ELSEVIER

Computer Communications 23 (2000) 1537–1547

computer communications

www.elsevier.com/locate/comcom

An end-to-end flow control approach based on round trip time

Jin-Ru Chen, Yaw-Chung Chen*, Chun-Liang Lee

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan

Received 20 August 1999; revised 23 March 2000; accepted 23 March 2000

Abstract

Traditional TCP flow control schemes, such as Tahoe and Reno, continuously increase the window size until possible packet loss is detected. These schemes generate congestion unnecessarily by themselves and utilize the buffer space inefficiently. Therefore, Vegas was proposed to accommodate the drawbacks. It uses the measured round-trip time (RTT) to obtain the information required for the flow control. In this work, we propose a flow control scheme, RTT-based TCP, which uses the RTT directly to control the window size adjustment. RTT-based TCP comprises four mechanisms: the modified exponential window increment, the RTT measurement, the queue occupancy estimation, and the quick queue-occupancy reduction. Simulation results show that RTT-based TCP outperforms Vegas in the aspects of throughput, fairness and efficiency of buffer usage. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: TCP; Congestion avoidance; Fairness

1. Introduction

The TCP flow control mechanisms have evolved for more than two decades. From the observation of the evolution, we found that the packet retransmission and the congestion avoidance are two major mechanisms for TCP to provide reliable and efficient data transport. Therefore, a variety of approaches have been proposed to deal with these two issues to improve TCP performance. FACK [13] and SACK [14] improve the TCP retransmission mechanism to reduce the bandwidth waste when the packet loss occurs. Other TCP studies [1,5,18] deal with the congestion avoidance as does this work.

The earliest window-based congestion control improvement was implemented on TCP Tahoe, which maintained a proper window size to control the number of outstanding packets. Both probing and allocation of the available bandwidth were accomplished through increasing the window size gradually until packet loss occurs. Most of the window size adjustment schemes that improve the TCP performance are based on this approach, and its main advantage is the simplicity in implementation. TCP Reno improves the throughput of TCP Tahoe by introducing the fast recovery algorithm [16], which sets the window size to a threshold value instead of resetting it after a segment loss.

Both TCP Tahoe and Reno detect the available bandwidth by self-generated congestion and packet losses. Although their window adjustment schemes avoid the occurrence of further congestion, both schemes are unable to fully utilize the available bandwidth. To remedy the periodic congestion, other schemes have been proposed. When queue occupancies in intermediate nodes increase, the round-trip times (RTTs) of the successfully acknowledged packets increase as well. DUAL algorithm [6] is based on reacting to such RTT increment. In Ref. [12], Jain proposed an approach based on an analytic derivation of an optimum window size for a deterministic network. Wang and Crowcroft proposed another scheme called Tri-S [17], which takes advantage of the fact that when the network approaches congestion, the observed increment in throughput caused by the window size increment will be flattened.

TCP Vegas [1] is the most known congestion avoidance scheme, which achieves much better throughput than Reno. It is similar to Tri-S but uses a different way to calculate throughputs. Vegas features three improvements, early timeout detection, window size adjustment and safe slow-start. The latter two can be further enhanced to increase the throughput, to improve the fairness and to use the buffer more efficiently. In this article, we propose a flow control scheme, RTT-based TCP, which uses the TCP optional RTTM field [7] and the RTT threshold to decide how to adjust the window. In our proposed scheme, safe slow-start proposed in Vegas is no longer used, so that the response time can be reduced when congestion does

* Corresponding author. Tel.: +886-3-573-1864; fax: +886-3-5727-842.
E-mail addresses: jzchen@csie.nctu.edu.tw (J.-R. Chen), ycchen@csie.nctu.edu.tw (Y.-C. Chen), leecl@csie.nctu.edu.tw (C.-L. Lee).

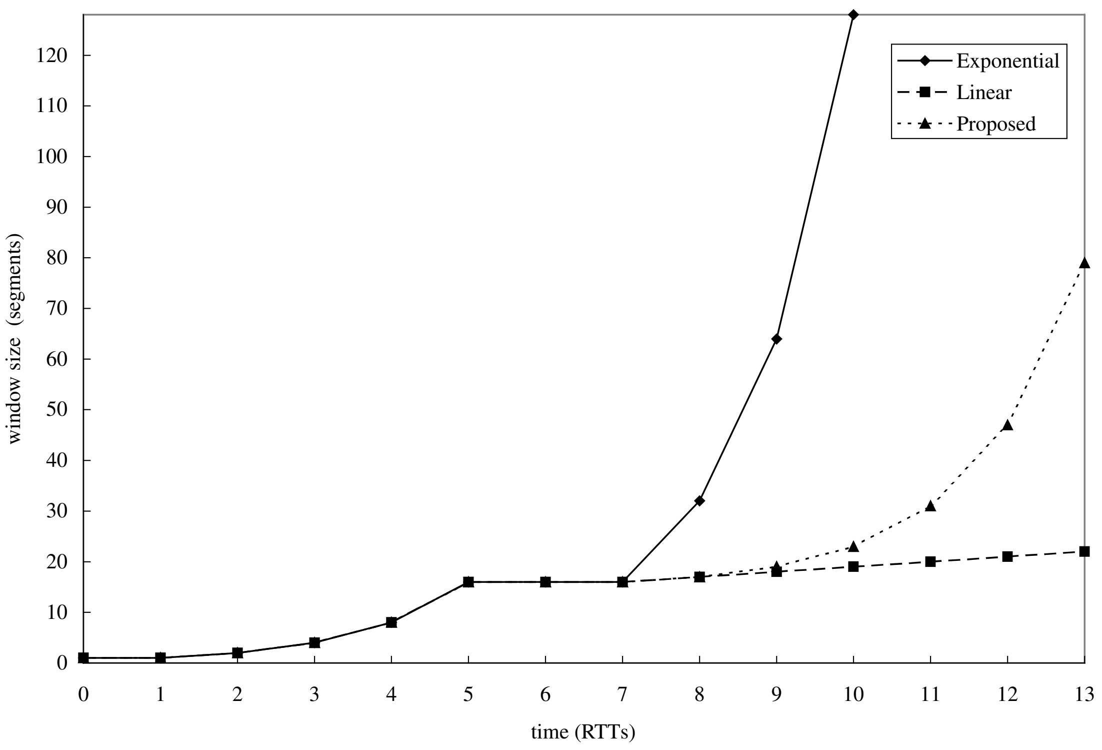

Fig. 1. Comparison of different window increment approaches.

occur. Furthermore, the fair bandwidth sharing at the bottleneck switch can be achieved by keeping equal queue occupancy for each connection.

Another class of congestion avoidance scheme, the rate-based scheme [2,3,8], uses a detection mechanism to estimate the allowed network service rate; in addition, it controls the queue occupancy to avoid the congestion. This is similar to some window-based congestion avoidance schemes in which the dependency between the RTT and the window size is accounted, so that periodic congestion can be avoided. Generally speaking, rate-based congestion avoidance schemes not only avoid the periodic congestion, but also improve throughput and fairness. However, the performance of existing schemes is not satisfactory enough. Therefore, we try to use the probing approach and the queue occupancy control to enhance the window-based congestion avoidance scheme. This may achieve better performance in throughput, fairness and efficiency of buffer usage.

The rest of this paper is organized as follows. Section 2 addresses the aforementioned four mechanisms employed by RTT-based TCP. Section 3 presents the implementation approach. The simulation results are discussed in Section 4. Section 5 concludes the work and discusses future research directions for this work.

2. Proposed RTT-based TCP

Our proposed approach consists of four mechanisms, which are designed to increase the throughput, to reduce the bandwidth waste, to achieve fairness and to improve the buffer usage. They are described as follows:

1. *Modified exponential window increment:* in most of existing studies, the window size is increased exponentially during the slow-start phase, and increased linearly otherwise. However, the window increment strategy outside the slow-start phase may be more aggressive without incurring congestion. Our mechanism modifies the exponential increment scheme to increase the window size.
2. *RTT-based window adjustment:* this mechanism adjusts the window size based on the RTT instead of the traditional window threshold. Without increasing the implementation complexity, this mechanism features faster and safer window adjustment than the existing schemes (e.g. TCP Vegas).
3. *Queue occupancy estimation:* it enables a source to estimate the queue occupancy in the network. With the precise estimation and the stable control of queue occupancy, better fairness can be achieved.
4. *Quick queue occupancy reduction:* to avoid the waste of buffer space usage, this mechanism quickly reduces the queue occupancy when the queue occupancy is too large.

The details of these mechanisms and the relationship between them will be discussed in subsequent subsections.

2.1. Modified exponential window increment

The operation in traditional TCP window increment schemes consists of two phases: the slow-start and the congestion avoidance. The window size of a source increases exponentially every RTT in the slow-start phase, and linearly in the congestion avoidance phase. The purpose of the former is to approach the network service rate quickly, and that of the latter is to reach the network service rate safely. The timing for switching between these two phases varies with different schemes. Their common objective is to increase the source window size as fast as possible without self-generated congestion.

In our proposed window increment scheme, the traditional two phases are replaced by a single phase, called "increment phase". Within this new phase, the window size still increases exponentially, but with a different manner. As mentioned above, the traditional schemes always double the window size based on the current window every RTT, while in our scheme, the portion of exponential increment always starts from one segment, and it doubles every RTT regardless the starting window size. For example, let SWND denote the starting window size and WND_i denote the window size after i RTTs. Assume that the network service rate keeps increasing and there is no indication of congestion during these i RTTs. Then we have the following equations for calculating WND_i under different window increment approaches:

(a) Under the linear increment, $\mathrm{WND}_i = \mathrm{SWND} + i$.
(b) Under the traditional exponential increment, $\mathrm{WND}_i = \mathrm{SWND} \times 2^i$.
(c) Under our proposed exponential increment,

$$\mathrm{WND}_i = \begin{cases} \mathrm{WND}_{i-1} + 2^{i-1} & \text{if } i \geq 1. \\ \mathrm{SWND} & \text{if } i = 0. \end{cases}$$

When the window starts growing from one segment (i.e. SWND = 1), both exponential increment approaches operate exactly the same. The window sizes of both approaches grow from one segment to two segments, from two segments to four segments, and so on. The difference between the traditional exponential increment and our proposed exponential increment can be observed when the starting window is larger than one segment. Also, the larger the starting window, the bigger the difference. In TCP Vegas, when the available service rate increases, a source may enter either the slow-start phase or the congestion avoidance phase, then increases its window size exponentially or linearly, respectively.

Fig. 1 shows the window adjustments under three aforementioned approaches. Suppose that the slow-start phase is from $T = 0$ to 5 RTTs, and the window size keeps stable from $T = 5$ to 7 RTTs. After $T = 7$ RTTs, the available service rate increases, and then the source is allowed to increase its window size. As shown in Fig. 1, the traditional exponential increment approach doubles the window size every RTT, and the linear increment approach increases the window size by one segment for every RTT. However, neither exponential nor linear approach works satisfactorily. The former increases the transmission rate too fast, and it may cause congestion when the available service rate increases more slowly. The latter needs a long time to catch up the available service rate. In contrast with these two approaches, our approach is more conservative than the traditional exponential increment, but more aggressive than the linear increment.

As compared with Vegas, our proposed exponential window increment approach would be more efficient. When the available network service rate increases, our approach fills the rate gap exponentially, where the gap is defined as the difference between the original rate and the newly available rate. According to our approach, the portion of exponential window increment always starts from one segment, hence only few segments will be added to the original window size within the first several RTTs. Therefore, if the available service rate increase slowly, our approach would be safer than the traditional exponential window increment approach for avoiding the congestion. Since our approach still increases the window size exponentially, it is faster than the linear increment under the congestion avoidance phase.

The following paragraphs discuss the differences between the traditional exponential window increment, our proposed exponential window increment and the linear window increment. Assume that the available service rate has been increased. Let n denote the current window size and m denote the window size for the newly available service rate. Both m and n are measured in the segments. Obviously, m is larger than n.

1. *The time duration required to reach the new service rate:*
   (a) Traditional exponential window increment: $\lceil \log_2(m/n) \rceil$ RTTs.
   (b) Our proposed exponential window increment: $\lceil \log_2(m - n) \rceil + 1$ RTTs.
   (c) Linear window increment: $(m - n)$ RTTs.
2. *The amount of over-increased window size:* as described above, the traditional exponential window increment is dependent of the starting window size, while the proposed exponential increment is not. Therefore, if m is slightly larger than n, the amount of over-increased window size in our method will be smaller than that in the traditional approach. In other words, when the available service rate only slightly increases, our method can avoid the risk of self-generated congestion, which is caused by over-increasing the window size. Even in the case that the available service rate increased significantly, our method still can perform as good as the traditional approach.

Although the window size increases more slowly under our proposed exponential increment approach, the amount of over-increased window will be mostly smaller than or at least equal to that under the traditional exponential increment approach.

2.2. RTT-based window adjustment

The RTT of a connection can be computed by subtracting the time a packet is sent by the source from the time that the corresponding ACK is received. Traditionally, this information is used to improve the preciseness of the RTT measurement, as well as to achieve congestion avoidance because the delay-based control schemes use the delay information to adjust the window size. In the proposed approach, we use the RTT threshold instead of the window threshold for congestion avoidance. This not only can achieve shorter response time but can also reduce the implementation complexity.

Vegas-like congestion avoidance schemes [1,3,4,9] use probing packets to detect the variation of RTTs. In these schemes, a source always keeps a probing packet in the network. The reason for doing so is to reduce the implementation complexity and to avoid confusion in the measured information. However, it needs at least two RTTs to detect whether the window adjustment increases or decreases the RTT. Further details about this control information delay are addressed in Section 2.4.

In order to solve the above problem, TCP Vegas proposed the safe slow-start to avoid the side effect of delayed control information. The drawback of this mechanism is the slower window size increment. Compared with the traditional exponential window increment, the safe slow-start requires twice the time to reach the same window size, thus it reduces the efficiency of the bandwidth usage. Therefore, if both proper congestion detection and window reduction scheme are effective, it is unnecessary to slow down the window increment for avoiding the packet loss. Our RTT-based mechanism serves for this purpose.

In our RTT-based mechanism, the RTT threshold is an indicator used to stop the window increment. The source calculates a new RTT threshold as well as records the current window size every RTT. The RTT threshold is derived from the virtual queue occupancy (VQO) and the VQO threshold, which will be discussed later. Once an acknowledgement packet is received, if the source is in the window increment phase, the corresponding RTT will be compared with the threshold. If the newly calculated RTT is larger than the threshold, the window must not be increased. Furthermore, the window size will be set to the value recorded in the previous period to avoid further increase of the buffer occupancy. As a result, the effect of control information delay can be reduced.

As described above, the comparison between the RTT and the RTT threshold is only meaningful in the window increment phase. If the window stays in either stable phase or decrement phase, there is no need to make the comparison.

2.3. Queue occupancy estimation

The VQO [3,4] is introduced to reflect the queue occupancy in the network. It is calculated every RTT to avoid both computational complexity and ambiguity of the control results. The main idea behind the VQO estimation is that most of the queue occupancies would appear at the bottleneck node after the bandwidth contention period.

The VQO is derived from the measured RTT, which basically consists of the fixed delay and the queuing delay. The former comprises the round-trip propagation delay and protocol processing time. This part is treated as a fixed value, which is considered as the minimum RTT ever measured. The latter is the packet queuing time in intermediate nodes along the end-to-end path. Therefore, the queuing delay is the measured RTT subtracting the ever-measured minimum RTT.

When a packet is acknowledged, its total queuing delay is the sum of the queuing delay experienced in all the intermediate nodes along the round-trip path. Given an intermediate node, if its queue occupancy is N and the network service rate is μ, the queuing delay of a newly arrived packet at this node would be N/μ. Since the service rates for all the intermediate nodes are unknown to a source, the above queuing delay calculation is meaningless unless it can be modified so that we can use the parameters known by the source. Assume that N_i denote the amount of queued data belonging to a specific flow, and the buffer management policy used in a node is common-FIFO. According to the service characteristics of common-FIFO, we can have the shared service rate μ_i of that specific data flow as follows:

$$\mu_i = \mu \times \frac{N_i}{N}. \tag{1}$$

According to Eq. (1), we can use (N_i/μ_i) instead of (N/μ) to calculate the queuing delay. Therefore, it is reasonable to represent the queuing delay based on the per-flow information.

Assume there are n queues along the end-to-end path, the total queuing delay, D_{total}, is

$$D_{\text{total}} = \sum_{j=1}^{n} \frac{N_{ij}}{\mu_j}, \tag{2}$$

where N_{ij} denotes the queue occupancy in the jth node for connection i, and μ_j denotes the service rate of the jth node. Although the service rate of each queue is hard to know, the bottleneck service rate can be derived from the packet acknowledgement rate. If the service rate for connection i in the bottleneck queue is $\mu_{\min}$, Eq. (2) can be rewritten as

$$D_{\text{total}} \le \frac{\sum_{j=1}^{n} N_{ij}}{\mu_{\min}}. \tag{3}$$

Therefore, the maximum total queue occupancy can be calculated as

$$\sum_{j=1}^{n} N_{ij} \geq D_{\text{total}} \times \mu_{\min}. \tag{4}$$

The way to calculate the VQO is through continuously monitoring a single monitor packet, whose transmission time and corresponding acknowledgment time are recorded to estimate the D_{total}. The amount of data acknowledged during the monitoring period is also recorded. As a result, the service rate $\mu_{\min}$ of the bottleneck queue can be obtained. The VQO estimation based on the monitor packets can then be calculated using Eq. (4).

2.4. Quick queue occupancy reduction

When the VQO grows larger than the preset threshold, α, the queue occupancy control will be activated to restrict the number of packets transmitted into the network. The control mechanism must decide how to reduce the queue occupancy. We will describe the characteristic of the control-information delay before discussing the queue occupancy reduction.

At the start of each control cycle, the first packet to transmit is assigned as a monitor packet, whose queuing delay is then used to calculate the VQO using Eq. (4). Starting from the second cycle, the source will adjust its window size based on the calculated result. However, the queuing delay experienced by the monitor packet for the second cycle is due to the queue occupancy carried over from the first cycle, and the effect of the reduction will not be seen until the end of the third cycle. Therefore, the control mechanism should take the effect of control delay into consideration. The way to derive the reduction ratio is based on the assumption that the queue occupancy remains nearly constant throughout the control period. Since the occurrence of congestion would only increase the queue occupancy, this assumption would work in avoiding over-reduction of the queue occupancy. Two parameters are used:

1. *K:* the ratio of the queue occupancy to be reduced, $0 \leq K \leq 1$.
2. *N:* the queue occupancy to be reduced at the start of the control cycle.

The value N can be estimated in the first and the second cycles. Then, it can be reduced by $N \times K$ in each cycle. Since the reduction effect will appear in the third cycle, the estimated queue occupancy would be $N \times (1 - K)$ packets and the queue occupancy reduced is $N \times (1-K)K$. Therefore, the reduced number of packets in each cycle is

$$f(1) = N \times K$$

$$f(2) = N \times K$$

$$f(3) = N \times (1 - K) \times K$$

$$\vdots$$

$$f(i) = \left(N - \sum_{j=1}^{i-2} f(j) \right) \times K, \ \forall i \geq 3$$

These functions can be derived recursively. It is guaranteed that the queue occupancy will never be over-reduced if the following equation is held

$$\sum_{i=1}^{\infty} f(i) \leq N. \tag{5}$$

From Eq. (5), we are able to derive the maximum value of K, which is about 0.25. Therefore, when the queue occupancy is larger than the threshold, α, the reduced amount R of the queue occupancy can be calculated as follows:

$$R = (\text{VQO} - \alpha) \times 0.25, \tag{6}$$

where

$$N = \text{VQO} - \alpha.$$

3. Implementation approach

In our proposed mechanisms, each source may operate in either normal state or error recovery state.

3.1. Normal state

The algorithm for normal state consists of two parts: the packet transmission and the packet reception.

3.1.1. Packet transmission

When the total size of unacknowledged packets is smaller than the window size, more packets are allowed to transmit. If there is no monitor packet for detecting the RTT, the next packet to be transmitted will be the monitor packet and its transmission time will be recorded.

3.1.2. Packet reception

In the following, we use a variable *phase* to indicate the state of window adjustment. For each source, the window adjustment will be in one of the three phases: the *increment* phase, the *decrement* phase, and the *stable* phase. Also, a variable *RTT_Threshold* is used to decide whether to leave the increment phase. To implement the exponential window increment regardless of the starting window size, two variables, *offset* and *bound*, are used to restrict the ratio of window increment. Initially, *phase* is set to *increment*; *RTT_Threshold* is set to the maximum value; *offset* is set to one segment; *bound* is set to zero. A parameter α is used to control the target queue occupancy.

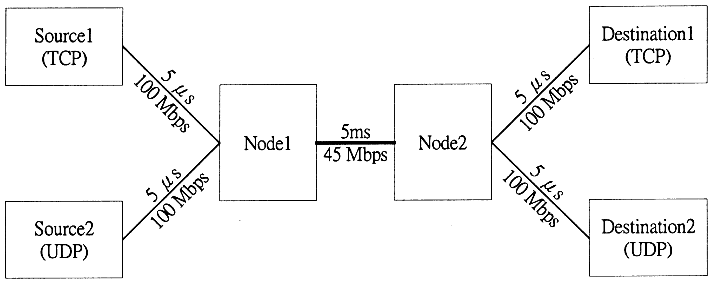

Fig. 2. Simple configuration.

The algorithm is described as follows:

```
I. If an acknowledgement packet is received and it is able
to advance the window
if ($phase = increment$) then
   Calculate the RTT of this packet;
   if (RTT < $RTT\_Threshold$) and ($offset \geq bound$) then
   window_size = window_size + 1;
      $offset = offset + 1$;
      $bound = bound + 2$;
   endif
   if (RTT > $RTT\_Threshold$) then
      $phase = stable$;
      $window\_size = previous\_window$;
   endif
else if ($phase = decrement$) and ($offset \leq bound$) then
   $window\_size = window\_size - 1$;
   $offset = offset + 1$;
   $bound = bound + 2$;
endif

II. If the acknowledged packet is a monitor packet
$bound = 0$;
$previous\_window = window\_size$;
Calculate VQO, $RTT\_Threshold$, and the RTT of this
packet;
if (VQO > $\alpha$) then
   $phase = decrement$;
   $offset = (\text{VQO} - \alpha)*0.25$;
else if (VQO < $\alpha$) and ($phase \neq increment$) then
   $phase = increment$;
   $offset = 0$;
else if (VQO = $\alpha$) then
   $phase = stable$;
endif
```

3.2. Error recovery

Since the design objectives of RTT-based TCP are to avoid the self-generated packet loss and to keep the fair bandwidth sharing, the error recovery procedure can follow the well-developed fast recovery algorithm, New-Reno [5], which successfully overcomes the weakness of Reno algorithm used in TCP Vegas under the multiple-packet-loss situation.

The RTT-based TCP can avoid the periodic congestion and the resulted packet loss. Therefore, the reason for packet loss would be limited to the transmission error or congestion generated by other flows. In the former case, New-Reno can retransmit the lost packet without wasting the bandwidth. On the other hand, in the latter case, the New-Reno improvement is able to recover the packet loss without having to redo the fast recovery phase.

After the error recovery phase, RTT-based TCP can keep

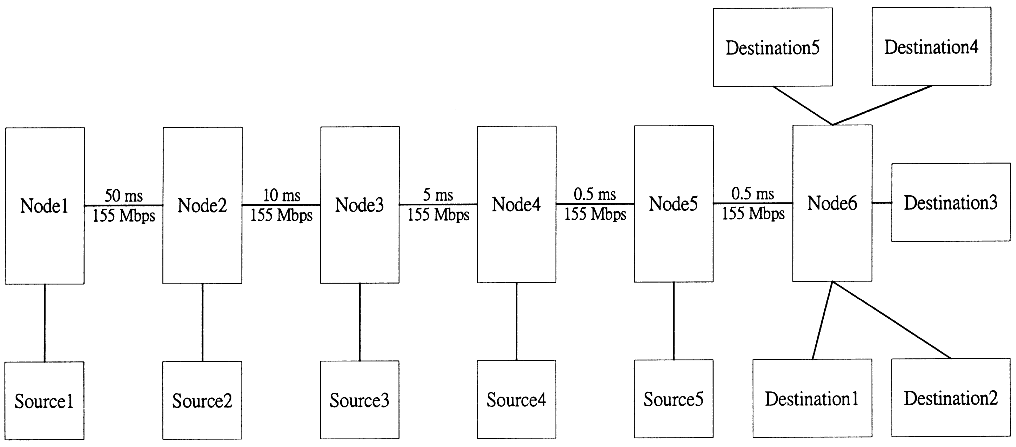

Fig. 3. Parking-lot configuration.

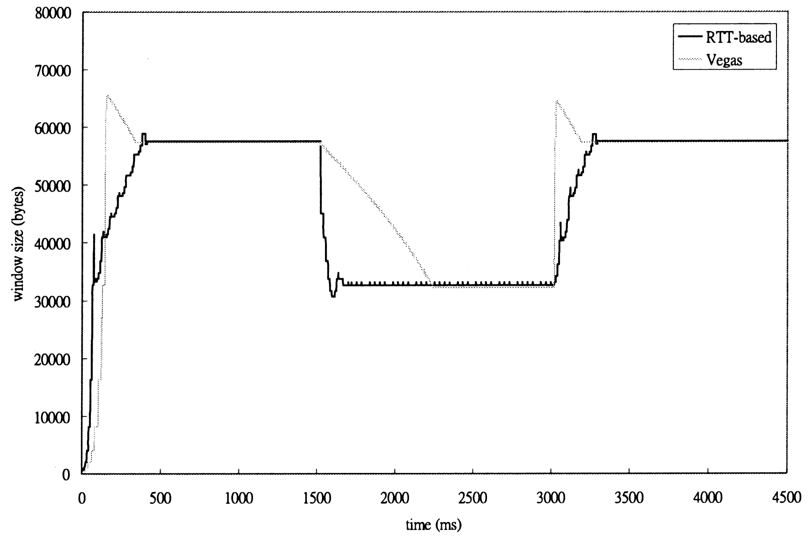

Fig. 4. Comparison of window size.

its window stable and restart the calculation of RTT threshold. Once a new RTT threshold is obtained, the window size adjustment mechanism will start to catch up the available service rate quickly.

4. Simulation and numerical results

4.1. Performance metrics

The performance of RTT-based TCP is evaluated based on the throughput and the fairness. To evaluate the fairness among sources with different round trip times, we use Jain's fairness index [11], which is defined as follows. Given a set of throughputs (x_1, x_2,…,x_n), the following formula calculates the fairness index of the set:

$$f(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}.$$

Since the throughput values are nonnegative, the fairness index always results in values between 0 and 1. In particular, the fairness index value 1 represents that all links share the bandwidth equally.

4.2. Network configurations for the simulation

Two network configurations are used to evaluate the proposed approach. The first is a simple configuration,

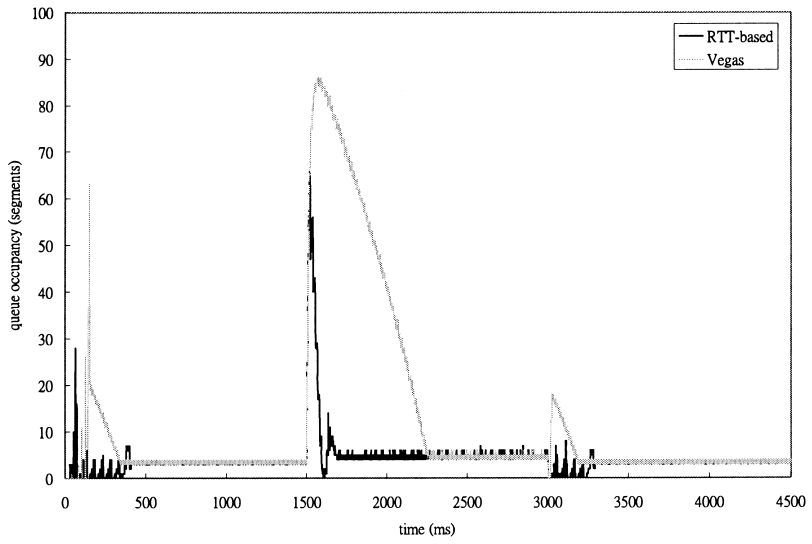

Fig. 5. Comparison of queue occupancy.

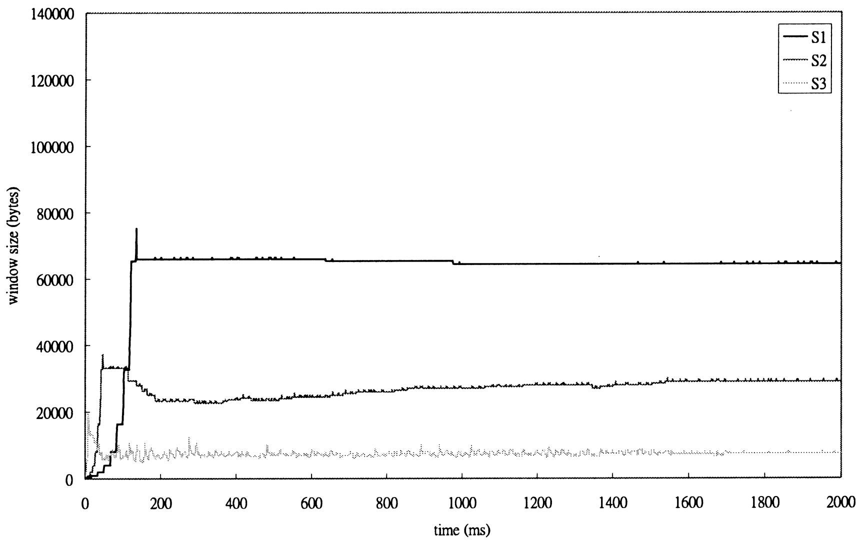

Fig. 6. Window size of RTT-based TCP.

which consists of two intermediate nodes and four end systems, as shown in Fig. 2. A TCP connection and an UDP connection are established. This configuration is used to compare the basic differences between RTT-based TCP and TCP Vegas. The second is a parking-lot configuration, which is shown in Fig. 3. It is used to compare the throughput, fairness and queue occupancy between these two schemes.

The buffer management approach used in each intermediate node is common-FIFO. The transmission rates of all sources are only constrained by the congestion avoidance schemes. The simulation times are 4.5 s and 2 s. The link error probability is ignored due to the advance of transmission technology, which is able to provide very low channel error rate.

4.3. Numerical results

In the simple configuration, the UDP source generates 20 Mbps data traffic from $T = 1.5$ to 3 s. Therefore, the basic behavior for each control scheme under both large service rate reduction and large rate increment can be demonstrated. Both RTT-based TCP and TCP Vegas need to reduce its window size first and then increase it. The following figures demonstrate the comparison results.

As shown in Fig. 4, RTT-based TCP responses faster than TCP Vegas. Therefore, when the service rate is reduced dramatically, the maximum queue occupancy in RTT-based TCP will be smaller than that in Vegas, as shown in Fig. 5. When the available service rate increases at 3 s,

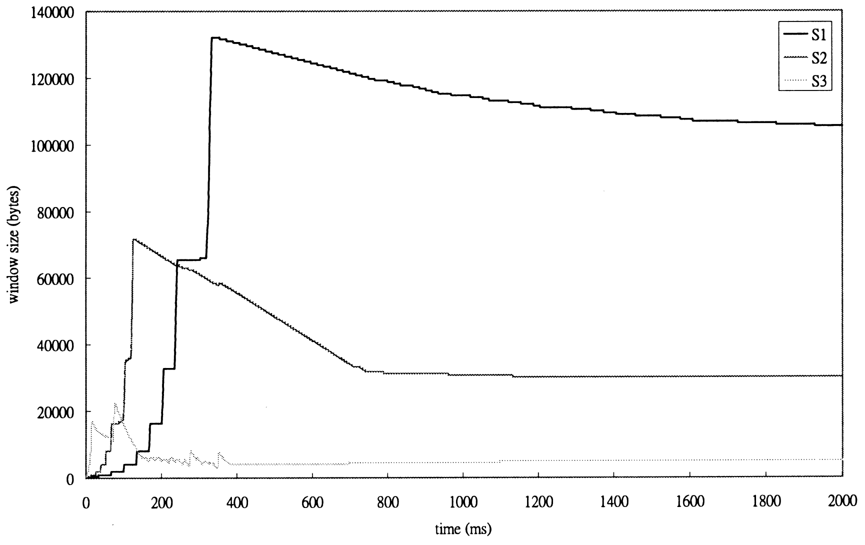

Fig. 7. Window size of TCP Vegas.

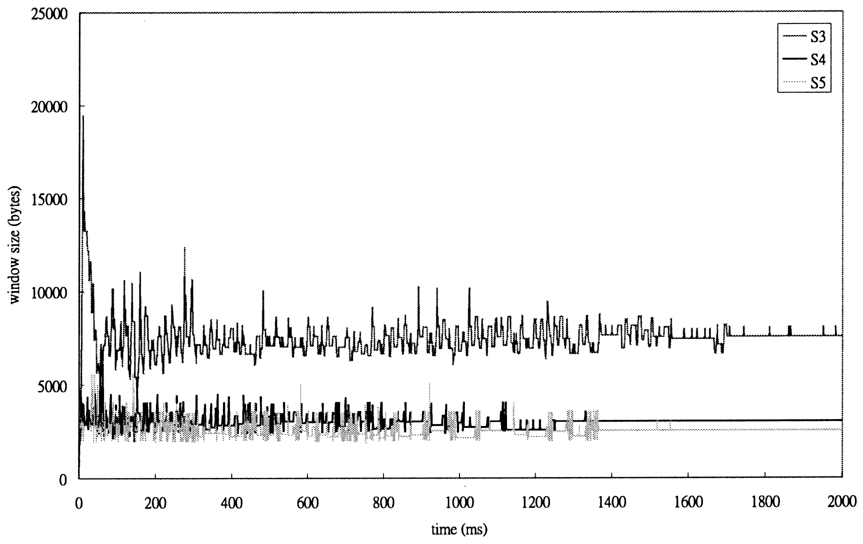

Fig. 8. Window size of RTT-based TCP.

RTT-based TCP also responses faster than TCP Vegas without increasing too much queue occupancy.

Next, the contention between sources with different RTTs is demonstrated under parking-lot configuration. For the sake of clarity, five sources shown in Fig. 3 are divided into two groups. The window sizes for three sources with longer access links are demonstrated in Figs. 6 and 7. As seen in the figures, RTT-based TCP features both shorter ramp-up time and faster window size adjustment than TCP Vegas.

Figs. 8 and 9 show the window sizes for source3, source4, and source5. Since RTT-based TCP increases the window size as well as reduces the queue occupancy faster than Vegas, it features a period of window size fluctuation. However, this would not cause any harm, because the fluctuation is small and the window size becomes stable after a period of time.

Fig. 10 demonstrates the queue occupancies of two different congestion avoidance schemes under the parking-lot configuration. Although RTT-based TCP increases the queue occupancy more quickly due to its faster window size increment, its faster window size reduction reduces the maximum queue occupancy at the bottleneck switch. As shown in Fig. 10, the maximum queue occupancy in Vegas is 65 segments, while it is 45 segments in RTT-based TCP. For the routers with small buffer space, RTT-based TCP reduces the possibility of packet loss caused by congestion.

The performance of TCP Vegas and RTT-based TCP is compared in Table 1. Since TCP Vegas has already

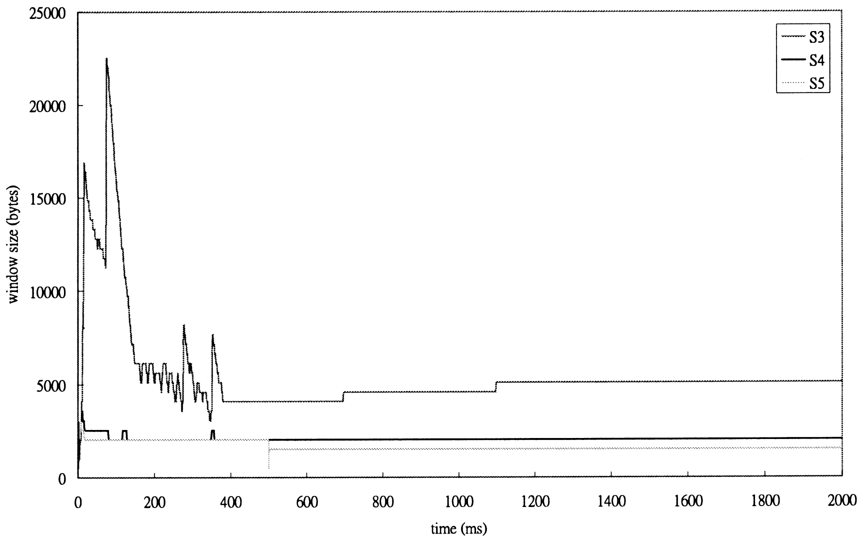

Fig. 9. Window size of TCP Vegas.

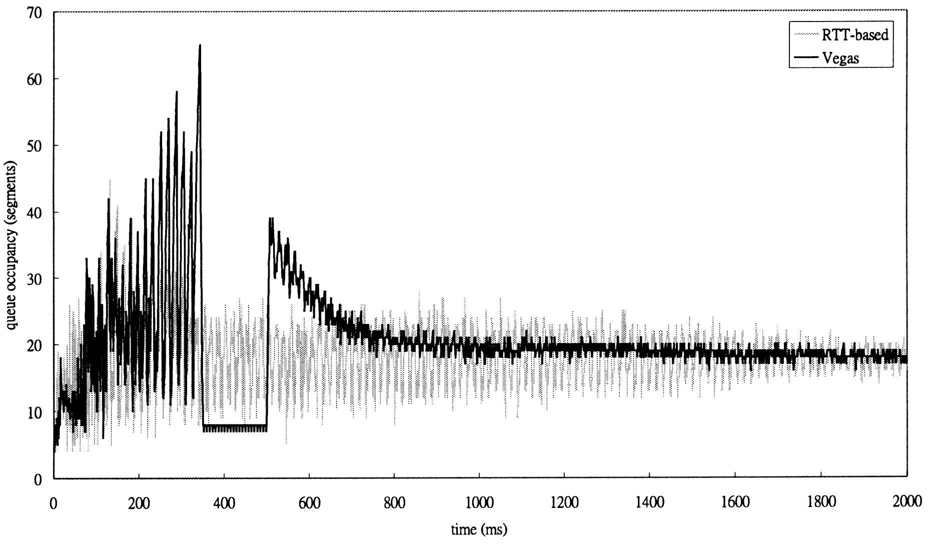

Fig. 10. Total queue occupancy under parking-lot configuration.

demonstrated its excellent throughput [1], RTT-based TCP is only a little better than TCP Vegas in terms of throughput. However, the fairness in RTT-based TCP is significantly improved comparing with TCP Vegas. This can be attributed to the quick window-size adjustment and the use of a single control parameter.

5. Conclusion and future work

In this paper, we propose an end-to-end flow control scheme, RTT-based TCP. It is derived from the original idea of TCP Vegas, but it provides better performance than TCP Vegas. RTT-based TCP consists of four mechanisms. First of all, the modified exponential increment scheme is used to increase the window size. It provides safer and faster window increment than TCP Vegas. Second, the RTT measurement is used as the control objective instead of the window threshold used in TCP Vegas. This can reduce the implementation complexity and achieve shorter response time. Third, the idea of VQO is introduced to estimate the queue occupancy of the intermediate nodes in the network. The estimation results are used for a source to adjust its window. Finally, quick queue occupancy reduction is used to decrease the window size of a source when the VQO is larger than the threshold. These four proposed mechanisms together improve both throughput and fairness of TCP. Besides, it reduces the maximum queue occupancy, and this in turn reduces the possibility of congestion.

Table 1
Performance index

	TCP Vegas	RTT-based TCP
Throughput	97.2%	97.3%
Fairness	0.871	0.999

The proposed scheme adjusts the window size of a source based on the measured RTT information. It is initially designed for wired networks. In wireless networks, some reasons may cause unexpected RTT variations, which lead to performance degradation. For example, if a mobile host roams from a base station to another base station, extra delay may incur before the handoff is complete. Also, since the route is changed, the round-trip propagation delay may also be changed. Another RTT variation may happen if dynamic channel allocation scheme, such as CDPD [15], is used. As a result, the proposed approach may not work as efficiently as in wired networks if it is directly applied in wireless networks without any modifications. We are currently working on developing an RTT-based TCP applicable in both wired and wireless networks.

References

[1] L.S. Brakmo, L.L. Peterson, T.C.P. Vegas, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE Journal on Selected Areas in Communications 13 (8) (1995) 1465–1480.
[2] R.F. Chang, L. Huynh, J. Gray, Adaptive rate-based congestion control versus TCP-SS: a performance comparison, International Conference on Network Protocols (1993) 186–197.
[3] J.R. Chen, Y.C. Chen, C.T. Chan, A distributed end-to-end rate control scheme for ABR service, IEEE GLOBECOM (1998) 2446–2451.
[4] J.R. Chen, Y.C. Chen, C.T. Chan, Performance evaluation of an end-to-end rate control approach for ABR services, IEICE Transactions on Communications 12 (1998) 2400–2412.
[5] S. Floyd, T. Henderson, The New Reno modification to TCP's fast recovery algorithm, RFC2582, April 1999.
[6] L. Huynh, R.F. Chang, W. Chou, Performance comparison between TCP slow-start and a new adaptive rate-based congestion avoidance scheme, MASCOTS'94 (1994) 300–307.
[7] V. Jacobson, R. Braden, D. Borman, TCP extensions for high performance, RFC1323, May 1992.
[8] S. Keshav, A control-theoretic approach to flow control, ACM SIGCOMM, August 1991, pp. 189–201.

[9] L. Kalampoukas, A. Varma, K.K. Ramakrishnan, Explicit window adaptation: a method to enhance TCP performance, IEEE INFOCOM (1998) 242–251.

[11] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.

[12] R. Jain, A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks, ACM Computer Communication Review 19 (5) (1989) 56–71.

[13] M. Mathis, J. Mahdavi, Forward acknowledgment: refining TCP congestion control, ACM Computer Communication Review 26 (4) (1996) 281–291.

[14] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP selective acknowledgement options, RFC2018, October 1996.

[15] M. Streetharan, R. Kumar, Cellular Digital Packet Data, Artech House, 1996.

[16] W. Stevens, TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, RFC2001, January 1997.

[17] Z. Wang, J. Crowcroft, A new congestion control scheme: slow start and search (Tri-s), ACM Computer Communication Review 21 (1) (1991) 32–43.

[18] Z. Wang, J. Crowcroft, Eliminating periodic packet loss in 4.3 Tahoe BSD TCP congestion control algorithm, ACM Computer Communication Review 22 (2) (1992) 9–16.