TCP-GT: A New Approach to Congestion Control Based on Goodput and Throughput

Hyungsoo Jung, Shin-Gyu Kim, Heon Young Yeom, and Sooyong Kang

Abstract: **A plethora of transmission control protocol (TCP) congestion control algorithms have been devoted to achieving the ultimate goal of high link utilization and fair bandwidth sharing in high bandwidth-delay product (HBDP) networks. We present a new insight into the TCP congestion control problem; in particular an end-to-end delay-based approach for an HBDP network. Our main focus is to design an end-to-end mechanism that can achieve the goal without the assistance of any network feedback. Without a router's aid in notifying the network load factor of a bottleneck link, we utilize goodput and throughput values in order to estimate the load factor. The obtained load factor affects the congestion window adjustment. The new protocol, which is called TCP-goodput and throughput (GT), adopts the carefully designed inversely-proportional increase multiplicative decrease window control policy. Our protocol is stable and efficient regardless of the link capacity, the number of flows, and the round-trip delay. Simulation results show that TCP-GT achieves high utilization, good fairness, small standing queue size, and no packet loss in an HBDP environment.**

Index Terms: **Congestion control, high bandwidth-delay product networks (HBDP), protocol design.**

I. INTRODUCTION

Congestion control is the most fundamental issue to resolve in order to utilize a bottleneck link efficiently and share the link fairly. As current network technology endows the Internet with a large number of high bandwidth-delay product (HBDP) networks, solving the congestion control problem becomes a matter of utmost importance because the additive increase multiplicative decrease (AIMD) [1] congestion control algorithm adopted by transmission control protocol (TCP) [2] is known to be uneconomical in achieving high link utilization in HBDP networks.

The TCP's defect of underutilizing an HBDP network has set a new research avenue for the network community. With various algorithmic techniques, many research efforts have suggested different protocols for this problem, each of which has merits and drawbacks. We can classify these protocols into two categories: *End-to-end* and *router-supported* approaches. End-to-end congestion control algorithms such as HighSpeed TCP (HSTCP) [3], FAST [4], binary increase congestion control (BIC) [5], and CUBIC [6] can be attractive solutions since they do not require any support from routers, and this results in a lesser deployment problem. However, in HBDP networks, using the loss of segments and/or the variation in packet delay as the only congestion signal sets fundamental limitations in achieving high utilization and fairness while maintaining low bottleneck queue length and minimizing congestion induced packet drop rate.

To overcome the shortcomings of end-to-end congestion protocols, many researchers have approached the issue with different viewpoints, some of which propose the use of network feedback. Explicit control protocol (XCP) [7] tackles this problem by measuring a congestion level at the bottleneck to calculate a desired flow rate; routers inform the senders about the degree of congestion at the bottleneck. As a result, XCP can achieve high utilization, good fairness, small queue size, and almost no packet drop in HBDP networks. To lessen the deployment problem of XCP, which requires several bytes to encode the congestion-related information between routers and end-hosts, variable-structure congestion control protocol (VCP) [8] attempts to achieve XCP's performance by using only the two explicit congestion notification (ECN) bits to encode the congestion feedback. But, it sacrifices the speed of convergence to fairness nontrivially. This has been improved further by the recent work [9]. In summary, the most obvious problem of these protocols is deployment; routers should perform intelligent work.

In this paper, we propose TCP-goodput and throughput (GT), an end-to-end delay-based congestion control protocol with high utilization, good fairness, small queue size, and almost no packet drop in HBDP networks. The crucial characteristic of TCP-GT is using a load factor, which is proposed as a congestion signal in [10]. In router-supported protocols, routers usually compute the load factor and use it to identify the congested link state. However, in end-to-end protocols, obtaining an accurate load factor is almost impossible without the support from routers. We deal successfully with this problem by approximating a load factor via an end-to-end measurement scheme. A pair of end-hosts (sender/receiver) cooperates to estimate goodput and throughput[1]. The difference between goodput and throughput gives us very useful information about the state of a congested link. The difference indicates the level of congestion (or spare bandwidth) at a bottleneck link. We denote this difference value as ϕ and compute it as follows: $\phi = \text{goodput} - \text{throughput}$. The use of a load factor estimated by an end-host makes it possible to approach the TCP congestion problem with a simple window control policy; this has been an essential technique in many router-supported protocols.

Manuscript received February 16, 2009; approved for publication by Saewoong Bahk, Division III Editor, February 22, 2010.

This work was supported by the National Research Foundation (NRF) grant funded by the Korea government (MEST) (no. 2009-0074573).

H. Jung, S.-G. Kim, and H. Y. Yeom are with the School of Computer Science and Engineering, Seoul National University, Seoul, Korea, email: {jhs, sgkim, yeom}@dcslab.snu.ac.kr.

S. Kang (corresponding author) is with the Division of Computer Science and Engineering, Hanyang University, Seoul, Korea, email: sykang@hanyang.ac.kr.

[1]Goodput is effective (or actual) throughput observed by a receiver, and throughput is ideal (or expected) throughput anticipated by a sender.

TCP-GT guarantees the convergence to fairness between flows by its unique congestion control policy, i.e., the inversely-proportional increase multiplicative decrease (IIMD) policy. With the AIMD policy, TCP flows converge to fairness only in the decreasing phase. Meanwhile, TCP-GT flows converge to fairness not only in a decreasing phase, but also in an increasing phase. Convergence in an increasing phase is a very attractive feature insofar as it works well under the variety of bandwidth conditions. Particularly, special care must be taken in designing an IIMD policy because a large congestion window can prevent the protocol from converging fast; when we see the following inversely-proportional increase (II) equation, constant values of α and k could hardly solve the slow convergence; $W_{t+\Delta t} = W_t + \alpha/(W_t)^k$; $\alpha,\ k > 0$. We get around this issue by the window splitting technique: Additively splitting the congestion window into smaller ones. Splitting windows makes an effect of creating virtual flows that share the same congestion window. When increasing the congestion window, we use the aggregate increment from all split windows instead of using the very small increment from a large window. In an increasing phase, this makes the congestion window of a TCP-GT flow behave like a quadratic equation, and it enhances the convergence speed of TCP-GT. TCP-GT handles the round trip time (RTT) unfairness[2] moderately by devising the RTT compensation technique. Flows with different RTTs can converge to fairness inasmuch as all IIMD equations are compensated properly. The last point we should address is that the use of a load factor as a congestion signal differentiates TCP-GT from existing drop-based end-to-end protocols, and this makes TCP-GT hard to be compatible with drop-based protocols.

Using extensive ns-2 simulations, we show that TCP-GT achieves high utilization, good fairness, low persistent queue size, and no packet loss regardless of bottleneck capacity, round-trip delay, and the number of flows. We also demonstrate that TCP-GT converges to fairness with heterogeneous RTTs, and TCP-GT adapts smoothly to a sudden increase and quickly to a sudden decrease in traffic. The surprising result is that TCP-GT never dropped any packets in our simulations. To the best of our knowledge, TCP-GT is the only end-to-end protocol exhibiting almost no packet drop.

The rest of the paper is organized as follows: We review related work in Section II. In Section III, we provide a detailed description of TCP-GT. In Section IV, we evaluate the performance of TCP-GT using extensive simulations. In Section V, we present future work and the conclusion.

II. RELATED WORK

Network-supported congestion control schemes like XCP, VCP, and coupling logistic TCP (CLTCP) [11] show excellent performance and efficient convergence in HBDP networks. TCP-GT is motivated by these protocols and built upon a number of previous research efforts in end-to-end congestion control [4], [12]–[16]. Roughly speaking, these efforts can be divided into two categories: Delay-based congestion control and packet loss-based congestion control.

[2]Flows with different RTTs share bandwidth unfairly.

A. Delay-Based Congestion Control

Jain first proposed delay-based congestion control in [17]. In 1994, TCP-Vegas was proposed with the claim of achieving throughput improvement ranging from 37% to 71% compared with TCP-Reno [18], [19]. An innovative idea in Vegas is that it detects congestion by observing the change of a throughout rate and prevents packet losses proactively. TCP-GT makes use of a mechanism similar to that in Vegas to estimate a congested link state. However, there is a big difference between Vegas and TCP-GT in the manner of changing the congestion window. Regardless of the degree of congestion, Vegas increases or decreases the congestion window by a fixed size in every control interval. In contrast, TCP-GT adjusts the congestion window in proportion to a congestion level. It controls the window in a more fine-grained way, and as a result, it achieves good fairness.

Some delay-based enhancements, which include TCP-Vegas and FAST [4], adopt a minimum RTT to detect the network condition. Because RTT reflects the bottleneck queuing delay, this mechanism is effective in determining the network congestion status. But, the use of a minimum of all measured RTT results in a fairness problem [20], [21]. TCP-GT addresses this problem and is built upon the experience gained from these studies.

B. Packet Loss-Based Congestion Control

Most TCP implementations belong to this category. A packet loss is an evident notification which indicates that the network is highly congested and the bottleneck queue is full. The common features include slow start and fast recovery. While each protocol has a unique policy when increasing or decreasing the congestion window, it is based on the AIMD policy. Retaining the AIMD policy guarantees TCP-friendliness. However, the pure *additive increase* policy significantly degrades utilization in HBDP networks. To improve performance in this environment, many solutions have been proposed.

HSTCP [3] extends the standard TCP by adaptively setting the increasing/decreasing parameters according to the congestion window size. HTCP [22] employs a similar control policy to HSTCP, but modifies the increase parameter based on the elapsed time since the last congestion event. Scalable TCP (STCP) [23] has an multiplicative increase multiplicative decrease (MIMD) control policy to ensure that the congestion window can be doubled in a fixed number of RTTs. BIC [5] and CUBIC [6] focuses on RTT fairness properties by adding a binary search and a curve-fitting algorithm into the additive increase and multiplicative decrease phase. Layered TCP (LTCP) [24] modifies the LTCP flow to behave like it is a collection of virtual flows, and layers congestion control.

III. THE TCP-GT PROTOCOL

In this section, we provide a detailed description of TCP-GT. We start by explaining the design rationale of TCP-GT and describe how we materialize the protocol.

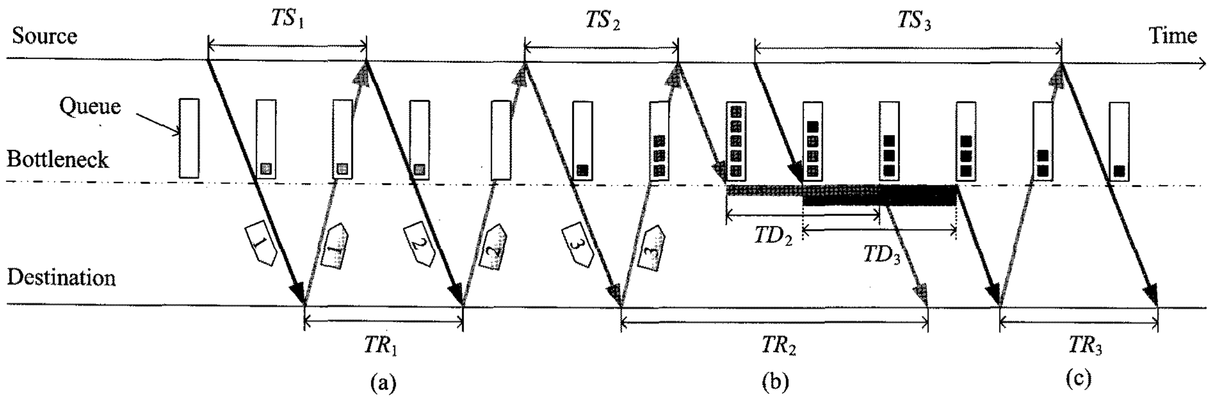


Fig. 1. From the epoch durations TS and TR observed at the sender and the receiver, respectively, a TCP-GT sender calculates goodput and throughput within an epoch. Queuing delay (TD_2 and TD_3) reflects the congestion status of the bottleneck link, and a TCP-GT sender can identify the congestion status by comparing goodput and throughput: (a) Case I, (b) case II, and (c) case III.

A. Design Rationale

The key design rationale of TCP-GT is to make TCP-GT a practical end-to-end approach for HBDP networks having nontrivial heterogeneity in link capacities and RTTs. As has been pointed out in [7], using a packet loss as a congestion signal is ill-suited for accurate flow control in HBDP networks. Nevertheless, most end-to-end approaches still use a packet loss as a congestion sign. Because the binary signal (a loss or no loss) expresses only two extreme states of a network link, an end-to-end approach needs a delicate and effective beacon which can determine bottleneck status continuously.

Unlike many router-supported approaches, an end-to-end approach has a fundamental limitation in quantitatively recognizing the load status of a bottleneck link. The meaning of end-to-end control implies that there is no way of getting link information. We, therefore, use a goodput-throughput dynamic as a new congestion signal. The actual sign we rely on is the difference between goodput and throughput measured by a sender. This value is not a binary signal, rather it indicates the link state accurately by revealing the exact amount of excessive (or spare) bandwidth. By increasing (or decreasing) a congestion window to this exact amount, a sender can respond much more quickly to the change in a bottleneck link, and this also allows multiple TCP-GT flows to share the bottleneck link fairly without bulk packet losses.

B. Measurement of Goodput and Throughput

TCP-GT measures goodput and throughput within a specific time duration, which we call an epoch, and regulates its sending rate by adjusting the congestion window in proportion to the difference between them. The precise measurement of goodput and throughput is the key factor in accurate congestion control. To get more accurate goodput and throughput values, each TCP-GT packet carries an epoch header in the optional field of the TCP header. The epoch header has a layout similar to the round-trip time measurement (RTTM) [25].

Fig. 1 illustrates the details of how to measure goodput and throughput by using epochs. When sending a packet to the destination, a TCP-GT sender writes its current epoch number in the epoch header (downward numbers in Fig. 1). Whenever a new packet arrives at the destination, a TCP-GT receiver echoes the epoch number of the new packet (upward numbers in Fig. 1). If the returned epoch number is equal to its current epoch number, the TCP-GT sender initiates a new epoch by increasing its epoch number by one. At this time, the sender records the current timestamp and the timestamp of the acknowledgment (ACK) packet using the RTTM technique. From these timestamps, the TCP-GT sender measures the duration of an epoch observed at the sender (TS) and the duration of an epoch observed at the receiver (TR). The TCP-GT sender can calculate goodput (G) and throughput (T) of a previous epoch

$$G = \frac{N_{\text{bytes}}}{TR}, \tag{1}$$

$$T = \frac{N_{\text{bytes}}}{TS} \tag{2}$$

where N_{bytes} is the total size of packets which are transferred from the sender to the receiver during one epoch. The length of one epoch observed at the sender (TS) is equal to the round-trip time.

We want to point out that the measurement period can not be shorter than the length of an epoch (i.e., RTT). The reason follows; the size of the congestion window can be understood as the total transfer size during an RTT. However, packets do not start uniformly within an RTT because packets are transferred as a packet train, which means that they are not equidistant each other, rather they are sent at the same time. Therefore, it does not make sense to measure goodput and throughput between successive acks.

C. Recognizing Congestion Status of a Bottleneck Link

To identify the bottleneck congestion status, TCP-GT utilizes the difference ϕ between goodput and throughput

$$\phi = G - T. \tag{3}$$

Fig. 1 explains the relationship between ϕ and the congestion status. There are three situations that need to be considered. Fig. 1(a) illustrates the case where the congestion level has been kept at almost the same level during an epoch. In most of this case, the congestion level is very low. In other words, throughput is lower than the bandwidth of the bottleneck link, and as a result the bottleneck queue is empty. In this case, the TCP-GT

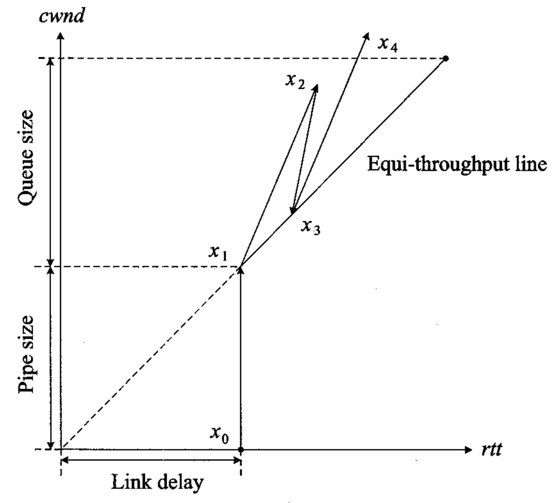


Fig. 2. Point x_0 is the start point, and point x_1 is the optimal point. Goodput and throughput are equal along the equi-throughput line.

sender increases its congestion window to occupy spare bandwidth. Fig. 1(b) illustrates the case when the congestion level has been increased during an epoch. Increased queuing delay at the bottleneck link results in an increased RTT and an increased epoch duration at the destination ($TR_2 \approx TS_2 + TD_2$) and throughput becomes larger than goodput ($\phi < 0$). To relieve the congested status at the bottleneck, the TCP-GT sender decreases its congestion window. Fig. 1(c) illustrates the case when the congestion level has been decreased during an epoch. Decreased queuing delay at the bottleneck link brings about a decreased RTT and a decreased epoch duration at the destination ($TR_3 \approx TS_3 - TD_3$). Consequently, goodput becomes larger than throughput ($\phi > 0$). This phenomenon indicates that the congestion started at the beginning of an epoch has been relieved before the end of the same epoch. At the time when a TCP-GT sender measures goodput and throughput, the bottleneck link might be underutilized. To ameliorate the link utilization, the TCP-GT sender increases its congestion window.

Fig. 2 illustrates the relation between the congestion window and RTT. A new TCP-GT flow enters into the network at x_0, and goes toward x_1. While the flow walk along the line from x_0 to x_1, its goodput and throughput increase at the same speed, and its RTT remains at the lowest value (i.e., the sum of link delays). The point x_1 is the optimal point where the bottleneck bandwidth is fully utilized and the queue starts to build up. At the optimal point, the difference ϕ is almost zero (case I in Fig. 1). If a TCP-GT sender increases its congestion window, the operating point moves to the point x_2. At point x_2, throughput becomes larger than goodput, and TCP-GT decreases its congestion window by the difference between them. We expect the operating point to go back to x_1. But if a TCP-GT sender measures small throughput by some errors (i.e., a large TS in Fig. 1), it brings the operating point to x_3 instead of x_1. In this case, the operating point never goes back from x_3 to x_1. Because x_3 is also on the equi-throughput line, its goodput and throughput are equal, and as a result, TCP-GT continues to increase the congestion window. After some cycles pass, the congestion window may eventually go beyond the sum of the pipe size and the bottleneck queue size. From that point, the bottleneck queue starts to drop packets.

The primary reason for this problem comes from the equi-throughput line. Because ϕ has the same value along the line, TCP-GT fails to recognize the queuing delay. To remedy this catastrophe, we introduced a minimum RTT $rtt_{\min}$ in the measurement of throughput

$$T = \frac{N_{\text{bytes}}}{rtt_{\min}}. \tag{4}$$

By replacing the epoch duration with $rtt_{\min}$ in (2), we now can guarantee that the operating point of TCP-GT goes safely back to the point x_1 instead of the point x_3. The technique of using $rtt_{\min}$ was originally introduced in [4] and [18], and it was reported that the use of $rtt_{\min}$ can incur a fairness problem in [15]. The main cause of the fairness problem is the disagreement on $rtt_{\min}$ between flows sharing the network. If a flow joins a network when the network is not congested, it has a lower RTT than latter flows. The larger $rtt_{\min}$ a flow has, the more bandwidth it occupies. This problem was analytically derived in [14]. To rectify this challenge, TCP-GT uses a local minimum RTT instead of global minimum RTT; TCP-GT updates its minimum RTT to the smallest RTT within the last 10 epochs.

$$rtt_{\min} = \min(rtt \text{ within the last 10 epochs}). \tag{5}$$

Unlike the global minimum RTT, the local minimum RTT increases when the network is congested. It helps all flows have similar minimum RTT values, and improves fairness between flows.

D. The IIMD Equations of TCP-GT

Multiplicative decrease (MD): To relieve the congestion, a TCP-GT sender must decrease its congestion window. The basic principle of decreasing the congestion window of a flow i, i.e., $cwnd_i$, is to adjust $cwnd_i$ according to the congestion level and its current sending rate. Because ϕ_i is principally affected by the queuing delay, the change of a congestion level necessitates adjusting ϕ_i. We change $cwnd_i$ based on ϕ_i; i.e., $\Delta cwnd_i \propto \phi_i$. The change $\Delta cwnd_i$ is the sum of per packet changes in one epoch. We obtain the per packet negative change n_i in the congestion window of flow i by dividing the change in congestion window $\Delta cwnd_i$ by its current congestion window $cwnd_i$. The difference ϕ_i is the amount of change in sending rate and $\Delta cwnd_i$ is the amount of change in the congestion window during an epoch (or RTT). To translate ϕ_i into $\Delta cwnd_i$, $\Delta cwnd_i$ is multiplied with the average RTT of flow i and divided by the size of maximum transmission unit (MTU) ($\Delta cwnd_i = \phi_i rtt_i / mtu$). The per packet negative change n_i in the congestion window of flow i is given by

$$n_i = \frac{\phi_i rtt_i}{cwnd_i mtu}. \tag{6}$$

n_i is computed every epoch, and is added to the current congestion window whenever an ACK packet arrives at a TCP-GT sender. Because the size of ϕ is directly proportional to the flow's current sending rate, TCP-GT operates in the MD mode in a decreasing phase.

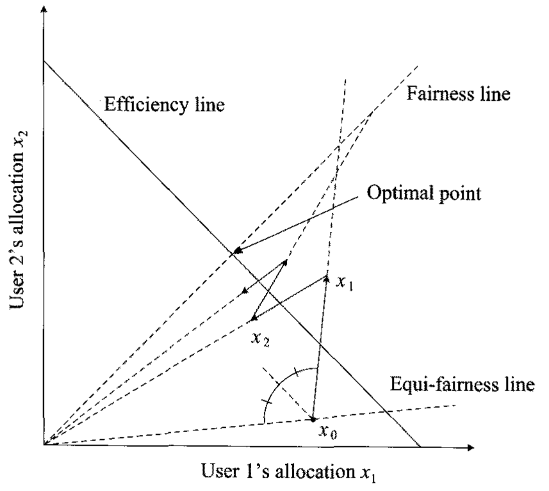


Fig. 3. A TCP-GT flow converges to fairness in both increasing and decreasing phase by its IIMD control policy. The IIMD policy is more efficient than the AIMD policy.

Inversely-proportional increase (II): Under the AIMD principle, flows converge to fairness only in a decreasing phase by the MD policy along the equi-fairness line (Fig. 3) [1]. To improve the efficiency of convergence to fairness, we want the change in the congestion window $\Delta cwnd$ to be inversely proportional to the sending rate (goodput), which is $\Delta cwnd \propto \frac{1}{G_i}$. Again, the translation from G_i to $\Delta cwnd_i$ is applied. The per packet positive change p_i is given by

$$p_i = \frac{mtu}{G_i rtt_i cwnd_i}. \tag{7}$$

When there are two flows i and j with $cwnd_i > cwnd_j$, the ratio of $\frac{\Delta cwnd_i}{\Delta cwnd_j}$ is $\frac{G_j}{G_i} (\propto \frac{cwnd_j}{cwnd_i} < 1)$, assuming $rtt_i = rtt_j$, and this guarantees convergence to fairness in an increasing phase. TCP-GT operates in the II mode in an increasing phase as well.

Fig. 3 shows a complete trajectory of the two-user system starting from point x_0 using the aforementioned IIMD equations. This representation of the dynamics of the set of controls was introduced by Chiu *et al.* in [1]. In an increasing phase, the point x_0 is below the efficiency line and both users are asked to increase their congestion windows. Because TCP-GT increases the congestion window inversely proportional to its congestion window, the users move along the line that is axial symmetric with the equi-fairness line with respect to the perpendicular line from x_0 to the fairness line. This brings them to x_1 that happens to be above the efficiency line. In the decreasing phase, the users are asked to decrease and they do so multiplicatively. This corresponds to moving towards the origin on the line joining x_1 and the origin. This brings them to point x_2, which happens to be below the efficiency line and the cycle repeats. With every cycle, the fairness increases slightly, and eventually the system converges to the optimal state. There is one big difference between AIMD and IIMD. Because IIMD's symmetric equi-fairness line is warped toward the fairness line, IIMD converges faster than AIMD[3]. IIMD goes toward the optimal point in all phases.

Notwithstanding the beauty of the II policy that it converges

[3]It is not always true, but TCP-GT makes it true.

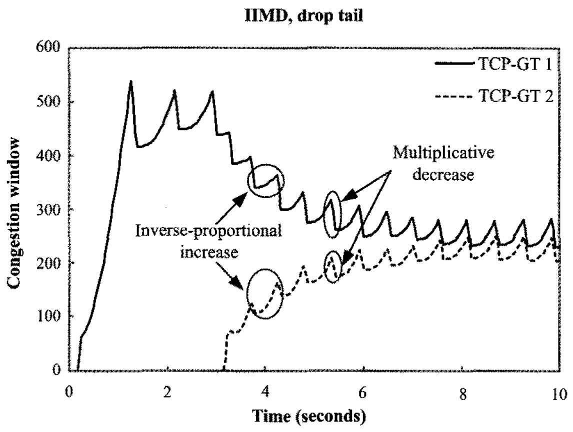


Fig. 4. The behavior of TCP-GT.

to fairness even in an increasing phase, as the congestion window becomes large, $\Delta cwnd$ decreases according to (7). This leads a flow to take a long time[4] to fully utilize a high bandwidth link; even worse, the convergence to fairness between flows is severely deteriorated. We overcome these issues by two techniques: The *window splitting technique* and *aggressive probing*.

The window splitting technique is splitting $cwnd_i$ additively into small windows when $\phi > 0$. When $\phi < 0$, it combines all split windows into a single one, which means that there is no change in the MD equation. If there are s split windows, this means that there are s split virtual flows that share the $cwnd_i$ evenly, i.e., $cwnd_i/s$ (or G_i/s since $cwnd_i \propto G_i$)

$$\overbrace{\frac{cwnd}{s} + \cdots + \frac{cwnd}{s}}^{s}. \tag{8}$$

These s split virtual flows affect the $\Delta cwnd_i$ computation; each virtual flow has the spare bandwidth G_i/s and contributes its increment $mtu/((G_i/s)rtt_i)$ to the $\Delta cwnd_i$ computation, and as a result, s flows contribute $(s^2 mtu)/(G_i rtt_i)$. Then, the new p_i can be obtained by the following equation

$$p_i = \frac{s^2 mtu}{G_i rtt_i cwnd_i}. \tag{9}$$

p_i is indeed a quadratic equation over the discretized variable s, which is incremented by one while $\phi > 0$. When we have two flows i and j, where $cwnd_i < cwnd_j$, the window splitting technique does not change the ratio of p_i/p_j compared to the one we can derive from (7). The accelerated convergence speed to fairness is the main effect what we expect from adopting the splitting technique.

The second technique is aggressive probing. The aggressive probing expedites the convergence speed in a decreasing phase as the window splitting technique does in an increasing phase. For aggressive probing, we keep the II phase going within the small amount of ϕ; in other words, TCP-GT increases $cwnd_i$ while $\phi_i \geq tT_i$[5]. Aggressive probing induces a larger bandwidth

[4]It is getting extremely slower than the AIMD policy as $cwnd$ is becoming larger.

[5]We set t to -0.05 (5%) in TCP-GT.

drop, and as a consequence, leads to faster convergence, and it creates larger spare bandwidth, which is a big margin to the II equation. Then, we have the following equation.

$$p_i = \frac{(s_i)^2 mtu}{G_i rtt_i cwnd_i} \quad (\text{if } -0.05T_i \le \phi_i < 0). \tag{10}$$

Fig. 4 shows the simulated trajectory of *cwnd* converging to fairness with 100 Mbps bandwith and 40 ms RTT. Due to the splitting technique, *cwnd* shows a quadratic behavior in an increasing phase and converges to fairness efficiently in a large *cwnd* environment.

E. RTT Unfairness

Over many decades of protocol development, working out the RTT unfairness problem has always been a notorious task. The IIMD policy of TCP-GT also has the RTT unfairness problem because flows with different RTTs have no consensus on the control cycle of *cwnd*.

The first problem lies in the computation of the splitting variable s. During the period Δt of an increasing phase starting from t_0, a flow with a longer RTT rtt_i has smaller $s_{i,t_0+\Delta t}$ than $s_{j,t_0+\Delta t}$ of a flow with a shorter RTT rtt_j

$$\frac{s_{i,t_0+\Delta t}}{s_{j,t_0+\Delta t}} = \frac{1+\frac{\Delta t}{rtt_i}}{1+\frac{\Delta t}{rtt_j}} < 1 \text{ (unfair)}. \tag{11}$$

As time elapses, this leads to a big difference between p_i and p_j

$$\frac{p_i}{p_j} = \frac{(1+\frac{\Delta t}{rtt_i})^2}{(1+\frac{\Delta t}{rtt_j})^2} < 1 \text{ (unfair)}. \tag{12}$$

The second problem arises in the p_i computation. A flow with a longer RTT increases a less amount of p_i even though s_i and s_j are assumed to be the same, moreover p_i has two unfairness causes, i.e., rtt_i and $cwnd_i$, and this makes larger unfairness

$$\frac{p_i}{p_j} = \frac{rtt_j cwnd_j}{rtt_i cwnd_i} \propto \frac{(rtt_j)^2}{(rtt_i)^2} < 1 \text{ (unfair)}. \tag{13}$$

The third problem comes out in the aggressive probing and n_i; a flow with a shorter RTT has a higher chance of detecting the congestion (i.e., $\phi_i < -tT_i$). Accordingly, a longer RTT flow detects the congestion less often and consumes more bandwidth. If a long RTT flow lowers the threshold t for the early detection of congestion, the amount of n_i is curtailed as well. When we take a close look at n_i, n_i itself does not need the RTT compensation because the unfairness term cancels out due to $rtt/cwnd$. The change of the threshold t indeed incurs the change of n_i. To remedy these coupled problems, we cut the threshold t and raise n_i in a certain degree at the same time.

We attempt to alleviate these difficulties moderately by using a compensation factor f_i, which is the ratio of rtt_i to the baseline RTT rtt_{base}. The use of the RTT compensation is also used in [24] so as to lessen the RTT unfairness problem. Considering the tradeoff between the convergence speed and the stability of flows, we set rtt_{base} to 50 ms from simulations. We apply the compensation factor $f_i(= rtt_i/rtt_{\text{base}})$ to s_i, p_i, t, and n_i as follows

$$s_{i,t+rtt_i} = s_{i,t} + f_i, \tag{14}$$

$$p_i = \frac{(f_i)^2(s_i)^2 mtu}{G_i rtt_{\text{base}} cwnd_i}, \tag{15}$$

$$\phi_i \ge \frac{t(=-0.05)}{f_i} T_i, \tag{16}$$

$$n_i = \frac{f_i \phi_i rtt_{\text{base}}}{cwnd_i mtu}. \tag{17}$$

Algorithm 1 Congestion control algorithm of TCP-GT

```
1:  Output: $p$, $n$
2:  /* Increase/reset the splitting constant */
3:  if $\phi \ge \frac{-0.05T}{f}$ then
4:  $s = s + f$;
5:  else
6:  $s = 1$;
7:  end if
8:  /* The IIMD algorithm */
9:  if $\phi \ge \frac{-0.05T}{f}$ then
10: $p = \frac{f^2 s^2 mtu}{G rtt_{\text{base}} cwnd}$;
11: else
12: $n = \frac{f \phi rtt_{\text{base}}}{cwnd mtu}$;
13: end if
```

One additional effect of using the compensation factor f is that, with the same bandwidth condition, the time to converge to fairness is almost the same under a variety of RTT conditions. This will be shown in the simulation study.

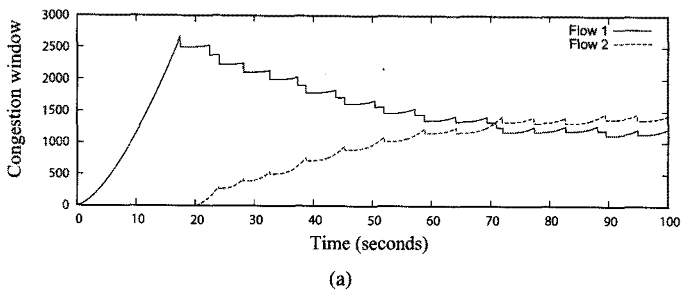
Algorithm 1 shows the congestion control algorithm of TCP-GT. The algorithm contains all equations needed and is very simple to implement.

IV. PERFORMANCE EVALUATION

In this section, we present extensive simulations presenting dynamics of TCP-GT. Every simulation uses the dumbbell topology, and we choose the *drop-tail* policy for the bottleneck queuing scheme, which is most widely used in the real world. The bottleneck queue size is set to the amount of bandwidth-delay product, and the bandwidth at a bottleneck link is set to 500 Mbps. In all simulations the packet size (MTU) is 1000 bytes, and the ACK packet size is 40 bytes.

A. Convergence Speeds in Different RTT Conditions

In this experiment, two file transfer protocol (FTP) flows share a 500 Mbps bottleneck and have a common RTT of 40 ms and 80 ms in each simulation. We configure that the first flow starts at 0 second, and the second flow enters into the network at 20 seconds. Fig. 5 shows the convergence dynamics of two TCP-GT flows in different RTT conditions. In 40 ms RTT condition (Fig. 5(a)), it takes 60 seconds to converge to fairness, and in Fig. 5(b) of 80 ms RTT, the convergence time is almost the same. This is due to the compensation factor f that attempts to adjust the RTT unfairness effect.



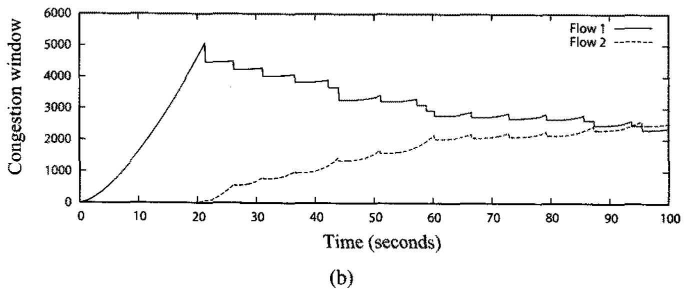


Fig. 5. TCP-GT converges to fairness at same speed regardless of RTT. Two TCP-GT flows share a 500 Mbps bottleneck: (a) 40 ms and (b) 80 ms.

Table 1. Jain's fairness index among five TCP-GT flows.

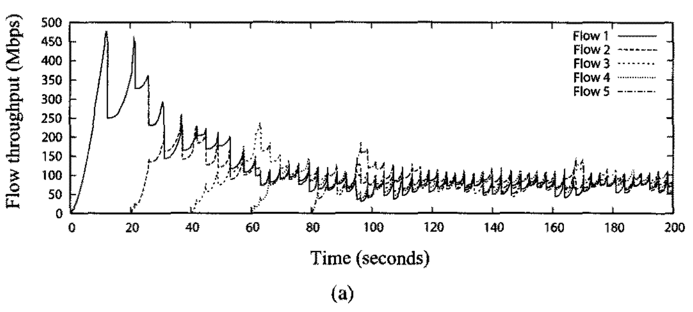
Differences in RTT	10	20	30	40
Jain's index	0.98	0.92	0.88	0.86

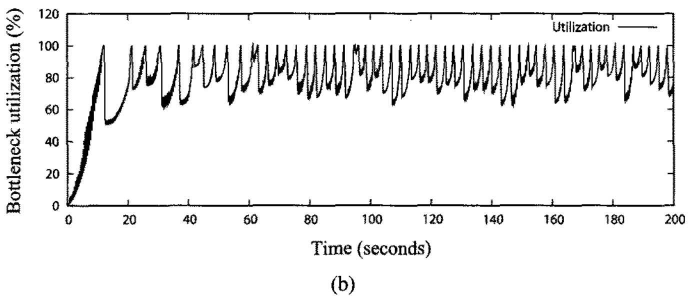
B. Convergence Dynamics with Common RTTs

In this simulation, five long-lived flows share a 500 Mbps bottleneck and have a common RTT of 80 ms. The flows start their transfers 20 seconds apart at 0, 20, 40, 60, and 80 seconds. Fig. 6(a) shows that whenever new flows come in the network, old flows and new flows converge to fairness without affecting its high utilization (Fig. 6(b)) or causing a large instantaneous queue (Fig. 6(c)). When we see the utilization graph in Fig. 6(b), all flows utilize the bottleneck link efficiently. The queue length graph in Fig. 6(c) substantiates the arguement that the queue size is not overshot. In this simulation, we observed no packet drop and low persistent queue size. These unique features come from the sensitive measurement of goodput and throughput. To the best of our knowledge, there is no end-to-end delay-based TCP implementation that shows the above behavior. Only XCP and VCP, which are supported by routers, achieve this smooth convergence. Fig. 6 also demonstrates that TCP-GT does not have a problem related with the global minimum RTT. The use of the local minimum RTT solves the problem of building up the bottleneck queue.

C. Convergence Dynamics with Heterogeneous RTTs

We have seen the situation that TCP-GT flows share bottleneck bandwidth fairly inasmuch as their RTTs are not significantly different. The convergence to fairness, however, is deteriorated when flows have large RTT heterogeneity. Fig. 7 shows the case that the RTT difference is small (less than twice). In this simulation, we set the RTT values of the five flows to different values ranging from 40 ms to 80 ms, which are 10 ms apart. All other parameters have the same values used in the previous





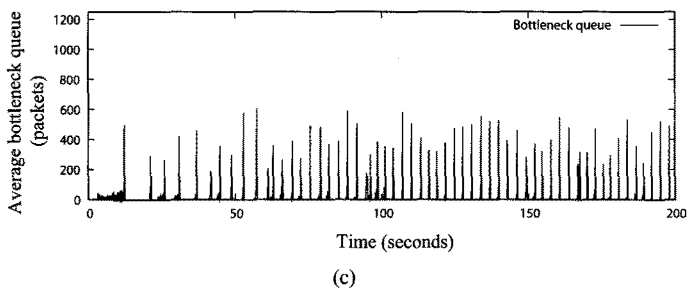


Fig. 6. Flows converge to fairness efficiently with high utilization and small queue size. Five TCP-GT flows share a 500Mbps bottleneck. They start their transmission at times 0, 20, 40, 60, and 80 seconds: (a) Throughput, (b) bottleneck utilization, and (c) bottleneck queue size.

simulation. Fig. 7(a) shows that TCP-GT converges to fairness between heterogeneous RTTs without affecting its high utilization (Fig. 7(b)) or causing large instantaneous queue (Fig. 7(c)).

Table 1 shows Jain's fairness index when five TCP-GT flows share a bottleneck link. The RTT differences between the competing flows are increased from 10 ms to 40 ms by 10 ms, and the smallest RTT is fixed to 40 ms in all experiments. In the case of 10 ms difference, the RTTs of five flows are 40, 50, 60, 70, and 80 ms. We note that since the congestion control algorithm of TCP-GT is moderately compensated by the RTT compensation factor f, the severe RTT unfairness phenomenon is not observed in this evaluation. One thing, however, we should note is that when flows have largely different RTTs, our protocol is susceptible to the RTT unfairness because very large RTTs could render the IIMD equations sensitive. One conspicuous thing is that unlike the traditional RTT unfairness that a flow with a shorter RTT consumes more bandwidth than a flow with a longer RTT, the opposite phenomenon happens in TCP-GT; a longer RTT flow consumes more bandwidth than a shorter RTT flow. This is because a flow with a long RTT usually has a less chance to decrease its congestion window, and this elongates the inversely proportional increase period. Another limitation to solve the RTT unfairness is ironically the RTT compensation

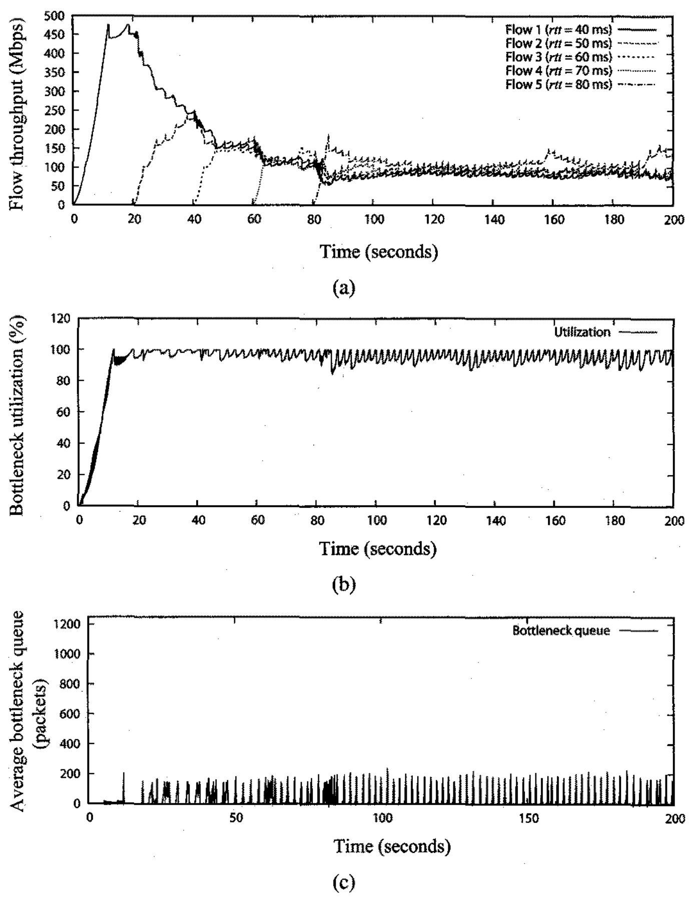


Fig. 7. TCP-GT also converges onto good fairness, high utilization, and small queue with heterogeneous RTTs. Each five TCP-GT flows have RTTs of 40, 50, 60, 70, and 80 ms, respectively: (a) Throughput, (b) bottleneck utilization, and (c) bottleneck queue size.

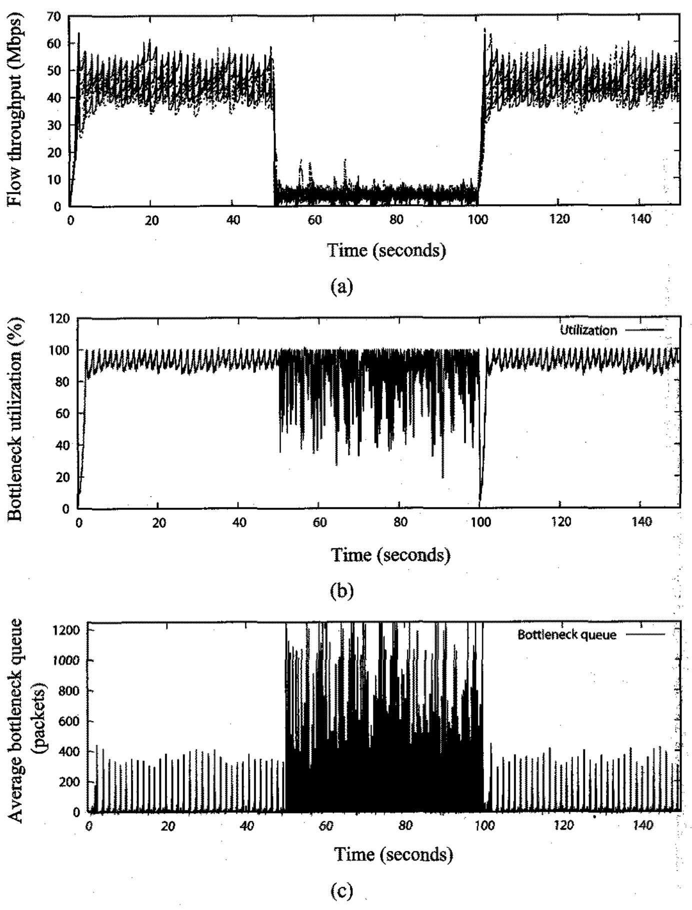


Fig. 8. TCP-GT is robust against sudden increase or decrease in traffic demands. Ten FTP flows share a bottleneck. At time t = 50 seconds, we start 100 additional flows. At t = 100 seconds, these 100 flows are suddenly stopped and the original 10 flows are left to stabilize again: (a) Throughput, (b) bottleneck utilization, and (c) bottleneck queue size.

factor itself. As an RTT grows, f also has to be large, and as a result, f^2 becomes huge enough to make our algorithm work unreliably due to the increased sensitivity of the large compensation factor. One possible solution to this is to update *cwnd* frequently enough not to have huge f.

D. *Robustness to Sudden Change in Traffic Demands*

We start the simulation with 10 long-lived FTP flows sharing a 500 Mbps bottleneck with an RTT of 40 ms. Each flow's start time is distributed between zero and one seconds uniformly. At $t = 50$ seconds, we start 100 new flows and let them stabilize. At t = 100 seconds, we stop these 100 flows, leaving the original 10 flows in the system.

Fig. 8(a) shows that TCP-GT adapts quickly to a sudden changes in traffic. We skip the graphs for these 100 short flows in Fig. 8(a) since they show the same behavior as the FTP flows do. Fig. 8(c) shows that the bottleneck queue size increases when 100 new flows enter into the network. The number of total flows becomes 10 times larger, but the bottleneck queue size increases less than 10 times. TCP-GT effectively prevents the bottleneck queue from building up when there is a sudden increase in traffic.

E. *Comparision with Other TCP Implementations*

We compare TCP-GT with other end-to-end TCP implementations, which are selective ACK (SACK), HSTCP, FAST, and CUBIC.

E.1 Impact of Capacity

In this experiment, 50 long-lived FTP flows share a bottleneck. The bottleneck capacity varies from 50 Mbps to 2 Gbps, and the RTT is 80 ms.

As shown in Fig. 9(a), we notice that the bottleneck utilization is higher than 90% in all participants. However, there are obvious differences in the bottleneck queue size (Fig. 9(b)) and the number of packet drops (Fig. 9(c)). As capacity increases, bottleneck queue sizes of competitors increase significantly. In contrast, TCP-GT's bottleneck queue size is significantly smaller than others. Furthermore, TCP-GT never drops any packets, whereas other TCP implementations drop thousands of packets. Because TCP-GT has no buffer section in the bottleneck queue (i.e., the gap between α and β in TCP-Vegas [18]), it never permits the bottleneck queue to build up. This unique behavior is also shown in the following experiments.

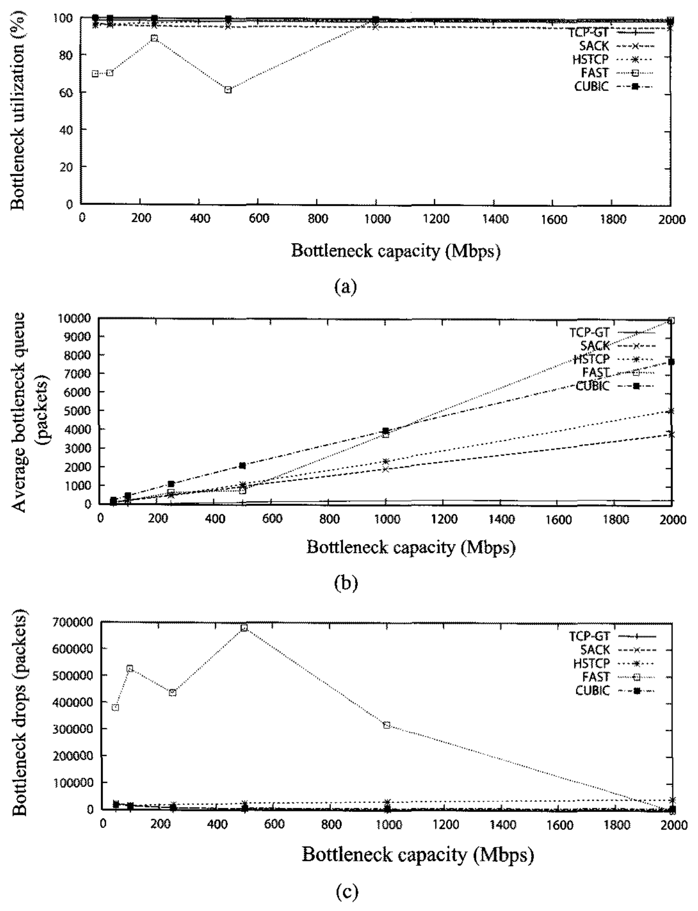


Fig. 9. TCP-GT and other high-speed TCP implementations achieve high utilization with 50 FTP flows in a range of bandwidth from 50 Mbps to 2 Gbps. TCP-GT has a distinguished low queue size and never drops packets in any simulations: (a) Throughput, (b) bottleneck queue size, and (c) bottleneck drops.

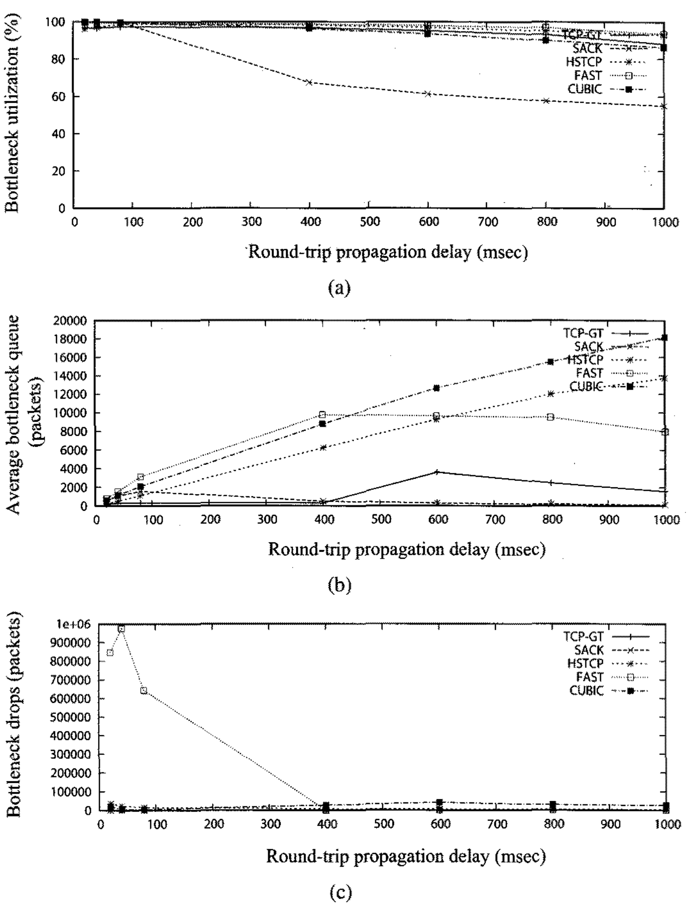


Fig. 10. TCP-GT and other high-speed TCP implementations achieve high utilization with 50 FTP flows in a range of RTT from 20 ms to 1000 ms. TCP-GT exhibits the uniqueness of small queue size and no packet loss: (a) Throughput, (b) bottleneck queue size, and (c) bottleneck drops.

E.2 Impact of RTT

We fixed the bottleneck capacity at 500 Mbps and study the impact of increased delay on the performance of congestion control. We vary the RTT from 20 ms to 1000 ms. All other parameters have the same values used in the previous experiment.

Fig. 10(a) shows that as the RTT increases, the bottleneck utilization decreases in all TCP implementations including TCP-GT. TCP-GT achieves a higher utilization than other TCP implementations regardless of the number of flows. This feature comes from the fast convergence characteristic of TCP-GT. As in the previous experiments, TCP-GT's bottleneck queue size remains at a low level (Fig. 10(b)), and it does not drop any packets (Fig. 10(c)).

E.3 Impact of Number of Flows

We fixed the bottleneck capacity to 300 Mbps and the RTT to 80 ms. We repeat the same experiment with varying numbers of FTP sources. Other parameters have the same values used in the previous experiment. Fig. 11 shows that overall, TCP-GT exhibits high utilization (Fig. 11(a)), small queue size (Fig. 11(b)), and no packet losses (Fig. 11(c)). The queue size of TCP-GT does not change significantly in any cases. This behavior derives largely from the decrease policy in such a way that each TCP-GT flow decreases the congestion window by the exact amount that is estimated by means of the ϕ calculation.

E.4 Impact of Short Web-Like Traffic

Since a large number of flows in the Internet are short web-like flows, it is important to investigate the impact of such dynamic flows on congestion control. In this experiment, we have 50 long-lived FTP flows traversing the bottleneck link. We fixed the bottleneck capacity at 500 Mbps and the RTT at 40 ms. We repeat the same experiment with varying the number of web sessions from 50 to 1000. The number of users, inter-page arrival time, the number of objects per page, and inter-object arrival time were 10, 4 seconds, 10, and 0.01 seconds, respectively. Their transfer size was derived from a Pareto distribution with an average of 3000 bytes (ns-implementation with shape = 1.2). Other parameters have the same values used in the previous experiment.

Fig.12 shows the results. As the number of sessions exceeds 600, TCP-GT starts to drop packets (Fig. 12(c)). However, the utilization is still kept at near 100% (Fig. 12(a)), and its queue size is at lowest level among all competitors (Fig. 12(b)). Though the web traffics create a severe fluctuation in the measurement of goodput and throughput, TCP-GT's window split-

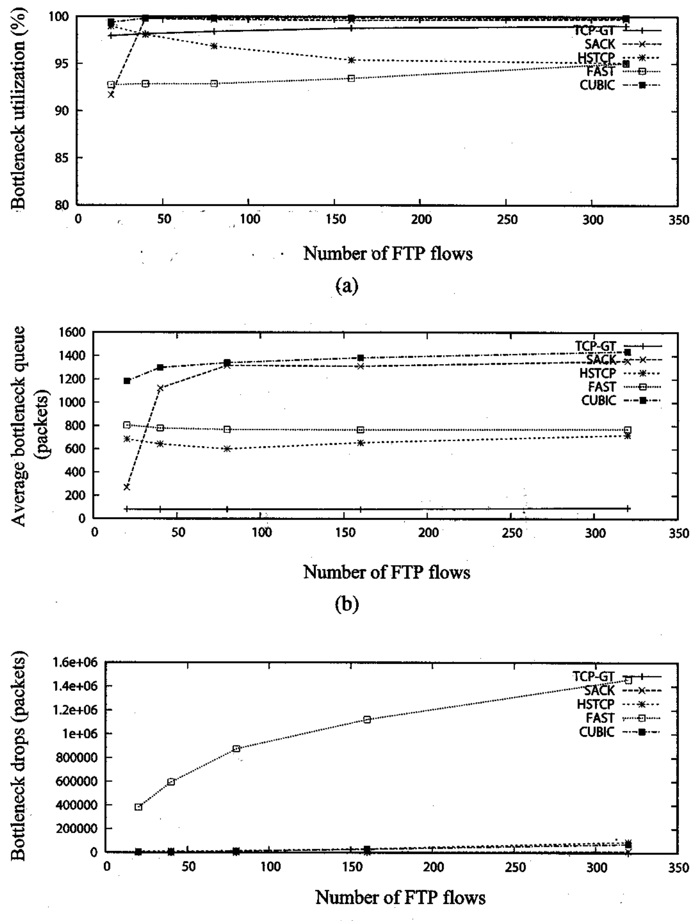


Fig. 11. TCP-GT and other high-speed TCP implementations achieve high utilization as the number of flows increases. With small number of flows, TCP-GT shows best utilization. In all simulations, TCP-GT never drops a packet: (a) Throughput, (b) bottleneck queue size, and (c) bottleneck drops.

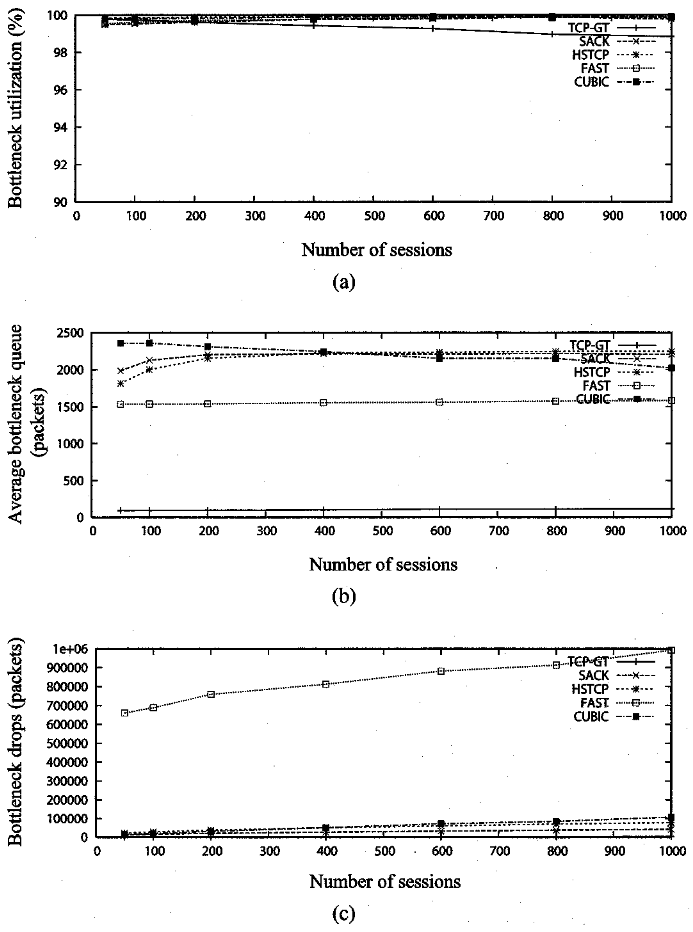


Fig. 12. TCP-GT and all other participants achieve high utilization with web-like flows. The number of web sessions varies from 50 to 1000. As the number of web sessions goes beyond 600, TCP-GT also starts to drop packets. But, TCP-GT always drops fewer packets than other protocols do: (a) Throughput, (b) bottleneck queue size, and (c) bottleneck drops.

ting technique and aggresive probing help to keep high utilization in this situation.

F. Limitations and Discussion of TCP-GT

Unstable behavior: Throughout the extensive experimentations, we realized that TCP-GT showed unstable behaviors under the circumstance that the bottleneck capacity is less than 50 Mbps, or an RTT is less than 20 ms. The reason of this unstable behavior can be found in the per-packet positive feedback (9). Because the per-packet positive feedback is inversely proportional to goodput (G) and RTT (rtt), the size of the feedback becomes larger as bandwidth of a bottleneck link and RTT get lower. In this environment, TCP-GT overshoots and drops many packets.

TCP-GT is affected by the accuracy of measured RTT values. If there are some errors in the RTT measurement, TCP-GT outputs wrong p and n value. However, this problem becomes serious only when the error keeps occurring in the RTT measurement. If the measurement error occurs rarely, the unintended misbehavior will be quickly recovered by subsequent correct control. Indeed, the method of measuring RTT is simple and straightforward, and an error occurs very rarely with a small deviation. Based on our experience and observations, rarely happening errors in measured factors are not a serious matter in TCP-GT.

TCP-unfriendliness: TCP-GT is a delay-based protocol that is known not to be compatible with packet loss-based protocols because packet loss-based protocols usually increase the congestion window until they detect any packet drop. While loss-based protocols increase the window, TCP-GT keeps decreasing its congestion window since in most cases ϕ would be negative.

Measuring a local minimum RTT: We use a local minimum RTT to resolve the fairness problem of TCP-GT. Our choice is totally based on extensive simulations (empirical study) like what we can see in [26] and [27]. Instead of 10 epochs, 9 or 11 epochs can be used for this purpose as well. Our choice is a tradeoff between response time and stability. If we choose longer epochs, TCP-GT reacts slowly to the change of the number of competing flows or network conditions. In contrast, if we set shorter epochs, TCP-GT surely shows an unstable behavior as was addressed previously.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose TCP-GT, a delay-based end-to-end protocol. Motivated by XCP and TCP-Vegas, we choose the difference between goodput and throughput as a degree of congestion in the network. We show that TCP-GT converges to fairness efficiently. The window splitting technique and aggressive probing improve convergence to fairness, and the RTT compensation mitigates the RTT unfairness moderately. The RTT compensation factor might be unreliable because flows with largely different RTTs could make the IIMD equations very sensitive. We are currently working on this issue further along with the TCP-friendliness issue. Because TCP-GT is a delay-sensitive congestion control protocol, it is not able to work properly with other loss-based protocols. Whenever there are flows which use loss-based protocols, flows using TCP-GT always yield bandwidth to other flows.

REFERENCES

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

[2] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[3] S. Floyd, "Highspeed TCP for large congestion windows," RFC 3649.

[4] C. Jin, D. X. Wei, and S. H. Low, "Fast TCP: Motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, Mar. 2004.

[5] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long distance networks," in *Proc. IEEE INFOCOM*, Mar. 2004.

[6] I. Rhee and L. Xu, "Cubic: A new tcp-friendly high-speed TCP variant," in *Proc. PFLDnet*, Feb. 2005.

[7] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, 2002, pp. 89–102.

[8] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," in *Proc. ACM SIGCOMM*, 2005, pp. 37–48.

[9] I. A. Qazi and T. Znati, "On the design of load factor based congestion control protocols for next-generation networks," in *Proc. IEEE INFOCOM*, 2008.

[10] R. Jain, S. Kalyanaraman, and R. Viswanathan, "The osu scheme for congestion avoidance in atm networks: Lessons learnt and extensions," *Performance Evaluation*, vol. 31, no. 1, pp. 67–88, 1997.

[11] X. Huang, C. Lin, F. Ren, G. Yang, P. Ungsunan, and Y. Wang, "Improving the convergence and stability of congestion control algorithm," in *Proc. IEEE ICNP*, 2007, pp. 206–215.

[12] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for TCP," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 356–369, 2003.

[13] U. Hengartner, J. Bolliger, and T. Gross, "TCP vegas revisited," in *Proc. IEEE INFOCOM*, 2000.

[14] J.-S. Li and C.-W. Ma, "Improving fairness of TCP vegas," *Int. J. Netw. Manag.*, vol. 15, no. 1, pp. 3–10, 2005.

[15] C. Boutremans and J.-Y. L. Boudec, "A note on the fairness of TCP vegas," in *Proc. Int. Zurich Seminar on Broadband Commun.*, Feb. 2000.

[16] S. H. Low, L. Peterson, and L. Wang, "Understanding TCP vegas: A duality model," in *Proc. SIGMETRICS*, NY, USA: ACM, 2001, pp. 226–235.

[17] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, pp. 56–71, 1989.

[18] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, 1994, pp. 24–35.

[19] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP vegas: Emulation and experiment," in *Proc. ACM SIGCOMM*, NY, USA, 1995, pp. 185–195.

[20] J.-S. Li and C.-W. Ma, "Improving fairness of TCP vegas," *Int. J. Netw. Manag.*, vol. 15, no. 1, pp. 3–10, 2005.

[21] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," in *Proc. IEEE INFOCOM*, 1999.

[22] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proc. PFLDNet*, 2004.

[23] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, 2003.

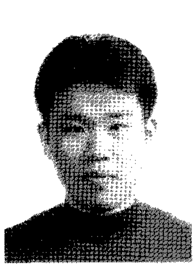
[24] S. Bhandarkar, S. Jain, and A. Reddy, "Improving TCP performance in high bandwidth high RTT links using layered congestion control," in *Proc. PFLDNet*, Feb. 2005.

[25] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, May 1992.

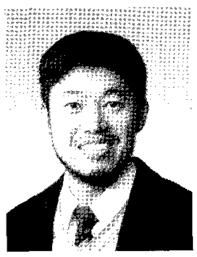
[26] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCPreno and vegas," in *Proc. IEEE INFOCOM*, 1999, pp. 1556–1563.

[27] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, 2006.

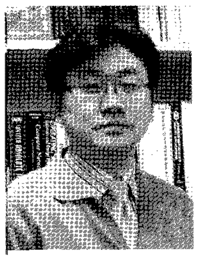
Hyungsoo Jung received his B.S. degree in Mechanical Engineering from the Korea University, Seoul, Korea, in 2002 and his M.S. degree in the Computer Science and Engineering from the Seoul National University, Seoul, Korea, in 2004. Currently, he is a Ph.D. candidate at Seoul National University. His research interests are computer networks, distributed systems, and large-scale data management systems.



Shin-Gyu Kim received his B.S. and M.S. degrees in the Computer Science and Engineering from the Seoul National University, Seoul, Korea, in 2006 and 2008, respectively. Currently, he is a Ph.D. candidate at Seoul National University. His research interests are distributed computing systems and large-scale data processing systems.



Heon Young Yeom is a Professor with the Department of Computer Science and Engineering, Seoul National University. He received his B.S. degree in the Computer Science from the Seoul National University in 1984 and received the M.S. and Ph.D. degrees in Computer Science from Texas A&M University in 1986 and 1992, respectively. From 1992 to 1993, he was with Samsung Data Systems as a Research Scientist. He joined the Department of Computer Science, Seoul National University in 1993, where he currently teaches and researches on distributed systems, multimedia systems, and transaction processing, etc.



Sooyong Kang received his B.S. degree in the Mathematics and the M.S. and Ph.D. degrees in the Computer Science, from the Seoul National University, Seoul, Korea, in 1996, 1998, and 2002, respectively. He was then a Postdoctoral Researcher in the School of Computer Science and Engineering, Seoul National University. He is now with the Division of Computer Science and Engineering, Hanyang University, Seoul. His research interests include operating systems, multimedia systems, storage systems, flash memories and next generation nonvolatile memories, and distributed computing systems.