Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 27 (2004) 1624–1636
computer communications
www.elsevier.com/locate/comcom

RoVegas: a router-based congestion avoidance mechanism for TCP Vegas

Yi-Cheng Chan[a,*], Chia-Tai Chan[b], Yaw-Chung Chen[a]

[a]Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan
[b]Telecommunication Laboratories, Chunghwa TeIecom Company Ltd, Taipei 106, Taiwan

Received 19 September 2003; revised 12 June 2004; accepted 23 June 2004

Available online 23 July 2004

Abstract

Transmission control protocol (TCP) Vegas detects network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. It has been demonstrated that TCP Vegas outperforms TCP Reno in many aspects. However, TCP Vegas suffers several problems that inhere in its congestion avoidance mechanism, these include issues of rerouting, persistent congestion, fairness, and network asymmetry. In this paper, we propose a router-based congestion avoidance mechanism (RoVegas) for TCP Vegas. By performing the proposed scheme in routers along the round-trip path, RoVegas can solve the problems of rerouting and persistent congestion, enhance the fairness among the competitive connections, and improve the throughput when congestion occurs on the backward path. Through the results of both analysis and simulation, we demonstrate the effectiveness of RoVegas.

Keywords: Congestion control; TCP Vegas; Transport protocol

1. Introduction

With the fast growth of Internet traffic, how to efficiently utilize network resources is essential to a successful congestion control. Being a widely used end-to-end transport protocol on the Internet, transmission control protocol (TCP) has several implementation versions (i.e. Tahoe, Reno, Vegas,…) which intend to improve network utilization. Among these TCP variants, there are two notable approaches. One is Reno [1] which has been widely deployed on the Internet; the other is Vegas [2,3] with a claim of 37–71% throughput improvement over Reno was achieved.

TCP Reno uses a window-based flow control mechanism. Its window-adjustment algorithm consists of three phases; slow-start, congestion avoidance, and fast retransmit and recovery. A connection begins with the slow-start phase. Upon receiving an acknowledgement packet (ACK), the congestion window size (*CWND*) is increased by one packet. Since the *CWND* in the slow-start phase expands exponentially, the packets sent at this increasing rate would quickly lead to network congestion. To avoid this, the congestion avoidance phase begins when *CWND* reaches the slow-start threshold (*SSTHRESH*). In congestion avoidance phase, the *CWND* is added by 1/*CWND* packet every once receiving an ACK, this makes window size grow linearly. The process continues until a packet loss is detected.

To estimate the available bandwidth in the network, TCP Reno uses packet loss as an indicator for congestion. When congestion is detected by reception of duplicate ACKs, the fast retransmit and recovery algorithm is performed. The source retransmits the lost packet immediately without waiting for a coarse-grained timer to expire. In the meantime, the *SSTHRESH* is set to half of *CWND*, which is then set to *SSTHRESH* plus the number of duplicate ACKs. The *CWND* is increased by one packet every once receiving a duplicate ACK. When the ACK of a retransmitted packet is received, the *CWND* is set to *SSTHRESH* and the source reenters the congestion avoidance phase. If congestion is detected by a retransmission timeout, the *SSTHRESH* is set to half of *CWND* and then the *CWND* is reset to one and finally the source restarts from slow-start phase.

The fast retransmit and recovery algorithm of TCP Reno allows the connection to quickly recover from isolated

* Corresponding author. Tel.: +886-3-5731851; fax: +886-3-5714031.
E-mail addresses: ycchan@csie.nctu.edu.tw (Y.-C. Chan), ctchan@cht.com.tw (C.-T. Chan), ycchen@csie.nctu.edu.tw (Y.-C. Chen).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.06.008

packet losses. However, when multiple packets are dropped from a window of data, Reno may suffer serious performance problems. Since Reno retransmits at most one dropped packet per round-trip time (*RTT*), and further the *CWND* may be decreased more than once due to multiple packet losses occurred during one *RTT* interval. In this situation, TCP Reno operates at a very low rate and loses a significant amount of throughput. TCP New Reno [4] is a modification to the fast retransmit and recovery. In TCP New Reno, a sender can recover from multiple packet losses without having to time out. TCP with selective acknowledgement (SACK) options [5] also has been proposed to efficiently recover from multiple packets loss. However, the additive increase and multiplicative decrease approach (AIMD) of Reno leads to periodic oscillations in the congestion window size, round-trip delay, and queue length of the bottleneck node. Recent works have shown that the oscillation may induce chaotic behavior into the network thus adversely affects overall network performance [6,7].

TCP Vegas employs a fundamentally different congestion avoidance mechanism. It uses the difference between the expected and actual throughputs to estimate the available bandwidth in the network. The idea is that when the network is not congested, the actual throughout will be close to the expected throughput. Otherwise the actual throughput will be smaller than the expected throughput. TCP Vegas uses the difference in throughput to gauge the congestion level in the network and update the *CWND* accordingly. As a result, TCP Vegas is able to detect network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in Reno. Many studies have demonstrated that Vegas outperforms Reno in the aspects of overall network utilization [2,3,8], stability [9,10], fairness [9,10], and throughput [2,3,6,8].

Although Vegas is superior to Reno in the aforementioned aspects, it suffers some problems that inhere in its congestion avoidance scheme, these include issues of rerouting [9], persistent congestion [9], fairness [10–12], network asymmetry, [13–15], and incompatibility between Reno and Vegas [9,16,17]. All these problems may be obstacles for Vegas to achieve a success.

In this work, we propose a router-based congestion avoidance mechanism for TCP Vegas (abbreviated as RoVegas hereafter). Through the proposed mechanism performed in routers along the round-trip path, RoVegas may solve the problems of rerouting and persistent congestion, enhance the fairness among the competitive connections, and improve the throughput when congestion occurs on the backward path. Based on the results of analysis and simulation, we demonstrate the effectiveness of RoVegas.

The rest of this paper is organized as follows. Section 2 describes Vegas and its problems. Section 3 discusses the RoVegas. In Section 4, related work is reviewed. Sections 5 and 6 present the analysis and simulation results, respectively. Lastly, we conclude this work in Section 7.

2. TCP Vegas and its problem statements

2.1. TCP Vegas

Compared with Reno, TCP Vegas adopts a more sophisticated bandwidth estimation scheme that tries to avoid rather than to react to congestion. Vegas uses the measured *RTT* to accurately calculate the amount of data packets that a source can send. 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 *CWND* accordingly. It records the *RTT* and sets *BaseRTT* to the minimum of ever measured RTTs. The amount of extra data is between two thresholds α and β, as shown in the following:

$$\alpha \leq (Expected - Actual) \times BaseRTT \leq \beta, \tag{1}$$

where *Expected* throughput is the current *CWND* divided by *BaseRTT*, and *Actual* throughput represents the current *CWND* divided by the newly measured *RTT*. The congestion window is kept constant when the amount of extra data is between α and β. If the amount 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 amount is smaller than α, the connection may be under utilizing the available bandwidth. Hence, the *CWND* will be increased.

During the slow-start phase, Vegas is similar to Reno that allows a connection to quickly ramp up to the available bandwidth. However, to ensure that the sending rate will not increase too fast, Vegas doubles the size of its congestion window only every other *RTT*. A similar congestion detection mechanism is applied during the slow-start to decide when to switch the phase. If the estimated extra data is greater than γ, Vegas leaves the slow-start phase, reduces its *CWND* by 1/8 and enters the congestion avoidance phase.

To retransmit the lost packets quickly and adjust the congestion window correctly, Vegas extends TCP 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 time-out period of the oldest unacknowledgement packet is expired. If so, the packet is retransmitted. 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. Moreover, in case of multiple packet losses occurred during one *RTT* that trigger more than one fast retransmission, the congestion window will be reduced only for the first retransmission.

2.2. Problem statements of Vegas

In Vegas, there are several problems inhere in its congestion avoidance mechanism that may have a serious impact on the performance. The problems are summarized as follows.

2.2.1. Rerouting

Since Vegas estimates the *BaseRTT* to compute the expected throughput and adjust its window size accordingly. Thus it is very important to estimate the quantity accurately for Vegas connections. Rerouting may cause a change of the fixed delay[1] that could result in substantial throughput degradation. When the route of a connection is changed, if the new route has a shorter fixed delay, it will not cause any serious problem for Vegas because most likely some packets will experience shorter *RTT*, and *BaseRTT* will be updated eventually. On the other hand, if the new route for the connection has a longer fixed delay, it would be unable to tell whether the increased *RTT* is due to network congestion or route change. The source host may misinterpret the increased *RTT* as a signal of congestion in the network and decrease its window size. This is just the opposite of what the source should do.

2.2.2. Persistent Congestion

Persistent congestion is another problem caused by the incorrect estimation of *BaseRTT* [9]. Overestimation of the *BaseRTT* in Vegas may cause a substantial influence on the performance. Suppose that a connection starts while many other active connections also exist, the network is congested and the packets are accumulated in the bottleneck. Then, due to the queuing delay caused by packets of other connections, the packets from the new connection may experience a *RTT* that are considerably larger than the actual fixed delay along the path. Hence, the window size of the new connection will be set to a value such that its expected amount of extra data lies between α and β; in fact, there may be much more extra data in the bottleneck queue due to the inaccurate estimation of the fixed delay. This scenario will repeat for each newly added connection, and it may cause the bottleneck node to remain in a persistent congestion. Persistent congestion is likely to happen in TCP Vegas due to its fine-tuned congestion avoidance mechanism.

2.2.3. Unfairness

Different from TCP Reno, Vegas is not biased against the connections with longer *RTT* [9,10]. However, there is still unfairness comes with the nature of Vegas. According to the difference between the expected and actual throughputs, a Vegas source attempts to maintain an amount of extra data between two thresholds α and β by adjusting its *CWND*.

The range between α and β induces uncertainty to the achievable throughput of connections. Since Vegas may keep different amount of extra data in the bottleneck even for the connections with the same round-trip path. Thus, it prohibits better fairness achievement among the competitive connections.

Furthermore, the inaccurate computation of expected throughput may also lead to unfairness. Recall that the computation of expected throughput is based on the measurement of *BaseRTT*. If Vegas connections cannot estimate the *BaseRTT* accurately, it may affect the fairness achievement. When a new connection starts sending data while many other connections are also active, it may cause overestimation of the fixed delay and result in unfair distribution of bandwidth among the Vegas connections.

2.2.4. Network asymmetry

Based on the estimated extra data kept on the bottleneck, Vegas updates its congestion window to avoid congestion as well as to maintain high throughput. However, a roughly measured *RTT* may lead to a coarse adjustment of *CWND*. If the network congestion occurs in the direction of ACK (backward path), it may underestimate the actual throughput and cause an unnecessary decreasing of the *CWND*. 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 *CWND* accordingly.

Some prevalent networking technologies, with asymmetry network characteristics, such as asymmetric digital subscriber line (ADSL), cable modem, and satellite-based networks, greatly increase the possibilities of backward path congestion. In these networks, it is very likely that the capacity of the forward direction is larger than that of backward direction. Both Reno and Vegas may suffer severe performance degradation in the case of backward path congestion, especially for Vegas [15]. Therefore, how to amend the deficiency of TCP Vegas in such situation becomes an important issue.

2.2.5. Incompatibility

TCP Vegas adopts a proactive congestion avoidance scheme, it reduces its congestion window before an actual packet loss occurs. TCP Reno, on the other hand, employs a reactive congestion control mechanism. It keeps increasing its congestion window until a packet loss is detected. Researchers [9,16,17] have found that when Reno and Vegas perform head-to-head, Reno generally steals bandwidth from Vegas. This incompatibility between Vegas and Reno depress the adoption of Vegas on the Internet.

[1] The fixed delay is the sum of propagation delay and packet processing time along the round-trip path. In other words, the fixed delay is the *RTT* without queuing delay.

3. TCP RoVegas

From the above discussion, we can find that the coarse estimation of fixed delay along the round-trip path, *BaseRTT*, results in problems such as issues of rerouting, persistent congestion, and unfairness. A Vegas source is unable to distinguish whether congestion occurs in the forward path or not, this further leads to unnecessary throughput degradation when the congestion occurs on the backward path. In this work, we propose a router-based congestion avoidance mechanism (RoVegas) for TCP Vegas to deal with these problems. The details of the proposed mechanism are described as follows.

3.1. Proposed mechanism

TCP Vegas estimates a proper amount of extra data to be kept in the network pipe and controls the *CWND* accordingly. The amount is between two thresholds α and β as shown in Eq. (1). When backward congestion occurs, the increased backward queuing time will affect the *Actual* throughput and enlarge the difference between the *Expected* throughput and *Actual* throughputs. It results in decreasing the *CWND*. Since the network resources in the backward path should not affect the traffic in the forward path, it is unnecessary to reduce the *CWND* when only backward congestion occurs.

A measured *RTT* can be divided into four parts; forward fixed delay (i.e. propagation delay and packet processing time), forward queuing time, backward fixed delay, and backward queuing time. To utilize the network bandwidth efficiently, we redefine the *Actual* throughput as

$$Actual' = \frac{CWND}{RTT - QT_b}, \tag{2}$$

where *RTT* is the newly measured *RTT*, QT_b is the backward queuing time, and *CWND* is the current *CWND*. Consequently, the $Actual'$ is a throughput that can be achieved if there is no backward queuing delay along the path.

To realize our scheme, we define a new IP option named accumulate queuing time (AQT) to collect the queuing time along the path. According to the general format of IP options described in Ref. [20], the fields of an AQT option are created as in Fig. 1. The option type and length fields indicate the type and length of this IP option. The AQT field expresses the accumulated queuing time that a packet experienced along the route path. The AQT-Echo field echoes the accumulated queuing time value of an AQT option that was sent by the remote TCP.

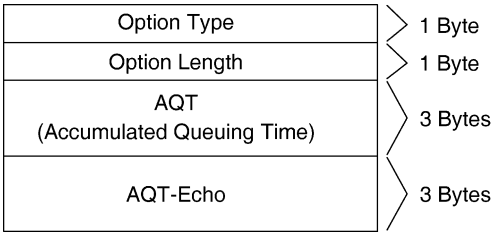


Fig. 1. Fields of an AQT option.

A probing packet is a normal TCP packet (data or ACK) with AQT option in its IP header. When a RoVegas source sends out a probing packet, it sets the AQT field to zero. An AQT-enabled router (i.e. a router that is capable of AQT option processing) adds the queuing delay of a received probing packet to the AQT field. The queuing time is computed based on the queuing disciplines. The details regarding how to compute the queuing time of each received probing packet in various queuing disciplines is beyond the scope of this paper.

Whenever a RoVegas destination acknowledges a probing packet, it inserts an AQT option into the ACK. The AQT field is set to zero, and the AQT-Echo field is set to the value of the AQT field of the received packet. Through the AQT-enabled routers along the round-trip path, a RoVegas source is able to obtain both the forward queuing time (the value of AQT-Echo field) and backward queuing time (the value of AQT field) from the received probing packet. Moreover, for each probing packet received by a RoVegas source, the *BaseRTT* can be derived as follows:

$$BaseRTT = RTT - (\text{AQT} + \text{AQT-Echo}). \tag{3}$$

Notice that, the derived *BaseRTT* of a connection will be identical for each probing packet received when both the route and size of the probing packets are fixed. The derived *BaseRTT* of RoVegas represents the actual fixed delay along the round-trip path, if the path of a connection is rerouted and the fixed delay is changed, the newly derived *BaseRTT* may reflect the rerouting information. As a result, the issue of rerouting can be solved. Furthermore, since each connection of RoVegas is able to measure the fixed delay without bias, the problem of persistent congestion can be avoided and the fairness among the competitive connections can also be improved.

To avoid the unnecessary reduction of *CWND*, the proposed router-based congestion avoidance mechanism is described as follows:

- Derive the *Expected* throughput that is defined as the current *CWND* divided by *BaseRTT*.
- Calculate the $Actual'$ as the current *CWND* divided by the difference between the newly measured *RTT* and backward queuing time.
- Let $Diff = (Expected - Actual') \times BaseRTT$
- Let w_{cur} and w_{next} be the *CWND*s for the current *RTT* and the next *RTT*, respectively. The rule for congestion window adjustment is as follows:

$$w_{next} = \begin{cases} w_{cur} + 1, & if \quad Diff < \alpha \\ w_{cur} - 1, & if \quad Diff > \beta \\ w_{cur}, & if \quad \alpha \le Diff \le \beta \end{cases}. \tag{4}$$

3.2. Implementation issue

RoVegas relies on probing packets to probe the network status, therefore, how often a probing packet will be sent for a connection is an important issue. Since the window adjustment of RoVegas is performed on per-*RTT* basis. Inserting probing packets frequently makes the proposed mechanism robust against the network congestion, however, it also imposes more overhead on RoVegas. For the overhead induced by the probing packets, we consider the worst case that every packet with the AQT option. If the data packet size is 1500 bytes, which is the maximum transmission unit of Ethernet, the overhead ratio of data packets is 8/1500, which is about 0.53%. In the practical implementation, the number of the probing packets per-*RTT* can be dynamically adjusted depends on the network status. That is, the more severe the backward congestion is, the more frequent the AQT option will be inserted into a data packet. Through this way, the overhead induced by the AQT option can be reduced to an even smaller amount.

We make every packet to be a probing packet and demonstrate that the proposed mechanism is effective to improve the performance of TCP Vegas by the results of both analysis and simulation shown in Sections 5 and 6.

4. Related work

Congestion control for TCP is an active research area. Since Brakmo et al. [2,3] proposed TCP Vegas in 1994 with claiming to achieve higher throughput and one-fifth to one-half the losses of TCP Reno, there have been quite a lot of studies focusing on the TCP Vegas.

Ahn et al. [8] performed some live Internet experiments with TCP Vegas. They reproduced claims in Refs. [2,3] with varying background traffic and concluded that Vegas indeed offers improved throughput of at least 3–8% over Reno. TCP Vegas is also found to retransmit fewer packets and to have a lower average and a lower variance of *RTT* than Reno.

By using the fluid model and simulations, Mo et al. [9] show that Vegas is not biased against connections with longer *RTT* like Reno does. It achieves better fairness of bandwidth sharing among the competitive connections with different propagation delays. However, they also pointed out that TCP Vegas does not receive a fair share of bandwidth in the presence of a TCP Reno connection.

Two problems of Vegas that could have serious impact on its performance are also described in Ref. [9]. One is the rerouting problem. Rerouting may lead to the change of fixed delay and therefore bring about inaccurate estimation of *BaseRTT*. This may erroneously affect the adjustment of the *CWND*. The other is the persistent congestion, which is still caused by the inaccurate estimation of *BaseRTT*.

Hasegawa et al. [10] focus on the fairness and stability of the congestion control mechanisms for TCP. They use an analytical model to derive that TCP Vegas can offer higher performance and much stable operation than Reno. However, because of the default values of α and β in the implementation, connections of Vegas could not share the total bandwidth in a fair manner. Thus Hasegawa et al. propose an enhanced Vegas which sets α equal to β to remove the uncertainty induced by the range between α and β.

Through the analytical study and simulation, Boutremans et al. [11] show that in addition to the setting of α and β, the fairness of TCP Vegas critically requires an accurate estimation of propagation delay. Nevertheless, they think there is no obvious way to achieve this.

To prevent the performance degradation of TCP Vegas in asymmetric networks, Elloumi et al. [13] proposed a modified algorithm. It divides a *RTT* into a forward trip time and a backward trip time in order to remove the effects of backward path congestion. However, it seems unlikely to work without clock synchronization.

Another mechanism for solving the issue of Vegas in asymmetric networks is proposed in Refs. [14,15]. Fu et al. employ an end-to-end method to measure the actual flow rate on the forward path at a source of TCP Vegas. Based on the differences between the expected rate along the round-trip path and the actual flow rate on the forward path, the source adjusts the *CWND* accordingly. However, in a backward congestion environment the self-clocking behavior of TCP will be disturbed. Then the TCP traffic with bursty nature will make the source hard to decide the measure interval between two consecutive tagged packets. Moreover, the actual flow rate on the forward path that measured by the source may be usually greater than the expected rate along the round-trip path. It may lead to an over-increased *CWND*, and causes congestion on the forward path.

To enhance the throughput of Vegas when it performs with TCP Reno head-to-head, Lai [16] suggests two approaches, one is using the random early detection (RED) mechanism in the router, the other is adjusting parameters of Vegas. Both may improve the performance of Vegas.

Feng et al. [17] show that the default configuration of Vegas is indeed incompatible with TCP Reno. However, with a careful analysis of how Reno and Vegas use buffer space in the routers, Vegas and Reno can be compatible with one another if Vegas is configured properly. Nevertheless, no mechanism has been proposed to configure Vegas automatically.

5. Performance analysis

The performance of TCP Vegas in forward path congestion environments has been studied and modelled [18,19]. In this section, we present a steady-state performance analysis of both Vegas and RoVegas when congestion occurs in backward path. By investigating the queue length of the bottleneck buffer through the analytical approach, we

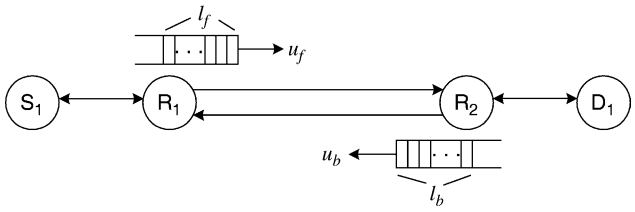


Fig. 2. Network model for analysis.

can clarify the essential nature of these two mechanisms. The network model used in the analysis is depicted in Fig. 2.

Assuming the source S_1 is a greedy source. The destination D_1 generates an ACK immediately upon receiving a data packet sent from S_1. Either the forward or backward link between two routers R_1 and R_2 is the bottleneck along the path. The forward link between two routers has a capacity of u_f (data packets per second) and backward link has a capacity u_b (ACKs per second). To facilitate the analysis as the backward path is congested, a normalized asymmetric factor k, $k=u_f/u_b$, is introduced [21]. The network is defined as asymmetric if the asymmetric factor is greater than one.

The service discipline is assumed to be First-In-First-Out (FIFO). Let τ be the *BaseRTT* (without any queuing delay), l_f and l_b be the mean numbers of packets queued in the forward and backward bottleneck buffer, respectively. Since the window size of Vegas converges to a fixed value in steady state, the mean number of packets queued in bottleneck buffer should also be converged to a fixed level [10].

5.1. Analysis on Vegas

The congestion avoidance mechanism of Vegas shown in Eq. (1) can be rewritten as below:

$$\frac{RTT}{RTT-BaseRTT}\alpha \le CWND \le \frac{RTT}{RTT-BaseRTT}\beta. \quad (5)$$

The *BaseRTT* and *RTT* can be expressed as follows:

$$BaseRTT = \tau, \quad (6)$$

$$RTT = \tau + \frac{l_f}{u_f} + \frac{l_b}{u_b}. \quad (7)$$

After substitution of Eqs. (6) and (7), Eq. (5) can be rewritten as:

$$\frac{\tau u_f u_b + l_f u_b + l_b u_f}{l_f u_b + l_b u_f}\alpha \le CWND \le \frac{\tau u_f u_b + l_f u_b + l_b u_f}{l_f u_b + l_b u_f}\beta. \quad (8)$$

To elaborate on the following analysis, we now observe the TCP Vegas behavior as shown in Figs. 3 and 4. A single Vegas connection runs on a network in which the round-trip propagation delay is 40 ms, bottleneck queue size is 10 packets, and forward bottleneck capacity is 200 packets/s. Figs. 3 and 4 show the evolutions of window size and

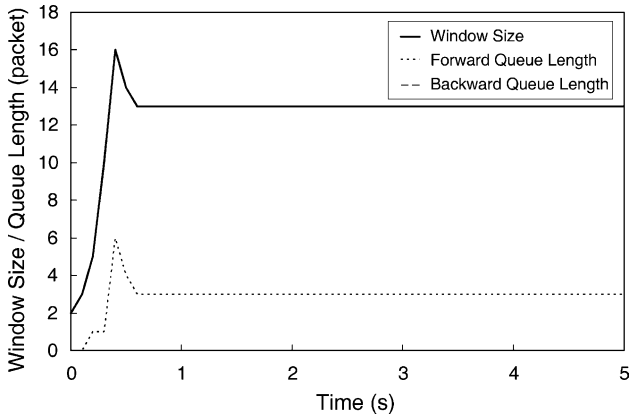


Fig. 3. Window size and bottleneck queue length of Vegas with $k=0.5$.

bottleneck queue length with different normalized asymmetric factor k. When the Vegas connection runs on the symmetric network ($k=0.5$), its stable window size (13 packets) is larger than that of it runs on the asymmetric network ($k=2$). The forward queue length is stable at three packets and no packet is queued in the backward bottleneck. On the contrary, as the Vegas connection runs on the asymmetric network, the packets is queued in the backward bottleneck.

5.1.1. Symmetric network ($k \le 1$).

If the bottleneck is in the forward path, packets will be accumulated in the forward bottleneck queue and no packets will be queued in the backward path, that is $l_b=0$, thus Eq. (8) can be simplified as:

$$\frac{\tau u_f + l_f}{l_f}\alpha \le CWND \le \frac{\tau u_f + l_f}{l_f}\beta. \quad (9)$$

Since S_1 is the only traffic source in the network thus it may occupy all the bandwidth of the bottleneck. Based on the fluid approximation, the *CWND* of S_1 can be obtained through the bandwidth-delay product of the bottleneck as follows:

$$CWND = u_f \times \left(\tau + \frac{l_f}{u_f}\right). \quad (10)$$

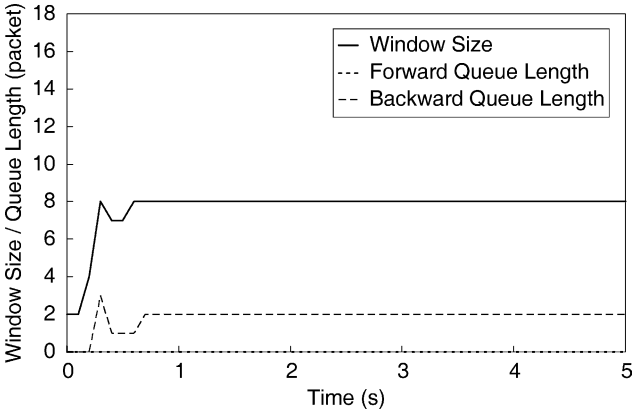


Fig. 4. Window size and bottleneck queue length of Vegas with $k=2$.

By substituting Eq. (10) into Eq. (9), we have

$$\alpha \le l_f \le \beta. \tag{11}$$

The throughput T of S_1 can also be derived from Eqs. (7) and (10) as:

$$T = \frac{CWND}{RTT} = u_f. \tag{12}$$

From Eqs. (11) and (12), we observe that when the bottleneck appears in the forward path, the mean number of packets queued in forward bottleneck buffer is kept stable between α and β, and the link bandwidth is always fully utilized in steady state. This observation matches the design goal of Vegas.

5.1.2. Asymmetric network ($k>1$).

If the bottleneck exists in the backward path then the queue of the backward bottleneck node will be built up and no packets will be queued in the forward path, that is $l_f=0$, therefore Eq. (8) can be rewritten as:

$$\frac{\tau u_b + l_b}{l_b}\alpha \le CWND \le \frac{\tau u_b + l_b}{l_b}\beta. \tag{13}$$

Similar to the Eq. (10), the window size of S_1 can also be obtained by the bandwidth-delay product of the bottleneck link:

$$CWND = u_b \times \left(\tau + \frac{l_b}{u_b}\right). \tag{14}$$

By substituting Eq. (14) into Eq. (13), we have

$$\alpha \le l_b \le \beta. \tag{15}$$

In the meantime, the throughput T of S_1 can be derived from Eqs. (7) and (14) as:

$$T = \frac{CWND}{RTT} = u_b = \frac{u_f}{k}. \tag{16}$$

From Eq. (15) we can find that, Vegas is unable to distinguish whether congestion occurs in the forward path or not. It keeps a quantity of extra data steady between α and β on the backward path. This may lead to poor utilization of forward path. As shown in Eq. (16), the throughput of S_1 is limited by the capacity of backward path. Notably, an ACK in the backward path implies that a data packet has arrived at its destination. Therefore, the throughput of S_1 is u_f/k (data packets per second).

5.2. Analysis on RoVegas

The congestion avoidance mechanism of RoVegas can be briefly expressed as follows:

$$\frac{\alpha}{BaseRTT} \le \frac{CWND}{BaseRTT} - \frac{CWND}{RTT - \frac{l_b}{u_b}} \le \frac{\beta}{BaseRTT}. \tag{17}$$

By Eq. (17), we have the $CWND$ of RoVegas as:

$$\frac{RTT}{RTT - BaseRTT - \frac{l_b}{u_b}}\alpha \le CWND \le \frac{RTT}{RTT - BaseRTT - \frac{l_b}{u_b}}\beta. \tag{18}$$

From Eqs. (6) and (7), Eq. (18) can be rewritten as:

$$\frac{\tau u_f + l_f}{l_f}\alpha \le CWND \le \frac{\tau u_f + l_f}{l_f}\beta. \tag{19}$$

Since the result of Eq. (19) is identical to that of Eq. (9). If the bottleneck is in the forward path (i.e. $l_b=0$), the behavior of RoVegas will be same as Vegas. However, the result of Eq. (19) reveals that the throughput of RoVegas in the case of backward congestion is not simply limited by the bandwidth of backward path as that of Vegas.

As shown in Fig. 5, when a RoVegas connection runs on the asymmetric network ($k=2$), its stable window size (33 packets) is larger than that of a Vegas connection runs on the same network as depicted in Fig. 4. RoVegas keeps a stable queue length (two packets) in the forward bottleneck and its backward bottleneck queue is always full.

Both Fig. 5 and Eq. (17) demonstrate that RoVegas always attempts to maintain a proper amount of extra data in the forward path regardless of where the congestion occurs. However, TCP is a 'self-clocking' protocol, that is, it uses ACKs as a 'clock' to strobe new packets into the network [22]. Hence, as the backward path is congested, the rate of data flow in the forward direction will be throttled in a manner by the rate of ACK flow.

There exists a further restriction in Vegas that may limit the growth of the congestion window. The congestion window will not be increased if the source is unable to keep up with, that is, the difference between the $CWND$ and the amount of outstanding data is larger than two maximum-sized packets [23]. Be a variant of TCP Vegas, RoVegas also complies with this restriction.

In an asymmetric network, for example $k=8$, assuming that in steady state the forward path can be fully utilized by

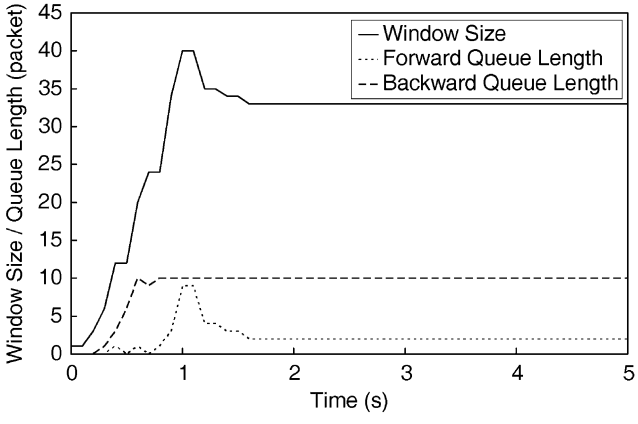


Fig. 5. Window size and bottleneck queue length of RoVegas with $k=2$.

S_1, it means that 7/8 of ACKs will be dropped in the backward path. With TCP, the ACKs are cumulative [24], that is, later ACKs carry all the information contained in earlier ACKs. In this case, a survived ACK may represent that eight data packets have arrived at the destination. Once a survived ACK is received by the source, the difference between the *CWND* and the amount of outstanding data is eight packets. It will restrict the growth of the congestion window. Actually, the forward path may not be fully utilized by RoVegas with $k=8$.

For an asymmetric network, if the dropping ratio of ACKs reaches 2/3, the congestion window of RoVegas will not be increased. Since for each ACK received by the RoVegas source, the difference between the *CWND* and the amount of outstanding data will be three packets. In such situation, RoVegas enters the steady state and the growth of congestion window stops. For each ACK received, the RoVegas source may send three packets back-to-back.

Let F be the throughput ratio of RoVegas to Vegas (i.e. $F=$(throughput of RoVegas/throughput of Vegas)). In asymmetric networks, we have the throughput relationship of Vegas and RoVegas as follows:

$$1 < F \leq 3, \quad \forall k > 1. \tag{20}$$

Note that, the throughput of RoVegas, contains the overhead induced by the AQT option. So the actual throughput ratio of RoVegas to Vegas should be slightly smaller than F. Eq. (20) will be further verified by the following performance evaluation.

6. Performance evaluation

In this section, we compare the performance of TCP RoVegas with TCP Vegas by using the network simulator ns-2 [25]. We show the performance results in backward congestion environments, the bias experiments, the fairness investigations among the competitive connections, and the study of gradual deployment.

The FIFO service discipline is assumed. Every packet of RoVegas is a probing packet. Whenever a throughput of RoVegas is computed, the overhead induced by the AQT option will be subtracted from the throughput. Several VBR sources are used to generate backward traffic. These VBR sources are exponentially distributed ON–OFF sources. During ON periods, the VBR source sends data at 3.2 Mb/s. Unless stated otherwise, the size of each FIFO queue used in routers is 50 packets, the size of data packet is 1 KB, and the sizes of ACKs are 40 and 48 bytes for Vegas and RoVegas, respectively. To ease the comparison, we assume that the sources always have data to send.

6.1. Throughput improvement

Improving the throughput of connection when the congestion occurs in the backward path is one of the design goals of RoVegas. In this subsection, we investigate the throughputs of Vegas and RoVegas in two types of backward congestion. One is the congestion caused by network asymmetry, the other is the congestion caused by additional backward traffic.

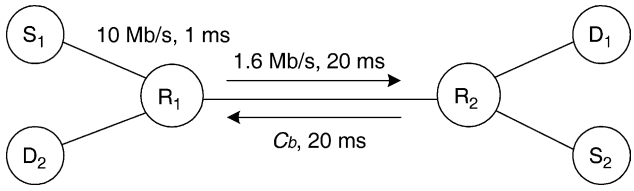


Fig. 6. A single bottleneck network topology for investigating throughputs of Vegas and RoVegas when the congestion occurs on backward path.

The first network topology 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 10 Mb/s and 1 ms for each full-duplex access link, 1.6 Mb/s and 20 ms for the connection link from R_1 to R_2, and C_b and 20 ms for the connection link from R_2 to R_1, respectively. The C_b is set based on the normalized asymmetric factor k. For example, if $k=4$ and the size of data packet and ACK are 1 KB and 40 bytes, respectively, then the C_b is set to 16 Kb/s.

6.1.1. Asymmetric networks

To evaluate the throughputs of Vegas and RoVegas in asymmetric networks, different values of k are used. A source S_1 of either Vegas or RoVegas sends data packet to its destination D_1. The size of each FIFO queue used in routers is 10 packets. Figs. 7 and 8 exhibit the throughput performance of Vegas and RoVegas in asymmetric networks, respectively.

By observing the results shown in Fig. 7, with the increasing value of k from 2 to 32, the throughput of Vegas degrades accordingly. As our analysis depicted in Eq. (16), the throughput of Vegas in this scenario should be u_f/k (data packets per second). Obviously, the simulation results conform to our previous analysis.

Comparing the results of Fig. 8 with that of Fig. 7, we can find that the throughput of RoVegas is much greater

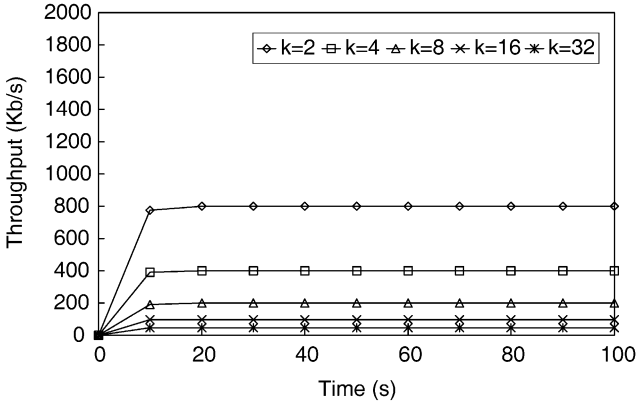


Fig. 7. Throughput of Vegas in asymmetric networks.

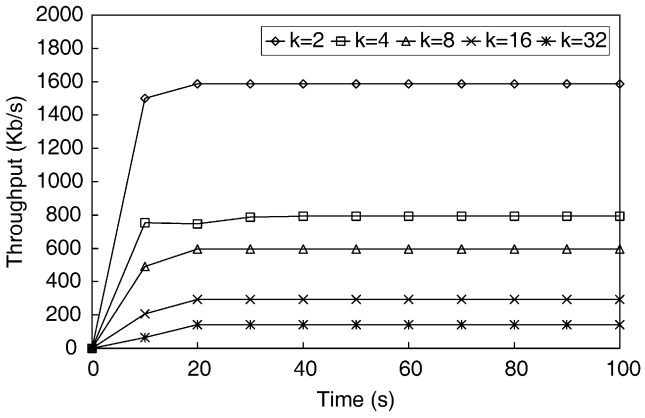


Fig. 8. Throughput of RoVegas in asymmetric networks.

than that of Vegas. With $k=2$, RoVegas maintains a high throughput at 1587.2 Kb/s in which the backward congestion seems not existing. The throughput ratios of RoVegas to Vegas in steady state are about 2 and 3 for $k=2,4$ and $k=8,16,32$, respectively. Notably, all the simulation results shown in Figs. 7 and 8 are consistent with our previous analysis.

6.1.2. Symmetric network with backward traffic

Asymmetric networks should not be the only reason causing backward congestion. Actually, even in a symmetric network the backward congestion may still occur. We use a VBR source with 1.44 Mb/s averaged sending rate to examine the throughputs of Vegas and RoVegas separately in the single bottleneck network as shown in Fig. 6. The capacity of the backward bottleneck, C_b, is set to 1.6 Mb/s. A source of either Vegas or RoVegas is attached to S_1, and a VBR source is attached to S_2. The S_1 starts sending data at 0 s, while S_2 starts at 50 s. Fig. 9 depicts the throughput comparison between Vegas and RoVegas.

As shown in Fig. 9, when the traffic source is Vegas only (0–50 s), it achieves high throughput and stabilizes at 1.6 Mb/s. However, the performance of Vegas degrades dramatically as the VBR source starts sending data. Although the overhead induced by AQT option slightly lower

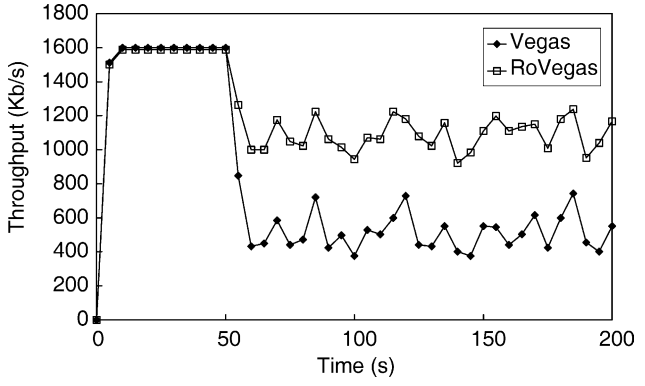


Fig. 9. Throughput comparison between Vegas and RoVegas with the backward traffic load is 0.9 in the single bottleneck network topology.

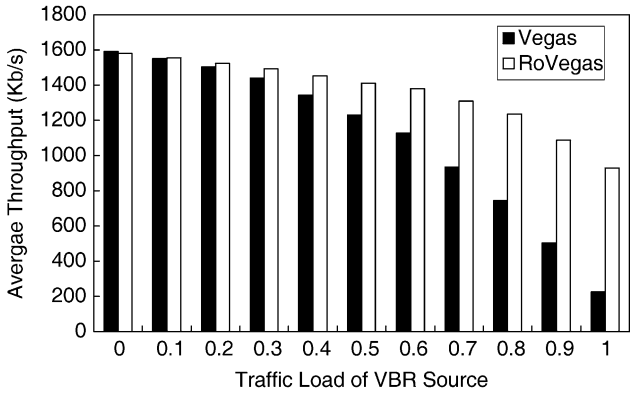


Fig. 10. Average throughput versus different backward traffic loads for Vegas and RoVegas in the single bottleneck network topology.

the throughput of RoVegas (0.8%) during the preceding 50 s, nevertheless, RoVegas maintains a much higher throughput than that of Vegas while the backward congestion occurs. With the inference of the backward VBR traffic, the average throughput of Vegas is 521 Kb/s and RoVegas is 1092 Kb/s. Since we use the same traffic pattern of the VBR source while the throughput of Vegas or RoVegas is examined. Thus there appear some synchronized fluctuations of throughput between Vegas and RoVegas.

To evaluate the average throughputs of Vegas and RoVegas with different backward traffic loads, we set the VBR traffic loads to vary from 0 to 1. The traffic sources are the same as the above descriptions but the sources of either Vegas or RoVegas and VBR start at 0 s. The simulation period is 200 s for each sample point. From the simulation results shown in Fig. 10, we can find that when the backward traffic load is not zero, RoVegas always achieves a higher average throughout than Vegas. For example, as the backward traffic load is 1 RoVegas achieves a 4.1 times higher average throughput in comparison with that of Vegas.

In the parking lot configuration as shown in Fig. 11 we use three VBR sources each with 1.28 Mb/s averaged sending rate to examine the throughputs of Vegas and RoVegas. The bandwidth and propagation delay of each full-duplex access link and connection link are 10 Mb/s, 1 ms and 1.6 Mb/s, 10 ms, respectively. The source of either Vegas or RoVegas are attached to S_1, and three VBRs are attached to S_2–S_4, respectively., The TCP source from either Vegas or RoVegas starts sending data at 0 s, and then three VBR sources from S_2 to S_4 successively enter the network every 100 s.

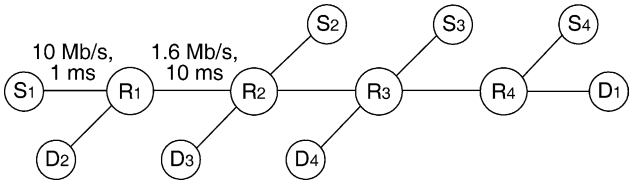


Fig. 11. A parking lot network topology for investigating throughputs of Vegas and RoVegas when the congestion occurs on the backward path.

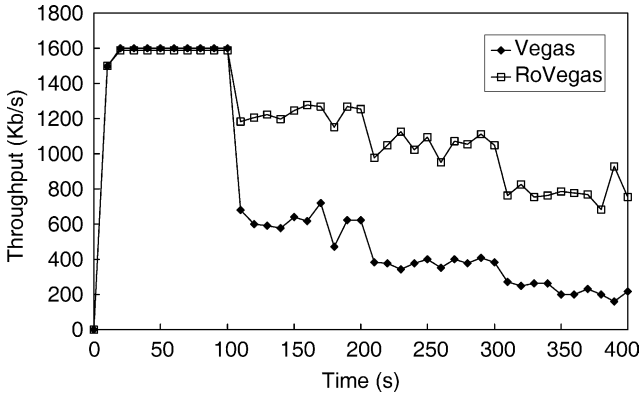


Fig. 12. Throughput comparison between Vegas and RoVegas with the backward traffic load is 0.8 in the parking lot network topology.

From the simulation results presented in Fig. 12 we can observe that when the traffic source is TCP only (0–100 s) both Vegas and RoVegas could fully utilize the bandwidth (due to the overhead induced by AQT option, the throughput of RoVegas is slightly lower than Vegas). However, as the VBR sources successively enter the network, Vegas suffers a serious throughput reduction. Under the same environment, RoVegas features a much better throughput performance compared with that of Vegas. The average throughput ratio of RoVegas to Vegas during 100–200, 200–300, and 300–400 s are 2.00, 2.77, and 3.46, respectively.

The average throughputs of Vegas and RoVegas with different backward traffic loads in the parking lot network are also examined. The traffic sources of either Vegas or RoVegas and three VBRs start at 0 s. The VBR traffic loads, vary from 0 to 1 accordingly. From the simulation results shown in Fig. 13, we can find that as the backward traffic load is not zero, RoVegas always achieves a higher average throughput than Vegas, especially, when the backward traffic load is heavy. For example, as the backward traffic load is 1, the average throughput ratio of RoVegas to Vegas is 14.09.

Obviously, we have demonstrated that RoVegas significantly improves the connection throughput when the backward path is congested.

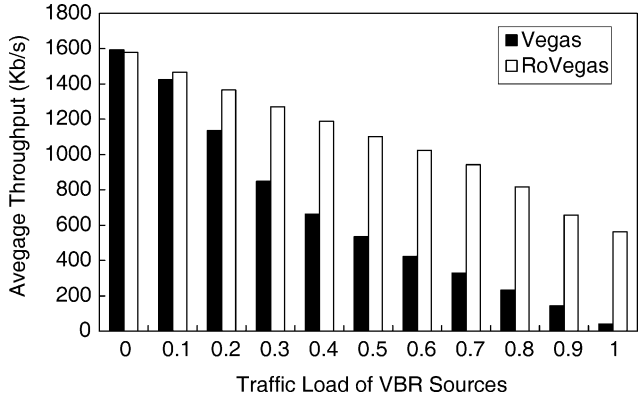


Fig. 13. Average throughput versus different backward traffic loads for Vegas and RoVegas in the parking lot network topology.

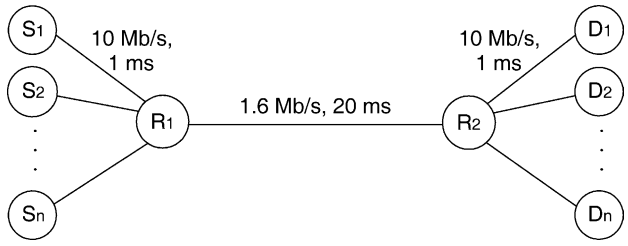


Fig. 14. Network topology for studying the bias and fairness issues of Vegas and RoVegas.

6.2. Persistent congestion

As a connection starts when there exists many other connections, the new connection may experience *RTTs* that are considerably larger than the actual fixed delay along the path. Thus the *BaseRTT* of this new connection will be larger than it should be. Therefore the new connection will put a larger amount of extra data than its expected amount on the network. This bias may possibly drive the system to a persistent congestion.

In this subsection, we study the bias of Vegas through simulation. The simulation network topology is shown as in Fig. 14 in which the bandwidth and propagation delay for each full-duplex link are depicted.

Eight connections of Vegas from S_1 to S_8 successively enter the network every 20 s. The α and β of Vegas are set to 1 and 3, respectively. Thus, the amount of extra data for each connection should be kept between 1 and 3 packets. From the results shown in Fig. 15, we can observe that when the fourth connection of Vegas joins the network, the queue length of the bottleneck increases to 15 packets. This amount of extra data is larger than the expected maximum amount (12 packets). Even worse as the eighth connection goes into the network, the queue length of the bottleneck is 40 packets. That is, averagely each connection of Vegas contributes five packets to the bottleneck. This situation will become worse and worse when more connections enter the network.

In contrast to the Vegas connections, each RoVegas connection keeps a proper amount of extra data in

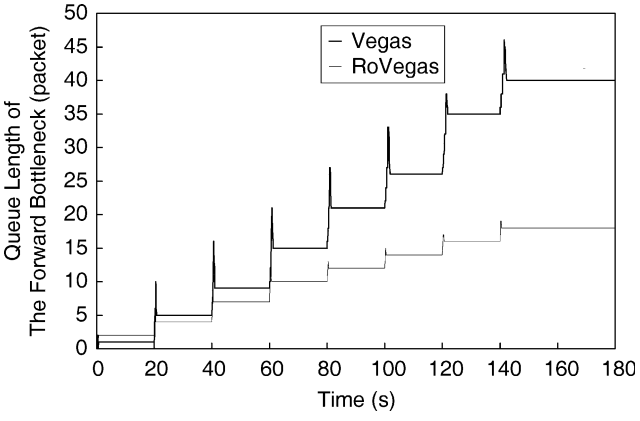


Fig. 15. Queue length of the forward. bottleneck for Vegas and RoVegas.

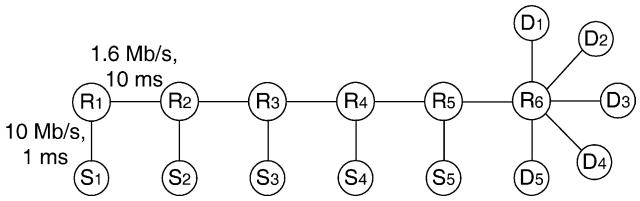


Fig. 16. Network topology for exploring the fairness issue of Vegas and RoVegas, in which the traffic pairs are featured by different propagation delay.

the bottleneck. When the eighth connection of RoVegas joins the network, the queue length of the bottleneck is 18 packets. Since each connection of RoVegas is able to derive the fixed delay along the round-trip path, the bias of Vegas no longer exists in RoVegas.

6.3. Fairness enhancement

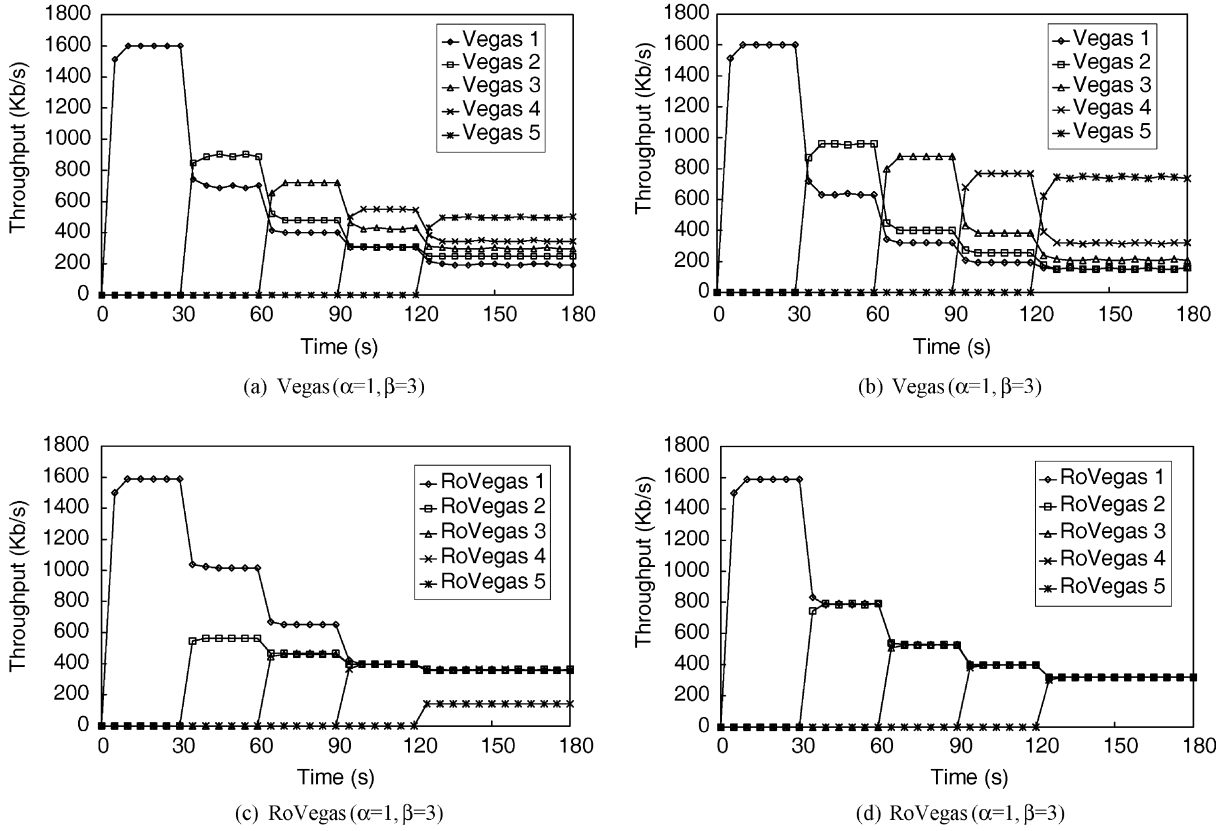
Fairness is another important issue of Vegas. Although Vegas is not biased against the connections with longer *RTT* like Reno does, there is still unfairness occurred in Vegas. In this subsection, we investigate the fairness metric of Vegas and RoVegas. Two network topologies used in the simulations are depicted in Figs. 14 and 16.

The first network topology for the simulation is shown in Fig. 14, in which all traffic pairs features the same propagation delay. Five connections of either Vegas or RoVegas from S_1 to S_5 successively join the network every 30 s. In order to remove the uncertainty induced by the range between α and β, we set α equal to β in two simulation scenarios. Fig. 17 shows the results of simulations.

From the simulation results of Vegas presented in Fig. 17(a) and (b) we can see that no matter the values of α and β equal or not, connections are unable to share the bandwidth fairly. According to our previous discussion, there are two criteria for achieving the fairness among the competitive connections of Vegas. One is that the measured *BaseRTT* must be precise enough. The other is that the uncertainty induced by the range between α and β must be removed. Connections in Fig. 17(a) do not meet both criteria. Connections in Fig. 17(b) do not conform to the first criterion. Therefore, both connections in these two figures are unable to fairly share the bandwidth of the bottleneck.

Observing the results of RoVegas shown in Fig. 17(c) and (d). Since connections in Fig. 17(c) do not meet the second criterion of fairness, the throughputs of these connections are not identical. Finally, connections in Fig. 17(d) meet both criteria, hence, all the connections share the bandwidth fairly.

The second network topology for exploring the fairness issue of Vegas and RoVegas is depicted in Fig. 16, in which traffic pairs are featured by different propagation delays. The bandwidth and propagation delay of each full-duplex access link and connection link are 10 Mb/s, 1 ms and 1.6 Mb/s, 10 ms, respectively.



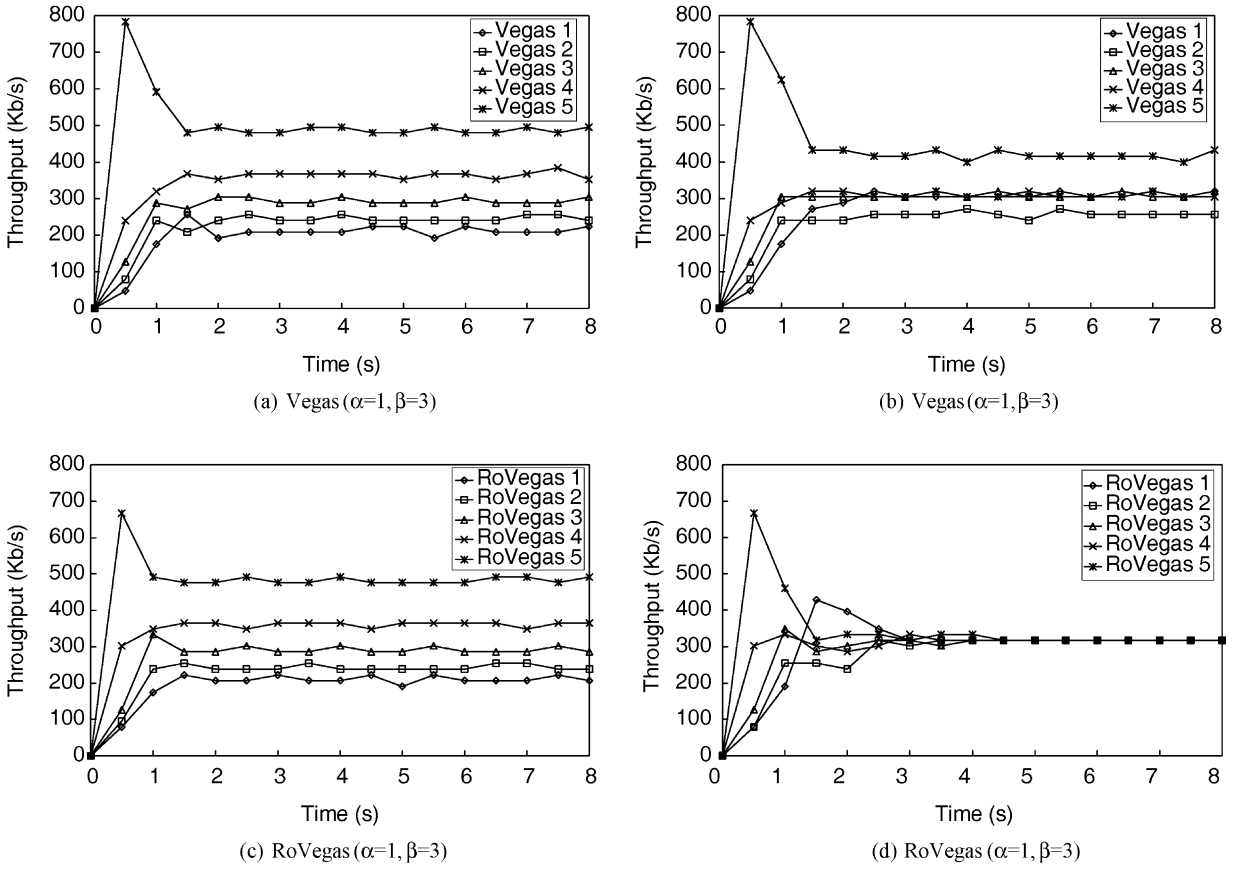
(a) Vegas (α=1, β=3)

(b) Vegas (α=1, β=3)

(c) RoVegas (α=1, β=3)

(d) RoVegas (α=1, β=3)

Fig. 17. Fairness investigation of Vegas and RoVegas in which connections with the same propagation delay and successively enter the network every 30 s.



(a) Vegas (α=1, β=3) (b) Vegas (α=1, β=3)

(c) RoVegas (α=1, β=3) (d) RoVegas (α=1, β=3)

Fig. 18. Fairness investigation of Vegas and RoVegas in which connections with different propagation delay and enter the network at the same time.

Five connections of either Vegas or Rovegas from S_1 to S_5 start at the same time. Same as the previous simulations, we set α equal to β in two simulation scenarios to remove the uncertainty induced by the range between α and β. Fig. 18 represents the results of simulations.

Since the Vegas connections in Fig. 18(a) do not meet both criteria of fairness and those in Fig. 18(b) do not conform to the first criterion. Therefore, the Vegas connections could not share the bandwidth fairly with each other no matter the values of α and β are equal or not. The RoVegas connections in Fig. 18(c) do not obey the second criterion, thus the bandwidth sharing of bottleneck is unfair. However, the RoVegas connections in Fig. 18(d) comply with both criteria, and hence the fairness among the competitive connections is indeed achieved.

6.4. Gradual deployment

It cannot be expected that all routers on the Internet are AQT-enabled while the AQT option is a newly defined IP option. To consider the gradual deployment issue, for each ACK received by a RoVegas source, the *BaseRTT* should be measured as the following pseudocodes:

if (the ACK is a probing packet)
  $BaseRTT_{temp} = RTT$–(AQT + AQT-Echo)
  /* where *RTT* is the newly measured round-trip time */
  if ($BaseRTT_{temp} < BaseRTT$)
    $BaseRTT = BaseRTT_{temp}$
else /* the ACK is not a probing packet) */
  if ($RTT < BaseRTT$)
    $BaseRTT = RTT$

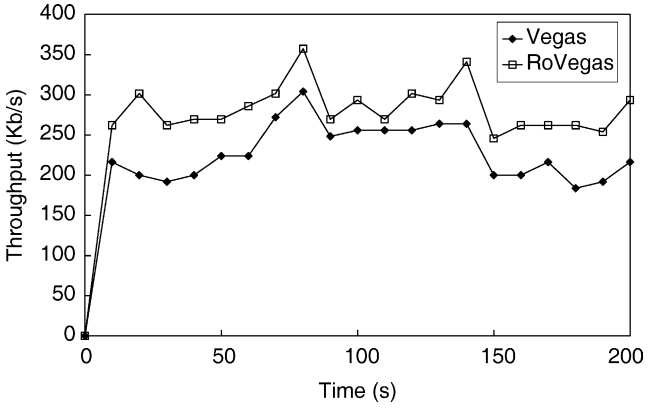


Fig. 19. Throughput comparison between Vegas and RoVegas for only R2 is AQT-enabled in the parking lot network.

If all bottleneck routers along the round-trip path are not AQT-enabled, RoVegas may behave like Vegas. Since RoVegas cannot obtain the backward queuing time (QT_b) to reduce the impacts of backward congestion, and may not estimate a precise *BaseRTT* to enhance the fairness and solve the persistent congestion. However, we try to explore whether a single AQT-enabled router on the end-to-end path may achieve the benefits from the RoVegas mechanism.

A parking lot network as shown in Fig. 11 is used to examine the throughputs of Vegas and RoVegas separately, here only R_2 is AQT-enabled. Three VBR sources each with 1.28 Mb/s averaged sending to generate backward traffic. A source of either Vegas or RoVegas from S_1, and three VBRs from S_2 to S_4 start sending data at 0 s. Despite only one AQT-enabled router R_2 located on the routing path, we can find that RoVegas still maintains a higher throughput than that of Vegas, as depicted in Fig. 19. The simulation results imply that the proposed mechanism is amenable to gradual deployment for reducing the impacts of backward congestion. This feature may encourage the gradual adoption of RoVegas on the Internet.

7. Conclusion

In this research, we propose a router-based congestion avoidance mechanism, RoVegas, for TCP Vegas. Comparing with other previous studies, RoVegas provides a more effective way to solve the problems of rerouting and persistent congestion, to enhance the fairness among the competitive connections, and to improve the throughput when congestion occurs on the backward path. Through both analysis and simulation, the effectiveness of RoVegas has been shown. Nevertheless, there is still an issue that needs more attentions. It is enhancing the throughput of Vegas when it performs with Reno head-to-head. Therefore, how to enable compatibility between Reno and Vegas would be our future work.

References

[1] V. Jacobson, Modified TCP congestion avoidance algorithm, Techical Report, April 1990.
[2] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, , TCP Vegas: new techniques for congestion detection and avoidance, ACM SIGCOMM'94 August (1994) 24–35.
[3] L.S. Brakmo, L.L. Peterson, Vegas:, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE J. Select. Areas Commun. 13 (1995) 1465–1480.
[4] J.C. Hoe, Start-up dynamics of TCP's congestion control and avoidance schemes, Master's Thesis, MIT, June 1995.
[5] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP selective acknowledgement options, Internet Draft, April 1996.
[6] W. Feng, P. Tinnakornsrisuphap, The failure of TCP in high-performance computational grids, SC 2000, High-performance Networking and Computing Conference, November 2000.
[7] A. Veres, M. Boda, The chaotic nature of TCP congestion control, IEEE INFOCOM'2000 March (2000) 1715–1723.
[8] J.S. Ahn, P.B. Danzig, Z. Liu, L. Yan, Evaluation of TCP Vegas: emulation and experiment, ACM SIGCOMM'95 March (1995) 185–195.
[9] J. Mo, R.J. La, V. Anantharam, J. Walrand, Analysis and comparison of TCP Reno and Vegas, IEEE INFOCOM'99 March (1999) 1556–1563.
[10] G. Hasegawa, M. Murata, H. Miyahara, Fairness and stability of congestion control mechanism of TCP, Telecommun. Syst. J. November (2000) 167–184.
[11] C. Boutremans, J.L. Boudec, A note on the fairness of TCP Vegas, 2000, International Zurich Seminar on Broadband Communications, February 2000 pp. 163–170.
[12] D.D. Luong, J. Biro, On the proportional fairness of TCP Vegas, IEEE GLOBECOM'01 November (2001) 1718–1722.
[13] O. Elloumi, H. Afifi, M. Hamdi, Improving congestion avoidance algorithms for asymmetric networks, IEEE Int. Conf. Commun. June (1997) 1417–1421.
[14] C.P. Fu, S.C. Liew, A remedy for performance degradation of TCP Vegas in asymmetric networks, IEEE Commun. Lett. 7 (2003) 42–44.
[15] C.P. Fu, L.C. Chung, S.C. Liew, Performance degradation of TCP Vegas in asymmetric networks and its remedies, IEEE Int. Conf. Commun. June (2001) 3229–3236.
[16] Y. Lai, Improving the performance of TCP Vegas in heterogeneous environment, Int. Conf. Parallel Distrib. Syst. June (2001) 581–587.
[17] W. Feng, S. Vanichpun, Enabling compatibility between TCP Reno and TCP Vegas, IEEE Symp. Appl. Internet January (2003) 301–308.
[18] K. Takagaki, H. Ohsaki, M. Murata, Analysis of a window-based flow control mechanism based on TCP Vegas in heterogeneous network environment, IEICE Trans. Commun. E85-B (1) (2002) 89–97.
[19] C. Samios, M.K. Vernon, Modeling the throughput of TCP Vegas, ACM SIGMETRICS'03 June (2003) 71–81.
[20] J. Postel, Internet Protocol, RFC791, September 1981.
[21] T.V. Lakshman, U. Madhow, B. Suter, Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance, IEEE INFOCOM'97 April (1997) 1199–1209.
[22] V. Jacobson, Congestion avoidance and control, ACM SIGCOMM'88 August (1988) 314–329.
[23] U. Hengartner, J. Bolliger, Th. Gross, TCP Vegas revisited, IEEE INFOCOM'2000 March (2000) 1546–1555.
[24] W.R. Stevens, TCP/IP Illustrated, The Protocols, vol. 1, Addison-Wesley, New York, 1994.
[25] UCB/LBNL/VINT Network Simulator—ns (version 2), http://www.isi.edu/nsnam/ns/