IEICE TRANS. COMMUN., VOL.E90–B, NO.3 MARCH 2007

PAPER

TCP-Ho: A Congestion Control Algorithm with Design and Performance Evaluation

Cheng-Yuan HO[†a)], ***Student Member***, **Yi-Cheng CHAN**[††], ***and*** **Yaw-Chung CHEN**[†], ***Members***

SUMMARY A critical design issue of Transmission Control Protocol (TCP) is its congestion control that allows the protocol to adjust the end-to-end communication rate based on the detection of packet loss. However, TCP congestion control may function poorly during its slow start and congestion avoidance phases. This is because TCP sends bursts of packets with the fast window increase and the ACK-clock based transmission in slow start, and respond slowly with large congestion windows especially in high bandwidth-delay product (BDP) networks during congestion avoidance. In this article, we propose an improved version of TCP, TCP-Ho, that uses an efficient congestion window control algorithm for a TCP source. According to the estimated available bandwidth and measured round-trip times (RTTs), the proposed algorithm adjusts the congestion window size with a rate between exponential growth and linear growth intelligently. Our extensive simulation results show that TCP-Ho significantly improves the performance of connections as well as remaining fair and stable when the BDP increases. Furthermore, it is feasible to implement because only sending part needs to be modified.

key words: *congestion control, TCP, slow start, congestion avoidance, transport protocol*

1. Introduction

With the fast growth of Internet traffic, we need a congestion control scheme that can efficiently utilize the network's resources. TCP is the most popular transport protocol for the current Internet because it provides a reliable data transport between two end hosts of a connection as well as controls the connection's bandwidth usage to avoid network congestion.

The essential strategy of TCP is sending packets to the network without a reservation and then reacting to observable events occurred. Original TCP is officially defined in [1]. It has a simple sliding window flow control mechanism without any congestion control. After observing a series of congestion collapses in 1980's, Jacobson introduced several innovative congestion control mechanisms into TCP in 1988. This TCP version, called TCP Tahoe [2], includes the slow start, additive increase and multiplicative decrease (AIMD), and fast retransmit algorithms. Two years later, the fast recovery algorithm was added to Tahoe to form a new TCP version called TCP Reno [3]. Reno is currently the dominating TCP version deployed in the Internet. TCP Reno can be thought as a reactive congestion control scheme. It uses packet loss as an indicator for congestion. In order to probe the available bandwidth along the end-to-end path, TCP congestion window will be increased until a packet loss is detected, at which point the congestion window is halved and a linear increase algorithm will take over until further packet loss is experienced. It is known that TCP Reno may periodically generate packet loss by itself and can not efficiently recover multiple packet losses from a window of data [4]–[8]. Moreover, the AIMD strategy of TCP Reno leads to periodic oscillations in the aspects of congestion window size, round-trip delay, and queue length of the bottleneck node [9]–[14].

To alleviate the performance degradation problem of packet loss, many researchers attempted to refine the fast recovery algorithm embedded in TCP Reno. New proposals includes TCP NewReno [4], SACK [5], FACK [6], Net Reno [7], and LT [8]. All these algorithms bring performance improvement for a connection after a packet loss is detected. To combat the inherent oscillation problem of Reno, many congestion avoidance mechanisms are proposed. These works include DUAL [9], CARD [10], Tri-S [11], Packet-Pair [12], TCP Vegas [13], and TCP Santa Cruz [14]. Among these creative mechanisms, TCP Vegas is a notable approach because it can successfully avoid the congestion in the network.

Although TCP has several implementation versions which intend to improve network utilization, TCP still suffers problems that inhere in its congestion control algorithm. For example, the sender has no prior knowledge regarding the available bandwidth on the networks, this leads to the abrupt transition of congestion window with exponential growth and transmission of highly bursty traffic from the source, and it in turn would cause buffer overflow at the bottleneck link during the slow start phase. When the per-flow product of bandwidth and latency increases, the congestion avoidance scheme of TCP becomes inefficient. This will be problematic for TCP as the bandwidth-delay product (BDP) of Internet continues to grow. All these problems may prevent TCP from achieving a success.

In this work, we propose a modified congestion control mechanism for TCP (abbreviated as TCP-Ho hereafter). By estimating the available bandwidth of the bottleneck link and measuring the RTT for every packet, TCP-Ho updates the congestion window size with a rate between exponential growth and linear growth intelligently, so that a smooth

Manuscript received September 1, 2005.
Manuscript revised March 8, 2006.

[†]The authors are with the Department of Computer Science, National Chiao Tung University, No. 1001, Ta Hsueh Road, Hsinchu City, 30050, Taiwan.
[††]The author is with the Department of Computer Science and Information Engineering, National Changhua University of Education, No. 1, Jin-De Road, Changhua City, 50007, Taiwan.
a) E-mail: cyho@csie.nctu.edu.tw

DOI: 10.1093/ietcom/e90–b.3.516

transmission can be achieved. Furthermore, TCP-Ho integrates slow start and congestion avoidance phases into one fundamental mechanism. We demonstrate the effectiveness of the proposed algorithm based on the results of analysis and simulation. Moreover, the implementation of TCP-Ho is simple because only the sending part requires modifications, thus it facilitates incremental deployment in today's Internet.

The remainder of this paper is organized as follows. Related work is described in Sect. 2. Section 3 explains our motivation and goals for using the TCP-Ho mechanism and describes it in detail. The mathematical analysis of TCP-Ho is presented in Sect. 4. Section 5 demonstrates the simulation results. Lastly, we conclude this work in Sect. 6 with a summary of the results and highlights of the future works.

2. Related Work

Several studies have been made to improve the connection performance over Internet. These approaches can be divided into two categories. One needs some changes or modifications in the slow start phase such as Smooth-start [15], γ Selection [16], Smooth Slow Start [17], and Limited Slow-Start [24]. The other varies the congestion avoidance mechanism, for example, HighSpeed TCP [18], Scalable TCP [19], XCP [20], DUAL [9], CARD [10], and FAST TCP [25].

2.1 Modifications in Slow Start Phase

The first approach (Smooth-start) is to set the maximum slow start threshold to avoid buffer overflow and limit the sending rate, but it not only reduces the throughput of a sender but also sets the maximum slow start threshold to 64 kbytes. In a large BDP network, this value may be too small and causes TCP switching to congestion avoidance phase early. On the other hand, this fixed slow start threshold may be of no use in a small BDP network. Another way (γ Selection) is selecting γ dynamically to suit various kinds of BDP networks, but it needs to estimate the available bandwidth of the network at the steady state. However, to estimate the available bandwidth based on end-to-end congestion avoidance mechanism on a global internet is difficult and this mechanism is only for TCP Vegas.

Next, Smooth Slow Start uses a smooth slow start algorithm to reduce burst data transfer. However, it uses 200 msec timer interrupt to control data transfer and it only fits some network topology. Furthermore, using timer interrupt increases the overhead of the operating system. Overall, these proposed mechanisms only work well in some network models, and the burstiness problem is still not solved completely. The last method (Limited Slow-Start) introduces a parameter, max_ssthresh which is smaller than slow start threshold (SSTHRESH). During slow start, the sender doubles its congestion window when the congestion window size (CWND) is not greater than max_ssthresh. If the value of CWND is between max_ssthresh and SSTHRESH, the congestion window is increased by $1/2$ maximum segment size (MSS) or less for each arriving acknowledgement (ACK). Otherwise, the sender leaves its slow start phase and goes into the congestion avoidance phase. Although Limited Slow-Start algorithm can reduce the number of drops, a proper value of max_ssthresh is hard to decide. The authors recommend a sender to set the max_ssthresh value at 100 MSS. It is obvious that a fixed value may be not adapted to varied network topologies.

2.2 Congestion Avoidance Schemes

HighSpeed TCP involves a subtle change in the congestion avoidance response function to allow connections to capture available bandwidth more readily. Scalable TCP is similar to HighSpeed TCP in that the congestion window response function for large windows is modified to recover more quickly from loss events and hence reduce the penalty for probing the available bandwidth. The same as TCP Reno, both HighSpeed TCP and Scalable TCP use packet loss as an indication for congestion. This causes periodic oscillations in the congestion window size, round-trip delay, and queue length of the bottleneck node. These drawbacks may not be appropriate for emerging Internet applications. XCP is a new transport protocol designed for high BDP networks. It separates the efficiency and fairness policies of congestion control, and enables connections to quickly make use of available bandwidth. However, because XCP requires all routers along the path to participate, deployment feasibility is a concern.

The window in Jain's CARD approach is increased by one packet size and decreased by one-eighth based on the gradient of delay-window curve, which is used to evaluate the optimal point of the system[†]. The performance of the window control mechanism was studied with a deterministic simulation model of a connection in a wide-area network. DUAL scheme defines one optimal point with queue length and uses the corresponding delay as the congestion signal. The congestion window normally uses fine-tuning to adjust window size, namely increases by 1/CWND for each ACK received. The algorithm decreases the congestion window by one-eighth if the current RTT is greater than the average of the minimum and maximum RTTs observed so far for every two RTTs. If a timeout is detected, the algorithm assumes that substantial traffic increase and severe congestion have occurred. It uses quick-turning to reduce the window size, similar to TCP Tahoe timeout action.

FAST TCP incorporates multiplicative increase if the buffer occupied by the connection at the bottleneck is far less than some pre-defined threshold α and switch to linear increase if it is near α. Then, FAST tries to maintain the buffer occupancy around α and reduces sending rate if delay is further increased. Theoretical analysis and experiments show that FAST TCP has better properties than pure

[†]Note that the window changes during every adjustment, that is, it oscillates around its optimal point.

loss-based approaches, such as higher utilization, less self-induced packet losses, faster convergence speed, better RTT fairness and stabilization. However, FAST may not be able to obtain fair share when it is competing with loss-based approaches like TCP Reno. Moreover, how to set a suitable α value for a FAST source may be another possible problem since the α is a function of buffer size and the number of concurrent connections, which are generally unknown in a real world network.

3. TCP-Ho

3.1 Motivation

There are some problems of TCP in the slow start and congestion avoidance phases. TCP Reno takes packet loss as an indiction of congestion. In order to probe available bandwidth along the end-to-end path, it periodically creates packet losses by itself. It is well-known that TCP Reno may feature poor utilization of bottleneck link under high BDP networks. Since TCP Reno uses AIMD algorithm to adjust its window size, when packet losses occur, it cuts the congestion window size to half and linearly increases the congestion window until next congestion event is detected. The additive increase policy limits TCP's ability to acquire spare bandwidth at one packet per round. The BDP of a single connection over very high bandwidth links may be thousands of packets, thus TCP Reno might waste thousands of rounds to ramp up to full utilization of the link. For example, the time of a connection to converge to an optimal bandwidth value can take the order of minutes in a high BDP network which with 1 Gbps available bandwidth and 100 ms RTT. Thus, if TCP's convergence mechanism is too sluggish, TCP will eventually become a performance bottleneck itself.

Unlike TCP Reno which uses binary congestion signal, packet loss, to adjust its window size, TCP Vegas adopts a more fine-grained signal, queuing delay, to avoid congestion. Although it successfully detects network congestion in the early stage, the burstiness causes TCP Vegas to tend to prematurely stop the exponentially increasing slow start phase and enter the slower congestion avoidance phase until it reaches its equilibrium congestion window size, especially in high BDP networks. As a result, a new TCP Vegas connection may experience a very long transient period and throughput suffers. In addition, the availability of network resources and the number of competing users may vary over time unpredictably. It is sure that the available bandwidth does not vary linearly. Since TCP Vegas adjusts its congestion window linearly in the congestion avoidance phase, this prevents TCP Vegas from quickly adapt to the changing environments.

3.2 The Scheme of TCP-Ho

Doubling the congestion window size during slow start causes the traffic burstiness in the network and a linear increase in the congestion avoidance phase limits a sender's ability to ramp up to full utilization of the link; therefore, in TCP-Ho, we increase the congestion window between exponential growth and linear growth. We call this growth *Stable growth* as in Fig. 1. Furthermore, the 'quick shift' phase is introduced into our proposed mechanism instead of using the slow start and congestion avoidance phases. In other words, we combine slow start and congestion avoidance to quick shift. The details of TCP-Ho including the *quick shift* phase and *Stable growth* are as follows.

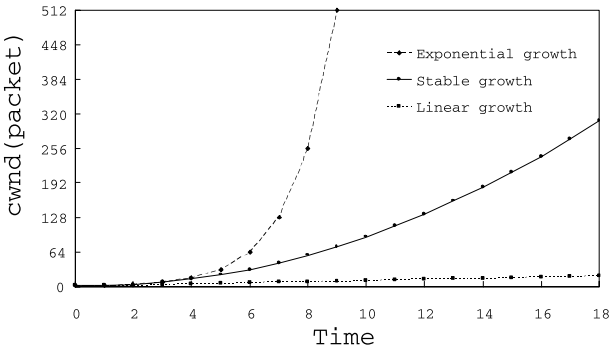

Fig. 1 The growth of congestion window size.

TCP-Ho's window adjustment algorithm consists of two phases: quick shift (QS), and fast retransmit and fast recovery (FF). The congestion window is updated based on the currently executing phase. Before describing the algorithm of TCP-Ho to adjust the congestion window, we have to define some parameters and terms. Let *'incr'* be the current window increment, with value 0 at the beginning of a new connection or after a retransmission timeout, and *'maxincr'* be a dynamic value representing the maximum value of the congestion window increment. The minimum RTT and the maximum RTT measured from packets with one round are called *'Minimal RTT'* and *'Maximal RTT'* respectively. It is because the minimum RTT in this round does not necessarily equal the minimum of ever measured round-trip times. Similarly, the maximum RTT in this round does not necessarily equal the maximum of ever measured round-trip times. *'Prior'* means the $(n-2)$th round's and *'Current'* represent the nth round's. For example, Current Maximal Throughput (CMaT) and Current Minimal Throughput (CMiT) are the maximal throughput and minimal throughput in the nth round, and Prior Maximal Throughput (PMaT) and Prior Minimal Throughput (PMiT) are the maximal throughput and minimal throughput in the $(n-2)$th round, where Maximal Throughput is the *CWND* divided by Minimal RTT and Minimal Throughput is the *CWND* divided by Maximal RTT. In addition, CMiT $\leq$ CMaT and PMiT $\leq$ PMaT. In order to record the comparison result of CMaT, CMiT, PMaT, and PMiT at the last time, we also create a parameter *'status,'* whose default value is 1. Table 1 shows the meanings of the status' value.

During the quick shift phase, TCP-Ho does not continually increase the congestion window. Instead, it tries to

Table 1 The meanings of the status' values.

status' value	comparison situation
1	PMaT < CMiT
2	CMaT < PMiT
3	PMiT < CMiT and PMaT < CMaT
4	CMiT < PMiT and CMaT < PMaT
5	Otherwise

Table 2 Comparison results, two steps, status' values, and corresponding motivations.

comparison results	steps	status values	corresponding motivations
PMaT < CMiT	One	(1) 1 or 3	*'incr'* += 1. If *'incr'* > *'maxincr,'* *'maxincr'* = *'incr.'*
		(2) o/w	*'maxincr'* = 1, & *'incr'* = 1.
	Two		CWND += *'incr,'* & *'status'* = 1.
CMaT < PMiT	One	2 or 4	*'maxincr'* -= 1, & *'maxincr'* ≥ 1.
	Two		CWND -= *'maxincr,'* & CWND ≥ 2, *'incr'* = 0, & *'status'* = 2.
PMiT < CMiT & PMaT < CMaT	One	(1) 1 or 3	*'incr'* /= 2, & *'incr'* ≥ 1.
		(2) o/w	*'incr'* = 1.
	Two		CWND += *'incr,'* & *'status'* = 3.
CMiT < PMiT & CMaT < PMaT	One	2 or 4	*'maxincr'* /= 2 & *'maxincr'* ≥ 1.
	Two		CWND -= *'maxincr,'* & CWND ≥ 2, *'incr'* = 0, & *'status'* = 4.
Otherwise		1–5	*'maxincr'* = 0, *'incr'* = 0, & *'status'* = 5.

detect relative congestion by comparing CMiT and CMaT with PMiT and PMaT. We do not increase congestion window size in the first round, which is the beginning of a new connection and/or after a retransmission timeout, because we have no idea about the available bandwidth in the end-to-end path. When a sending source increases (or decreases) its congestion window at the ith round, the influence to the network can be detected at the $(i + 2)$th round. As a result, from the second round, we adjust the increase (or decrease) amount every other round (if it is possible). In other words, the increase (or decrease) amounts in the ith and $(i + 1)$th rounds will be the same. The increase (or decrease) amount and CWND is updated according to the following descriptions, as briefly presented in Table 2. After comparing CMiT and CMaT with PMiT and PMaT, there are two steps for adjusting increase or decrease amount and CWND. (1) PMaT is smaller than CMiT: Step one, when *'status'* value equals 1 or 3, *'incr'* is increased by one and set the *'maxincr'* to *'incr'* if *'incr'* is bigger than *'maxincr'*; otherwise, set the value of *'incr'* and *'maxincr'* to 1. Step two, CWND is increased by the *'incr'* value and set *'status'* to 1; (2) CMaT is smaller than PMiT: Step one, while *'status'* value equals 2 or 4, *'maxincr'* is decreased by one and *'maxincr'* is not smaller than 1. Step two, CWND is decreased by the *'maxincr'* value and CWND is not smaller than 2, let *'incr'* to 0, and set *'status'* to 2; (3) PMiT is smaller than CMiT and PMaT is smaller than CMaT: Step one, when *'status'* value equals 1 or 3, *'incr'* value is halved and *'incr'* is not smaller than 1; otherwise, set the value of *'incr'* to 1. Step two, CWND is increased by the *'incr'* value and set *'status'* to 3; (4) CMiT is smaller than PMiT and CMaT is smaller than PMaT: Step one, while *'status'* value equals 2 or 4, *'maxincr'* is halved and *'maxincr'* is not smaller than 1. Step two, CWND is decreased by the *'maxincr'* value and CWND is not smaller than 2, let *'incr'* to 0, and *'status'* to 4; and (5) otherwise, set the value of *'incr'* and *'maxincr'* to 0, and *'status'* to 5.

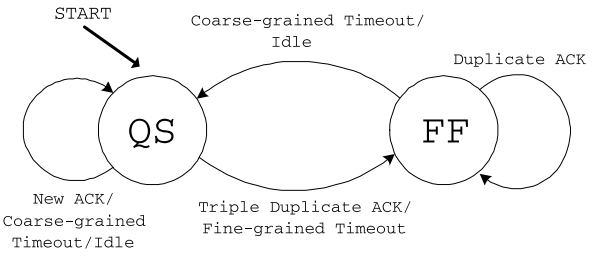

Fig. 2 Phase transition diagram of TCP-Ho.

This approach for increasing and decreasing congestion window is more efficient comparing with TCP Reno and TCP Vegas, and the effect can be observed by both simulation and analysis. Since we focus on the congestion control and flow control in this work, the fast retransmit and fast recovery of TCP-Ho are same as those of TCP Vegas. In addition, in the FF phase, we reset all parameters used in the QS phase to their default values. Figure 2 shows the phase transition diagram of TCP-Ho. A connection begins with the QS phase. The window-adjustment phase transition is owing to the specific events as depicted along the edges.

In summary, TCP-Ho transmits one packets when receiving one ACK, and it sends an extra packet to increase the congestion window size after getting two or more ACKs. Through this method, we smooth out the burstiness without using timer. It is because comparing with congestion window, the increment is always small. Thus, TCP-Ho reduces the burstiness in transmission. Furthermore, TCP-Ho adjusts its congestion window size with *Stable growth* according to the difference between measured available bandwidths in the mth and $(m + 2)$th rounds. Therefore, the throughput of TCP-Ho is much larger and grows faster than TCP Reno and TCP Vegas.

4. Numerical Analysis

In this section, we present behavior analysis of TCP Vegas and TCP-Ho. A simple case is considered when a single connection tries to fill up an empty network with N links connecting the source and the destination. Figure 3 shows

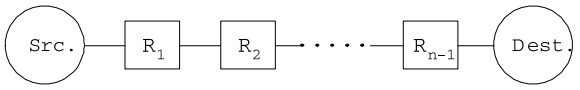

Fig. 3 Network topology for analysis.

the network topology for analysis. We denote the transmission rate of N links (in packets/s) as X_i, i = 1,...,N, and the total round-trip propagation delay of the route path (in seconds) as τ. Similar to the work in [16], we assume that there is one bottleneck in the route path and $X_1 \le X_2 \le ... \le X_N$. Since X_1 is the smallest transmission rate (i.e., link 1 behaves as the bottleneck link), we let μ be equal to X_1. The un-congested BDP of this network is then given by μd where

$$d = \tau + (1 + a) \sum_{i=1}^{N} \frac{1}{X_i}, \tag{1}$$

with a being the ACK size relative to the data packet size. Without loss of generality, we assume that a is much smaller than the data packet size, so we use 1 to approximate $1+a$ (i.e., $d = \tau + \sum_{i=1}^{N} \frac{1}{X_i}$).

Throughout our analysis, we assume a fluid model and the source always has a packet to transmit, the buffer sizes in routers are large enough so that packet loss can be ignored when analyzing the behavior of TCP Vegas and TCP-Ho. Moreover, the ith round starts with the transmission of W_i packets where W_i is the size of congestion window in this round. The ith round ends when the source receives the ACK of the first packet in this round, then the source starts transmitting a new packet of the next round. Suppose that there is no congestion in ACK path. The mth round-trip time is named 'conspicuous round-trip time' when the "steady throughput" of a sender reaches the available bandwidth at the first time. In the following subsections, we derive mathematically the conspicuous round-trip time for TCP Vegas and TCP-Ho. Then we ascertain the better transient performance for TCP-Ho by examples and simulations.

4.1 TCP Vegas

In this subsection, we derive the conspicuous round-trip time for TCP Vegas. Table 3 shows the value of W_i and z_i at the ith round if Δ is smaller than γ, where z_i is the increase amount at the ith round and

$$\begin{cases} W_i = 2^{\frac{i+1}{2}}, z_i = 0, & \text{if } i \text{ is odd or 0} \\ W_i = z_i = 2^{\frac{i}{2}}, & \text{if } i \text{ is even.} \end{cases}$$

Vegas doubles its congestion window every other round. Assuming that in the lth round, Δ is smaller than γ at the last time and W_l is the congestion window size, where l is an integer in Table 3. According to the work [16], the *BaseRTT*, newly measured *RTT*, and the congestion window size W_l from which Vegas gets out of slow start phase are given by d, $D = d + \frac{W_l}{2\mu}$, and $W_l \ge \frac{1+\sqrt{1+8\mu d}}{2}$, respectively. In an actual Vegas implementation, D is the smoothed RTT rather than the RTT of a specific packet. Thus, D for the last packet is the average of the actual RTTs of all packets in the same round, i.e., D=d+$W_l/4\mu$, rather than D=d+$W_l/2\mu$, as given above. By using the smoothed RTT, we have $W_l \ge \frac{1+\sqrt{1+16\mu d}}{2}$, and TCP Vegas changes the slow start phase to congestion avoidance in the lth round, where $l \ge 2\log_2(1+\sqrt{1+16\mu d})-1$ because l is even. Then, it takes $\lceil \mu d - 7W_l/8 \rceil$ rounds to the conspicuous round-trip time; therefore, at the

$$(l + \lceil \mu d - 7\, W_l/8 \rceil)\text{th} \tag{2}$$

round, TCP Vegas attains the available bandwidth.

Table 3 The congestion window size and the increase amount for Vegas at the ith round.

ith	0	1	2	3	4	5	6	7	8	9	10	...
W_i	2	2	2	4	4	8	8	16	16	32	32	...
z_i	0	0	2	0	4	0	8	0	16	0	32	...

Table 4 The congestion window size and the increase amount for TCP-Ho at the ith round.

ith	0	1	2	3	4	5	6	7	8	9	10	...
W_i	2	2	2	3	4	6	8	11	14	18	22	...
z_i	0	0	1	1	2	2	3	3	4	4	5	...

4.2 TCP-Ho

In this subsection, we derive the conspicuous round-trip time for TCP-Ho. Table 4 shows the value of W_i and z_i at the ith round when PMaT < CMiT where

$$\begin{cases} W_i = (i^2 - 2i + 9)/4, z_i = (i-1)/2, & \text{if } i \text{ is odd} \\ W_i = (i^2 - 2i + 8)/4, z_i = i/2, & \text{otherwise.} \end{cases}$$

Moreover, $z_{i-1} + z_i = i - 1$, for $i \ge 1$.

TCP-Ho increases its congestion window with *Stable growth* every other round. Moreover, with *Stable growth*, the extra packet[†] will be transmitted to the network when two or more ACKs of the previous round are received. For example, at the twelfth round in Table 4, TCP-Ho sends an extra packet when it receives the sixth ACK, then it transmits another extra packet after five ACKs (i.e., the eleventh ACK is received). Since in this round, TCP-Ho increases $\frac{6}{32}$ packet to the congestion window size whenever an ACK of the previous round is received, and sends an extra packet to the network when the additional value is no smaller than one (as $\frac{6}{32} \times 6 = \frac{36}{32} > 1$). In the light of the work [16], the spacing between each ACK of the previous round is $1/\mu$ seconds because μ is the smallest transmission rate (and we assume that there is no congestion along the ACK path). Assuming that the kth round is the conspicuous round-trip time, then the congestion window size is W_k in this round,

[†]When a source receives an ACK of the previous round, it transmits two packets to the network. The difference between two packets and one packet is called the extra packet. For example, in Table 4, the source transmits two packets when it receives the second ACK at the fourth round.

the last packet will see z_{k-1} ($= \frac{k-2}{2} - 1$) packets waiting ahead of it in the sender queue. However, for the queues at other nodes along the connection, the last packet will see no packet from the same connection in the queues because $1/X_1 \geq 1/X_2 \geq ... \geq 1/X_N$. Therefore, this last packet experiences the highest RTT. The minimal (or minimum) RTT and newly measured *RTT* are given by d and

$$D = d + \frac{z_{k-1}}{\mu} = d + \frac{k-4}{2\mu} \tag{3}$$

respectively. TCP-Ho changes its way to increase the congestion window size when

$$\frac{W_k}{D} \leq \frac{W_{k-2}}{d}. \tag{4}$$

By combining Eq. (3) and Eq. (4), we could get the following formula:

$$k^3 - 10k^2 + (40 - 8\mu d)k + (16\mu d - 64) \geq 0. \tag{5}$$

We could get k by Cardan's formula [21], [22] because Eq. (5) is a cubic equation, which is the closed-form solution for the roots of a cubic polynomial.

$$k \geq 4\left(\sqrt[3]{Q + 3\sqrt{P}} + \sqrt[3]{Q - 3\sqrt{P}}\right), \tag{6}$$

where $Q = 18\mu d + 8$ and $P = 21 - 18\mu d + 96\mu^2 d^2$. Thus, we solve the value of conspicuous round-trip time k and TCP-Ho reaches the available bandwidth with steady throughput in this round.

4.3 Analysis with Examples

We use three examples to quantify our analysis and show that the performance of TCP-Ho is more efficient than TCP Vegas. In the following examples, the packet size = 1 kbytes and d=0.1 sec.

Example 1: μ=187.5 packets/s (equals 1.5 Mbps.)

In TCP-Ho, we will get $k \geq 14.8$ by Eq. (5). Since k is even, $k = 16$. Thus, TCP-Ho takes 1.6 seconds to the available bandwidth. By using the values of μ and d to the Eq. (2), we can get that the conspicuous round-trip times of TCP Vegas is 1.3 seconds. In addition, from simulations, the actual conspicuous round-trip times of TCP Vegas and TCP-Ho are 1.5 seconds and 1.5 seconds respectively.

Example 2: μ=2500 packets/s (equals 20 Mbps.)

Similar to Example 1, by using the values of μ and d to the Eq. (2) and Eq. (5), we can get that the conspicuous round-trip times of TCP Vegas and TCP-Ho are 20.7 seconds and 3.4 seconds, respectively. In addition, from simulations, the actual conspicuous round-trip times of TCP Vegas and TCP-Ho are 20.9 seconds, and 3.8 seconds respectively.

Example 3: μ=6250 packets/s (equals 50 Mbps.)

Similarly, by using the values of μ and d to the Eq. (2) and Eq. (5), we can get that the conspicuous round-trip times of TCP Vegas and TCP-Ho are 58.1 seconds and 4.7 seconds, respectively. In addition, from simulations, the actual conspicuous round-trip times of TCP Vegas and TCP-Ho are 58.4 seconds and 5.1 seconds respectively.

From the above examples, we could see that the difference between analysis results and simulation results are small, and the value of TCP-Ho's conspicuous round-trip time is smaller than that of TCP Vegas. Thus, the utilization of bandwidth in TCP-Ho is more efficient than that in TCP Vegas.

5. Performance Evaluation

In this section, we compare the performance of TCP-Ho with TCP Reno, TCP Vegas, and FAST TCP[†] by using the network simulator ns-2 [26]. We show the simulation results for the basic behavior of a single source, and the fairness and friendliness among the competing connections with same or different TCP versions. The sizes of data packets and ACKs are 1 kbytes and 40 bytes respectively. To ease the comparison, we assume that the sources always have data to send. The network configuration for the simulations is shown in Fig. 4. Sources, destinations, and routers are expressed as S_i, D_i, and R_i respectively. A source and a destination with the same subscript value represent a traffic pair. The bandwidth and propagation delay are 1 Gbps and 1 ms for each full-duplex access link, and X Mbps and Y ms for the full-duplex connection link between R_1 and R_2. The X and Y are set based on the need of simulation scenarios.

5.1 Basic Behavior

In this subsection, we compare the basic behavior among TCP Reno, TCP Vegas, FAST TCP, and TCP-Ho in the aspects of convergence time [16], congestion window size, and throughput. In the following simulation scenarios, the objectives are to explore how fast for a new connection can ramp up to equilibrium and how fast a connection can converge to a steady state as the available bandwidth is changed.

5.1.1 A New Connection

In this simulation, the buffer size of a router is 100 packets. We use the convergence time with different BDPs (from

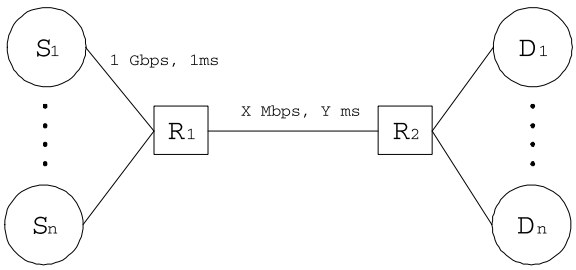

Fig. 4 Network configuration for the simulations.

[†]The value of α for FAST TCP is 100 in simulations.

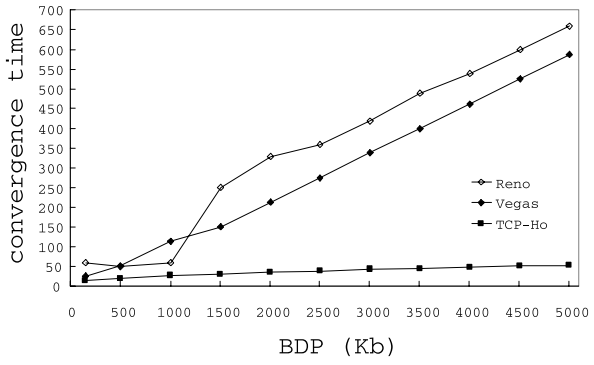

Fig. 5 The convergence time with different BDPs of communication networks.

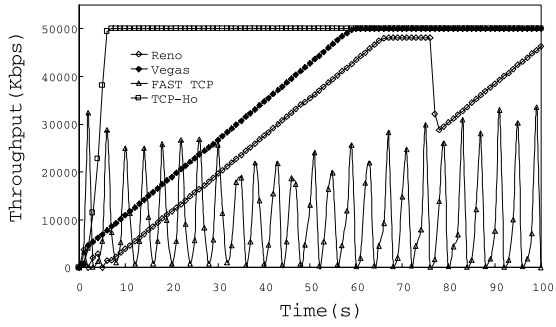

Fig. 6 Throughput comparison with 50 Mbps bottleneck bandwidth, and 48 ms link propagation delay.

150 Kb to 5000 Kb) of communication networks to compare TCP-Ho with TCP Reno and TCP Vegas. The result is shown in Fig. 5. FAST's state is unstable due to the large buffer size requirement, so the convergence time of FAST is not presented here. We can see that the convergence time of TCP-Ho grows slowly (or linearly) while BDP increases quickly. On the other hand, the convergence time of both Reno and Vegas climb up very fast. For instance, at 5000 Kb, the convergence time of Reno is more than 12.5 times of TCP-Ho and, similarly, the convergence time of Vegas is about 11 times of TCP-Ho. In addition, Fig. 6 depicts the throughput comparison with Reno, Vegas, FAST and TCP-Ho when the bottleneck bandwidth and link propagation delay are 50 Mbps and 48 ms respectively.

By observing the throughput shown in Fig. 6, we can find that the transient period for a new Reno or Vegas connection is quite long. For example, TCP Reno takes about 66 seconds to reach the available bandwidth (48 Mbps); however, it halves both the congestion window size and throughput in the 76.2th second because its AIMD algorithm causes the buffer overflow. Similarly, Vegas prematurely stops the exponentially-increasing slow start phase at the 1.3th second because doubling the sending rate in short interval causes Δ bias so that it enters the linearly-increasing congestion avoidance phase. Therefore, it takes 58.8 seconds to reach equilibrium. As mentioned before, FAST needs more buffer size in a router, so a lot of packets are lost. This results in bad performance and unstable state of FAST. In comparison with Reno, Vegas, and FAST, TCP-Ho reacts faster and better. The ramp up time of TCP-Ho to the maximum congestion window size is 5.4 seconds, and that to reach the available bandwidth (50 Mbps) is less than 7 seconds, which is one-tenth of Reno and Vegas' cost times to reach equilibrium. We could conclude that TCP-Ho is as good as Vegas or Reno in the small BDP and much better than these two in the large BDP with the demonstration in Figs. 5 and 6.

5.1.2 Various Available Bandwidth

The bottleneck capacity X is set at 50 Mbps, propagation delay Y is set at 48 ms and the buffer size of a router is set at 1000 packets[†]. A TCP connection of Reno, Vegas, FAST or TCP-Ho from S_1 to D_1 starts sending data at 0 second and a constant bit rate (CBR) traffic flow from S_2 to D_2 with 25 Mbps rate starts at the 80th second and stops at the 160th second. Figures 7, 8, 9, and 10 exhibit the basic behavior of TCP Reno, TCP Vegas, FAST TCP, and TCP-Ho respectively. By observing the congestion window evolution shown in Figs. 7, 8, 9, and 10, we can find that the transient period for a new connection of Reno or Vegas is quite long. In the beginning, Reno increases its congestion window size to 3017 packets, which is much larger than the number of packets held by the bottleneck bandwidth and buffer, so it faces the coarse-grained timeout and retransmission, and finally Reno reaches the available bandwidth after 6 seconds. On the other hand, Vegas prematurely stops the slow start phase at the 1.3th second, then enters congestion avoidance and takes 57.5 seconds to reach equilibrium. When the available bandwidth is halved at the 80th second, Vegas takes 47.7 seconds to converge to a new steady state. As the available bandwidth is doubled at the 160th second, there is a 31.6 seconds transient period for Vegas. On the other hand, Reno re-enters the slow start phase due to buffer overflow and coarse-grained timeout when the CBR traffic flow sends data and then it spends about 33 seconds to get the available bandwidth (24 Mbps). The performance of Reno is so ineffective because Reno uses packet loss to detect the available bandwidth and increases the congestion window size with linear growth in the congestion avoidance phase. In comparison with TCP Reno and TCP Vegas, when a new connection enters the empty link or the available bandwidth is either halved or doubled, TCP-Ho only converges the congestion window size in a short time (less than 7 seconds). Furthermore, even when the buffer is large enough, the throughput of TCP-Ho is a bit better than that of FAST and TCP-Ho's packets queued in the router (at most 51 packets) are fewer than FAST's (about 98 packets) in the steady state, although the time for a FAST's connection to adjust its throughput to the available bandwidth is little shorter than that of TCP-Ho.

With high BDP networks, the transient period of TCP can greatly affect overall performance. Now, we use a metric convergence time to capture the transient performance of

[†]The reasons for setting such a large buffer size are that we want to compare TCP-Ho with FAST under the steady state and FAST needs more buffer size in a router.

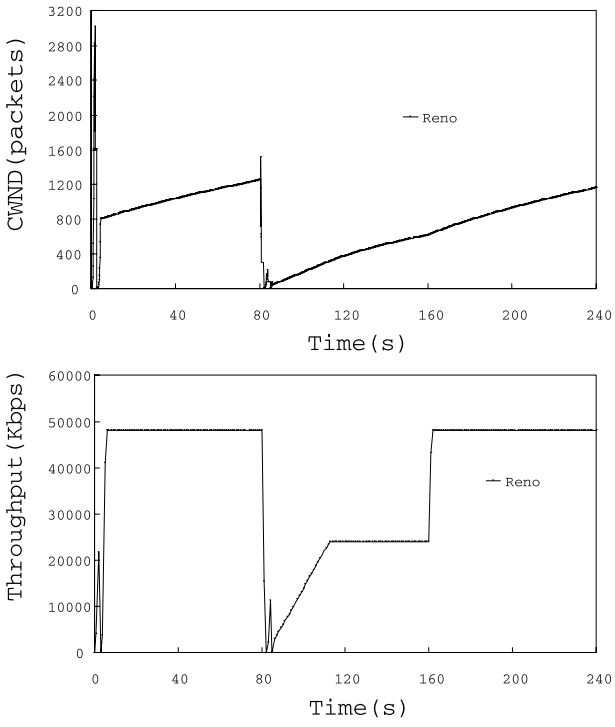

Fig. 7 Basic behavior of TCP Reno.

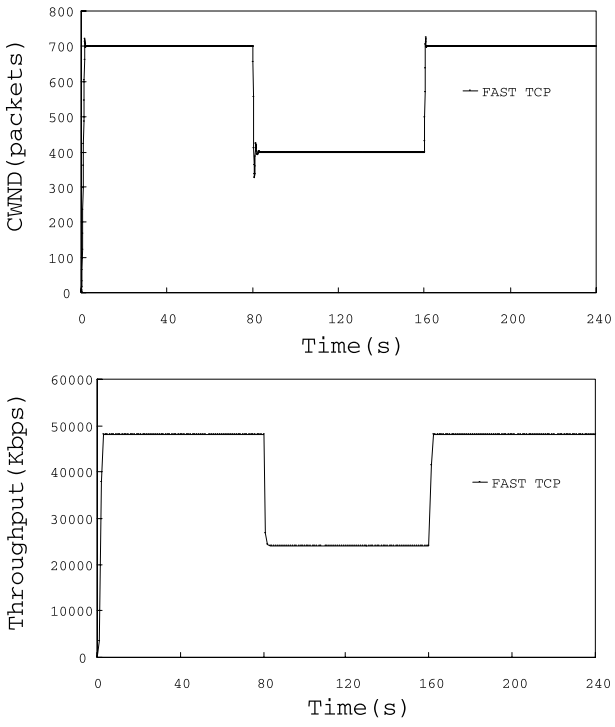

Fig. 9 Basic behavior of FAST TCP.

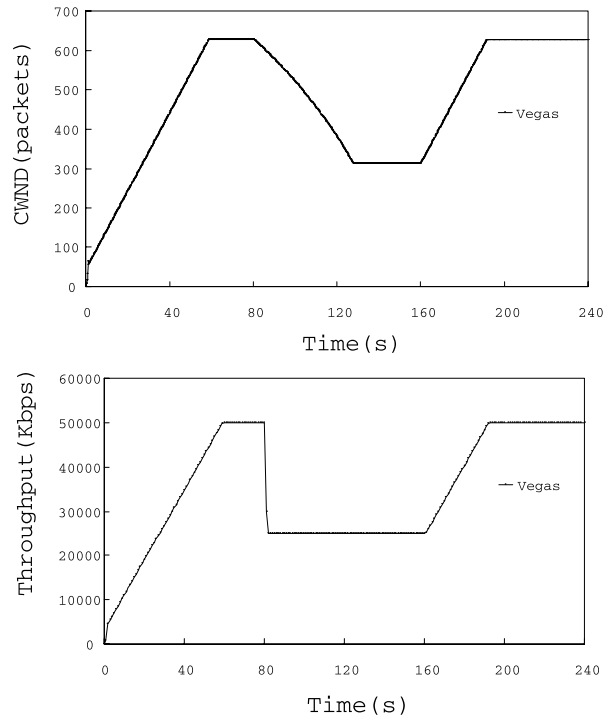

Fig. 8 Basic behavior of TCP Vegas.

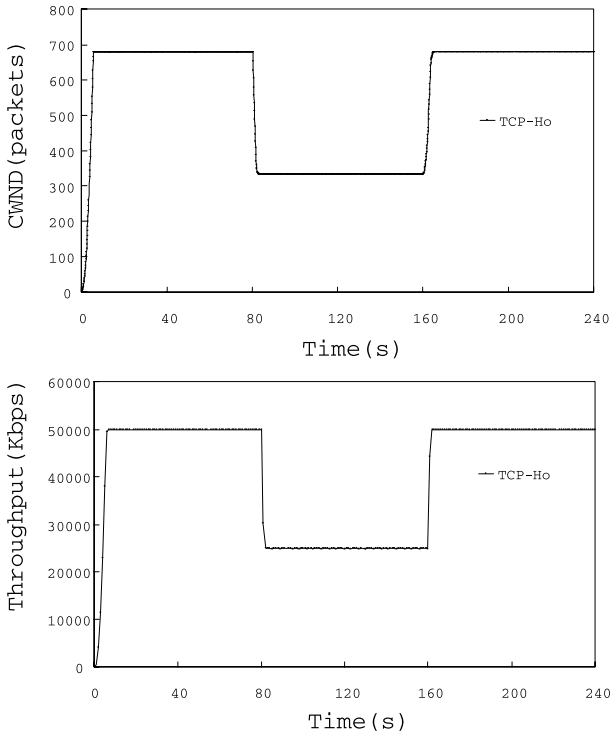

Fig. 10 Basic behavior of TCP-Ho.

TCP. The traffic sources are same as those in the previous simulation. The bottleneck capacity is varied for different BDP. At some time instant, the CBR traffic source starts or stops sending packets to halve or double the available bandwidth, respectively. Figures 11 and 12 display the convergence time as the available bandwidth is halved and doubled respectively. Obviously, TCP-Ho greatly improves the transient performance of connection in both scenarios as compared to TCP Reno and TCP Vegas. On the other hand, the

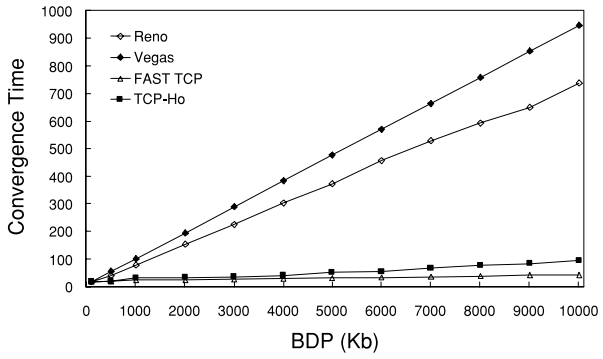

Fig. 11 Convergence time of connections when available bandwidth is halved.

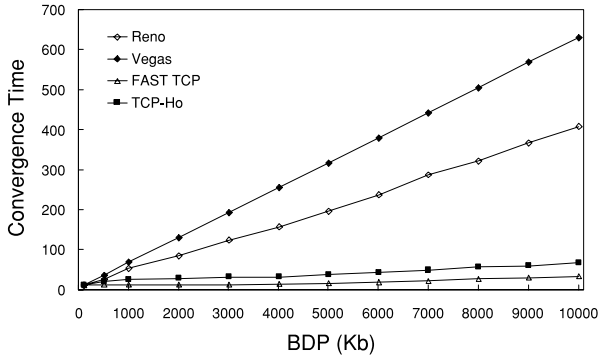

Fig. 12 Convergence time of connections when available bandwidth is doubled.

difference in convergence time between TCP-Ho and FAST is not clear in both scenarios if we give FAST enough buffer size. Moreover, from Figs. 11 and 12, we can see that the convergence time of TCP-Ho is under 100 no matter how large the BDP is.

5.2 Fairness and Friendliness

Other important issues of TCP are the fairness and friendliness. Multiple connections of same TCP scheme must interoperate nicely and converge to their fair shares. Similarly, a friendly TCP scheme should be able to coexist with other TCP variants without causing them starvation. We use the fairness index function, proposed in [23], to justify the fairness and friendliness of TCP schemes. The fairness index function is expressed as $F(x) = \frac{(\sum x_i)^2}{n(\sum x_i^2)}$, where x_i is the throughput of the ith connection, and n is the number of connections. $F(x)$ ranges from $1/n$ to 1.0. A perfectly fair bandwidth allocation would result in a fairness index of 1.0. On the contrary, if all bandwidth are consumed by one connection, fairness index would yield $1/n$.

5.2.1 Fairness

First, we are interested in the performance of sources with same TCP method and RTT. The network topology for simulations is shown in Fig. 4, where the bandwidth and propagation delay are 100 Mbps and 23 ms for the full-duplex trunk link respectively, five sources (n = 5) using same TCP scheme (i.e., Reno, Vegas, FAST, or TCP-Ho) all start at 0 second, and buffer size is 300 packets. We use the fairness index and utilization of bottleneck bandwidth to represent the result, as shown in Table 5. From Table 5, we find that the fairness index of TCP-Ho (99.9995%) is better than that of Reno (99.9663%), Vegas (97.3188%), and FAST (99.9979%), and the utilization of bandwidth for TCP-Ho (100%) is not worse than that for Reno (only uses 80–96%), Vegas (100%), and FAST (18.19%). Furthermore, Reno takes at least 6 seconds to approach the available bandwidth, Vegas spends 6.5 seconds to converge, FAST is never close to the available bandwidth (i.e., never converges), and TCP-Ho uses 2 seconds, which is one-third of 6 or 6.5, to achieve the balance of connections. Then, we simulate the sources with different RTTs, and the network topology is shown in Fig. 13. The bandwidth and propagation delay are 1 Gbps and 1 ms for the full-duplex access link, and 100 Mbps and 23 ms for the full-duplex trunk link respectively. The sources are TCP Reno, TCP Vegas, FAST TCP, or TCP-Ho. Table 6 depicts the fairness index and utilization of bottleneck bandwidth for four TCP versions with n = 5. We observe that the fairness index of TCP-Ho is about 1.1 times of Vegas, more than 1.2 times of Reno, and about 0.83 times of FAST, and the utilization of bandwidth for TCP-Ho is better than TCP Reno and FAST TCP from Table 6.

From these simulation results, we can observe that the TCP-Ho is more suitable than Reno and Vegas for multiple sources with same mechanism.

Table 5 Fairness index for Reno, Vegas, FAST and TCP-Ho.

	Reno	Vegas	FAST	TCP-Ho
Fairness Index	99.97%	97.32%	99.99%	99.99%
Utilization	80–96%	100%	18.2%	100%

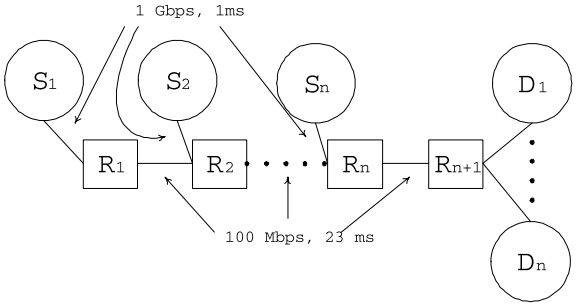

Fig. 13 The network topology for sources with different RTTs.

Table 6 The fairness index for four TCP versions.

	Reno	Vegas	FAST	TCP-Ho
Fairness Index	67%	75%	99%	82%
Utilization	96%	100%	96%	100%

Table 7 Friendliness of different TCP versions.

Fairness Index / Mean Throughput (kbps)			
	Reno–Vegas	TCP-Ho–Vegas	TCP-Ho–Reno
15 - 05	80%/(608–89)	95%/(533–403)	85%/(474–570)
10 - 10	59%/(840–120)	86%/(564–438)	89%/(453–557)
05 - 15	39%/(1526–138)	89%/(592–471)	96%/(435–524)
	FAST–TCP-Ho	FAST–Vegas	FAST–Reno
15 - 05	80%/(376–158)	72%/(504–118)	88%/(395–154)
10 - 10	54%/(665–222)	52%/(731–148)	79%/(554–317)
05 - 15	39%/(1139–251)	27%/(1380–166)	76%/(591–250)

5.2.2 Friendliness

To verify the friendliness of our proposed mechanism, we construct the network topology shown in Fig. 4, where Reno coexists with Vegas, FAST coexists with Reno, or Vegas, and TCP-Ho coexists with Reno, Vegas, or FAST. There are 20 pairs of connections, of which some connections use one TCP algorithm and the others are another TCP mechanism connections. The bandwidth of connection link is 10 Mbps, propagation delay is 1 ms, and the buffer size of a router is 120 packets. We vary the proportion of these TCP schemes in the network by adjusting the number of sources. Without the presence of congestion, all 20 connections are expected to share the bottleneck bandwidth equally. The friendliness results and mean throughput of each TCP scheme are listed in Table 7, where the mean throughput is the average throughput of sources with same mechanism under the steady state. We observe following phenomenons. First, while Vegas users compete with other TCP Reno users, they do not receive a fair share of bandwidth due to the different congestion avoidance mechanisms used by Vegas and Reno. Next, FAST sources keep inserting much more packets into the buffer and stealing more bandwidth no matter which mechanism coexists with FAST. Finally, when TCP-Ho competes with TCP Reno, it still can insert much packets into the buffer and get its fair share of bandwidth; nevertheless, TCP-Ho do not steal more bandwidth when it coexists with TCP Vegas. Therefore, TCP-Ho is friendlier towards TCP Reno and TCP Vegas schemes than FAST is. In other words, TCP-Ho is a friendly transport protocol.

6. Conclusion

In this article, we propose and evaluate a new variant of TCP, TCP-Ho, to reduce the burstiness, to adjust the rate to the available bandwidth in shorter time, and to improve the throughput for a TCP source. We achieve a significantly higher performance comparing with TCP Reno and Vegas. From simulation results, although TCP-Ho is more suitable for large bandwidth or long-delay networks, it still increases transmit performance in small bandwidth or short-delay networks. The design of TCP-Ho is simple and implementation feasible on existing operating systems. Further work involves studying the performance and spatial characteristics analysis of this algorithm under a wider range of parameters, network topologies and real traffic traces, obtaining more accurate theoretical models and insights, and considering hardware implementation issues.

References

[1] J. Postel, "Transmission control protocol," IETF RFC 793, Sept. 1981.
[2] V. Jacobson, "Congestion avoidance and control," Proc. ACM SIGCOMM, pp.314–329, Aug. 1988.
[3] V. Jacobson, "Modified TCP congestion avoidance algorithm," End-2-End-Interest Mailing List, Tech. Rep., LBL, April 1990.
[4] J.C. Hoe, Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes, Master's Thesis, MIT, June 1995.
[5] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," IETF RFC 2018, Oct. 1996.
[6] M. Mathis and J. Mahdavi, "Forward acknowlegement: Refining TCP congestion control," Proc. ACM SIGCOMM, pp.181–191, Aug. 1996.
[7] D. Lin and H.T. Kung, "TCP fast recovery strategies: Analysis and improvements," Proc. IEEE INFOCOM, pp.263–271, March 1998.
[8] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," IETF RFC 3042, Jan. 2001.
[9] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm," ACM Computer Communication Review, vol.22, no.2, pp.9–16, 1992.
[10] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," ACM Computer Communication Review, vol.19, no.5, pp.56–71, 1989.
[11] Z. Wang and J. Crowcroft, "A new congestion control scheme: Slow start and search (Tri-S)," ACM Computer Communication Review, vol.21, no.1, pp.32–43, 1991.
[12] S. Keshav, "A control-theoretic approach to flow control," ACM Computer Communication Review, vol.25, no.1, pp.188–201, 1995.
[13] L.S. Brakmo and L.L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," IEEE J. Sel. Areas Commun., vol.13, pp.1465–1480, 1995.
[14] C. Parsa and J.J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internet with heterogeneous transmission media," Proc. IEEE ICNP, pp.213–221, Nov. 1999.
[15] H. Wang, H. Xin, D.S. Reeves, and K.G. Shin, "A simple refinement of slow-start of TCP congestion control," Proc. IEEE ISCC 2000, pp.98–105, July 2000.
[16] S. Vanichpun and W. Feng, "On the transient behavior of TCP Vegas," Proc. IEEE ICCCN, pp.504–508, Oct. 2002.
[17] Y. Nishida, "Smooth slow-start: Refining TCP slow-start for large-bandwidth with long-delay networks," Proc. IEEE LCN, pp.52–60, Oct. 1998.
[18] S. Floyd, "HighSpeed TCP for large congestion windows," Internet draft draft-oyd-tcp-highspeed-02.txt, Feb. 2003.
[19] T. Kelly, "Sclable TCP: Improving performance in highspeed wide area networks," ACM Computer Communication Review, vol.33, no.2, pp.83–91, 2003.
[20] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," Proc. ACM SIGCOMM, pp.89–102, Aug. 2002.
[21] R. Calinger, Classics of Mathematics, pp.235–237, Moore Publishing, Oak Park, Illinois, 1982.
[22] J. Stillwell, Mathematics and its History, p.55, Springer-Verlag, New York, 1989.
[23] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC, Research Report TR-301, Sept. 1984.
[24] S. Floyd, "Limited slow-start for TCP with large congestion win-

dows," IETF RFC 3742, March 2004.
[25] C. Jin, D.X. Wei, and S.H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," Proc. IEEE INFOCOM, vol.4, pp.2490–2501, March 2004.
[26] http://www.isi.edu/nsnam/ns/

Cheng-Yuan Ho is currently a Ph.D. candidate in computer science at National Chiao Tung University, Hsinchu City, Taiwan. He also works with the Wireless and Networking Group of Microsoft Research Asia, Beijing, China since Dec., 2005. His research interests include the design, analysis, and modelling of the congestion control algorithms, high speed networking, QoS, and mobile and wireless networks.

Yi-Cheng Chan received his Ph.D. degree in computer science and engineering from National Chiao Tung University, Taiwan in 2004. He is now an assistant professor in the department of computer science and information engineering of National Changhua University of Education, Changhua City, Taiwan. His research interests include network protocols, wireless networks, and AQM.

Yaw-Chung Chen received his Ph.D. degree in computer science from Northwestern University, Evanston, Illionis, USA in 1987. During 1987–1990, he worked at AT&T Bell Laboratories. Now he is a professor in the department of computer science of National Chiao Tung University. His research interests include multimedia communications, high speed networking, and wireless networks.