An Enhanced Congestion Avoidance Mechanism for TCP Vegas

Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, *Member, IEEE*

***Abstract*—TCP Vegas detects network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in traditional schemes. It has been demonstrated that TCP Vegas achieves much higher throughput than TCP Reno. However, TCP Vegas cannot prevent unnecessary throughput degradation when congestion occurs in the backward path. In this letter, we propose an enhanced congestion avoidance mechanism for TCP Vegas. By distinguishing whether congestion occurs in the forward path or not, it significantly improves the connection throughput when the backward path is congested.**

***Index Terms*—Congestion avoidance, TCP vegas, transport protocol.**

I. Introduction

With the fast growth of Internet traffic, how to efficiently utilize network resources becomes an important issue. Transmission Control Protocol (TCP) is a widely used end-to-end transport protocol on the Internet, it has several implementation versions (i.e., Tahoe, Reno, Vegas,…) which intend to improve network utilization. Among these TCP versions, Vegas can achieve much higher throughput than that of others [1].

TCP Vegas attempts to control and avoid congestion by monitoring the difference between the measured throughput and expected throughput. It uses the congestion window size and measured round-trip time (*RTT*) to estimate the amount of data in the network pipe and maintain extra data with amount between the lower threshold (α) and the upper threshold (β). By adjusting source congestion window size, it keeps an appropriate amount of extra data in the network to avoid congestion as well as to maintain high throughput. However, a roughly measured *RTT* may lead to an improper adjustment of congestion window size. If the network congestion occurs in the direction of ACK packets (backward path), it may underestimate the actual rate and cause an unnecessary decreasing of the congestion window size. Ideally, congestion in the backward path should not affect the network throughput in the forward path, which is the data transfer direction. Obviously, the control mechanism must distinguish whether congestion occurs in the forward path or not and adjust the congestion window size precisely.

Several works have been proposed to improve TCP performance for asymmetric networks. These mechanisms obtain either the forward trip time [2] or the actual flow rate on the forward path [3] depending on TCP timestamps option. Although solutions such as ACC (ack congestion control), AF (ack filtering), SA (sender adaptation) and AR (ack reconstruction) have improved the Reno's performance under asymmetric networks [4], these approaches are not effective for handling the Vegas' asymmetry problem [3]. By using the relative delay estimation along the forward path, TCP Santa Cruz [5] is able to identify the direction of congestion. However, it is not for rate-based Vegas. In this work, we propose an enhanced congestion avoidance mechanism for TCP Vegas (Enhanced Vegas), our mechanism uses TCP timestamps option to estimate queueing delay on the forward and backward path separately without clock synchronization. By distinguishing the direction along where congestion occurs, Enhanced Vegas significantly reduces the impact and improves the throughput in the case of backward congestion.

The rest of this letter is organized as follows. Section II describes the Enhanced Vegas. Section III shows the simulation results. Section IV summarizes this work.

II. Enhanced Vegas

Different from Tahoe and Reno, which detect network congestion based on packet losses, TCP Vegas estimates a proper amount of extra data to be kept in the network pipe and controls the congestion window size accordingly. The amount is between two thresholds α and (β), as shown in the following:

$$\alpha \leq (Expected - Actual) \times BaseRTT \leq \beta \qquad (1)$$

where *Expected* throughput is the current congestion window size divided by *BaseRTT* (i.e., the minimum of ever measured *RTT*s), and *Actual* throughput represents the current congestion window size divided by the newly measured *RTT*.

When backward congestion occurs, the increasing backward queueing time will affect the *Actual* throughput and enlarge the difference between the *Expected* throughput and *Actual* throughput. This results in decreasing the congestion window size. Since the network resources in the backward path should not affect that in the forward path, it is unnecessary to reduce the congestion window size when backward congestion occurs.

A measured *RTT* can be divided into four parts: forward fixed delay time (i.e., propagation delay and packet processing time), forward queueing time, backward fixed delay time, and

Manuscript received January 10, 2003. The associate editor coordinating the review of this letter and approving it for publication was Prof. D. Petr. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC 91-2213-E-009-017.

Y.-C. Chan and Y.-C. Chen are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, R.O.C. (e-mail: ycchan@csie.nctu.edu.tw; ycchen@csie.nctu.edu.tw).

C.-T. Chan is with the Telecommunication Laboratories, Chunghwa Telecom Company, Ltd, Taipei 106, Taiwan, R.O.C. (e-mail: ctchan@cht.com.tw).

Digital Object Identifier 10.1109/LCOMM.2003.814715

backward queueing time. To utilize the network bandwidth efficiently, we redefine the *Actual* throughput as

$$Actual' = \frac{cwnd}{RTT_{new} - QD(backward)} \tag{2}$$

where RTT_{new} is the newly measured *RTT*, *QD(backward)* is the backward queueing time, and *cwnd* is the current congestion window size. Consequently, the $Actual'$ is a throughput that may be achieved if there is no backward queueing delay along the path.

To realize the implementation, we make use of the TCP timestamps option to obtain the backward queueing time. When a source sends a packet, it inserts a timestamp into the TCP header. As the destination acknowledges this packet, it copies the forward timestamp and adds a backward timestamp to the ACK packet. Assume there is a TCP source a and its destination b. Let t_{ab} and t_{ba} be the end-to-end trip time of the forward data packet and the backward ACK packet respectively. We can acquire t_{ab} by subtracting the forward timestamp from backward timestamp and t_{ba} by subtracting the backward timestamp from the receiving time in the source (i.e., system timestamp of source a when it receives this ACK packet). Assume td_{ab} is the difference of system clocks in source a and destination b, and $t_{ab}\,(Min)(t_{ba}(Min))$ denotes the minimum $t_{ab}(t_{ba})$ that the source a ever measured. The trip time of a packet between two hosts is consisted of the *fixed delay time* and the *queueing delay time*. Let $t_{ab}(FD)(t_{ba}(FD))$ denotes the fixed delay time from $a(b)$ to $b(a)$ and $t_{ab}(QD)\ (t_{ba}(QD))$ represents the forward (backward) queueing time from $a(b)$ to $b(a)$. We have the following equations:

$$\begin{cases} t_{ab} = t_{ab}(FD) + t_{ab}(QD) + td_{ab} \\ t_{ba} = t_{ba}(FD) + t_{ba}(QD) - td_{ab}. \end{cases} \tag{3}$$

Since $t_{ab}(Min)\ (t_{ba}(Min))$, the minimum delay from $a(b)$ to $b(a)$, occurs when the queueing delay $t_{ab}(QD)\ (t_{ba}(QD))$ approaches zero, therefore we obtain

$$\begin{cases} t_{ab}(FD) = t_{ab}(Min) - td_{ab} \\ t_{ba}(FD) = t_{ba}(Min) + td_{ab}. \end{cases} \tag{4}$$

Here, the fixed delay time from a to b is expressed as $t_{ab}(Min) - td_{ab}$. Since t_{ab}, t_{ba}, $t_{ab}(Min)$, and $t_{ba}(Min)$ all can be measured by source a. Thus the forward queueing time $t_{ab}(QD)$ and the backward queueing time $t_{ba}(QD)$ can be derived from (3) and (4) as follows:

$$\begin{cases} t_{ab}(QD) = t_{ab} - t_{ab}(Min) \\ t_{ba}(QD) = t_{ba} - t_{ba}(Min). \end{cases} \tag{5}$$

Furthermore, the *BaseRTT* is equal to the sum of $t_{ab}(FD)$ and $t_{ba}(FD)$ (i.e., the sum of $t_{ab}(Min)$ and $t_{ba}(Min)$).

Avoids the unnecessary reduction of TCP congestion window size, our proposed enhanced congestion avoidance mechanism is more effective in improving the throughput of TCP connections. The Enhanced Vegas mechanism is described as follows:

1) Define *BaseRTT* as the sum of estimated forward fixed delay time $t_{ab}(FD)$ and estimated backward fixed delay time $t_{ba}(FD)$. Calculate the *Expected* throughput as the current congestion window size divided by *BaseRTT*.
2) Calculate the $Actual'$ as the current congestion window size divided by the difference between newly measured *RTT* and backward queueing delay $t_{ba}(QD)$.
3) Let $Diff = (Expected - Actual') \times BaseRTT$.
4) Let w_{cur} and w_next be the congestion window sizes for the current *RTT* and the next *RTT*, respectively. The rule for congestion window adjustment is as follows:

$$w_{next} = \begin{cases} w_{cur} + 1 & \text{if } Diff < \alpha, \\ w_{cur} - 1 & \text{if } Diff > \beta, \\ w_{cur} & \text{if } \alpha \le Diff \le \beta. \end{cases} \tag{6}$$

The proposed mechanism estimates *BaseRTT* and queueing time on both directions based on tracking the minimum end-to-end trip time ($t_{ab}(Min)$, $t_{ba}(Min)$). However, if the clock speed is different between the source and the destination, the accumulated time differences caused by clock skews may result in incorrect measurement of minimum end-to-end trip time. Certain efficient algorithms have been proposed to estimate clock skews from network delay measurements [6], [7]. Let C_a and C_b be the clock speed of source a and destination b, respectively. The clock ratio is denoted by γ, $\gamma = C_b/C_a$. Then we have the following equations to adjust the minimum end-to-end trip time:

$$\begin{cases} t_{ab}^{x+\Delta x}(Min) = t_{ab}^{x}(Min) - \Delta x(1-\gamma) \\ t_{ba}^{x+\Delta x}(Min) = t_{ba}^{x}(Min) + \Delta x(1-\gamma) \end{cases} \quad \Delta x \ge 0 \tag{7}$$

where $t_{ab}^{x+\Delta x}(Min)\ (t_{ba}^{x+\Delta x}(Min))$ and $t_{ab}^{x}(Min)\ (t_{ba}^{x}(Min))$ denote the $t_{ab}(Min)\ (t_{ba}(Min))$ on the time $(x + \Delta x)$ and x, respectively.

III. Performance Analysis

We perform the simulations using *ns*-2 [8] to compare the throughputs between Vegas and our proposed Enhanced Vegas. A VBR source is used to generate the backward traffic. This VBR source is an exponentially distributed ON–OFF source. During ON periods, the VBR source sends data at 2 Mb/s. Several VBR sources with different average sending rates are used to examine our mechanism. All parameters of both Vegas and Enhanced Vegas are the same. Without loss of generality, the packet size is set at 1 kbytes. To ease the comparison, we assume that the sources always have data to send. The network configuration we used is shown in Fig. 1, in which the bandwidth and delay of each full duplex link are depicted.

In the first simulation, we use a VBR source with 900 kb/s averaged sending rate to examine the throughput of Vegas and Enhanced Vegas separately. A source from either Vegas or Enhanced Vegas starts sending data at 0 second, while VBR source starts at 50 second. By observing the result in Fig. 2, when traffic source is Vegas only, it can achieve high throughput and stabilize at 1,000 Kb/s until the VBR source starts sending data. However, it shows that performance of Vegas drops dramatically as the VBR traffic starts. On the contrary, Enhanced Vegas maintains a much higher throughput than Vegas. During the active period of VBR source, the average throughput of Vegas is 325 kb/s and Enhanced Vegas is 767 kb/s. Since the traffic pattern of the VBR source keeps the same when the throughput of Vegas

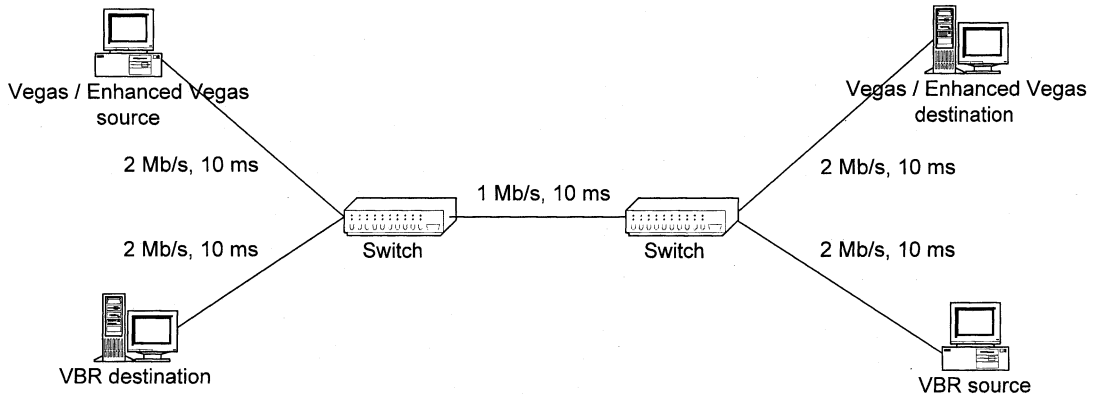

Fig. 1. Network configuration for the simulations.

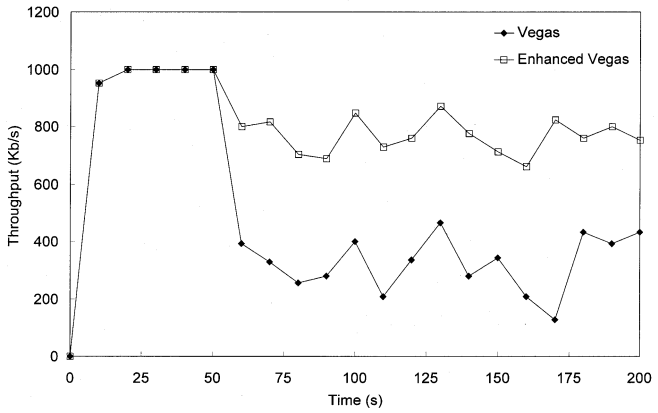

Fig. 2. Throughput comparison between Vegas and Enhanced Vegas with the backward traffic load is 0.9.

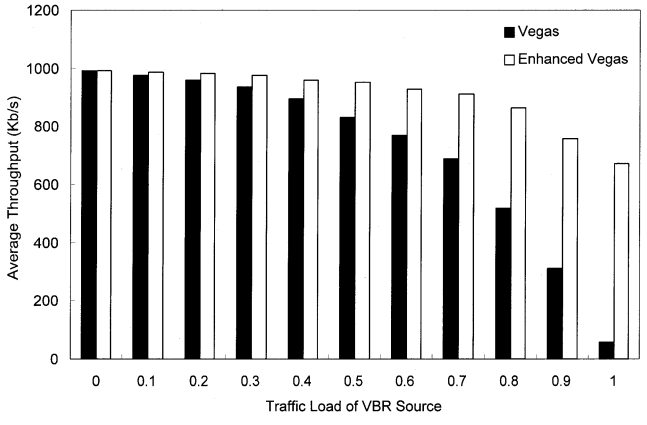

Fig. 3. Average throughput versus backward traffic load for Vegas and Enhanced Vegas.

or Enhanced Vegas is examined. Thus there appear some synchronized fluctuations of throughputs between Vegas and Enhanced Vegas. The simulation results demonstrate that the proposed scheme significantly improves the throughput of Vegas when the backward path is congested.

In the second simulation, we evaluate the average throughput of Vegas and Enhanced Vegas with different backward traffic loads separately. Sources of either Vegas or Enhanced Vegas and VBR start at 0 second. The VBR traffic loads vary from 0 to 1. The simulation period is 200 s for each sample point. From the simulation results shown in Fig. 3, we can find that the Enhanced Vegas obtains a much higher average throughput than TCP Vegas, especially when the backward traffic load is heavy. For example, as the backward traffic load is 1, Enhanced Vegas achieves a 12 times higher average throughput than that of Vegas.

IV. Conclusion and Future Work

In this letter, we propose an enhanced congestion avoidance mechanism for TCP Vegas. Comparing with other studies [2]–[5], Enhanced Vegas provides a much easier way to improve the connection throughput when the backward path is congested. The simulation results show the effectiveness of our proposed mechanism. Nevertheless, clock skew issue is still a problem of Enhanced Vegas, such as the convergence speed of the clock ratio. Therefore, how to eliminate the clock issues from Enhanced Vegas would be our future work.

References

[1] 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.

[2] O. Elloumi, H. Afifi, and M. Hamdi, "Improving congestion avoidance algorithms for asymmetric networks," in *Conf. Rec. 1997 IEEE Int. Conf. Communications*, pp. 1417–1421.

[3] C. P. Fu and S. C. Liew, "A remedy for performance degradation of TCP vegas in asymmetric networks," *IEEE Commun. Lett.*, vol. 7, pp. 42–44, Jan. 2003.

[4] H. Balakrishnan and V. N. Padmanabhan, "How network asymmetry affects TCP," *IEEE Commun. Mag.*, vol. 39, pp. 60–67, Apr. 2001.

[5] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP congestion control over internet with heterogeneous transmission media," in *Conf. Rec. 1999 IEEE Int. Conf. Network Protocols*, pp. 213–221.

[6] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurement," in *Proc. IEEE INFOCOM'99*, vol. 1, pp. 227–234.

[7] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," in *Proc. IEEE INFOCOM'2002*, vol. 1, pp. 160–169.

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