Design and performance evaluation of an improved TCP congestion avoidance scheme

Y.-C. Chan, C.-T. Chan and Y.-C. Chen

Abstract: TCP Vegas detects network congestion in its early stage and successfully prevents the periodic packet loss that usually occurs in traditional schemes. It has been demonstrated that TCP Vegas achieves a much higher throughput than TCP Reno. However, TCP Vegas cannot prevent unnecessary throughput degradation when congestion occurs on the backward path. A router-based congestion avoidance scheme for TCP Vegas is proposed. By distinguishing whether or not congestion occurs in the forward path, it significantly improves the connection throughput when the backward path is congested.

1 Introduction

The continuing growth of Internet traffic raises the important issue of how to efficiently utilise network resources. The transmission control protocol (TCP) is currently the most popular end-to-end transport protocol on the Internet, and it is implemented in several versions (i.e. Tahoe, Reno, Vegas…) all of which aim to improve the network utilisation. Among these TCP versions. Vegas can achieve a much higher throughput than that of the other versions [1].

TCP Vegas attempts to control and avoid congestion by monitoring the difference between the measured and expected throughputs. 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 between the lower threshold (α) and the upper threshold (β). By adjusting the source congestion window size, an appropriate amount of extra data is kept in the network to avoid congestion whilst maintaining a high throughput. However, a roughly measured *RTT* may lead to an incorrect change in of the 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 decrease in 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 or not congestion occurs in the forward path and adjust the congestion window size precisely.

Some current networking technologies with asymmetry network characteristics, such as asymmetric digital subscriber line (ADSL), cable modem, and satellite-based networks, greatly increase the possibility of backward path congestion. These technologies often have vastly different bandwidths in the two directions of the link. The connections of both TCP Reno and TCP Vegas can suffer severe performance degradation when backward path congestion occurs, especially for TCP Vegas [2]. Therefore, how to improve this deficiency in TCP Vegas now is an important issue.

We now propose a router-based congestion avoidance scheme for TCP Vegas (abbreviated as RoVegas hereafter). Using the proposed scheme in the routers along the round-trip path, a RoVegas source may obtain the queueing delay time on the forward and backward path separately. By judging the direction along which the congestion occurs, RoVegas significantly reduces the impact and improves the throughput when the backward path is congested. Furthermore, the simulation results reveal that RoVegas is compatible with Vegas as well as amenable to gradual deployment.

IEE Proceedings online no. 20040229

doi:10.1049/ip-com:20040229

Paper first received 13th February and in revised form 9th October 2003. Online publishing date: 4 February 2004

Y.-C. Chan and Y.-C. Chen are with the Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan, Republic of China

C.-T. Chan is with the Telecommunication Laboratories, Chunghwa Telecom Company Ltd., Taipei 106, Taiwan, Republic of China

2 Previous work

Various schemes have been proposed to improve the performance of TCP Reno for asymmetric networks [3, 4]. However, these mechanisms are not effective for handling the Vegas' asymmetry problems [2].

ACK filtering (AF) is a gateway-based technique [3]. A gateway identifies and maintains the states of individual TCP connections. Advantage is taken of the fact that ACK packets are cumulative. It attempts to remove redundant ACK packets in the backward buffer to lighten the congestion.

The key idea of ACK congestion control (ACC) is that it extends congestion control to TCP ACK packets [3]. ACC uses a gateway on the backward path to aid congestion control. It tries to detect impending congestion by tracking the average queue size in the recent past and then informs the destination to dynamically decrease the frequency of the ACK packets. Thus, each ACK packet effectively acknowledges several packets.

Essentially, AF and ACC are designed for TCP Reno which provides a window-based congestion control mechanism. TCP Vegas, on the other hand, employs a rate-based congestion avoidance technique. Obviously, these two approaches are not effective for handling the asymmetry problems of TCP Vegas.

Elloumi *et al.* [5] proposed a modified algorithm for TCP Vegas. It divides a RTT into a forward trip time and a backward trip time in order to remove the effects of backward path congestion. Nevertheless, it seems unlikely to work without clock synchronisation.

Fu and Liew [6] employ an end-to-end method to measure the actual flow rate on the forward path at a source of TCP Vegas. The source adjusts the congestion window size depending on the differences between the expected rate and the actual flow rate on the forward path. However, the TCP traffic has a bursty nature that makes it difficult to decide when to measure the actual flow rate. Moreover, the actual flow rate measured by the source is often greater than the expected rate. This leads to the congestion window size being over-increased, and therefore causing congestion in the forward path.

3 TCP RoVegas

Different from Tahoe and Reno, which detect network congestion based on packet losses, TCP Vegas estimates a suitable 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 every is measured *RTT*. The amount of extra data is between two thresholds α and β as shown in the following:

$$\alpha \leq (expected - actual) \times baseRTT \leq \beta \qquad (1)$$

where *expected*, the expected throughput, is the current congestion window size divided by *baseRTT*, and, *actual* is the throughput which can be represented by the current congestion window size divided by the newly measured value of *RTT*.

When backward congestion occurs, the increased backward queueing time will affect, the actual throughput and enlarge the difference between the expected throughput and the actual throughput. This results in a decrease in the congestion window size. Since the network resources in the backward path should not affect the traffic in the forward path, it is not necessary to reduce the congestion window size when only backward congestion occurs.

A measured *RTT* can be divided into four parts: (i) the forward fixed delay (i.e. propagation delay and packet processing time); (ii) forward queueing time; (iii) the backward fixed delay; and (iv) the backward queueing time. To utilise the network bandwidth efficiently, we redefine the actual throughput as:

$$actual' = \frac{CWND}{RTT - QT_b} \qquad (2)$$

where *RTT* is the newly measured round-trip time. QT_b is the backward queueing time and *CWND* is the current congestion window size. Consequent, *actual'* is the throughput that can be achieved if there is no backward queueing delay along the path.

To realise our scheme, we define a new IP option named AQT (accumulated queueing time) to collect the queueing time along the path. According to the general format of IP options described in [7], the fields of an AQT option are created as in Fig. 1. The option type and option length fields indicate the type and length of this IP option. The AQT field expresses the accumulated queueing time that a packet experienced along the route path. The AQT-echo field echoes the accumulated queueing time value of an AQT option that was sent by the remote TCP.

A probing packet is a normal TCP packet (data or ACK) with an 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 queueing delay of a received probing packet to the AQT field. The queueing time is computed based on the queueing disciplines. The details regarding how to compute the queueing time of each received probing packet in various queueing disciplines is beyond the scope of this work.

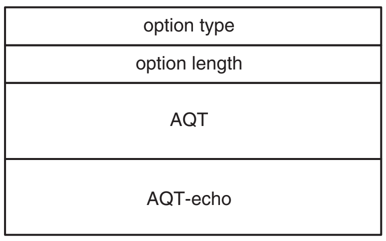

Fig. 1 *Fields of an AQT option*

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. Using the AQT-enabled routers along the round-trip path, a RoVegas source is able to obtain both the forward queueing time (the value of the AQT-echo field) and backward queueing time (the value of the AQT field) from the received probing packet. For each ACK packet received by a RoVegas source, the *baseRTT* can be measured based on the following pseudo-code:

```
if (the ACK is a probing packet)
  $baseRTT_{temp} = RTT - (AQT + AQT\text{-}echo)$
  /* 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 ($RTT < baseRTT$)
    $baseRTT = RTT$
```

The following router-based congestion avoidance mechanism is described to avoid any unnecessary reduction of the congestion window size:

- Derive *expected* (the expected throughput) defined as the current congestion window size divided by *baseRTT*.
- Calculate *actual'* as the current congestion window size divided by the difference between the newly measured *RTT* and backward queueing time.
- Let $diff = (expected - actual') \times baseRTT$.
- 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:

$$w_{next} = \begin{cases} w_{cur} + 1 & \text{if } diff < \alpha \\ w_{cur} - 1 & \text{if } diff > \beta \\ w_{cur} & \text{if } \alpha \leq diff \leq \beta \end{cases} \qquad (3)$$

We now demonstrate that the proposed scheme can improve the throughput of TCP Vegas using the performance evaluation results presented in the following Section.

4 Performance evaluation

We perform the simulations using the network simulator ns-2.1b9a [8] to compare the throughput between Vegas and proposed RoVegas. Two network topologies have been created as shown in Figs. 2 and 3 for our performance evaluations. The bandwidth and propagation delay of each full duplex link are depicted in the Figures. For an access link (i.e. a link between the host and a router), the bandwidth and propagation delay are 2 Mbit/s and 10 ms. For a connection link (i.e. a link between two routers), those are 1 Mbit/s and 20 ms. Sources, destinations and routers are expressed as S_i, D_i and R_i respectively. A source and a destination with the same suffix value represent a traffic pair. For example, S_1 sends packets to D_1, S_2 sends packets to D_2, and so on. The FIFO (first-in–first-out) service discipline is assumed and the size of each FIFO queue used in routers is 50 packets.

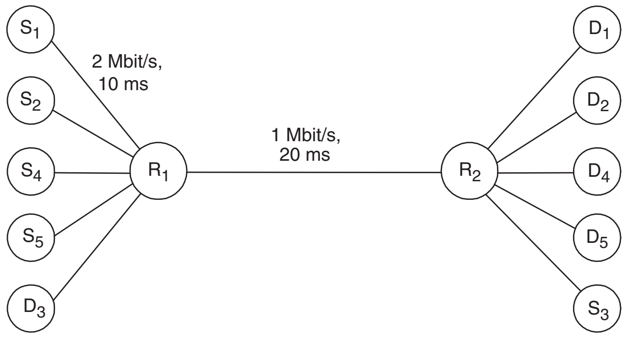

Fig. 2 *Single bottleneck network topology*

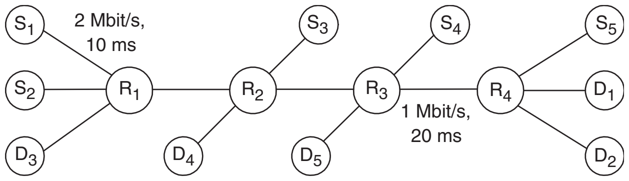

Fig. 3 *Parking lot network topology*

Several variable-bit-rate (VBR) sources are used to generate the backward traffic. These VBR sources are exponentially distributed ON–OFF sources. The average ON period is 10 ms. During ON periods, the VBR sources send data at 2 Mbit/s. All parameters of both Vegas and RoVegas are the same. Here $\alpha = 1$, $\beta = 3$, and without loss of generality, the data packet size is set at 1 kbyte. To ease the comparison, we assume that the sources always have data to send.

4.1 Throughput improvement

We use a VBR source with a 900 kbit/s averaged sending rate to examine the throughput of Vegas and RoVegas separately in the single bottleneck network as shown in Fig. 2. The Vegas, RoVegas and VBR sources are attached to S_1, S_2 and S_3 respectively. A source, either Vegas or RoVegas, starts sending data at 0 s, while the VBR source starts at 50 s.

The results in Fig. 4 show that when the traffic source is Vegas only, it achieves a high throughput that stabilises at 1000 kbit/s until the VBR source starts sending data. However, the performance of Vegas degrades dramatically when the VBR traffic starts. On the other hand, RoVegas maintains a much higher throughput than that of Vegas. During the active period of the VBR source, the average throughput of Vegas is 348 kbit/s whereas that of RoVegas is 776 kbit/s. Note that the traffic pattern of the VBR source is kept constant when work the throughput of Vegas or RoVegas is examined. Thus, there appears to be some

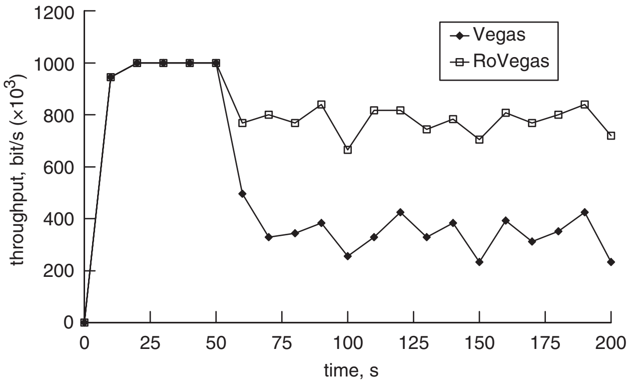

Fig. 4 *Throughput comparison between Vegas and RoVegas when the backward traffic load is 0.9 in the single bottleneck network topology*

synchronised fluctuations of throughput between Vegas and RoVegas.

To evaluate the average throughput of Vegas and RoVegas with different backward traffic loads, we set the VBR traffic loads to vary from zero to one. 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. 5, we can find that the RoVegas obtains a much higher average throughput than TCP Vegas, especially when the backward traffic load is heavy. For example, when the backward traffic load is one. RoVegas achieves a 6.75 times higher average throughput than that of Vegas.

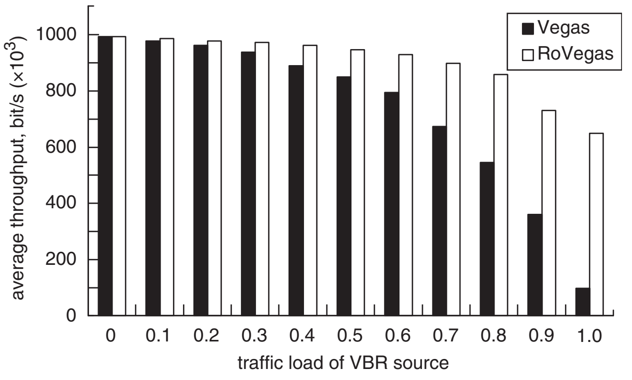

Fig. 5 *Average throughput as a function of backward traffic load for Vegas and RoVegas in the single bottleneck network topology*

In the parking lot network as shown in Fig. 3, we use three VBR sources each with a 800 kbit/s averaged sending rate to separately examine the throughput of Vegas and RoVegas. The Vegas, RoVegas and the three VBR sources send packets from S_1, S_2, and S_3–S_5 respectively. The TCP source, from either Vegas or RoVegas starts sending data at 0 whereas the three VBR sources start at 50 s. From the simulation results presented in Fig. 6, we can see that RoVegas maintains a much higher throughput than that of Vegas when backward congestion occurs.

The average throughput 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 the three VBR sources start at 0 s. The VBR traffic loads vary from zero to one accordingly. From the simulation results shown in Fig. 7, RoVegas acquires a much higher average throughput than TCP Vegas, especially when the backward traffic load is heavy.

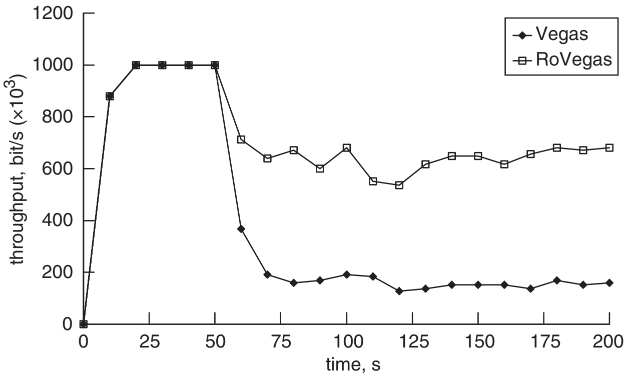

Fig. 6 *Throughput comparison between Vegas and RoVegas when the backward traffic load is 0.8 in the parking lot network topology*

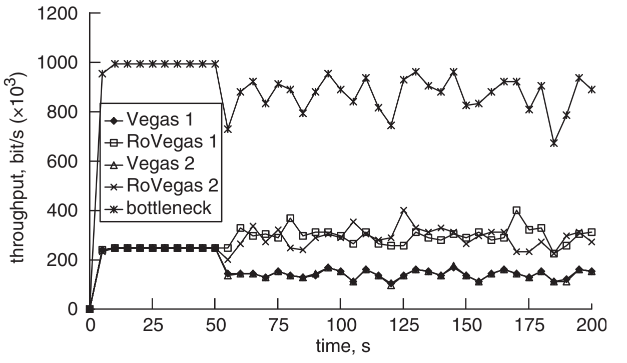

Fig. 8 *Compatibility test for two Vegas and two RoVegas connections when the backward traffic load is 0.9 in the single bottleneck network topology*

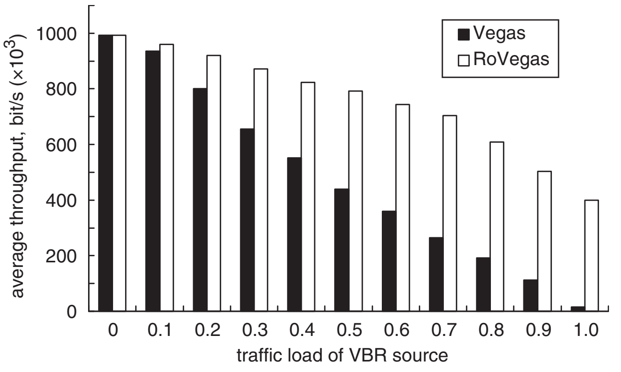

Fig. 7 *Average throughput as a function of backward traffic load for Vegas and RoVegas in the parking lot network topology*

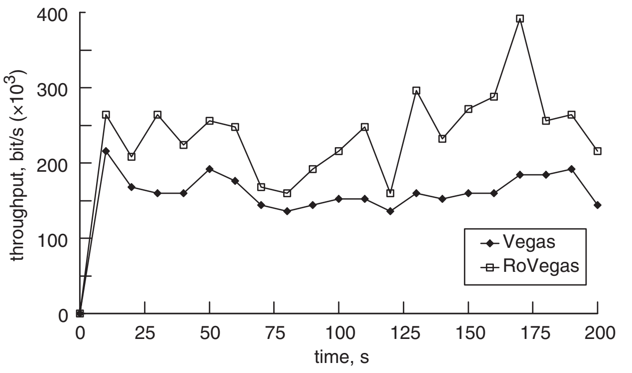

Fig. 9 *Gradual deployment test when only R_2 is AQT-enabled in the parking lot network topology*

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

4.2 Compatibility

In this Section, we investigate the RoVegas' compatibility to decide whether or not RoVegas is compatible with Vegas. By using the single bottleneck network, two Vegas sources are attached to S_1 and S_4, two RoVegas sources are attached to S_2 and S_5, and one VBR source with a 900 kbit/s averaged sending rate is attached to S_3. All the Vegas and RoVegas sources start sending data at 0 s, whereas the VBR source starts at 50 s.

By observing the result in Fig. 8, all the Vegas and RoVegas sources share the bandwidth of the bottleneck link fairly until the VBR source starts sending data. From the time point at 50 s, the throughput of the sources splits into two groups. The one with RoVegas sources achieves a much higher throughput than the one with Vegas sources. In the meantime, there is still some bandwidth left on the bottleneck link. During the active period of the VBR source, the average throughput of S_1, S_4 and S_2, S_5 are 140.5, 140.3 and 296.5, 289.6 kbit/s respectively. Through this simulation, we can claim that the Vegas and RoVegas sources are compatible with each other. However, RoVegas achieves a much higher throughput than that of Vegas when the backward path is congested.

4.3 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. In this simulation, we try to explore whether a single AQT-enabled router on the end-to-end path can achieve benefits from the RoVegas mechanism.

The parking lot network is used to examine the throughput of Vegas and RoVegas separately, and here only R_2 is AQT-enabled. Three VBR sources each with a 800 kbit/s averaged sending rate are used to generate backward traffic. The Vegas, RoVegas and three VBR source are attached to S_1, S_2 and S_3–S_5 respectively. A source, either Vegas or RoVegas starts sending data at 0 s, while the three VBR sources start at 1 s.

From the simulation results depicted in Fig. 9, we can find that despite only one AQT-enabled router R_2 is being located on the routing path, RoVegas still maintains a higher throughput than that of Vegas. The simulation results imply that the proposed mechanism is amenable to gradual deployment to reduce the impact of backward congestion. This feature may encourage the gradual adoption of RoVegas on the Internet.

5 Conclusions

A router-based congestion avoidance scheme for TCP Vegas has been proposed. In comparison to other schemes [3–6] RoVegas provides a more realistic and effective way to improve the connection throughput of TCP Vegas when the backward path is congested. Nevertheless, there is still some bandwidth left on the forward path when the backward congestion occurs. The question of how to advance the utilisation of the forward path in such a situation will be the subject of our further work.

6 References

1 Brakmo, L., and Peterson, L.: 'TCP Vegas: End-to-end congestion avoidance on a global Internet', *IEEE J. Sel. Areas Commun.*, 1995, **13**, (8), pp. 1465–1480

2 Fu, C., Chung, L., and Liew, S.: 'Performance degradation of TCP Vegas in asymmetric networks and its remedies'. Proc. IEEE Int. Conf. on Communications ICC'01, Helsinki, Finland, June 2001, pp. 3229–3236
3 Balakrishnan, H., Padmanabhan, V., and Katz, R.: 'The effects of asymmetry on TCP performance'. Proc. ACM MobiCom'97, Budapest, Hungary, Sept. 1997, pp. 77–89
4 Balakrishnan, H., and Padmanabhan, V.: 'How network asymmetry affects TCP', *IEEE Commun. Mag.*, 2001, **39**, (4), pp. 60–67
5 Elloumi, O., Afifi, H., and Hamdi, M.: 'Improving congestion avoidance algorithms for asymmetric networks'. Proc. IEEE Int. Conf. on Communications ICC'97, Montreal, Canada, June 1997, pp. 1417–1421
6 Fu, C., and Liew, S.: 'A remedy for performance degradation of TCP Vegas in asymmetric networks', *IEEE Commun. Lett.*, 2003, **7**, (1), pp. 42–44
7 Postel, J.: 'Internet Protocol' RFC791, Sept. 1981
8 http://www.isi.edu/nsnam/ns/