the effect of random multiplicative decreases explicitly enforced by RED, in comparison with the implicit random decrease of drop-tail buffer. The settings for RED were as per [5]. Specifically, *min_th* = buffer_size / 6, *max_th* = buffer_size / 2. Also we selected *max_p* = 0.1 and *w* = 0.002.

To evaluate the protocol's capability to exploit the bandwidth that becomes available rapidly (i.e. the responsiveness), we also created a scenario of temporary "blackouts" due to mobile handoffs, during which all transmitted packets were lost. An error model was inserted into the access links to the receivers (assumed in wireless domain), with 100% packet dropping rate.

We selected 7 protocol configurations: Reno (α = 1, β=0.5), Reno with RED configured at R_1, TCP Vegas, Reno with the γ parameter, Reno with γ and δ (= 2), and smooth TCP (α = 0.31, β = 0.875) [14], and TFRC [6]. We were particularly interested in the comparison between the γ mechanism and TCP(0.31, 0.875), whose smoothness is achieved by increasing the window decrease ratio, at the cost of lesser responsiveness. TCP Vegas was selected to demonstrate the unfairness of Additive Increase and Additive Decrease (AIAD) congestion avoidance. TFRC is an equation-based rate control mechanism for unicast applications. Since TFRC is not a reliable transport control, it's somewhat unreasonable to compare its performance with reliable TCP protocols. Nonetheless, TFRC was included as a reference of smoothness. Notably, our mechanism can be easily adapted for unreliable media-streaming.

4.2 Performance Metrics

Long-term Fairness is measured by the *Fairness Index*, defined by [3]:

$$FairnessIndex = \frac{\left(\sum_{i=1}^{n} throughput_i\right)^2}{n\sum_{i=1}^{n} throughput_i^2}$$

where $throughput_i$ is the throughput of the i^{th} flow, measured at a time scale of connection time. This Fairness Index provides a sort of "average-case" analysis used by most researchers. In order conduct a "worst-case" analysis and provide a tight bound on fairness, we propose the *Worst-Case Fairness* as:

$$WorstCaseFairness = \frac{\min_{1\le i\le n} throughput_i}{\max_{1\le i\le n} throughput_i}$$

The range of worst-case fairness is also in [0, 1], with 1 representing the greatest fairness. As an example demonstrating why worst-case fairness is introduced, consider a scenario of 6 flows, the throughputs of which are 9 Mbps, 9.5 Mbps, 8.5 Mbps, 9 Mbps, 9 Mbps, and 6 Mbps, respectively. The traditional "average-case" fairness index is 0.982, while the worst-case fairness is 0.667. Compare this scenario with a perfectly fair case in which all flows achieve 9.5 Mpbs, and both the "average-case" fairness index and worst-case fairness index are 1.0. The difference between the first scenario and the ideal case can't be obviously distinguished by the "average-case" fairness index. In the first scenario, the system is fair in general, but is particularly

unfair to the 6th flow. This unfairness to a very small fraction of flows can only be captured by the worst-case fairness.

To investigate the performance smoothness observed by end users, allotted throughput $throught_i(t)$ for the i[th] flow is sampled at a time scale of several RTTs, throughout the entire connection. In our simulations, the sample time period is set to be 0.5 seconds (3 to 6 RTTs). Following the metric in [13], we use *Coefficient of Variation (CoV)* to gauge the throughput smoothness experienced by flow *i*:

$$CoV_i = \frac{\sqrt{E_t\{thoughput_i^2(t)\} - E_t\{thoughput_i(t)\}^2}}{E_t\{thoughput_i(t)\}}$$

where $E_t\{\}$ denotes the computation of the mean along the time. For a system with multiple flows, the system CoV is the average of CoVs of all flows. Allotted throughput is also used to compute the short-term fairness, derived from the traditional Fairness Index:

$$ShortTermFairness = E_t\left\{\frac{\left(\sum_{i=1}^{n} throughput(t)_i\right)^2}{n\sum_{i=1}^{n} throughput(t)_i^2}\right\}$$

Allotted throughput, and hence CoV and short-term fairness, are measured 15 seconds after the simulation starts, in order to just capture the system performance after it enters equilibrium state.

Bottleneck queue lengths are also traced, since it affects the end-to-end delay and hence the subjective performance of delay-sensitive applications. Queue lengths are sampled periodically at a time scale of several RTTs. The queue length is normalized by the buffer size, with normalized queue length 1 representing a full queue.

4.3 Results and Analysis

4.3.1 Fairness and Smoothness

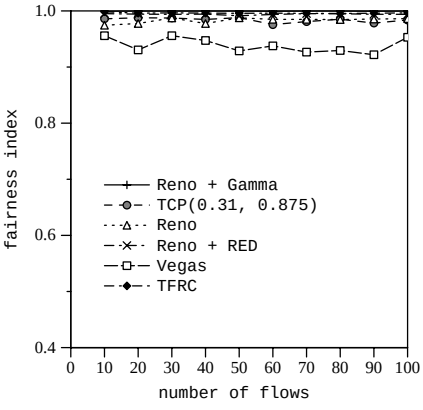


Figure 1. Fairness Index

We first conducted 10 simulations, with the number of flows varied from 10 to 100. The bottleneck link capacity scaled correspondingly, such that the fair share for each flow was 1 Mbps. Protocol performances with 100 second connection time are shown in Figures 1 – 4. If assessed by traditional fairness index (Figure. 1), all protocols achieve high level of fairness, except Vegas. However, Figure 2 reveals that random multiplicative decrease can not guarantee worst-case fairness, compared to the synchronized window adjustments of Reno + γ. As the number of flows increases, the worst-case fairness with random decrease strategy fluctuates and gradually degrades, reflecting the inherent characteristics of randomness. The deployment of RED can slightly improve the worst-case fairness of TCP flows. The overall fairness performance of Vegas is the worst.

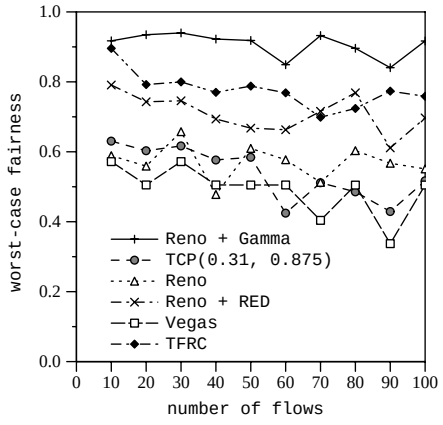


Figure 2. Worst Case Fairness

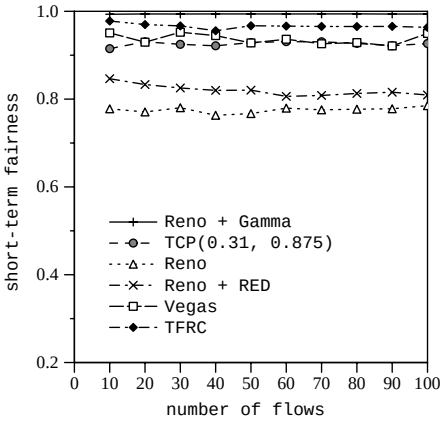


Figure 3. Short-term Fairness

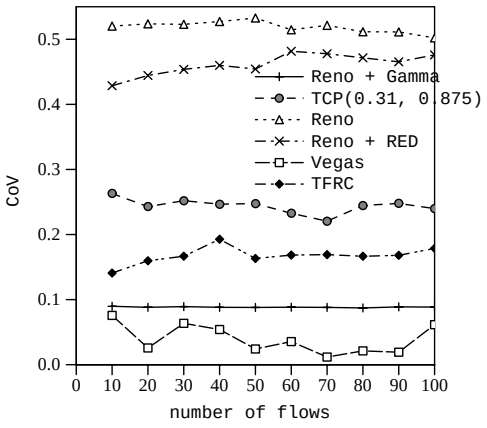


Figure 4. CoV

The short-term fairness and CoV measurements are shown in Figures 3 and 4, respectively. Reno + γ achieves higher short-term fairness and lower CoV (i.e. higher smoothness) than the protocols based on random window adjustments, including Reno, Reno + RED, TCP(0.31, 0.875) and even TFRC. Note that the smoothness of Reno + γ is achieved not by increasing the window

decreasing ratio as with TCP(0.31, 0.875). Since the buffer size is set to the round-trip propagation delay times the bottleneck bandwidth, the γ applied is around 2/3, according to equation (11). Rather, the smoothness is achieved by implementing coordinated multiplicative decreases, taken before the occurrence of congestion. Furthermore, how much to decrease is now adaptively set according to the measurement on the network condition. The lowest CoV is achieved by TCP Vegas because of its design philosophy to maintain the status quo. However, as analyzed in section 3.3, this is accomplished at the expense of fairness, due to its Additive Increase and Additive Decrease congestion avoidance strategy. Its unfairness is further amplified by the next set of tests.

Next, we fixed the number of flows at 30, and varied simulation time from 100 seconds to 1000 seconds, in order to watch the effect of connection time on performance metrics. The performances of CoV, fairness, and short-term fairness remain almost unchanged, as the connection time increases. However, Figure 5 shows that the worst-case fairness with random window adjustments grows with the connection time. This gives a clue to an explanation of the low worst-case fairness of random multiplicative decreases: The number of congestion epochs[4] experienced by the system is in direct proportion to the connection time. That is, the number of times to randomly select flows to drop is in proportion to the connection time. Assume the random-drop selection upon congestion is not biased against any flow, or the *probability* that a flow receives congestion indication upon a congestion event is *p*, where *p* is the same for all flows. According to the law of large numbers in statistics, however, the actual *percentage* of random selections in which a specific flow is chosen to drop gradually converges to *p*, only when the number of times to randomly select flows to drop grows to infinity, or more practically, when connections last for a very long time. Therefore, over time, the system appears more unbiased and the worst-case fairness is improved. However, even at time 1000 seconds, worst-case fairness with random window adjustments is still below that of Reno + γ. Furthermore, the fairness of TCP Vegas stays low, no matter how long the connection time is. The highest throughput achieved by individual flows is 75% higher "forever" than the lowest one, due to its tendency to maintain the status quo after the bandwidth efficiency is achieved. Without attempts to dynamically adjust and converge to fairness, the system loses the gravity to draw it back to a fair state once it initially enters an unfair steady state.

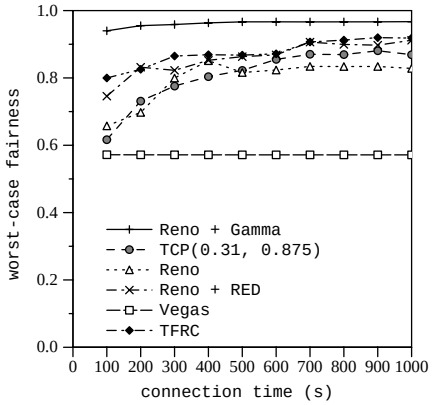


Figure 5. Worstcase Fairness with 30 Flows

[4] A congestion epoch is defined to be the time period between two successive congestion indications.

4.3.2 Queue Length

Queue lengths traces with 10 flows are displayed in Figures 11 – 16 attached to the end of this paper. As we can see, RED, Vegas and Reno + γ can control the bottleneck queue length, while standard TCP Reno periodically causes full queue length or buffer overflow. More seriously, without active queue management, TCP-friendly protocols result in a queue always close to full, due to their "gentle" regression upon buffer overflow. Both Reno + γ and Vegas control the queue lengths in an end-to-end fashion. Although Vegas has lower queuing delay, it is achieved by a window adjustment strategy that lacks concern for fairness.

4.3.3 Performance with Diverse RTTs

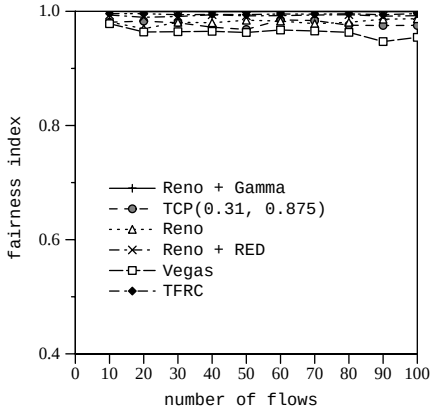


Figure 6. Fairness Index with Diverse RTTs

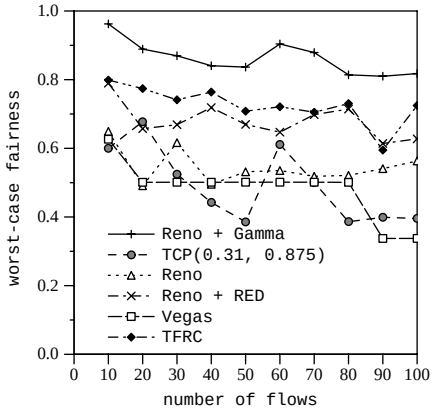


Figure 7. Worst Case Fairness with Diverse RTTs

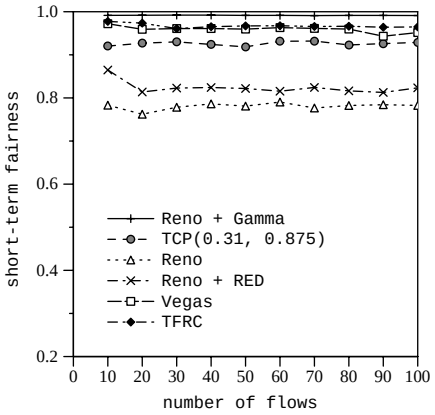


Figure 8. Short-term Fairness with Diverse RTTs

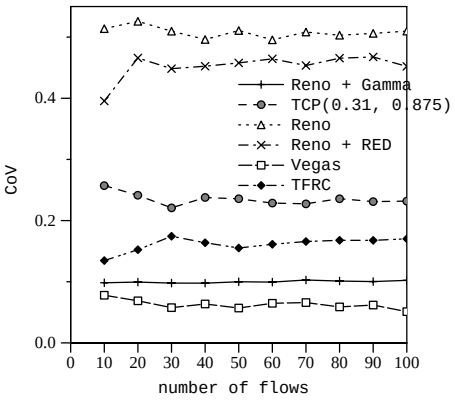


Figure 9. CoV with Diverse RTTs

We repeated the tests in section 4.3.1 with diverse RTTs. The propagation delays of access links to sink nodes are now uniformly distributed between 5 ms to 15 ms. The performances are demonstrated in Figures 6 – 9. Reno + γ still outperforms other protocols, and its synchronized congestion avoidance mechanism is not disrupted by the diverse RTTs.

4.3.4 Responsiveness in Dynamic Environment

Moreover, the high smoothness of coordinated window adjustments is achieved not at the cost of responsiveness, as shown by our next simulation over heterogeneous (wires/wireless) networks. A single flow runs on the 10Mbps bottleneck link. At time 20 seconds, the wireless access link was interrupted by a 1 second handoff period, during which all packets were lost. Since bandwidth becomes available immediately at the end of the handoff, a high sending rate increase is the desired behavior. The protocols' aggressiveness after the handoff is shown in Figure 10. Since both *cwnd* and *ssthresh* are reduced to minimum during the handoff, the additive increase speed is the dominant factor when the handoff is over. Due to the lesser responsiveness ($\alpha = 0.31$) as a result of TCP-friendly α/β tradeoff, it takes 57 seconds for TCP(0.31, 0.875) to fully recover the transmission speed, while the recovery time for Reno or Reno + γ is 19 seconds, with $\alpha = 1$. If the optional parameter $\delta = 2$ is enabled, the recovery time can be further reduced to 10 seconds. As the condition of bandwidth under-utilization is detected after the handoff is over, a faster additive increase step can be adopted. The recovery speed of Vegas is slightly below that of Reno. Notably the modified slow start [2] of Vegas hinders its capability to quickly exploit the available bandwidth even during the flow initialization stage (before the mobile handoff).

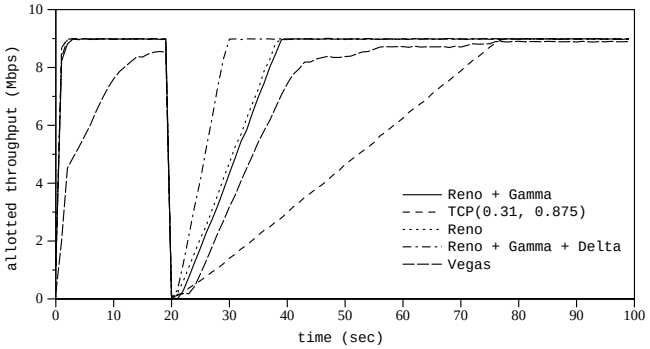


Figure 10. Thoughput with 1.0 Second Handoff

5. CONCLUSION AND FUTURE WORK

We argued that the major obstacle for achieving smoothness is the unsynchronized and random window adjustments, and that global synchronization may elude bandwidth under-utilization by adapting control parameters to the network conditions. We also revealed the negative impact of random window decreases on fairness, measured by the novel worst-case fairness index. Based on these observations, we proposed an experimental congestion avoidance mechanism that relies on fine-grained RTT measurements to coordinate window adjustments. TCP smoothness is enhanced, without compromising efficiency, fairness and responsiveness. The mechanism can also control the queuing delay of drop-tail buffers. With the current experimental protocol, the queuing delay on the reverse path could be reflected in the RTT measurements and thus affect the congestion avoidance mechanism. A receiver-oriented approach [12] can be incorporated to improve the system robustness with two way traffic.

6. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", RFC2581, April 1999.

[2] L.S. Brakmo and L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications, 13(8):1465-1480, Oct 1995

[3] D.-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", Computer Networks and ISDN Systems, 17(1):1-14, 1989.

[4] C. Diot, G. Iannaccone and M. May, "Aggregate Traffic Performance with Active Queue Management and Drop from Tail", Sprint ATL Technical Report, TR01-ATL-012501.

[5] S. Floyd, "RED: Discussions of Setting Parameters", November 1997, available from http://www.icir.org/floyd/REDparameters.txt

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications", Proceedings of ACM SIGCOMM 2000, August 2000.

[7] S. Floyd, and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.

[8] U. Hengartner, J. Bolliger and Th. Cross, "TCP Vegas Revisited", In Proceedings of IEEE INFOCOM 2000, March 2000.

[9] V. Misra, W. Gong, D. Towsley, "A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with Application to RED", ACM SIGCOMM'00, September 2000.

[10] NS-2, The Network Simulator: http://www.isi.edu/nsnam/ns/

[11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", In Proceedings of ACM SIGCOMM '98, August 1998.

[12] V. Tsaoussidis and C. Zhang, "TCP-Real: Receiver-Oriented Congestion Control", Computer Networks Journal (Elsevier), Vol. 40, No. 4, November 2002.

[13] Y.R. Yang, M.S. Kim and S.S. Lam, “Transient Behaviors of TCP-friendly Congestion Control Protocols”, In Proceedings of IEEE INFOCOM 2001, April 2001.

[14] Y.R. Yang and S.S. Lam, “General AIMD Congestion Control”, Proceedings of the 8th International Conference on Network Protocols, ICNP 2000, November 2000.

[15] C. Zhang and V. Tsaoussidis, “The Interrelation of TCP Smoothness and Responsiveness in Heterogeneous Networks”, Proceedings of the 7th IEEE Symposium on Computers and Communications, ISCC 2002, July 2002 (runner-up award winner).

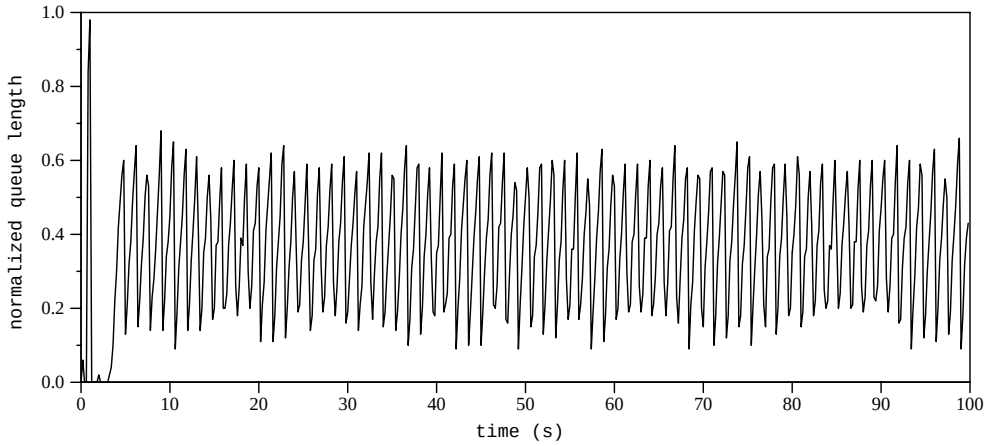


Figure 11. Queue Length of Reno + Gamma with 10 flows

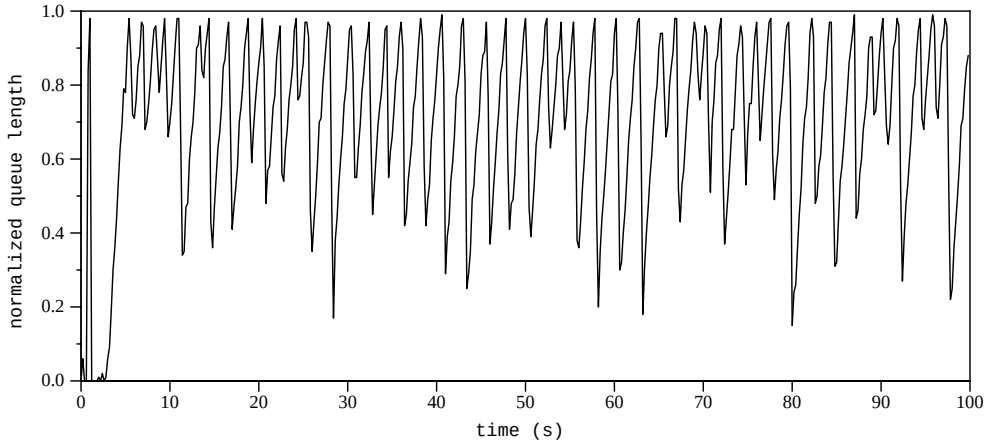


Figure 12. Queue Length of Reno with 10 flows

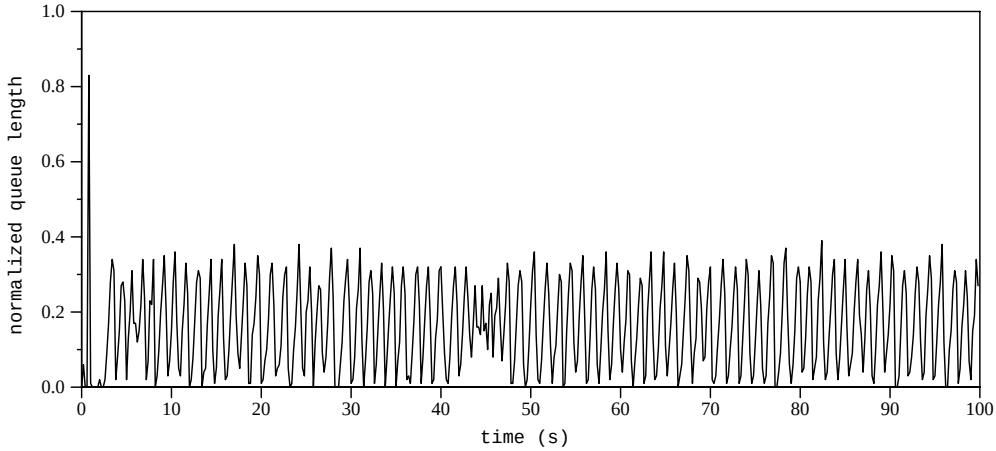


Figure 13. Queue Length of Reno + RED with 10 flows

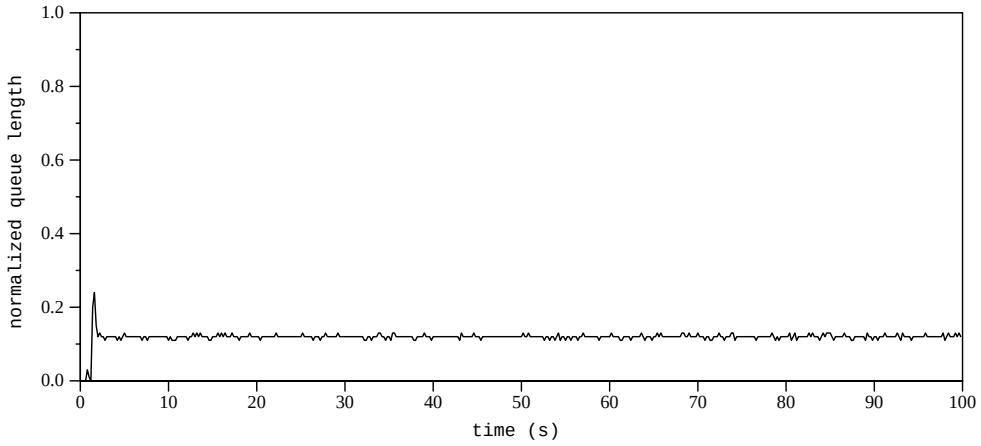


Figure 14. Queue Length of Vegas with 10 flows

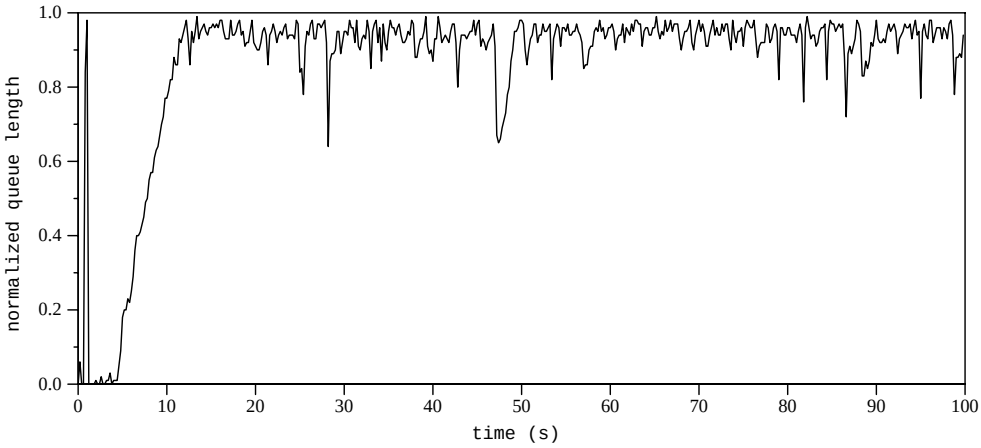


Figure 15. Queue Length of TCP(0.31, 0.875) with 10 flows

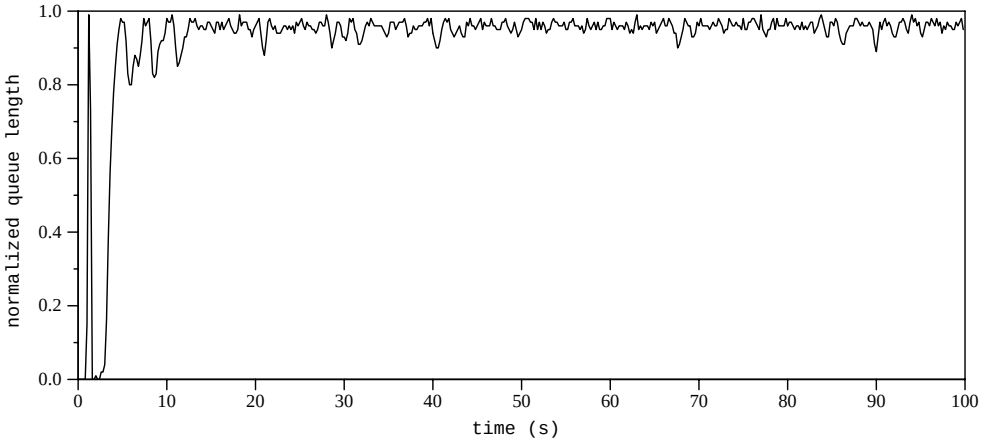


Figure 16. Queue Length of TFRC with 10 flows