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

Yi-Cheng Chan and Chia-Liang Lin
Department of Computer Science
and Information Engineering
National Changhua University of Education
Changhua 500, Taiwan
{ycchan@cc, s94612005@mail}.ncue.edu.tw

Chia-Tai Chan
Institute of Biomedical Engineering
National Yang-Ming University
Taipei 112, Taiwan
ctchan@bme.ym.edu.tw

Cheng-Yuan Ho
Department of Computer Science
and Information Engineering
National Chiao Tung University
Hsinchu 300, Taiwan
cyho@csie.nctu.edu.tw

Abstract— **A critical design issue of TCP is its congestion control that allows the protocol to adjust the end-to-end communication rate to the bandwidth on the bottleneck link. However, 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 improved version of TCP Vegas called Quick Vegas, in which we present an efficient congestion window control algorithm for a TCP source. Our algorithm is based on the increment history and estimated amount of extra data to update the congestion window intelligently. Simulation results show that Quick Vegas significantly improve the performance of connections as well as remain fair and stable when the bandwidth-delay product increases.**

Index Terms— **congestion control, high bandwidth-delay product networks, TCP Vegas, transport protocol.**

I. Introduction

Most of current Internet applications use the Transmission Control Protocol (TCP) as its transport protocol. The behavior of TCP is tightly coupled with the overall Internet performance. TCP performs at an acceptable efficiency over today's Internet. However, theory and experiments show that, 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 [2] 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 [3], [4] 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 [3], [4], [7], stability [8], [9], fairness [8], [9], throughput and packet loss [3], [4], [5], [7], and burstiness [5], [6]. 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 [10]. As a result, a new 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 is not varied linearly [11]. 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 improved version of TCP Vegas called Quick Vegas for high BDP networks. Based on the increment history and estimated amount of extra data, we present a simple change to the congestion window update algorithm in Quick Vegas. The modification allows TCP connections to react faster and better to high BDP networks and therefore improves the overall performance.

The rest of this paper is organized as follows. In Section 2, related work is reviewed. Section 3 addresses TCP Vegas and Quick Vegas. Section 4 presents the simulation results. Lastly, we conclude this work in Section 5.

II. 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, HighSpeed TCP [12], Scalable TCP [13], AdaVegas [14], and FAST TCP [15]. The other needs more complex changes with a new transport protocol, or more

This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC 94-2213-E-018-021.

explicit feedback from the routers, examples are XCP [1] and QuickStart [16].

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 [5], [6].

AdaVegas and FAST TCP are two congestion control algorithms based on TCP Vegas. AdaVegas uses some constant increments to increase its window size. It may be still too sluggish when the connection passes through a very high-speed and long-delay path. FAST TCP adopts a more aggressive way to update its window size. However, it needs a large bottleneck buffer to prevent packet losses.

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.

QuickStart is a mechanism that uses IP options for allowing an end host to request a high initial sending rate along the end-to-end path. The feasibility of QuickStart relies on the cooperation of the end host and routers. Again, the difficulty in deployment is an issue to be overcome.

III. 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.

A. 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:

$$\Delta = (Expected - Actual) \times BaseRTT, \qquad (1)$$

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:

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

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. Note that, packet retransmission due to an expired fine-grained timer is conditioned on the reception of certain ACKs.

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*. Notably, in case of multiple packet losses occurred

during one round-trip time that trigger more than one fast retransmission, the congestion window will be reduced only for the first retransmission.

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.

B. The Proposed Mechanism

In high BDP networks, the equilibrium congestion window size is larger than that of small BDP networks. 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 should be more aggressive than it has been.

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. The details of Quick Vegas is described as follows.

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:

$$CWND = CWND + (\beta - \Delta) \times succ. \tag{3}$$

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 amount of decrement for every estimation of $\Delta > \beta$. The decrement rule can be expressed as follows:

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

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. Therefore, Quick Vegas subtract the excess amount from the congestion window directly.

Compared with TCP Vegas, the window adjustment algorithm of Quick Vegas is more aggressive. To ensure that the estimation of extra data is valid and the adjustment does not overshoot the real need, like the updating in slow-start phase, Quick Vegas adjusts its congestion window only every other *RTT*. Besides, 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:

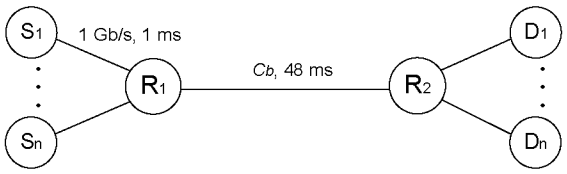


Fig. 1. Network configuration for the simulations.

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

IV. Performance Evaluation

We use the network simulator ns-2 [17] to execute the performance evaluation. All parameter settings of both Vegas and Quick Vegas are the same. Especially, $\gamma = 1$, $\alpha = 2$, and $\beta = 4$ that are same as in [4]. 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. 1. 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

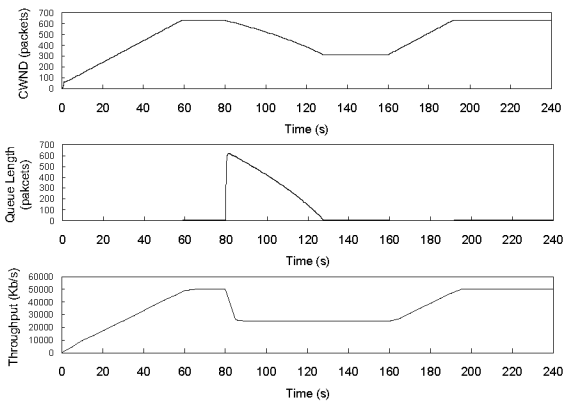


Fig. 2. Basic behavior of TCP Vegas.

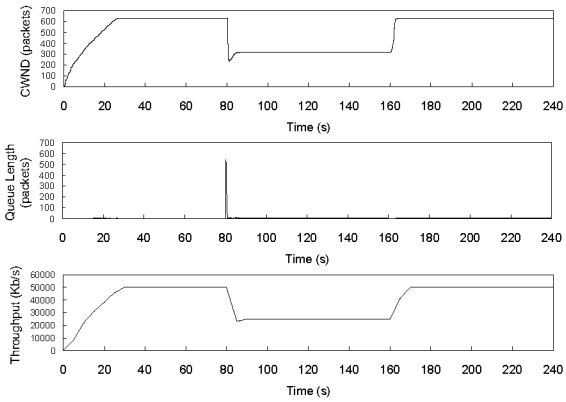


Fig. 3. Basic behavior of Quick Vegas.

link between R_1 and R_2. The C_b is set based on the need of simulation scenarios.

A. Basic Behavior

In this subsection, we compare the basic behavior between TCP Vegas and Quick Vegas 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 of either Vegas or Quick Vegas from S_1 to D_1 starts sending data at 0 second and a 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. Figure 2 and 3 exhibit the basic behavior of Vegas and Quick Vegas respectively.

By observing the congestion window evolution shown in Fig. 2 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 1.9 second and enter the linearly-increasing congestion avoidance phase. It 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. 2 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. 3. The ramp up time of Quick Vegas is 27 seconds, and it takes 6.7 and 3.9 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 [10]. 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. 3 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 540 packets that is also smaller than that of Vegas (620 packets).

Based on the simulation results of throughput shown in Fig. 2 and 3, obviously, Quick Vegas has a better performance than Vegas when a connection is either in the beginning (0–60 second) or the available bandwidth is doubled (160–190 second). Although the throughput of Quick Vegas (23.2 Mb/s) is smaller than that of Vegas (25.9 Mb/s) at 85 second, however, Vegas has a larger maximum queue length. In the simulation we define a large queue size at bottleneck so that packet losses will not occur. In more realistic scenarios, a larger maximum queue length means a higher probability of packet losses occur, which in turn would cause a lower throughput.

B. 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" [10] 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 4 presents the convergence time for a new connection to reach equilibrium. Theoretically, Quick Vegas doubles the increment rate 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.

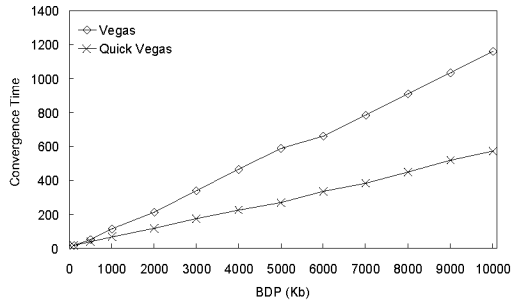


Fig. 4. Convergence time of new connections.

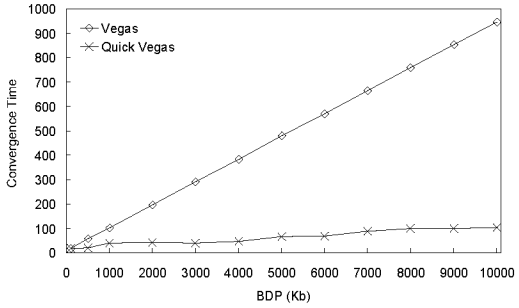


Fig. 5. Convergence time of connections when available bandwidth is halved.

Figure 5 and 6 display the convergence time as the available bandwidth is halved and doubled respectively. Obviously, Quick Vegas greatly improves the transient performance of connection in both scenarios as compared to Vegas.

C. 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 join and leave the network. 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. The size of bottleneck buffer is 1250 packets.

Table I shows the bottleneck link utilization in which

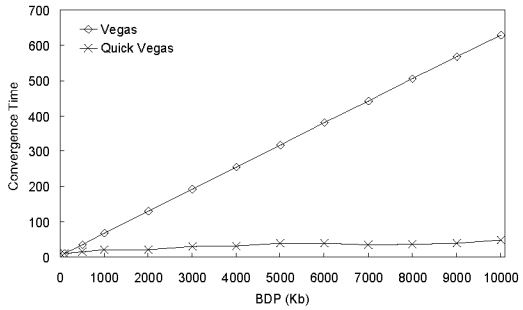
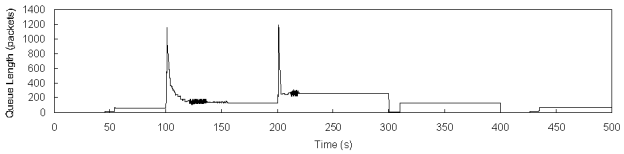


Fig. 6. Convergence time of connections when available bandwidth is doubled.

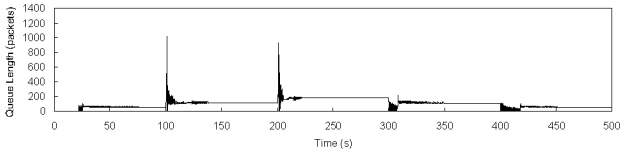
TABLE I

LINK UTILIZATION OF THE BOTTLENECK.

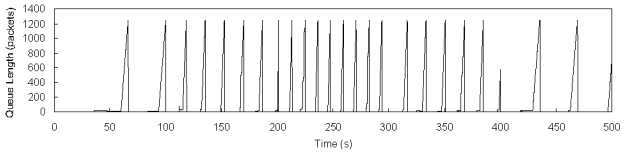
Time (s)	**Vegas**	**Quick Vegas**	**New Reno**
0–25	0.335	0.632	0.170
25–55	0.780	1.000	0.647
55–300	1.000	1.000	0.788
300–310	0.853	0.872	0.706
310–400	1.000	1.000	0.797
400–420	0.607	0.785	0.665
420–435	0.886	1.000	0.921
435–500	1.000	1.000	0.781



(a) Vegas



(b) Quick Vegas



(c) New Reno

Fig. 7. Queue status of the bottleneck.

connections of Vegas, Quick Vegas, and New Reno [18] are evaluated. When Vegas connections enter the empty network, it takes 55 seconds to reach equilibrium, while Quick Vegas takes 25 seconds. Since severe packet losses occur in the exponentially increasing slow-start phase, the link utilization of New Reno during 0–25 second is quite low (0.170).

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. However, by observing the queue status shown in Fig. 7 we can find that Quick Vegas features a smaller maximum queue length (1017 packets) as compared with that of Vegas (1188 packets). A smaller

TABLE II

AVERAGE QUEUE LENGTH (PACKETS).

Time (s)	0–100	100–200	200–300	300–400	400–500	0–500
Vegas	28	174	268	122	47	128
Quick Vegas	45	121	187	107	51	102

TABLE III
FAIRNESS INDEX.

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	0.965	0.960	0.980	0.977	0.981

maximum queue length implies that Quick Vegas can adapt to the changing network environment more quickly and prevent packet losses more effectively.

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 300–310 and 400–435 second are higher than that of Vegas.

Different from Vegas or Quick Vegas, New Reno can not maintain a stable queue length as shown in Fig. 7(c). Since New Reno needs to create packet losses by itself to probe the available bandwidth along the path. Therefore, packet losses occur periodically and certain amount of throughput is wasted. It is obvious that New Reno can not maintain such high link utilization like that of Vegas or Quick Vegas as depicted in Table I.

The average queue length at the bottleneck of Vegas and Quick Vegas are exhibited in Table II. Due to the ability of quickly grabs the idle bandwidth, Quick Vegas has slightly higher average queue length during 0–100 and 400-500 second. However, when the network is congested (100–300 second), Quick Vegas features not only smaller maximum queue length but also lower average queue length.

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:

$$f(x) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} {x_i}^2}. \quad (5)$$

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 III shows the fairness index of Vegas and Quick Vegas for each 100 seconds time period. Although Quick Vegas adopts a more aggressive strategy to adjust the congestion window size, however, Quick Vegas keeps the similar fairness index values as that of Vegas. The simulation result suggests that Quick Vegas preserves the good characteristic of fairness as that of original TCP Vegas.

V. CONCLUSIONS

In this research, we propose an improved version of TCP Vegas named Quick Vegas for high bandwidth-delay product networks. Quick Vegas presents a modification of congestion window update algorithm at a connection source. Based on the increment history and estimated amount of extra data, Quick Vegas adopts a more intelligent and aggressive way to adjust its window size. Simulation results show that Quick Vegas reacts faster and better to changing environments and therefore improves the overall performance.

However, there is still room for improvement. Due to the bursty nature of a new TCP connection, the estimation of extra data is disturbed. It makes Quick Vegas tend to stop the slow-start phase too early and has a longer transient period. Therefore, a new design of slow-start mechanism for Quick Vegas would be our future work.

REFERENCES

[1] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in Proc. ACM SIGCOMM'02, vol. 31, Issue 4, Aug. 2002.

[2] V. Jacobson, "Modified TCP congestion avoidance algorithm," Tech. Rep., Apr. 1990.

[3] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in Proc. ACM SIGCOMM'94, no. 4, pp. 24-35, Aug. 1994.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," IEEE J. Select. Areas Commun., vol. 13, pp. 1465-1480, Oct. 1995.

[5] W. Feng and P. Tinnakornsrisuphap, "The failure of TCP in high-performance computational grids," in Proc. SC 2000: High-performance Networking and Computing Conf., Nov. 2000.

[6] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," in Proc. IEEE INFORCOM'2000, vol. 3, pp. 1715-1723, Mar. 2000.

[7] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and experiment," in Proc. ACM SIGCOMM'95, vol. 25, pp. 185-195, Aug. 1995.

[8] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in Proc. IEEE INFORCOM'99, vol. 3, pp. 1556-1563, Mar. 1999.

[9] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanism of TCP," Telecommunication Systems Journal, pp. 167-184, Nov. 2000.

[10] S. Vanichpun and W. Feng, "On the transient behavior of TCP Vegas," in Proc. IEEE ICCCN'02, pp. 504-508, Oct. 2002.

[11] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar Nature of Ethernet Traffic," IEEE/ACM Trans. Networking, vol. 2, no. 1, Feb. 1994.

[12] S. Floyd, "HighSpeed TCP for large congestion windows," Internet draft draft-floyd-tcp-highspeed-02.txt, work in progress, Feb. 2003.

[13] T. Kelly, "Sclable TCP: Improving performance in highspeed wide area networks," in ACM SIGCOMM Computer Communication Review, vol. 33, Issue 2, pp. 83-91, April 2003.

[14] A. Maor and Y. Mansour, "AdaVegas: Adaptive control for TCP Vegas," in Proc. IEEE GLOBECOM'03, vol. 7, pp. 3647-3651, Dec. 2003.

[15] C. Jin, D. Wei and S. Low, "Fast TCP: Motivation, architecture, algorithm, performance," in Proc. IEEE INFORCOM 2004, vol. 4, pp. 2490-2501, Mar. 2004.

[16] A. Jain and S. Floyd, "QuickStart for TCP and IP," Internet draft draft-amit-quick-start-02.txt, Oct. 2002.

[17] UCB/LBNL/VINT Network Simulator – ns (version 2), http://www.isi.edu/nsnam/ns/.

[18] J. C. Hoe, "Start-up dynamics of TCP's congestion control and avoidance schemes," Master's thesis, MIT, Jun. 1995.

[19] R. Jain, The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling., New York: Wiley, 1991.