TCP Smoothness and Window Adjustment Strategy

Chi Zhang, *Member, IEEE*, and Vassilis Tsaoussidis, *Senior Member, IEEE*

***Abstract*—We observe that even when the system throughput is relatively stable, end users of media-streaming applications do not necessarily experience smooth throughput, due to the unsynchronized window adjustments triggered by random congestion indications. We analyze and evaluate the negative impact of random window adjustments on smoothness, short-term fairness, and long-term fairness. We further propose an experimental congestion avoidance mechanism, namely TCP(α, β, γ, δ), based on coordinated window adjustments. The flow-level smoothness is enhanced significantly for media-streaming applications, without a cost on fairness and responsiveness. Responsiveness is even boosted when bandwidth is underutilized.**

***Index Terms*—Congestion control, fairness, multimedia communication, smoothness, transport protocols.**

I. Introduction

Transmission control of standard TCP [1] is based on the additive increase/multiplicative decrease (AIMD) window adjustment strategy [2] 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 congestive packet loss is detected, which necessitates the subsequent error recovery. In contrast, window adjustments of TCP Vegas [3] take a congestion avoidance approach. The congestion window increases linearly if the sampled RTT is relatively low, or decreases linearly if the RTT is relatively high. Although Vegas may achieve higher throughput than standard TCP [3], it cannot guarantee fairness [4].

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 throughput oscillations and even transmission gaps [5]. Throughput smoothness is crucial to the subjective performance of media-streaming. TCP-friendly protocols [6]–[10] therefore have been proposed with two fundamental goals: i) to achieve smooth downward adjustments; this is done by increasing the window decrease ratio during congestion and ii) to compete fairly with TCP flows; this is accomplished by reducing the window increase step according to a steady-state TCP throughput equation [11]. That is, TCP friendly protocols favor *smoothness* for multimedia applications by using a gentle downward adjustment upon congestion, at the cost of lesser *responsiveness*—through moderated upward adjustments.

Rate adaptation protocol (RAP) [9] is an AIMD-based protocol friendly to TCP. While it decouples congestion control from application reliability, smoothness is not a design criterion for RAP. TCP friendly rate control (TFRC) is an equation-based TCP-friendly congestion control protocol for unicast applications [7]. The sender explicitly adjusts its sending rate as a function of the measured rate of loss events, to compete fairly with TCP. The benefit of TFRC is its "gentle" rate regression upon congestion.

GAIMD [8] is a TCP-friendly protocol that generalizes AIMD congestion control by parameterizing the additive increase value α and multiplicative decrease ratio β. For the family of TCP(α, β) protocols, [8] derives a simple relationship between α and β to be friendly to standard TCP ($\alpha = 1$, $\beta = 1/2$). They propose a $\beta = 0.875$ as an appropriate smooth decrease ratio, and a moderated increase value $\alpha = 0.31$ to achieve TCP friendliness. In [12], we uncovered undesirable behaviors of TCP(0.31, 0.875) with dynamic traffics. For example, moderated upward adjustments (as a result of the tradeoff for smoothness) confine the protocol's capability to exploit bandwidth that become available rapidly, when responsiveness is the dominant factor. Also, smooth downward adjustments of existing flows embarrass the fair and efficient growth of new incoming flows.

Therefore, the challenge does not lie in simply achieving smooth transmission control, but rather in providing smoothness along with bandwidth efficiency, fairness, responsiveness, as well as controlled queue lengths [12], [13]. Unfairness means lower throughputs for some sacrificed flows when the bandwidth is a scarce resource. Responsiveness to changes of bandwidth availability facilitates quick adjustments for rate-adaptive media-streaming. Queuing delay affects the subjective performance of delay-sensitive applications.

In this paper, we observe that previous theoretical analysis on congestion control [2], [11], [14] ignores some basic factors essential to the understanding of smoothness. We also realize that although multiplicative decrease is necessary to accomplish fairness, it does not necessarily sacrifice the system throughput, since window decreases can be balanced by the queue dynamics. However, in real systems multiplicative decreases are often unsynchronized among competing TCP flows, due to random congestive drops. We argue that even when the system throughput is relatively stable, individual users of media-streaming applications do not experience smooth throughputs, mainly due to the unsynchronized window adjustments. Random window adjustments also hurt short-term, and even long-term fairness. Based on these observations, we propose an measurement-based adaptive congestion avoidance mechanism, TCP(α, β, γ, δ), to improve *flow-level* throughput

Manuscript received July 17, 2004; revised June 9, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Madjid Merabti.

C. Zhang is with the School of Computer Science, Florida International University, Miami, FL 33199 USA (e-mail: czhang@cs.fiu.edu).

V. Tsaoussidis is with the Department of Electrical and Computer Engineering, Demokritos University, Xanthi 67100, Greece (e-mail: vtsaousi@ee.duth.gr).

Digital Object Identifier 10.1109/TMM.2006.870725

smoothness. The mechanism coordinates the upward and downward window adjustments to abolish the damage of unsynchronized and random window control on throughput smoothness. While the smoothness of TCP Vegas is achieved at the expense of fairness, our new mechanism enhances significantly both smoothness and fairness, without compromising responsiveness. Responsiveness can be even boosted when bandwidth underutilization is detected. Notably, our window-adjustment strategy can be easily adapted and incorporated into unreliable or rate-based transport protocols (such as RAP [9]) designed for media-streaming applications.

The rest of this paper is organized as follows: In Section II we discuss the dynamics of congestion control and the smoothness of TCP "sending rate." In Section III, we describe and justify the experimental congestion avoidance mechanism. Simulation results are presented in Section IV and our conclusion is summarized in Section V.

II. Discussion on TCP Smoothness

A. Dynamics of Congestion Control

[2] gives the general dynamics of throughput of a network as the network load increases. The concepts of "knee" and "cliff" were first defined in [2] under the context of generic networks. We now give an analytical and intuitive explanation to this dynamics by concentrating on the window-based transmission control of TCP, and by taking into account the role of bottleneck queue. Consider a simple network topology shown in Fig. 1, in which the link bandwidth and propagation delay are labeled. 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 first focus is the overall system behavior, we define the aggregated congestion window of the system at time t as

$$scwnd(t) = \sum_{i=1}^{n} cwnd_i(t) \tag{1}$$

where $cwnd_i(t)$ is the window size of the i^{th} flow. Consequently, the system throughput at time t should be

$$throughput(t) = \frac{scwnd(t)}{RTT(t)} = \frac{scwnd(t)}{RTT_0 + qdelay(t)} \tag{2}$$

where $qdelay(t)$ is the queuing delay at the bottleneck router R_1. Thus, throughput is not only a function of the congestion window, but also a function of the dynamic queuing delay, which was not incorporated into the analyzes of [2], [11].

Assume all flows are in the additive increase stage. If $scwnd(t)$ is below the point $knee$:

$$scwnd_{knee} = RTT_0 \cdot bw. \tag{3}$$

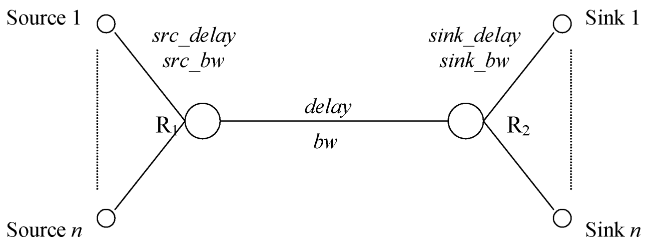


Fig. 1. Simple network topology.

Then there is no $steady$ queue buildup (i.e. $RTT(t) = RTT_0$. There could be *temporary* queue buildup due to the traffic burstiness. This is neglected to simplify our analysis) in R_1, and according to (2), the throughput grows in proportion to $scwnd$, since the bottleneck capacity is not fully utilized until $scwnd$ increases to $scwnd_{knee}$. If $scwnd(t)$ increases further beyond $scwnd_{knee}$, however, the system displays different dynamics. The bottleneck queue starts to build up, after the bottleneck capacity is saturated. Rewrite $scwnd(t)$ as

$$scwnd(t) = scwnd_{knee} + \Delta w(t)\,(\Delta w(t) > 0). \tag{4}$$

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

$$qdelay(t) = \frac{\Delta w(t)}{bw}. \tag{5}$$

Intuitively, the system throughput is bounded by the physical capacity bw, in spite of the increase of $scwnd(t)$ beyond the knee, because $qdelay(t)$ in the denominator of (2) grows as well. This can be validated by (6):

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

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

$$scwnd_{cliff} = (RTT_0 + \max qdelay) \cdot bw. \tag{7}$$

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

Equation (6) shows that increasing $scwnd$ beyond $scwnd_{knee}$ does not enhance further the system throughput, but only results in high queuing delay. However, although multiplicative decrease is necessary to accomplish fairness dynamically [2], [12], it does not sacrifice the system throughput, as long as $scwnd$ operates between $scwnd_{knee}$ and $scwnd_{cliff}$. In order to prevent $scwnd$ from operating below $scwnd_{knee}$ where bandwidth is underutilized, and meanwhile maintain adequate AIMD oscillation (for fast convergence to fairness [12]), an efficient window decreasing ratio would be

$$\begin{aligned}\beta_{opt} &= \frac{scwnd_{knee}}{scwnd_{cliff}} = \frac{1}{1+k}\\ where\ k &= \frac{\max qdelay}{RTT_0} = \frac{\max qdelay \cdot bw}{RTT_0 \cdot bw}\\ &= \frac{BufferSize}{RTT_0 \cdot bw}.\end{aligned} \tag{8}$$

With $\beta_{\rm opt}$ set by (8), $scwnd$ will decrease from the cliff down to the knee (for sufficient fairness-oriented AIMD oscillation), but not below the knee (where bandwidth is underutilized). When the bottleneck buffer size equals delay-bandwidth product, $k = 1$ and $\beta = 0.5$. Equation (8) implies that an efficient decrease ratio depends on the network settings.

B. Observations on Improving TCP Smoothness

Assume that $scwnd$ operates between the knee and the cliff in equilibrium, where the overall system throughput is maximum and thus relatively stable. However, this does NOT mean that each flow will observe a smooth throughput. The analysis of [2] assumed a synchronized model, i.e., all flows synchronously adjust downward upon congestion. However, our simulations (see Section IV-C4) confirms the early findings [15] that packet losses do not always occur to *all* competing flows when the buffer overflows, even with drop-tail buffer. Some flows experiencing early drops may reduce their windows quickly, which leads to *partial* queue draining. This could leave sufficient space for additive increase afterwards, and hence the remaining flows keep growing. Due to this *partial* window decrease upon congestion, the decrease ratio of $scwnd$ can be higher than the ratio β an individual flow adopts. The selection of which flows to drop is random by nature. With AQM such as RED [16], random congestion indications are explicitly performed.

Although the work based on the fluid model [14], which takes into account both the random window decreases and the role of bottleneck buffer, shows that the system still converges to fairness, our simulations (see Section IV) reveal that the convergence speed is slow. We are particularly interested in the impact of random window adjustments on the flow smoothness of each flow. Similar to (2), the throughput of the i^{th} flow at time t is

$$throughput_i(t) = \frac{cwnd_i(t)}{RTT(t)} = \frac{cwnd_i(t)}{RTT_0 + qdelay(t)}. \tag{9}$$

Obviously, unsynchronized multiplicative decrease degrades the short-term fairness (measured in several RTTs), due to random congestive drops that permit some flows to grow beyond their fair shares while the other flows are forced to decrease, in a short period of time. Assume an adequate level of long-term fairness (measured in the connection time) is achieved, since there is no bias in randomly selecting 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 of the long-term fairness accomplished without short-term guarantees, individual flows unavoidably see throughput oscillations, even when the system throughput is stable. Thus, we argue that the major obstacle for achieving smoothness with individual flows is the unsynchronized and random window adjustments.

If upward and downward window adjustments are synchronized in equilibrium, however, short-term fairness is not damaged. With a stable system throughput, the bandwidth allocated to each end user will be also smooth over time. Therefore, smoothness can be achieved along with bandwidth efficiency and fairness. From the perspective of an individual flow, multiplicative decrease of $cwnd_i(t)$ in (9) does not necessarily affect the flow throughput, if $qdelay(t)$ in the denominator decreases correspondingly due to synchronized window adjustments of all flows. Traditional wisdom might argue that global synchronization is more likely to cause $scwnd$ to operate below $scwnd_{knee}$ where bandwidth is underutilized. However, as shown by (8), this can be avoided by setting the window decrease ratio adaptively.

III. An Experimental Protocol

Based on Section II, we propose a novel congestion avoidance mechanism to improve the TCP smoothness. The sender measures the fine-grained RTT. Our fine-grained RTT measurement differs from that of TCP Vegas, in that it considers delayed ACKs and utilizes duplicate ACKs. The details can be found in [13, Sec. 3.2]. The sampled RTT measurements are smoothed by exponential weighted moving average, in order to eliminate temporary delay spikes due to unstable links. The sender records the minimum RTT and the maximum RTT perceived. The queuing delay can be derived by deducting $min\ RTT$ (which corresponds to the roundtrip propagation delay) from the current RTT. The slow start mechanism of TCP is not modified, to allow the queue length to fully grow to overflow during initialization. Therefore, $max\ RTT$ achieved with $max\ qdelay$ could be observed before the congestion avoidance stage takes over.

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

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

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

$$\begin{aligned}\gamma &= \frac{scwnd_{knee}}{scwnd(t)} = \frac{bw \cdot \min RTT}{bw \cdot RTT(t)}\\ &= \frac{\min RTT}{Th_{upper} \cdot \max RTT + (1 - Th_{upper}) \min RTT}\\ &= \frac{1}{1 + Th_{upper} \cdot k}\end{aligned} \tag{11}$$

where k is defined in (8). That is, multiplicative decrease is triggered when $scwnd$ is halfway between $scwnd_{knee}$ and

$scwnd_{cliff}$, and $scwnd$ decreases down to $scwnd_{knee}$. Note that γ-based decrease is carried out one RTT after the condition of inequality (10) is detected, in order to assure that no sender adjusts downward before the condition is observed by all the other senders. Equation (11) bears some similarities with (8). However, (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, downward adjustments based on (11) are triggered by a threshold on the queuing delay, which can be observed by all senders.

Another optional control parameter δ is introduced, in order to enhance the additive increase speed when $scwnd$ operates below the knee. 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 very likely the bandwidth is underutilized. The additive increase adopts a faster speed: $\delta = 2$. The threshold Th_{lower} is currently set to 0.1. Once the threshold is exceeded, δ hands over the control to α.

The threshold Th_{upper} in (10) provides an upper bound on the queue length. In our simulations, Th_{upper} is set to 0.5, and the upper-bound will be half the buffer size. The threshold Th_{lower} defines whether the queue is to close to idle and δ should be used. Although a high Th_{lower} makes the system converge faster [12], it can make the senders easily overshoot the queue upper-bound set by Th_{upper}, when $scwnd$ is well above the knee. Currently Th_{lower} is set to 0.1, and $\alpha = 1$ is used when the queue is more than 10% of the buffer size.

One concern with RTT-based congestion avoidance is the accuracy of $min\ RTT$. While $min\ RTT$ should reflect the round-trip propagation delay RTT_0, it can be overestimated, for example, if a new flow joins a network with a persistent queue. This will lead to a higher γ computed by (11) and unfairness to other flows. $min\ RTT$ overestimation is a well-known problem for TCP Vegas, as it stabilizes at a nonempty queue [3]. With our protocol, however, $scwnd$ periodically adjusts downward to $scwnd_{knee}$ where RTT_0 can be accurately measured. This is verified by our simulations with dynamic traffic in Section IV-C4. Here we theoretically analyze the system convergence. Assume when the i^{th} flow joins the system, its $min\ RTT$ is overestimated by ΔRTT, due to a bottleneck queue with $q_0 = \Delta RTT \cdot bw$ bytes. According to (11), the next γ-based window decrease will be triggered when $scwnd(t) = scwnd_{knee}/\gamma$. After that, the expected $scwnd$ in the next RTT will be

$$\begin{aligned} scwnd(t+RTT) &= (scwnd(t) - cwnd_i(t)) \\ &\quad \cdot \gamma + cwnd_i(t) \cdot \gamma_i \\ &= scwnd_{knee} + cwnd_i(t) \cdot (\gamma_i - \gamma) \end{aligned} \qquad (13)$$

where (according to (11))

$$\gamma = \frac{RTT_0}{RTT(t)}, \quad \gamma_i = \frac{RTT_0 + \Delta RTT}{RTT(t)}. \qquad (14)$$

The window decrease ratio γ_i for the i^{th} flow is higher than the γ for the other flows, due to the overestimation of $min\ RTT$. Based on (5), (13), (14) and (9), the queue size right after the coordinated window decreases will be

$$\begin{aligned} q(t+RTT) &= cwnd_i(t) \cdot (\gamma_i - \gamma) = \frac{cwnd_i(t)}{RTT(t)} \Delta RTT \\ &= throughput_i(t) \cdot \Delta RTT \\ &= s_i(t) \cdot (bw \cdot \Delta RTT) = s_i(t) \cdot q_0 \end{aligned} \qquad (15)$$

where $throughput_i(t)$ is the throughput of the i^{th} flow before congestion avoidance is triggered; $s_i(t) = throughput_i(t)/bw$ is the corresponding bandwidth share, in percentage, of the i^{th} flow. That is, after each coordinated window decrease, the minimum queue observed by the i^{th} flow will decrease by a factor of $s_i(t)$. Notably the bandwidth share $s_i(t)$ of an incoming new flow is often less than its fair share $(1/n)$, due to the small initial window size. Hence (15) implies that a flow overestimating $min\ RTT$ initially will quickly converge to an accurate estimation.

To summarize, TCP(α, β, γ, δ) has a number of features. a) It relies on fine-grained RTT measurements to estimate network conditions, beyond congestive packet drops. b) It follows a synchronized model by coordinating upward and downward window adjustments to enhance TCP smoothness, without compromising efficiency, fairness and responsiveness. c) It avoids congestion by scheduling window decreases well before the occurrence of congestion, to reduce packet drops. d) It introduces new control parameters γ and δ adaptable to the current network condition. e) It is an end-to-end solution that controls the average queue length through Th_{upper} and Th_{lower}, without deploying AQM in routers. Above all, these features do not violate the established fairness-oriented AIMD. Although, TCP Vegas also has (a), (c), and (e), its congestion avoidance uses Additive Increase/Additive Decrease (AIAD), which does not guarantee fairness [2]. This unfairness is amplified by the worst-case fairness defined in Section IV-B and analyzed in the Appendix.

IV. Evaluation and Analysis

A. Testing Plan and Methodology

The protocol is evaluated on the ns-2 network simulator [18], with the network topology shown in Fig. 1. By default, $delay = 30$ ms and $src\_delay = sink\_delay = 5$ ms. Simulations with diverse propagation delays are also conducted. bw varies from 10 Mbps to 100 Mbps. The buffer size at R_1 is set to the delay-bandwidth product. The deployment of RED at R_1 is also tested, in order to investigate the effect of random drops explicitly enforced by RED, in comparison with the implicit random decrease with drop-tail buffer. The settings for RED are $min\_th = buffer\_size/6$, $max\_th = buffer\_size/2$, $max\_p = 0.1$ and $w = 0.002$ [19].

Protocol behaviors are also tested with multiple bottlenecks and cross traffic shown Fig. 2. The router R1 is the bottleneck for the main traffic (n flows from “source” nodes to “sink” nodes), while the router R3 is the bottleneck for the cross traffic (m flows from “peripheral source” nodes to “peripheral sink”

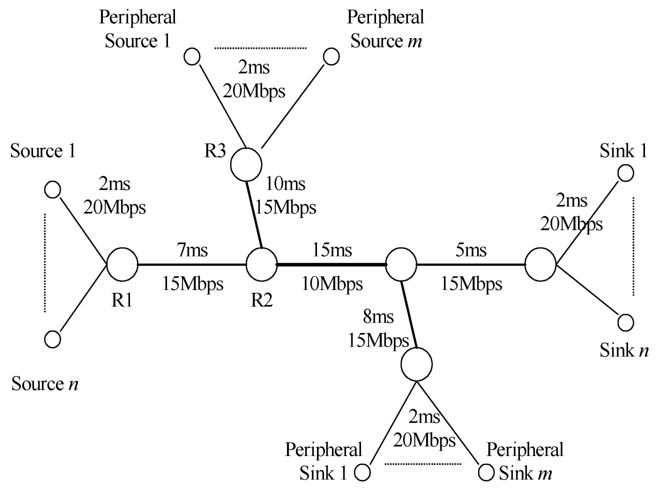


Fig. 2. Network topology with multiple bottlenecks and cross traffic.

nodes). The router R2 is the bottleneck for the competing main traffic and cross traffic.

To evaluate the protocol's responsiveness to the rapid changes of available bandwidth, we create a scenario of temporary "blackouts" due to mobile handoffs, during which all transmitted packets are lost. An error model is inserted into the access links to the receivers (assumed in the wireless domain), with 100% packet dropping rate.

We select seven protocol configurations: Reno ($\alpha = 1, \beta = 0.5$), Reno with RED, TCP Vegas, Reno plus γ, Reno plus γ and $\delta (= 2)$, smooth TCP ($\alpha = 0.31, \beta = 0.875$) [8], and TFRC [7]. In contrast to the adaptive γ, TCP(0.31, 0.875)'s smoothness is achieved by increasing the window decrease ratio, at the cost of lesser responsiveness, due to its static control parameters. TCP Vegas is selected to demonstrate the unfairness of AIAD. Since TFRC is a rate-based unreliable protocol, it's somewhat unreasonable to compare its performance with window-based reliable TCP protocols. Nonetheless, TFRC is included as a reference for smoothness. Notably, our mechanism can be easily adapted for unreliable media-streaming.

B. Performance Metrics

Long-term fairness is measured by the $Fairness\ Index$, defined by [2]:

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

where $throughput_i$ is the throughput of the ith flow during the entire connection. This index provides a sort of "average-case" analysis. For a "worst-case" analysis and a tighter bound on fairness, we propose to use $Worst - Case\ Fairness$:

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

The motivation for introducing the worst-case fairness is to capture unfairness to a very small fraction of flows (see [13, Sec. 4.2]).

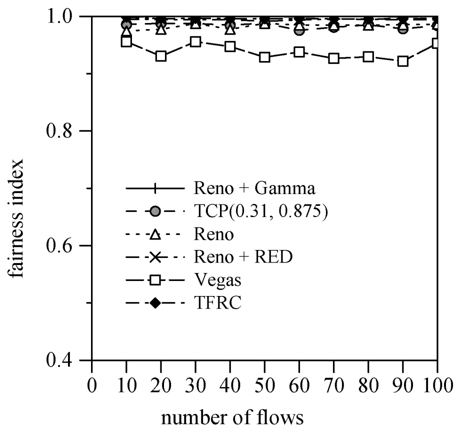


Fig. 3. Fairness index.

To investigate the performance smoothness observed by end users, allotted throughput $throughput_i(t)$ for the ith flow is sampled at a time scale of several RTTs (in our simulations, the sampling period is 0.5 s). Following the metric in [17], we use $Coefficient\ of\ Variation\ (CoV)$ to gauge the throughput smoothness experienced by flow i:

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

where $\mathrm{E_t}\{\}$ denotes the computation of the mean along the entire connection 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 s after the simulation starts, in order to just capture the system behaviors after it enters equilibrium. Bottleneck queue lengths are also traced.

C. Results and Analysis

1) Fairness and Smoothness: We first conduct ten simulations for 100–s connection time, with the number of flows varied from 10 to 100 (Figs. 3–6). The bottleneck bandwidth scales correspondingly, such that the fair share for each flow is 1 Mbps. If assessed by traditional fairness index (Fig. 3), all protocols achieve high level of fairness, except Vegas. However, random multiplicative decrease can not guarantee worst-case fairness (Fig. 4), compared to the synchronized window adjustments of $\mathrm{Reno} + \gamma$. The overall fairness of Vegas is the worst.

As shown in Figs. 5 and 6, $\mathrm{Reno} + \gamma$ also achieves higher short-term fairness and lower CoV (i.e. higher smoothness) than the protocols based on random window adjustments, including

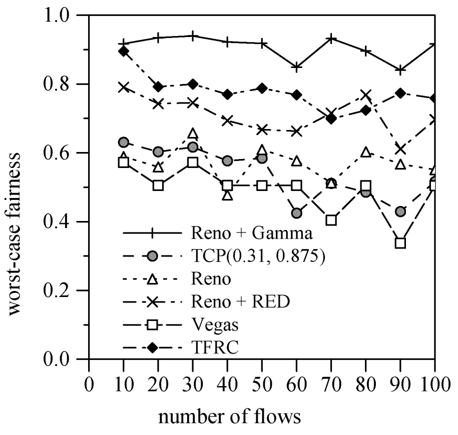


Fig. 4. Worst case fairness.

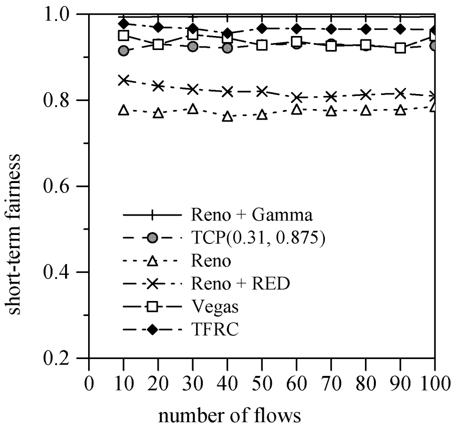


Fig. 5. Short-term fairness.

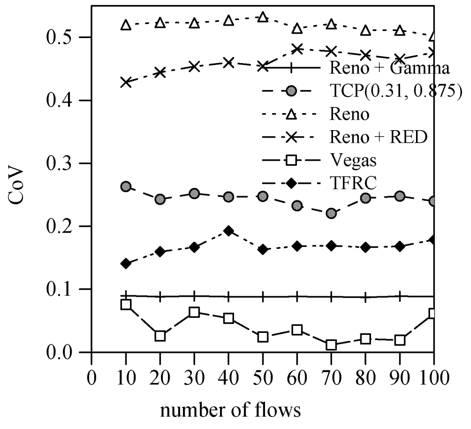


Fig. 6. CoV.

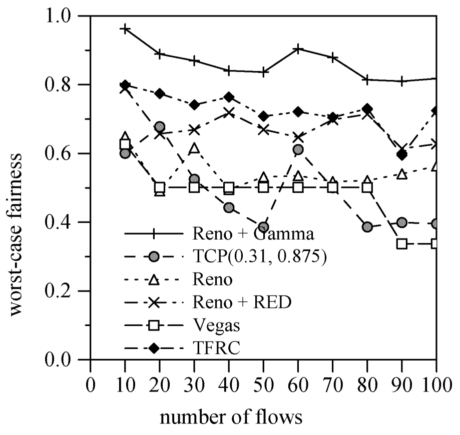


Fig. 7. Worst case fairness with diverse RTTs.

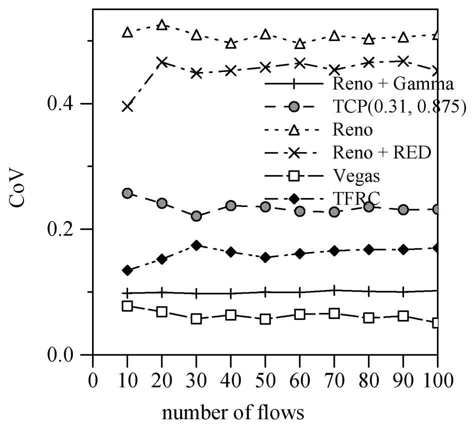


Fig. 8. CoV with diverse RTTs.

Reno, Reno + RED, TCP(0.31, 0.875) and even TFRC. Although Reno + γ achieves higher smoothness, its window decrease ratio (γ is around 2/3 with the simulation settings) is even lower than that of TCP(0.31, 0.875). The lowest CoV is achieved by TCP Vegas because of its AIAD strategy to stabilize at any *unfair* state, as long as each flow has a couple of packets in the buffer and the bottleneck bandwidth is fully utilized. In the Appendix, we will analyze why the worst-case fairness of Vegas can be as low as 1/3.

The unfairness of Vegas can be further demonstrated by a set of tests shown in Fig. 5 of our early publication [13]. The result is striking: the worst-case fairness with uncoordinated window adjustments grows slowly as the connection time increases to 1000 s, reflecting the inherent characteristics of randomness. However, the fairness of Vegas stays low, no matter how long the connection time is. The highest throughput achieved by individual flows is 75% higher than the lowest one throughout the simulation.

Queue lengths traces with 10 flows can also be found in [13]. Vegas and RENO + γ do control adequately the bottleneck queue length, while standard TCP Reno periodically causes buffer overflow. Without deploying AQM, TCP-friendly protocols (TCP(0.31, 0.875) and TFRC) force the bottleneck queue to stay at a position close to the cliff, due to their "gentle" regression upon buffer overflow.

2) Diverse RTTs and Multiple Bottlenecks: We repeat the tests in Section IV-A with diverse RTTs. The propagation delays of access links to sink nodes are uniformly distributed between 5 ms to 15 ms. The buffer size is set to the product of the bottleneck capacity and the minimum round-trip propagation delay. The worst-case fairness and CoV are demonstrated in Figs. 7 and 8. Reno + γ still outperforms the other protocols, and its synchronized window adjustments are not disrupted by the diverse RTTs. Protocols are further tested with multiple bottlenecks and cross traffic (see Fig. 2). Half of the flows form the main traffic, while the other half form the cross traffic. The result shown in Figs. 9 and 10 demonstrate that Reno + γ achieves higher worst-case fairness and smoothness than uncoordinated

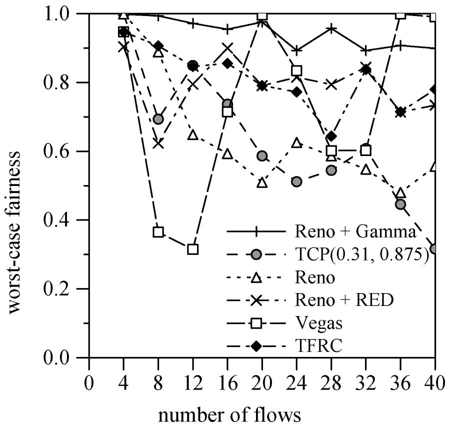


Fig. 9. Worst case fairness with multiple bottlenecks.

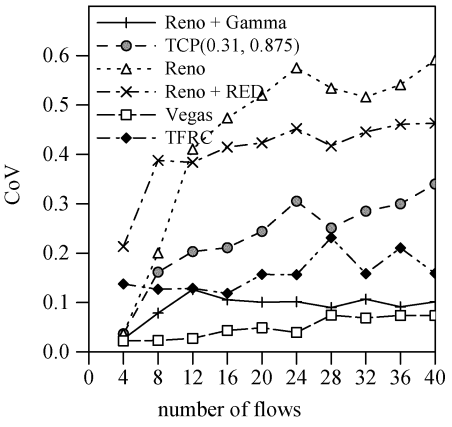


Fig. 10. CoV with multiple bottlenecks.

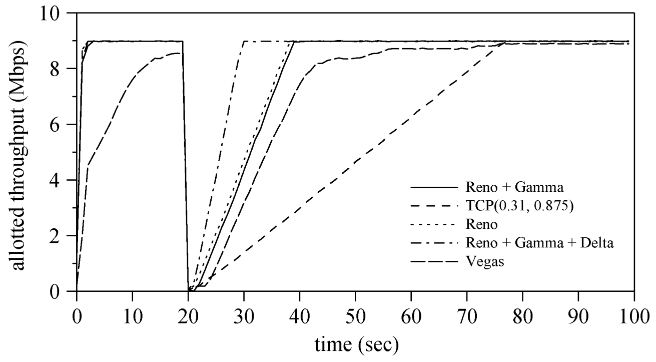


Fig. 11. Allotted thoughput with 1.0 s handoff.

AIMD. Noticeably the worst-case fairness of Vegas fluctuates as the number of flows changes, and can go as low as 0.31.

3) Responsiveness With Dynamic Traffic: Our next simulation over heterogeneous (wires/wireless) networks indicates that the high smoothness of coordinated window adjustments is achieved not at the cost of responsiveness. A single flow traverses a 10 Mbps bottleneck link. At time 20 s, the wireless access link is interrupted by a 1 second handoff period, during which all packets are lost. Since bandwidth becomes available immediately after the handoff, a high sending rate increase is the desired behavior. The protocols' aggressiveness with allotted throughputs after the handoff is shown in Fig. 11. Due to the lesser responsiveness ($\alpha = 0.31$), it takes 57 s for TCP(0.31, 0.875) to fully recover the transmission speed, while the recovery time for Reno or $\mathrm{Reno} + \gamma$ is 19 s, with $\alpha = 1$. If the optional parameter $\delta = 2$ is enabled, the recovery time can be further reduced to 10 s, since an idle bottleneck queue can be detected right after the handoff is over. The recovery speed of Vegas is slightly below that of Reno.

4) Microscopic Behaviors:

Microscopic behaviors in a static environment: We have also tracked the allotted throughputs of three flows competing at a 3 Mbps bottleneck link for 100 s, in order to observe the microscopic behaviors of protocols. The number of flows (3) is selected to ease the analysis. We trace the allotted throughput of each flow as well as the aggregated system throughput, with a 200 ms sampling period. It can be seen from Figs. 12–16 that after the initial slow start stage that leads to retransmission timeouts and subsequent minimum congestion windows, the senders rely mostly on additive increase to reach the equilibrium.

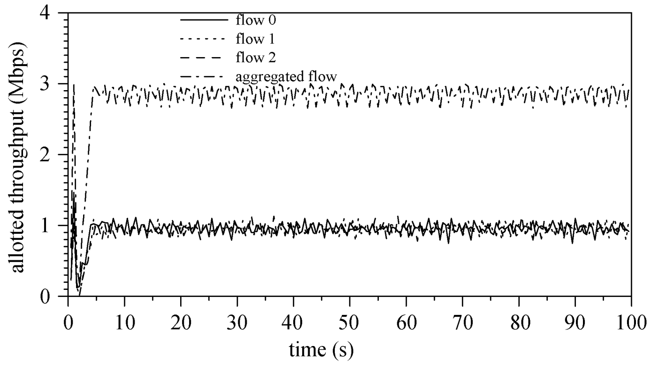


Fig. 12. Allotted throughput with $\mathrm{Reno} + \mathrm{Gamma}$.

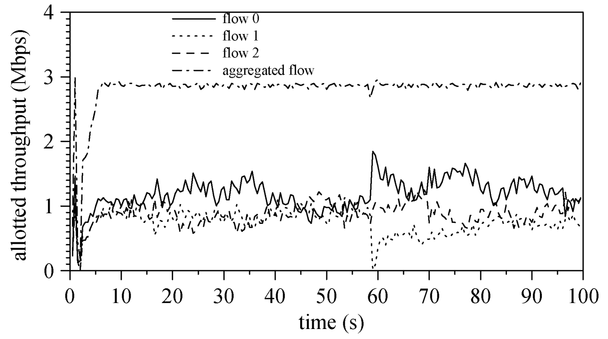


Fig. 13. Allotted throughput with TCP(0.31, 0.875).

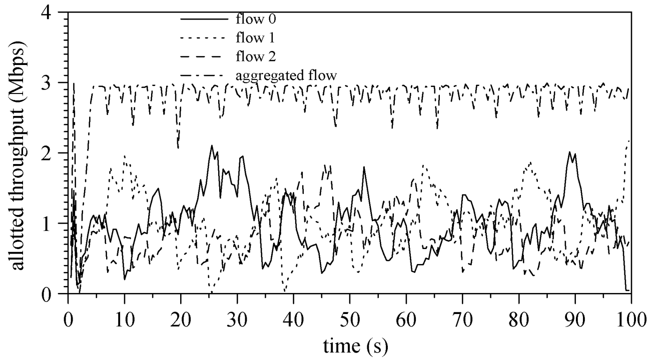


Fig. 14. Allotted throughput with Reno.

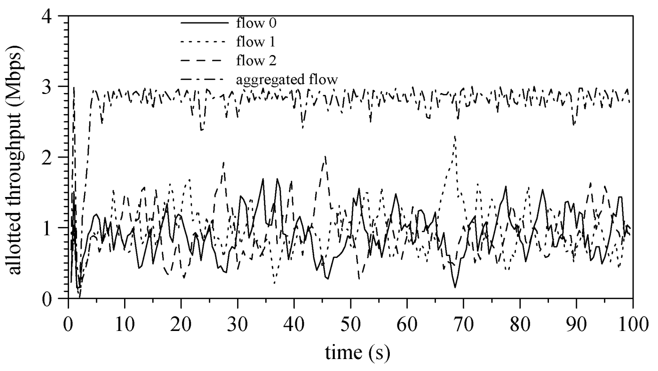


Fig. 15. Allotted throughput with $\mathrm{Reno} + \mathrm{RED}$.

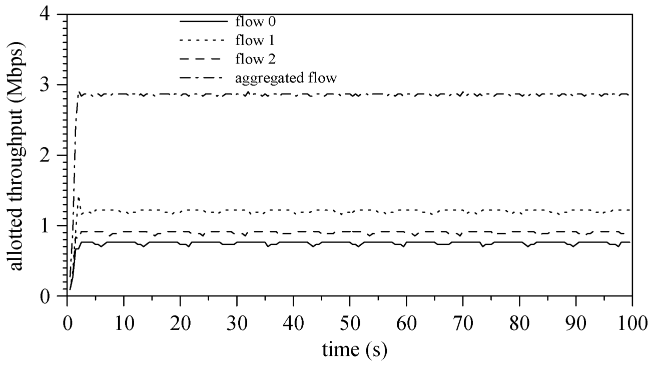


Fig. 16. Allotted throughput with Vegas.

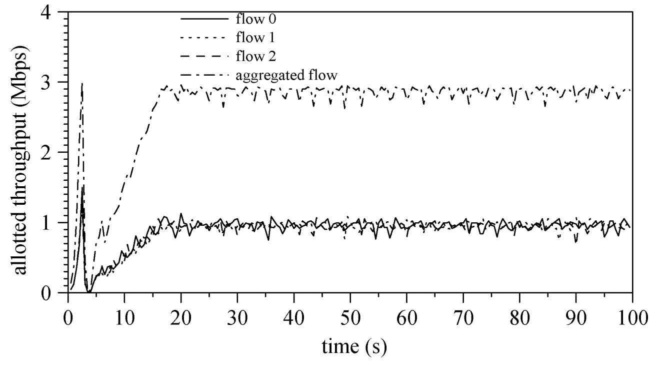


Fig. 17. Allotted throughput of $\mathrm{Reno} + \mathrm{Gamma}$ with long RTT.

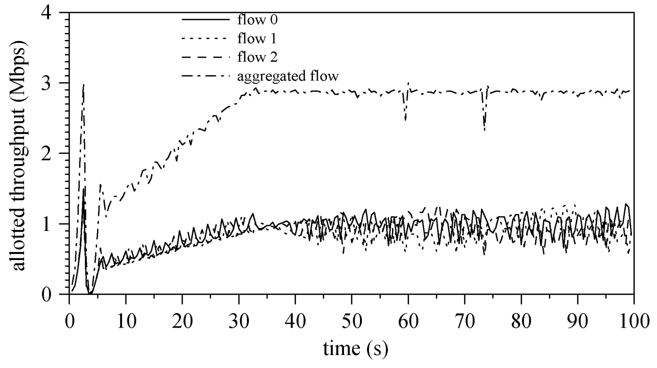


Fig. 18. Allotted throughput of TCP(0.31, 0.875) with long RTT.

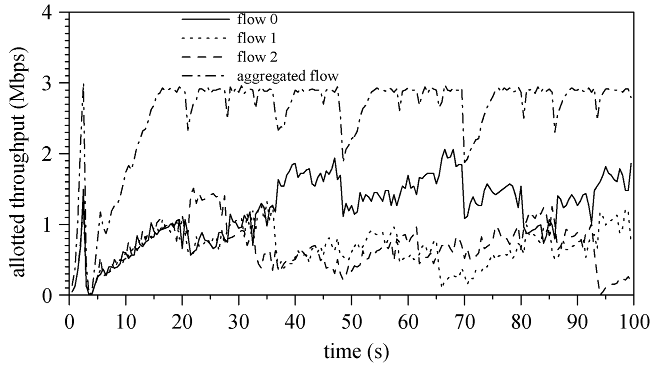


Fig. 19. Allotted throughput of Reno with long RTT.

The results confirm the analysis in Section II-A that magnitude of system throughput fluctuations is much smaller than the multiplicative decrease ratio, due to the queue dynamics. However, although the overall system throughput is relatively stable with all protocols (Figs. 12–16), individual flows do not experience smooth throughputs and the system is unfair in short-term, with TCP(0.31, 0.875), Reno, and $\mathrm{Reno} + \mathrm{RED}$. In contrast, $\mathrm{Reno} + \gamma$ provides relatively smooth throughputs to individual flows in steady-state, due to its coordinated window adjustments and the reduced congestive drops. Deploying RED can slightly improve smoothness and short-term fairness of Reno, as evident from Figs. 5, 6, 14 and 15. With packet drops prior to buffer overflow, the length of congestion epoch is reduced, which prevents a single flow from increasing its congestion window too far beyond its fair-share. We also observe that TCP(0.31, 0.875) may extend the time to converge to fairness. At time 58 s, a severe congestion causes the retransmission timeout and slow start of flow 1. Although this event almost does not affect the system throughput, it takes very long time for flow 1 to recover from the extreme unfairness afterwards, due to the low α. Although similar event also occurs to Reno (see the deep throughput gap of flow 1 around 25 s), its large window oscillations do not extend the convergence time. Vegas also provides smooth throughput for individual flows. However, its AIAD congestion avoidance can stabilize at an unfair state. If the bottleneck bandwidth is originally designed to accommodate, for example, three concurrent 1 Mbps video-streaming flows, with Vegas, flow 1 constantly achieves an *unnecessarily* higher throughput, at the expense of flow 0.

Microscopic behaviors with long RTT: We repeat the static simulations with longer bottleneck propagation delay (100 ms), shown in Figs. 17–21. Since the bottleneck buffer size is not adjusted accordingly, the buffer size (30 packets) is much smaller than the delay-bandwidth product: the k defined in (8) is around 36%. According to (8) and (11), a higher β_{opt} (0.73) or γ (0.85) is indicated for this network condition. The simulation is designed to investigate the impact of disproportionally long propagation delay (relative to the small buffer size) on the performance of the *static* congestion control parameters.

Conventional wisdom might argue that synchronized multiplicative decrease is more likely to cause the system to operate below the knee right after multiplicative decreases. However, results shown in Fig. 17 demonstrate that since γ is set adaptively according to the measured network conditions, the system throughput is high and the flow-level throughput is smooth with $\mathrm{Reno} + \gamma$. Although a static $\beta = 0.5$ of Reno can prevent the system from operating below the knee when k defined in (8) is around 100% (see results in Fig. 14), the same static ratio can even cause system-level throughput oscillations (Figs. 19

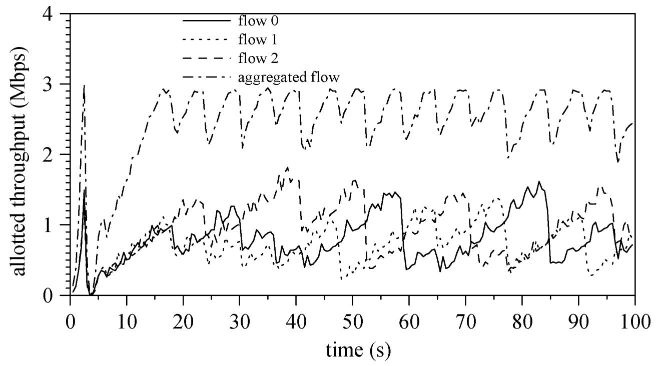


Fig. 20. Allotted throughput of $\mathrm{Reno} + \mathrm{RED}$ with long RTT.

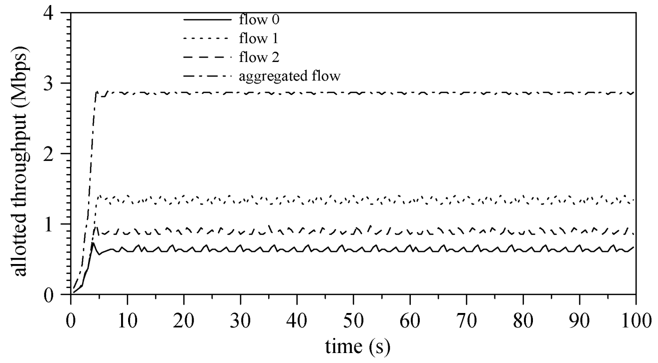


Fig. 21. **Author: ADD TEXT CITE FOR FIG. 21**Allotted throughput of Vegas with long RTT.

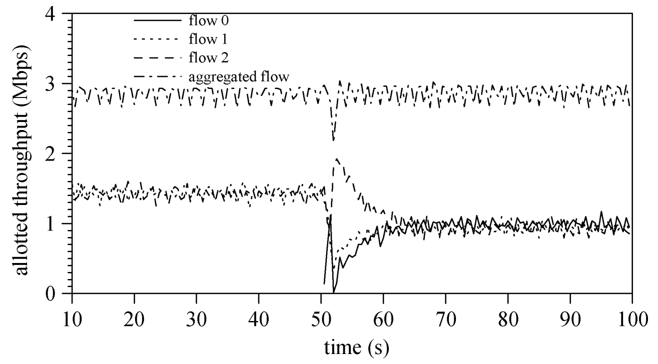


Fig. 22. Allotted throughput of $\mathrm{Reno} + \mathrm{Gamma}$ with increasing number of flows.

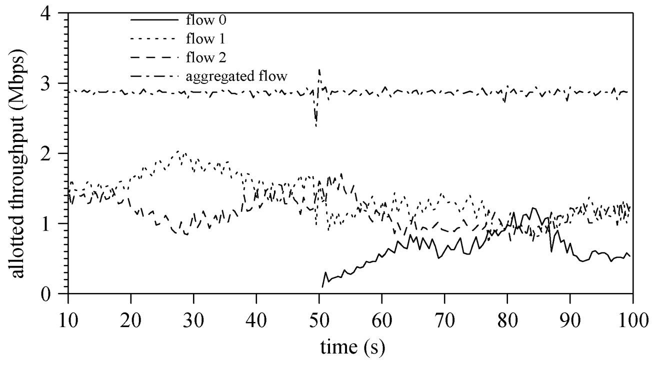


Fig. 23. Allotted throughput of TCP(0.31, 0.875) with increasing number of flows.

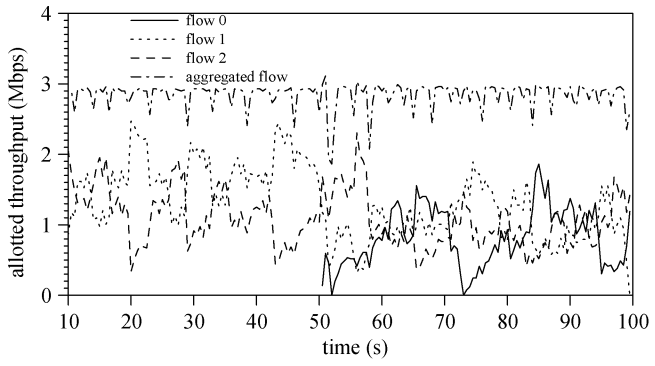


Fig. 24. Allotted throughput of Reno with increasing number of flows.

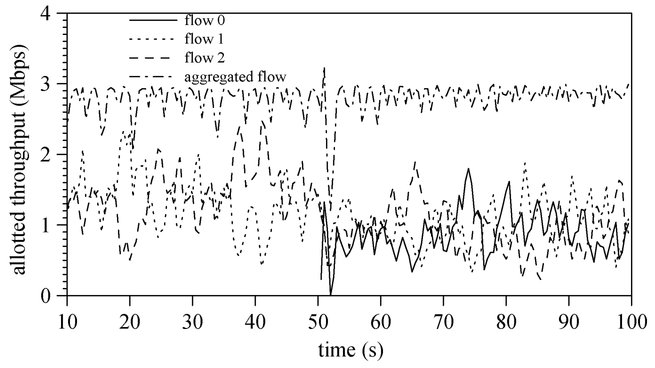


Fig. 25. Allotted throughput of $\mathrm{Reno}+\mathrm{RED}$ with increasing number of flows.

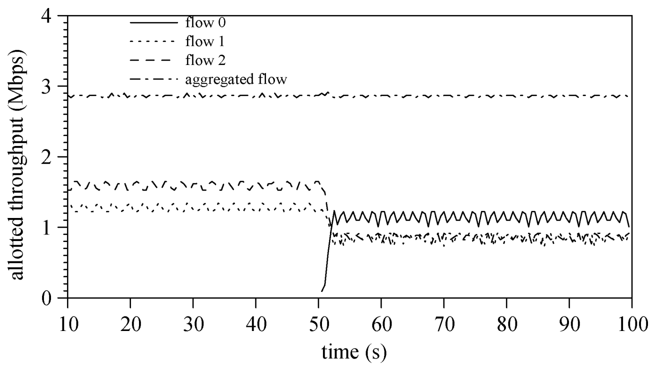


Fig. 26. Allotted throughput of Vegas with increasing number of flows.

and 20), let alone the throughputs of individual flows, when k is around 36%. The high β of TCP(0.31, 0.875) seems basically fitting for this specific long-RTT scenario, since its β is greater than the 0.73 suggested by (8). Even so, at around 73.5 second, an unusual dip of system throughput (from 2.92 Mbps to 2.31 Mbps, see Fig. 18) occurs to TCP(0.31, 0.875). This coincides with consecutive window decreases of flow 2, due to severe packet losses. This confirms again that blind window control based on packet losses can hurt the performance.

Microscopic behaviors in a dynamic environment: In the next simulations of dynamic traffic (Figs. 22–26), both flow 1 and flow 2 start at time 0. Flow 0 joins competition at time 50 s, which causes a sudden increase in contention. The initial slow start of flow 0 leads to severe packet losses and significant window decreases of some flows. With $\mathrm{Reno} + \gamma$, after the initial stage, the coordinated AIMD window adjustment takes over, and the system quickly converges to a smooth equilibrium, with a lower fair-share for each flow. The synchronized window adjustment is not disrupted by the incoming new flow. In line with the analysis on Vegas in the Appendix, flow 0 achieves an unfair high throughput, since the incoming Vegas flow overestimates RTT_0 after the system stabilizes at a nonempty queue,. In contrast, our scheme periodically adjusts downward to the knee where RTT_0 can be accurately measured.

V. Conclusion and Future Work

We argued that the major obstacle for achieving smoothness is the unsynchronized and random window adjustments. In deed, synchronized window adjustments may avoid bandwidth underutilization by adapting the control parameters to the network conditions. Based on these observations, we proposed an adaptive congestion avoidance mechanism to coordinate window adjustments. Flow-level throughput smoothness is enhanced, without compromising efficiency, fairness and responsiveness. Currently, the mechanism is integrated into TCP. We plan to further incorporate and adapt this mechanism into unreliable or rate-based transport protocols (such as RAP [9]). Furthermore, since DiffServ has been introduced, flows may have different utility functions and equal sharing is not always fair allocation. We will investigate how to adjust the parameters α, β, γ, and δ for different flows, in order to achieve proportional fairness.

Appendix

Analyzing the Worst-Case Fairness of TCP Vegas:

Using the scenario and the notations in Section II, we analyze TCP Vegas as follows. The control objective of Vegas is to keep each flow's congestion window in a range given by

$$\alpha \le \frac{cwnd_i(t)}{RTT_0} - \frac{cwnd_i(t)}{RTT(t)} \le \beta (i = 1, 2, \ldots\ldots n). \quad (A.1)$$

Note that the parameters α and β of Vegas are defined completely different from the parameters of TCP(α, β). With $RTT(t) - RTT_0 = qdelay(t)$ and (9), we have

$$\begin{aligned}\alpha &\le cwnd_i(t) \cdot \frac{qdelay(t)}{RTT_0 \cdot RTT(t)} \\ &= throughput_i(t) \cdot \frac{qdelay(t)}{RTT_0} \\ &\le \beta\, (i = 1, 2, \ldots\ldots n).\end{aligned} \quad (A.2)$$

In the original paper of TCP Vegas [3], $\beta = 3\alpha$. That is, $throughput_i(t)\ (i = 1, 2, \ldots n)$ can stabilize anywhere within

$$\left[\alpha \frac{qdelay(t)}{RTT_0}, \quad 3\alpha \frac{qdelay(t)}{RTT_0}\right]. \quad (A.3)$$

Therefore, the worst-case fairness can be anywhere between 1/3 and 1, assuming that RTT_0 is approximately the same for all flows. [10] points out that Vegas can be unfair to incoming new flows due to their over-estimations of RTT_0, since the system stabilizes at a nonempty queue. Our analysis further reveals that the unfairness is also caused by the fact that, as long as (A.3) holds, the windows can stabilize at an unfair state forever, as substantiated by the simulations described in Section IV.

References

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

[2] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Networks ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.

[3] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE J. Select. Areas Communi.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[4] U. Hengartner, J. Bolliger, and T. Cross, "TCP Vegas revisited," in *Proc. IEEE INFOCOM 2000*, Mar. 2000, pp. 1546–1555.

[5] D. Tan and A. Zakhor, "Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Trans. Multimedia*, vol. 1, no. 2, pp. 172–186, Jun. 1999.

[6] S. Floyd, Congestion Control Principles RFC 2914, Sep. 2000.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM 2000*, Aug. 2000, pp. 43–56.

[8] Y. R. Yang and S. S. Lam, "General AIMD congestion control," in *Proc. 8th Int. Conf. Network Protocols, Proc. IEEE ICNP 2000*, Nov. 2000, pp. 187–198.

[9] R. Rejaie, M. Handely, and D. Estrin, "RAP: an end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *Proc. IEEE INFOCOM 1999*, Apr. 1999, pp. 215–226.

[10] D. Sisalem and H. Schulzrinne, "The loss-delay adjustment algorithm: a TCP-friendly adaptation scheme," in *Proc. ACM NOSSDAV 1998*, July 1998, pp. 215–226.

[11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proc. ACM SIGCOMM 1998*, Aug. 1998, pp. 303–314.

[12] C. Zhang and V. Tsaoussidis, "The Interrelation of TCP smoothness and responsiveness in heterogeneous networks," in *Proc. 7th IEEE Symp. Computers and Communications, (ISCC 2002)*, Jul. 2002, pp. 291–297.

[13] ——, "Improving TCP smoothness by synchronized and measurement-based congestion avoidance," in *Proc. ACM NOSSDAV 2003*, Jun. 2003, pp. 131–140.

[14] V. Misra, W. Gong, and D. Towsley, "A fluid-based analysis of a network of AQM routers supporting TCP flows with application to RED," in *Proc. ACM SIGCOMM 2000*, Sep. 2000, pp. 151–160.

[15] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. ACM SIGCOMM 2004*, Aug. 2004, pp. 281–292.

[16] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[17] Y. R. Yang, M. S. Kim, and S. S. Lam, "Transient behaviors of TCP-friendly congestion control protocols," in *Proc. IEEE INFOCOM 2001*, Apr. 2001, pp. 1716–1725.

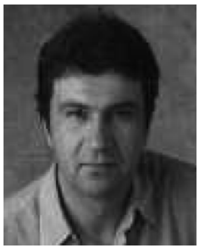
[18] NS-2, The Network Simulator [Online]. Available: http://www.isi.edu/nsnam/ns/

[19] S. Floyd, RED: Discussions of Setting Parameters November 1997 [Online]. Available: http://www.icir.org/floyd/REDparameters.txt

Chi Zhang (M'03) received the B.E. degree in electronic engineering from Shanghai Jiao Tong University, China, in 1996, and the Ph.D. degree in computer science from Northeastern University, Boston, MA, in 2003.

He is an Assistant Professor of Computer Science at Florida International University, Miami. His research interests lie in the areas of network protocols, mobile computing and QoS. He has published 19 papers and been issued one U.S. patent.

Dr. Zhang received the runner-up award in IEEE ISCC 2002 and is a member of Phi Kappa Phi.



Vassilis Tsaoussidis (M'97–SM'04) received the Ph.D degree in computer networks from Humboldt University, Berlin, Germany, in 1995.

He has held faculty positions in Rutgers University, State University of New York at Stony Brook and Northeastern University, Boston, MA. In May 2003, he joined the Department of Electrical and Computer Engineering of Demokritos University, Xanthi, Greece. His research interests lie in the area of transport/network protocols.

Dr. Tsaoussidis has edited three journal special issues and chaired IC 2002, the WWIC 2002, and WWIC 2004. He is an associate editor for the IEEE Transactions on Mobile Computing and an editor for the *Computer Networks* and *Wireless Communications and Mobile Computing*. He participates in several Technical Program Committees, such as INFOCOM, GLOBECOM, ICCCN, ISCC, WLN, and several others.