PAPER

Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks*

Yi-Cheng CHAN[†a)], ***Member***, **Chia-Liang LIN**[†], ***Nonmember***, ***and*** **Cheng-Yuan HO**[††], ***Student Member***

SUMMARY An important issue in designing a TCP congestion control algorithm is that it should allow the protocol to quickly adjust the end-to-end communication rate to the bandwidth on the bottleneck link. However, the TCP congestion control may function poorly in high bandwidth-delay product networks because of its slow response with large congestion windows. In this paper, we propose an enhanced version of TCP Vegas called Quick Vegas, in which we present an efficient congestion window control algorithm for a TCP source. Our algorithm improves the slow-start and congestion avoidance techniques of original Vegas. Simulation results show that Quick Vegas significantly improves the performance of connections as well as remaining fair when the bandwidth-delay product increases.

key words: *congestion control, high bandwidth-delay product networks, TCP Vegas, transport protocol*

1. Introduction

Most of the current Internet applications use the Transmission Control Protocol (TCP) as its transport protocol. Consequently, the behavior of TCP is tightly coupled with the overall Internet performance. TCP performs an acceptable efficiency over today's Internet. However, theory and experiments show when the per-flow product of bandwidth and latency increases, TCP becomes inefficient [1]. This will be problematic for TCP as the bandwidth-delay product (BDP) of Internet continues to grow.

TCP Reno is the most widely used TCP version in the current Internet. It takes packet loss as an indiction of congestion. In order to probe available bandwidth along the end-to-end path, TCP Reno 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 additive increase - multiplicative decrease (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-trip time (*RTT*). The BDP of a single connection over very high bandwidth links may be thousands of packets, thus TCP Reno might waste thousands of *RTTs* to ramp up to full utilization.

Unlike TCP Reno which uses binary congestion signal, packet loss, to adjust its window size, TCP Vegas [2] adopts a more fine-grained signal , queuing delay, to avoid congestion. Studies have demonstrated that Vegas outperforms Reno in the aspects of overall network utilization [2], [5], stability [6], [7], fairness [6], [7], throughput and packet loss [2], [3], [5], and burstiness [3], [4]. However, in high BDP networks, Vegas tends to prematurely stop the exponentially-increasing slow-start phase and enter the slower congestion avoidance phase until it reaches its equilibrium congestion window size [8]. As a result, a new Vegas connection may experience a very long transient period and thus 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 is not varied linearly [10]. Since Vegas adjusts its congestion window linearly in the congestion avoidance phase, this prevents Vegas from quickly adapt to the changing environments.

In this paper, we propose an enhanced version of TCP Vegas called Quick Vegas for high BDP networks. Quick Vegas is a sender-side modification that improves the slow-start and congestion avoidance techniques of original Vegas. Simulation results show that Quick Vegas significantly improves the performance of connections as well as remaining fair when the bandwidth-delay product increases.

The rest of this paper is organized as follows. Related work is reviewed in Sect. 2. Section 3 describes TCP Vegas and explains the proposed Quick Vegas. The mathematical analysis is given in Sect. 4 and simulation results are presented in Sect. 5. Lastly, we conclude this work in Sect. 6.

2. Related Work

Several studies have been made to improve the connection performance over high-speed and long-delay links. These approaches can be divided into two categories. One is simpler and needs only easily-deployable changes to the current protocols, for example, LTCP [11], TCP-Westwood [12], CUBIC [13], TCP-Africa [14], AdaVegas [15], and FAST [16]. The other needs more complex changes with a new transport protocol, or more explicit feedback from the routers, examples are XCP [1] and QuickStart [17]. XCP and QuickStart requires all routers along the path to participate, deployment feasibility is a concern.

Manuscript received April 12, 2007.
Manuscript revised July 3, 2007.

[†]The authors are with the Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua, Taiwan.
[††]The author is with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan.

*This work was sponsored by the National Science Council, Taiwan, R.O.C., under Grant NSC 96-2221-E-018-008.

a) E-mail: ycchan@cc.ncue.edu.tw

DOI: 10.1093/ietcom/e91–b.4.987

A protocol that falls in the first category commonly involves a subtle change in its congestion avoidance response function to allow connections capturing available bandwidth more readily and realistically at very high congestion windows. However, the variants of TCP Reno [11]–[14] use packet loss as an indication for congestion. This causes periodic packet losses and 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 [3], [4].

AdaVegas is an adaptive congestion control mechanism based on TCP Vegas. Using the number of successful transmissions as feedback, it dynamically changes the additive increase parameters to decide whether to switch to a more aggressive strategy. The modification makes connections react faster to a changing network environment.

FAST is also a variation of TCP Vegas. It incorporates multiplicative increase if the bottleneck buffer occupied by the connection is far less than a pre-defined threshold (i.e., 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 has better properties than pure loss-based approaches, such as higher utilization, less self-induced packet losses, faster convergence speed, better *RTT* fairness and stabilization.

The two TCP Vegas variants improve the connection performance when it works in a network with a large amount of available bandwidth. However, AdaVegas uses some constant increments to increase its window size. It may be still too sluggish when the connection passes through a very high BDP path. FAST adopts a more aggressive way to update its window size, nevertheless, it needs a large buffer on the bottleneck to prevent packet losses.

3. TCP Vegas and Proposed Mechanism

TCP Vegas features three improvements as compared with TCP Reno: (1) a new retransmission mechanism, (2) an improved congestion avoidance mechanism, and (3) a modified slow-start mechanism. In this section, we first review the design principles of TCP Vegas and then describe Quick Vegas in detail.

3.1 TCP Vegas

Vegas adopts a more sophisticated bandwidth estimation scheme that tries to avoid rather than to react to congestion. It uses the measured *RTT* to accurately calculate the amount of data packets that a source can send. Its window adjustment algorithm consists of three phases: slow-start, congestion avoidance, and fast retransmit and fast recovery. The congestion window is updated based on the currently executing phase.

During the congestion avoidance phase, TCP Vegas does not continually increase the congestion window. Instead, it tries to detect incipient congestion by comparing the actual throughput to the expected throughput. Vegas estimates a proper amount of extra data to be kept in the network pipe and controls the congestion window size accordingly. It records the *RTT* and sets *BaseRTT* to the minimum of ever measured round-trip times. The amount of extra data (Δ) is estimated as follows:

$$\Delta = (Expected - Actual) \times BaseRTT, \tag{1}$$

where *Expected* throughput is the current congestion window size (*CWND*) divided by *BaseRTT*, and *Actual* throughput represents the *CWND* divided by the newly measured smoothed-*RTT*. The *CWND* is kept constant when the Δ is between two thresholds α and β. If Δ is greater than β, it is taken as a sign for incipient congestion, thus the *CWND* will be reduced. On the other hand, if the Δ is smaller than α, the connection may be under utilizing the available bandwidth. Hence, the *CWND* will be increased. The updating of *CWND* is per-*RTT* basis. The rule for congestion window adjustment can be expressed as follows:

$$CWND = \begin{cases} CWND + 1, & if\ \Delta < \alpha \\ CWND - 1, & if\ \Delta > \beta \\ CWND, & if\ \alpha \le \Delta \le \beta \end{cases}. \tag{2}$$

During the slow-start phase, Vegas intends a connection to quickly ramp up to the available bandwidth. However, in order to detect and avoid congestion during slow-start, Vegas doubles the size of its congestion window only every other *RTT*. In between, the congestion window stays fixed so that a valid comparison of the *Expected* and *Actual* throughput can be made. A similar congestion detection mechanism is applied during the slow-start to decide when to switch the phase. If the estimated amount of extra data is greater than γ, Vegas leaves the slow-start phase, reduces its congestion window size by 1/8 and enters the congestion avoidance phase.

As in Reno, a triple-duplicate acknowledgement (ACK) always results in packet retransmission. However, in order to retransmit the lost packets quickly, Vegas extends Reno's fast retransmission strategy. Vegas measures the *RTT* for every packet sent based on fine-grained clock values. Using the fine-grained *RTT* measurements, a timeout period for each packet is computed. When a duplicate ACK is received, Vegas will check whether the timeout period of the oldest unacknowledgement packet is expired. If so, the packet is retransmitted. This modification leads to packet retransmission after just one or two duplicate ACKs. When a non-duplicate ACK that is the first or second ACK after a fast retransmission is received, Vegas will again check for the expiration of the timer and may retransmit another packet.

After a packet retransmission was triggered by a duplicate ACK and the ACK of the lost packet is received, the congestion window size will be reduced to alleviate the network congestion. There are two cases for Vegas to set the *CWND*. If the lost packet has been transmitted just once, the *CWND* will be three fourth of the previous congestion window size. Otherwise, it is taken as a sign for more serious

congestion, and one half of the previous congestion window size will be set to *CWND*.

If a loss episode is severe enough that no ACKs are received to trigger fast retransmit algorithm, eventually, the losses will be identified by Reno-style coarse-grained timeout. When this occurs, the slow-start threshold (*SSTHRESH*) will be set to one half of *CWND*, then the *CWND* will be reset to two, and finally the connection will restart from slow-start.

3.2 The Proposed Mechanism

TCP sends bursts of packets during its slow-start phase due to the fast window increase and the ACK-clock based transmission. This phenomenon causes TCP Vegas changing from slow-start phase to congestion avoidance phase too early in the large BDP links. Besides, network resources and competing users may vary over time unpredictably. In order to react faster and better to high BDP networks, the window adjustment algorithm of congestion avoidance phase should be more aggressive than it has been. The proposed Quick Vegas tries to address these issues.

3.2.1 Slow-Start

According to Eq. (1), Vegas calculates the extra data (Δ) and doubles its congestion window every other *RTT*. When the amount of Δ is greater than γ (usually set to 1), Vegas leaves the slow-start phase. The fundamental problem is that when the congestion window should be doubled, a Vegas sender sends two packets back-to-back whenever it receives an ACK. This leads the doubled amount of packets being sent in a short interval. The burst may create a temporary long queue that causes Δ bias [8]. As a result, Vegas leaves the slow-start phase too early and performance suffers. A simple ns-2 [18] simulation has been made to show the phenomenon. Figure 1 depicts the congestion window evolution of a Vegas connection in a communication path with 5 Mb BDP (50 Mb/s $\times$ 100 ms). The slow-start phase stops at 1.2 second with congestion window size 64 packets. Obviously, Vegas prematurely stop the exponentially-increasing slow-start phase and enter the slower congestion avoidance phase. It finally reaches the equilibrium window size (630 packets) at 59 second.

How to smooth out the transmission of packets in a fluid-like manner is the key of an unbiased Δ calculation. Pacing is a common way to solve the burstiness problem. A straightforward implementation of pacing would have the TCP sender schedule successive packet transmissions at a constant time interval, obtained by dividing the congestion window by the current *RTT*. In practice, this would require a timer with a very high resolution when TCP transmit in a large bandwidth link. Using timer interrupt in such high frequency may greatly increase the overhead of the operating system.

Two simple changes have been made in the slow-start mechanism of Quick Vegas. First, a sender sends out a extra packet whenever it receives two ACKs. Second, the congestion window is adjusted every *RTT* instead of every other *RTT*. The first modification makes the transmission less bursty when the congestion window should be increased and therefore alleviates the bias of Δ calculation. The second modification allows a sender quickly ramping up to the available bandwidth. Table 1 shows the value of congestion window for Vegas and Quick Vegas at each *RTT* round if Δ is not greater than γ.

The simulation results of congestion window evolution of Vegas and Quick Vegas during slows-tart phase is shown in Fig. 2. For the identical γ ($\gamma = 1$) setting, Quick Vegas increases half of *CWND* every *RTT* while Vegas doubles its *CWND* every other *RTT*. It is obvious that Quick Vegas has a higher critical window size (the congestion window size at upon leaving slow-start phase) as compared with that of

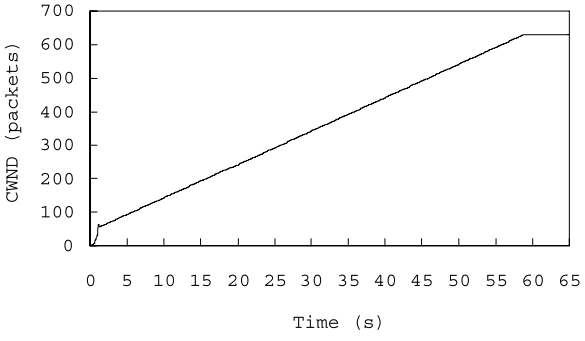

Fig. 1 Congestion window evolution of a Vegas connection.

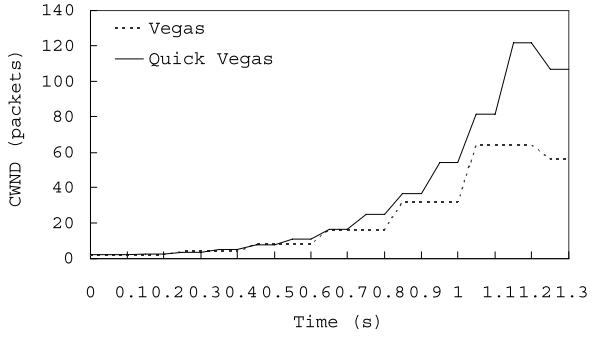

Fig. 2 Congestion window evolutions of Vegas and Quick Vegas during slows-tart phase.

Table 1 The congestion window size for Vegas and Quick Vegas at each *RTT* round.

RTT	0	1	2	3	4	5	6	7	8	...
Vegas	2	2	4	4	8	8	16	16	32	...
Quick Vegas	2	3	4.5	6.75	10.13	15.19	22.78	34.17	51.26	...

Vegas. The effectiveness of the modified slow-start scheme will be further examined in Sect. 4.

3.2.2 Congestion Avoidance

TCP Vegas updates its congestion window linearly in the congestion avoidance phase, it is too sluggish for a high BDP network. Depending on the information given by the estimated extra data, it is worth to try a more aggressive strategy.

For the increment of congestion window, Quick Vegas has the history to guide the window size changes. Since there is no direct knowledge of current available bandwidth, Quick Vegas records the number of consecutive increments due to $\Delta < \alpha$ and refers to this value as *succ*. Whenever the congestion window should be increased due to $\Delta < \alpha$, it is updated as follows:

$$CWND = CWND + (\beta - \Delta) \times succ. \quad (3)$$

Thus the congestion window size will be increased by $(\beta - \Delta)$ at the first estimation of $\Delta < \alpha$, and by $(\beta - \Delta) \times 2$ at the next consecutive estimation of $\Delta < \alpha$, and so on. The *succ* will be reset whenever $\Delta \geq \alpha$. The idea is that if the increment was successful, it might be the case that there is enough bandwidth and it is worthwhile to move to a more aggressive increasing strategy. However, to ensure that the congestion window will not be increased too fast, Quick Vegas can at most double the size of congestion window for every estimation of $\Delta < \alpha$.

For the decrement of congestion window, Quick Vegas uses the difference of Δ and $(\alpha + \beta)/2$ as the guide for every estimation of $\Delta > \beta$. The decrement rule can be expressed as follows:

$$CWND = CWND - \left(\frac{\Delta - \left(\frac{\alpha+\beta}{2}\right)}{2} \right). \quad (4)$$

Since the estimated amount of extra data gives us a good suggestion of how many extra data are beyond the ideal value that should be kept in the network pipe. However, when a TCP source decreases (or increases) its congestion window at the ith round, the influence to the network can be detected at the $(i + 2)$th round. To prevent over decrease, Quick Vegas subtracts the half of excess amount from the congestion window each time.

In order to achieve a higher fairness between the competing connections, Quick Vegas intends every connection to keep an equal amount, that is $(\alpha + \beta)/2$, of extra data in the network pipe. If the estimated amount of extra data is between α and β, Quick Vegas will adjust its congestion window linearly toward the ideal amount. The window adjustment algorithm of Quick Vegas can be presented as the following pseudo codes:

if $(\Delta > \beta)$
  $CWND = CWND - \left(\frac{(\Delta - \frac{\alpha+\beta}{2})}{2} \right)$
  $incr = 0$; $succ = 0$
else if $(\Delta < \alpha)$
  $succ = succ + 1$
  if $((\beta - \Delta) \times succ > CWND)$
    $incr = 1$
  else
    $incr = \frac{\beta - \Delta}{CWND} \times succ$
else if $\left(\Delta > \frac{\alpha+\beta}{2}\right)$
  $CWND = CWND - 1$; $incr = 0$; $succ = 0$
else if $\left(\Delta < \frac{\alpha+\beta}{2}\right)$
  incr $= \frac{1}{CWND}$; $succ = 0$
else /* $\Delta == \frac{\alpha+\beta}{2}$ */
  $incr = 0$; $succ = 0$

To reduce the bursty effect of increment, the *incr* is served as the increment amount of congestion window after each ACK is received by a Quick Vegas source.

4. Numerical Analysis

In this section, we present numerical analysis of TCP Vegas and Quick Vegas. Throughout our analysis, we assume a fluid model and the source always has a packet to transmit. There is no congestion in ACK path and the buffer size at the router is large enough so that packet loss is negligible. We try to model the TCP congestion window with respect to a "round" [9], or equivalently, "window transmission." A round starts with the transmission of W packets (back-to-back) where W is the size of congestion window in that round. A round ends when the source receives the ACK of the first packet in that round and then the source starts sending a new packet of the next round.

4.1 Slow-Start

In this subsection, we want to compare the sizes of congestion window that Vegas and Quick Vegas stop at upon leaving slow-start (critical window size). In high BDP networks, Vegas tends to prematurely stop the exponentially-increasing slow-start phase and enter the slower congestion avoidance phase. A larger critical window size means a shorter time is needed to reach its equilibrium window size.

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 the network topology for analysis. We denote the transmission rate of N links (in packets/s) as X_i, $i = 1, 2, ..., N$, and the total round-trip propagation delay of the route path (in seconds) as τ. Similar to the work in [8], we assume that there is one bottleneck in the route path and $X_1 \leq X_2 \leq ... \leq X_N$. Since X_1 is the smallest transmission rate (i.e., link 1 behaves as the

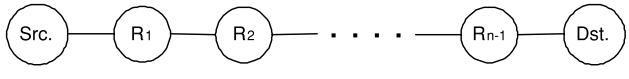

Fig. 3 Network topology for analysis.

bottleneck link), we let μ be equal to X_1. The un-congested BDP of this network is then given by μd where

$$d = \tau + \frac{1+a}{X_1} + ... + \frac{1+a}{X_N}, \tag{5}$$

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

The critical window size of Vegas has been derived in [8], it is given by

$$W^v = \frac{\gamma + \sqrt{\gamma^2 + 16\gamma\mu d}}{2}. \tag{6}$$

In the similar manner, we compute the critical window size of Quick Vegas as follows. Recall that in slow-start, Quick Vegas increases half of congestion window size every *RTT*. More precisely, Quick Vegas adds one extra packet in the sender queue each time two ACKs of previous round are received. Since μ is the smallest transmission rate and no congestion in reverse path, the spacing between each ACK of the previous round is $1/\mu$ seconds. When the last two ACKs of the previous round are received, the sender adds the last one extra packet of the current round in the sender queue and the last packet will see $W/3$ packets waiting ahead of it in the sender queue. Thus, this last packet experiences the highest *RTT* of the round and its *BaseRTT* and *RTT* are given by d in (5) and D respectively, where

$$D = d + \frac{W}{3\mu}. \tag{7}$$

By combining (1) and (7), Quick Vegas will stop its slow-start phase if

$$W \times \frac{\frac{W}{3\mu}}{\frac{W}{3\mu} + d} > \gamma. \tag{8}$$

By solving (8) for W, the window size that Quick Vegas stops its slow-start phase is given by

$$W^q = \frac{\gamma + \sqrt{\gamma^2 + 12\gamma\mu d}}{2}. \tag{9}$$

The same as actual Vegas implementation, D is the average *RTT* rather than the actual *RTT* of a packet. Thus, D should be the average of the actual *RTT*s of all packets in the same round, i.e., $D = d + \frac{W}{6\mu}$, rather than $D = d + \frac{W}{3\mu}$, as given in (7). By using the average *RTT*, we have

$$W^q = \frac{\gamma + \sqrt{\gamma^2 + 24\gamma\mu d}}{2}. \tag{10}$$

Obviously, the critical window size of Quick Vegas is larger than that of Vegas in the same network configuration. For a network path with 100 ms round-trip propagation delay with the default γ setting (i.e., $\gamma = 1$), we plot the critical window sizes of both Vegas and Quick Vegas in Fig. 4. The theoretical values of Vegas and Quick Vegas are computed according to (6) and (10) respectively. The experiment values are derived from ns-2 simulations. Since the burstiness problem still exists in slow-start phase, the experiment values do not exact match the fluid-based theoretical values. However from Fig. 4 we can find that, no matter from the theoretical or experimental point of view, Quick Vegas always has larger critical window sizes than that of Vegas in different BDPs. It means that Quick Vegas needs a shorter time to reach equilibrium as compared with that of Vegas.

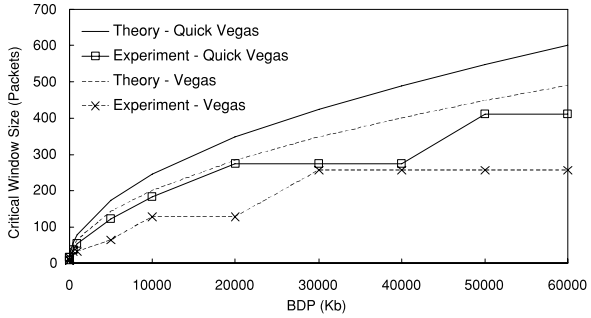

Fig. 4 Critical window sizes for Vegas and Quick Vegas in different BDPs.

4.2 Congestion Avoidance

In this subsection, we want to analyze the number of rounds (or *RTT*s) needed for Vegas and Quick Vegas to reach a new equilibrium state when more bandwidth becomes available. Consider a simplified case that the current equilibrium window size is W and the new equilibrium window size is nW.

According to (2), Vegas linearly updates its congestion window size every *RTT*. Obviously, it needs $(n-1)W$ *RTTs* to fully utilize the new available bandwidth. On the other hand, when the more available bandwidth is detected by Quick Vegas, the congestion window will be increased according to (3). If Quick Vegas needs i rounds to reach its new equilibrium state and then we have the following equation:

$$nW = W + \sum_{i=1}^{k} [(\beta - \Delta) \times i] = W + (\beta - \Delta) \times \frac{i(i+1)}{2}. \tag{11}$$

Assume the value of each Δ calculation approximates to zero during the transient period. Equation (11) can be solved for i as

$$i = \frac{-1 + \sqrt{\frac{\beta + 8W(n-1)}{\beta}}}{2}. \tag{12}$$

Figure 5 shows the number of *RTT*s needed for Vegas and Quick Vegas to reach its new equilibrium state when the available bandwidth becomes from 2 to 10 times of original used bandwidth (10 Mbps). The experiment values are obtained from ns-2 simulations and the theoretical values are computed according to the equations that are derived in this subsection.

Since the burstiness phenomenon is smoothed by the

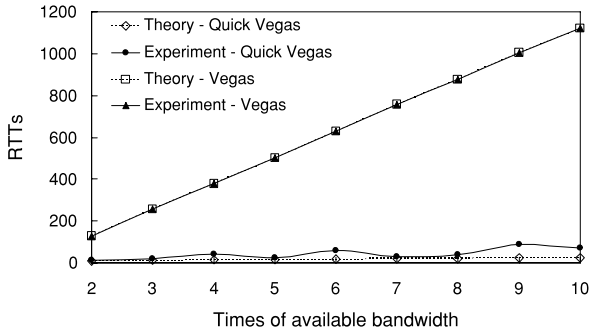

Fig. 5 The number of *RTT*s needed for Vegas and Quick Vegas reaches new equilibrium states.

network bottleneck when the connection enters its congestion avoidance phase and Vegas update its window size linearly in that phase, so the burstiness problem may be greatly eliminated. As a result, the theoretical values and the experiment values of Vegas in Fig. 5 are quite matched. On the other hand, Quick Vegas adopts a more aggressive way than that of original Vegas to adjust its window size. So it may create burstiness phenomenon to some extents. Therefore, the theoretical values and the experiment values of Quick Vegas are still with some deviations. Again, no matter from theoretical or experimental point of view, we can find that the *RTT*s needed for Quick Vegas to reach a new equilibrium state are much less than that of Vegas. The effectiveness of the enhanced congestion avoidance scheme in Quick Vegas will be further examined in the next section.

5. Performance Evaluation

We use the network simulator ns-2 [18] to execute the performance evaluation. TCP Vegas and Quick Vegas have the identical parameter values (i.e., $\gamma = 1$, $\alpha = 2$, and $\beta = 4$) those are the same as the study in [2]. The default parameter setting of FAST is $\alpha = 100$ that is also adopted in [16]. Unless stated otherwise, the buffer size in routers is large enough so that packet loss is negligible. 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. 6. 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 Gb/s and 1 ms for each full-duplex access link, and C_b and 48 ms for the full-duplex connection link between R_1 and R_2. The C_b is set based on the need of simulation scenarios.

5.1 Basic Behavior

In this subsection, we compare the basic behavior between Vegas, Quick Vegas, and FAST in the aspects of congestion window size, queue length, and throughput. The bottleneck capacity C_b is set at 50 Mb/s. A TCP connection 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 Mb/s rate starts at 80 second and stops at 160 second. The objective of the simulation scenario is 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. Figures 7, 8, and 9 exhibit the basic behavior of Vegas, Quick Vegas and FAST respectively.

By observing the congestion window evolution shown in Fig. 7 we can find that the transient period for a new Vegas connection is quite long. Vegas prematurely stop the exponentially-increasing slow-start phase at 1.2 second and enter the linearly-increasing congestion avoidance phase. It

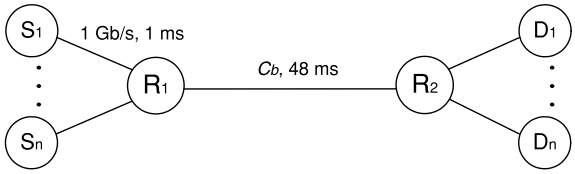

Fig. 6 Network configuration for the connections with the same *RTT*.

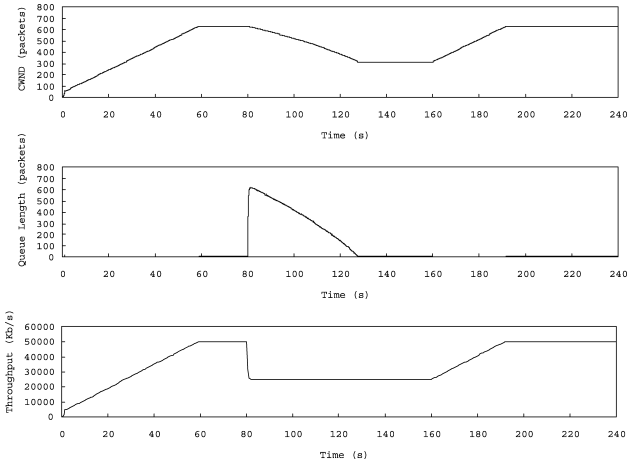

Fig. 7 Basic behavior of Vegas.

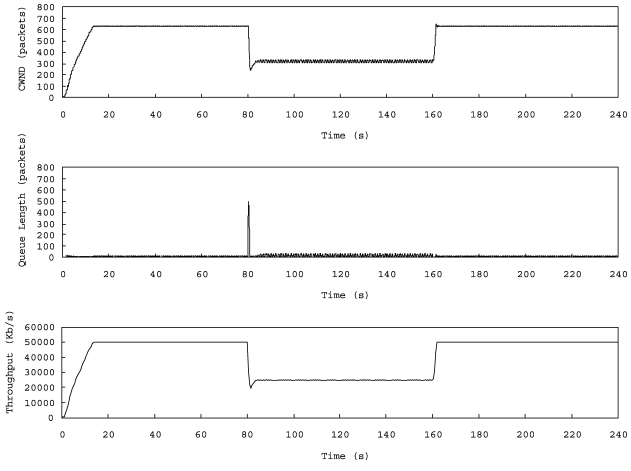

Fig. 8 Basic behavior of Quick Vegas.

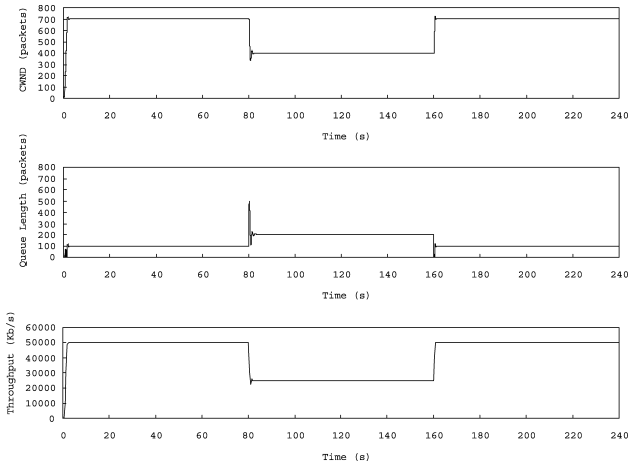

Fig. 9 Basic behavior of FAST.

takes 59 seconds to reach equilibrium. When the available bandwidth is halved at 80 seconds, Vegas takes 47.9 seconds to converge to a new steady state. As the available bandwidth is doubled at 160 second, there is a 31.8 seconds transient period for Vegas.

The queue length at bottleneck shown in Fig. 7 also reveals that Vegas can not quickly adapt to the changed bandwidth. When the available bandwidth is halved at 80 seconds, the queue is built up quickly. The maximum queue length is 620 packets and it also takes 47.9 seconds for Vegas to recover the normal queue length.

In comparison with Vegas, Quick Vegas react faster and better as shown in Fig. 8. The ramp up time of Quick Vegas is 13 seconds, and it takes 3.3 and 2.0 seconds to converge as the available bandwidth is halved and doubled respectively. Note that due to the bursty nature of a new TCP connection, the estimation of extra data will be disturbed [8]. The consecutive increment number (*succ*) may not be accumulated to a large number. Therefore, the ramp up time can not be greatly improved as compared with the convergence period of the available bandwidth is halved or doubled.

The queue length at bottleneck shown in Fig. 8 also exhibits that Quick Vegas can quickly adapt to the changed bandwidth. When the available bandwidth is halved at 80 seconds, the built up queue is quickly removed. The maximum queue length is 500 packets that is also smaller than that of Vegas (620 packets).

As for FAST shown in Fig. 9, we can find that the ramp up time is 2.6 seconds, and it takes 3.0 seconds and 1.6 seconds to converge as the available bandwidth is halved and doubled respectively. Although FAST takes less time to reach steady state than Quick Vegas in the ramp up phase, the queue length at the bottleneck router of FAST is much longer than that of Quick Vegas.

FAST maintains 100 packets of bottleneck queue length during the first 80 seconds. In the next 80 seconds, because of the presence of CBR traffic flow, the bandwidth is halved and the queue length is doubled to 200 packets as shown in Fig. 9. According to the design principle of FAST,

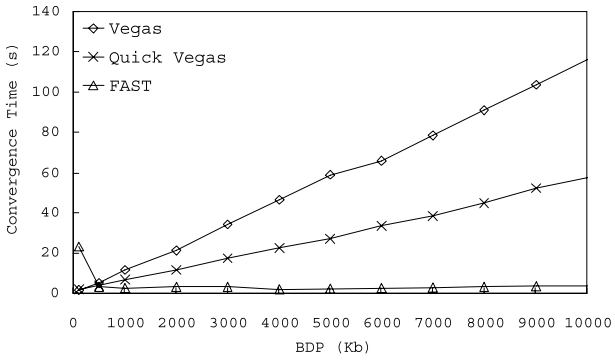

Fig. 10 Convergence time of new connections.

each connection tends to keep α packets in the bottleneck queue. Assume there are 100 FAST connections share the same bottleneck link, the buffer needed by these connections will be 10000 packets. If a CBR flow with a half of bottleneck transmission rate passes through this bottleneck, the usage of the buffer would be doubled, that is 20000 packets. In practical, it will be a serious problem to FAST because the buffer provided by router may not always be large enough to deal with FAST connections.

Based on the simulation results of throughput are shown in Figs. 7, 8, and 9, we can find Quick Vegas and FAST, especially for FAST, has a superior performance than that of Vegas. In these simulations, we define a large queue size at bottleneck so packet losses will not occur. In the later subsection, we will see more realistic scenarios. When the buffer size of the router is limited, FAST has a severe packet loss problem and thus its throughput suffers.

5.2 Convergence Time

With high BDP networks, the transient period of TCP can greatly affect overall performance. In this subsection, we use a metric "convergence time" [8] to capture the transient performance of TCP. Convergence time indicates how many *BaseRTTs* are required to reach a new stable state.

The traffic sources are the same as the previous subsection. The bottleneck capacity C_b is varied for different BDP. At some point of time, the CBR traffic source starts or stops sending packets to halve or double the available bandwidth, respectively.

Figure 10 presents the convergence time for a new connection to reach equilibrium. Theoretically, Quick Vegas doubles the increment rate in congestion avoidance phase that results in logarithm convergence time in contrast to Vegas which converges linearly. However, due to the bursty nature of a new TCP connection, the *succ* may not be consecutively accumulated. The convergence time of Quick Vegas is about half of that of Vegas as the BDP is greater than 500 Kb. On the other hand, due to the multiplicative increase scheme, FAST features a less convergence time between the three TCP variants. However, when the BDP is small (i.e., 100 Kb), FAST becomes hard to be stable. It seems that FAST may not be suitable for traditional low bandwidth-

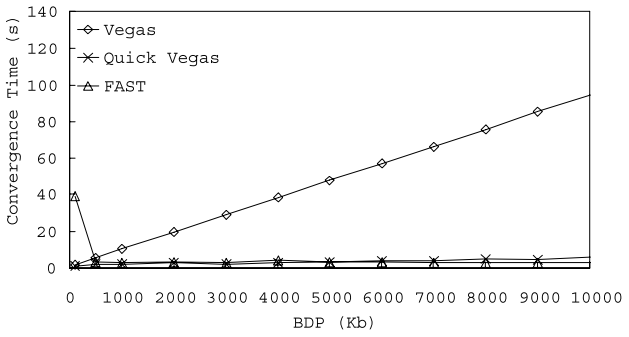

Fig. 11 Convergence time of connections when available bandwidth is halved.

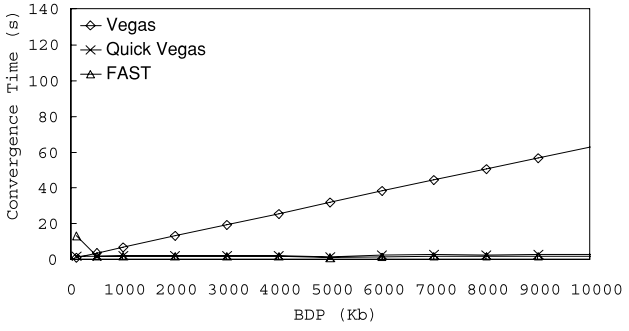

Fig. 12 Convergence time of connections when available bandwidth is doubled.

delay product networks.

Figures 11 and 12 display the convergence time as the available bandwidth is halved and doubled respectively. Obviously, both Quick Vegas and FAST greatly improves the transient performance of connections in both scenarios as compared to Vegas. Again, FAST seems not be suitable for small BDP networks.

5.3 Utilization, Queue Length, and Fairness

The simulations presented in this subsection intend to demonstrate link utilization of the bottleneck, fairness between the connections, and queue length at the bottleneck buffer where connections may join and leave the network. The buffer size of the bottleneck router is 1500 packets.

5.3.1 Connections with the Same *RTT*

We use the network topology as shown in Fig. 6 to execute the simulations. The bottleneck capacity C_b is set at 1 Gb/s. Connections C_1–C_{20}, C_{21}–C_{40}, and C_{41}–C_{60} start at 0, 100, and 200 second respectively. Each connection with the same active period is 300 seconds.

Figure 13 shows the bottleneck link utilization in which connections of Vegas, Quick Vegas, New Reno, and FAST are evaluated. When Vegas connections enter the empty network, it takes 65 seconds to reach equilibrium, while Quick Vegas takes 20 seconds. Since severe packet losses occur in the exponentially increasing slow-start phase, the link

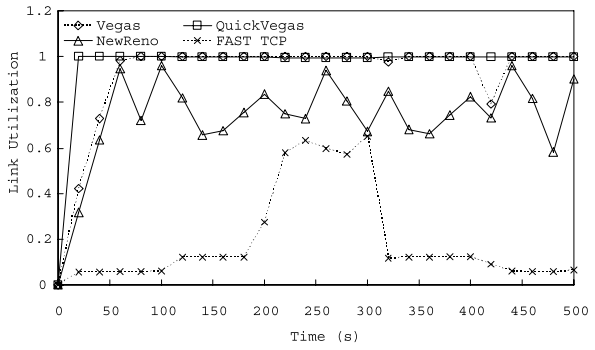

Fig. 13 Bottleneck Link Utilization for the connections with the same *RTT*.

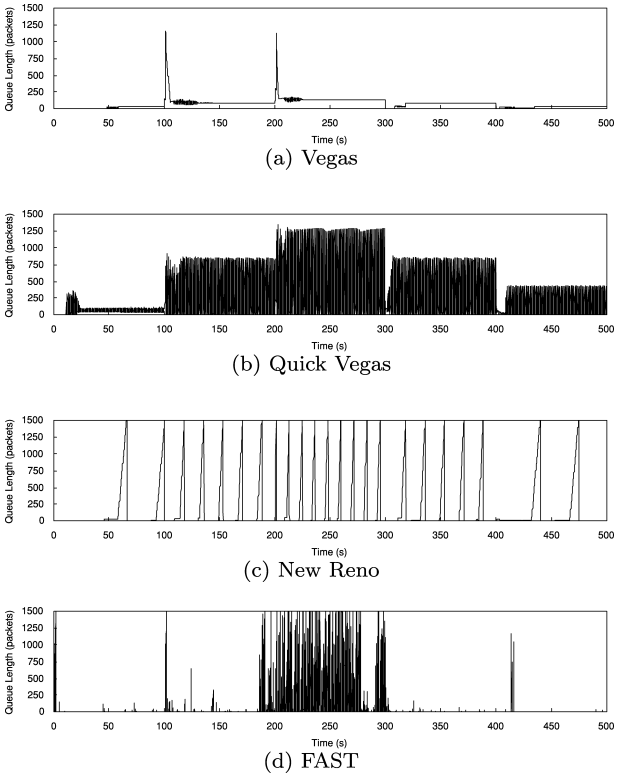

Fig. 14 Queue status of the bottleneck for the connections with the same *RTT*.

utilization of New Reno during 0–20 second is quite low (0.316). Fast TCP is limited by the buffer size of the bottleneck, it suffers a serious packet losses problem so it never reach equilibrium and the utilization is only 0.06 in the first 100 seconds.

As the new connections C_{21}–C_{40} and C_{41}–C_{60} enter the network at 100 and 200 second, both Vegas and Quick Vegas can fully utilize the bottleneck link. By observing the queue status shown in Fig. 14 we can find that Quick Vegas keeps a similar maximum queue length as compared with that of Vegas. A small maximum queue length implies that the congestion window update algorithms may prevent packet losses when the bottleneck buffer is limited. We can also find that the queue length of FAST oscillates between 0

Table 2 Fairness index for the connections with the same *RTT*.

Time (s)	0–100	100–200	200–300	300–400	400–500
Active Connections	C_1–C_{20}	C_1–C_{40}	C_1–C_{60}	C_{21}–C_{60}	C_{41}–C_{60}
Vegas	0.969	0.941	0.972	0.987	0.999
Quick Vegas	0.994	0.996	0.999	0.999	0.991
FAST	0.998	0.991	0.986	0.997	0.991
New Reno	0.996	0.946	0.993	0.999	0.999

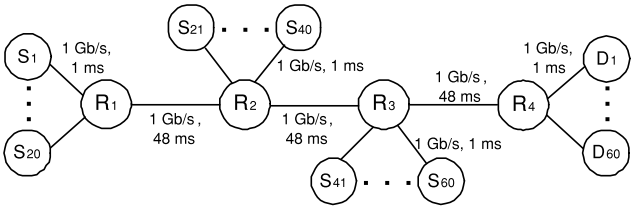

Fig. 15 Network configuration for the connections with different *RTT*s.

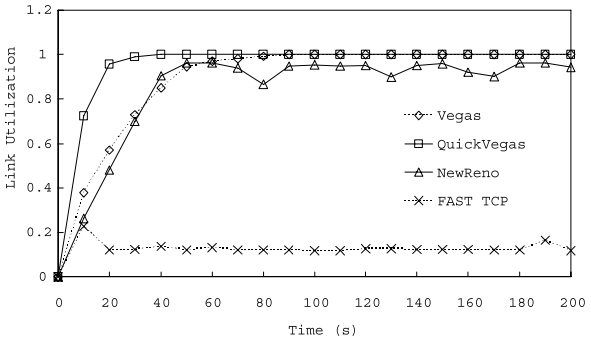

Fig. 16 Bottleneck link utilization for the connections with different *RTT*s.

and 1500, and thus cause a poor bottleneck utilization, even New Reno outperforms FAST.

When the available bandwidth increases substantially due to connections C_1–C_{20} and C_{21}–C_{40} leave the network at 300 and 400 second, the remaining connections of Quick Vegas can also quickly adapt to the new available bandwidth. As a result, the bottleneck link utilization of Quick Vegas during 300–340 and 400–440 second are higher than that of the other three TCP variants.

Different from Vegas or Quick Vegas, New Reno and FAST can not maintain a suitable queue length as shown in Fig. 14(c) and (d). Since New Reno needs to create packet losses by itself to probe the available bandwidth along the path and FAST needs to maintain at least 100 packets (i.e., $\alpha = 100$) at the bottleneck buffer for each connection. Therefore, packet losses occur periodically and certain amount of throughput is wasted. It is obvious that New Reno and FAST can not reach such high link utilization like that of Vegas or Quick Vegas as depicted in Fig. 13.

To evaluate the fairness among connections, we use the fairness index proposed in [19]. Given a set of throughput $(x_1, x_2, \ldots, x_n)$, the fairness index of the set is defined as:

$$f(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} {x_i}^2}. \tag{13}$$

The value of fairness index is between 0 and 1. If the throughput of all connections is the same, the index will take the value of 1.

Table 2 shows the fairness index of the four TCP variants for each 100 seconds time period. Although Quick Vegas adopts a more aggressive strategy to adjust the congestion window size, however, Quick Vegas keeps slightly superior fairness index values in comparison with that of Vegas. The simulation result suggests that Quick Vegas has a good characteristic of fairness when the contending connections with the same *RTT*.

Table 3 Fairness index for the connections with different *RTT*s.

Time (s)	0–50	50–100	100–150	150–200
Vegas	0.605	0.691	0.721	0.721
Quick Vegas	0.626	0.774	0.783	0.784
FAST	0.850	0.871	0.870	0.862
New Reno	0.622	0.431	0.475	0.469

5.3.2 Connections with Different *RTT*s

In this subsection, the simulations are executed for the three groups of connections with different *RTT*s those work in the network as shown in Fig. 15. Latency and bandwidth of each access link and connection link are depicted in the figure. A traffic pair contains a source and a destination with the same subscript value. All connections have emulated a 200 second FTP transfer between S_i and D_i and start at the same time. Routers utilize drop-tail queues with the buffer size being set to 1500 packets.

Figure 16 shows the link utilization of bottleneck (R_3-R_4) in which connections of Vegas, Quick Vegas, New Reno and FAST are separately evaluated. When Vegas connections enter the empty network, they take 90 seconds to reach stable state and full utilize the link while Quick Vegas' take only 40 seconds. On the other hand, due to the limitation by the buffer size, the average link utilization of FAST is about 0.15. Since the New Reno connections with the traditional congestion window update scheme can not quickly ramp up to the available bandwidth, the link utilization between 0 and 20 seconds is quite low. In the congestion avoidance phase, New Reno connections cause packet losses periodically and thus the bottleneck link cannot be full utilized.

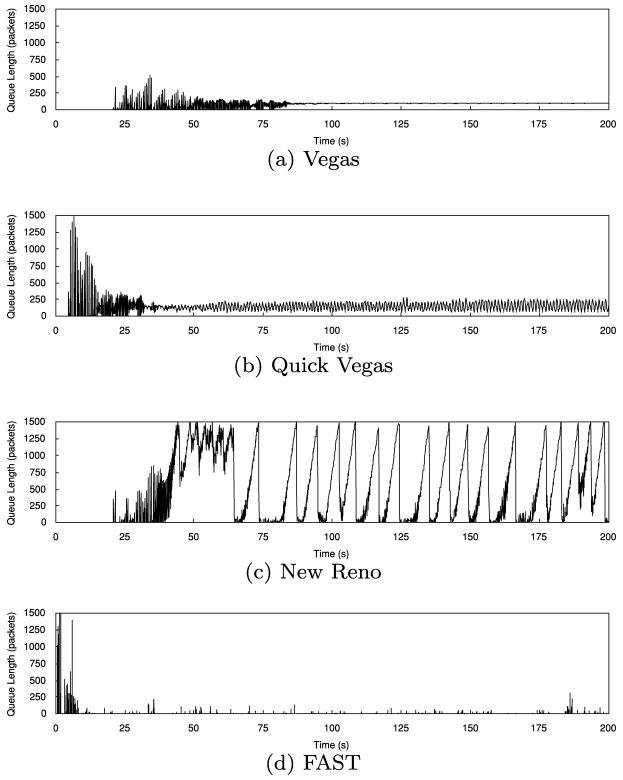

Fig. 17 Queue status of the bottleneck for the connections with different *RTT*s.

With different properties, New Reno and FAST can not maintain a stable queue length as shown in Figs. 17(c) and (d). It is obvious that, again, New Reno and FAST can not maintain such high link utilization like that of Vegas or Quick Vegas.

Table 3 shows the fairness index of the four TCP variants for each 50 seconds time period. Quick Vegas always keeps superior fairness index values than that of Vegas and New Reno. Although FAST has the most higher fairness index values in this table. However, its bottleneck link utilization is quite low.

6. Conclusions

In this research, we propose an enhanced version of TCP Vegas named Quick Vegas that improves the slow-start and congestion avoidance techniques of original Vegas. With the superior transient behavior, Quick Vegas outperforms Vegas when the connections work in high bandwidth-delay product networks. In comparison with FAST, Quick Vegas features a less bottleneck buffer utilization and keeps a better adaptability to traditional network environments.

To further advance this study, future work is needed. First, how to model the behavior of a Quick Vegas connection when it decreases its window size to alleviate the network congestion is still unanswered in this work. In other words, a more complete mathematical analysis of the new congestion avoidance scheme should be provided. Second, the most important one, the slow-start mechanism of Quick Vegas is needed to be modified. From the simulation results we are sure that there is still room for further improvement.

References

[1] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," Proc. ACM SIGCOMM'02, vol.31, no.4, pp.89–102, Aug. 2002.
[2] 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, no.8, pp.1465–1480, Oct. 1995.
[3] W. Feng and P. Tinnakornsrisuphap, "The failure of TCP in high-performance computational grids," Proc. SC 2000: High-Performance Networking and Computing Conf., Nov. 2000.
[4] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," Proc. IEEE INFORCOM'2000, vol.3, pp.1715–1723, March 2000.
[5] J.S. Ahn, P.B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and experiment," Proc. ACM SIGCOMM'95, vol.25, pp.185–195, Aug. 1995.
[6] J. Mo, R.J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," Proc. IEEE INFORCOM'99, vol.3, pp.1556–1563, March 1999.
[7] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanism of TCP," Telecommunication Systems Journal, vol.15, no.2, pp.167–184, Nov. 2000.
[8] S. Vanichpun and W. Feng, "On the transient behavior of TCP Vegas," Proc. IEEE ICCCN'02, pp.504–508, Oct. 2002.
[9] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," IEEE/ACM Trans. Netw., vol.8, no.2, pp.133–145, 2002.
[10] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, "On the self-similar nature of Ethernet traffic," IEEE/ACM Trans. Netw., vol.2, no.1, pp.1–15, Feb. 1994.
[11] S. Bhandarkar, S. Jain, and A.L. Narasimha, "LTCP: Improving the performance of TCP in highspeed networks," ACM SIGCOMM, vol.36, no.1, pp.41–50, Jan. 2006.
[12] R. Wang, K. Yamada, M.Y. Sanadidi, and M. Gerla, "TCP with sender-side intelligence to handle dynamic, large, leaky pipes," IEEE J. Sel. Areas Commun., vol.23, no.2, pp.235–248, Feb. 2005.
[13] I. Rhee and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP-Variant," Proc. PFLDnet 2005, 2005.
[14] R. King, R. Riedi, and R. Baraniuk, "TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP," Proc. INFOCOM 2005, 2005.
[15] A. Maor and Y. Mansour, "AdaVegas: Adaptive control for TCP Vegas," Proc. IEEE GLOBECOM'03, vol.7, pp.3647–3651, Dec. 2003.
[16] C. Jin, D. Wei, and S. Low, "Fast TCP: Motivation, architecture, algorithm, performance," Proc. IEEE INFORCOM 2004, vol.4, pp.2490–2501, March 2004.
[17] A. Jain and S. Floyd, "QuickStart for TCP and IP," Internet draft draft-amit-quick-start-02.txt, Oct. 2002.
[18] http://www.isi.edu/nsnam/ns/
[19] R. Jain, The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation and modeling., Wiley, New York, 1991.

Yi-Cheng Chan received his Ph.D. degree in computer science and Information 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, Taiwan. His research interests include Internet protocols, wireless networking, and AQM.

Chia-Liang Lin received his master degree in computer science and information engineering from National Changhua University of Education, Taiwan in 2007. He is currently a Ph.D. student in computer science at National Chiao Tung University, Taiwan. His research interests include the design and analysis of congestion control algorithms and wireless protocols.

Cheng-Yuan Ho is currently a Ph.D. student in computer science at National Chiao Tung University, Taiwan. He is research interests include the design, analysis, and modelling of congestion control algorithms, high speed networking, QoS, and mobile and wireless networks.