ELSEVIER

Computer Communications 22 (1999) 1493–1501

computer communications

Pseudo-rate TCP: a congestion avoidance scheme with nearly optimized fairness and throughput

J.-R. Chen, Y.-C. Chen*

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30056, Taiwan

Received 23 February 1999; received in revised form 1 June 1999; accepted 1 June 1999

Abstract

As the bandwidth requirement on the Internet grows, the efficiency of networking protocol becomes an important concern. This paper proposes a so-called pseudo-rate TCP, which features both better fairness and higher throughput than TCP Vegas by exponentially increasing the window size to a derived bound. The pseudo-rate TCP avoids the typical problem that sources with shorter round-trip times always get better chance in allocating the bandwidth. We discuss two key mechanisms designed for pseudo-rate TCP, and compare its performance with TCP Vegas through system simulation. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Pseudo-rate TCP; Fairness; TCP Vegas

1. Introduction

Packet retransmission and congestion control are two important mechanisms for TCP to provide reliable and efficient data transport. Due to the popularity of TCP, a variety of approaches have been proposed to improve TCP performance. The SACK [1] is based on the improvement of TCP retransmission mechanism to reduce bandwidth waste, while most TCP related research focus on the congestion control, so does this work. Our approach concerns the bandwidth-sharing control, which provides fairness service among sources with various round-trip times, and features as well a significantly higher throughput and better fairness than TCP Vegas.

The earlier congestion control improvement was implemented on TCP Tahoe, which maintained a proper window size to control the number of outstanding packets. Both allocation of bandwidth and probing the available bandwidth are accomplished by increasing the window size gradually. Most of the window size adjustment schemes that improve the TCP performance are based on this approach. The reason to use such a window adjustment approach is because of the simplicity in implementation.

Another class of congestion avoidance scheme, the rate-based scheme [2–4,12], uses a detection mechanism to estimate the allowed network service rate, in addition, it uses the queue occupancy control to avoid the congestion. This is similar to some of the window-based congestion avoidance schemes in which the dependency between the round-trip time 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. Therefore, we would take advantages of the rate-based flow control to enhance the window-based congestion avoidance scheme.

Existing schemes such as TCP Vegas try to linearly adjust the window size based on the variation of transmission rates. In contrast, pseudo-rate TCP uses the queuing delay to derive the bound of window size and to exponentially fill-up the gap, which is defined here as the difference between the actual window size and its estimated bound. Setting the window-size bound can prevent the traffic burst from generating congestion and keep the "self-clocked" characteristic of TCP to smoothen the traffic. Exponentially filling-up the window gap shortens the response time of pseudo-rate TCP. By dealing with the window gap instead of the window size, we are able to expand the window much faster with continuous tracking of the available bandwidth.

The rest of this paper is organized as follows. Section 2 briefly describes current TCP flow control schemes. The detailed pseudo-rate TCP scheme is discussed in Section 3. Simulation and performance comparison of different TCP schemes is addressed in Section 4. Section 5 concludes the work.

* 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)

PII: S0140-3664(99)00103-6

2. Current TCP flow control schemes

2.1. Pure window-based flow control schemes

In TCP Tahoe, the window increment process is performed in two phases, slow-start and congestion avoidance. The window size increases exponentially in the former and linearly in the latter. A source maintains a threshold to differentiate the two phases. Once the source detects a packet loss, it sets the threshold to half of its current window size, which is then reset to the initial value. Therefore, in TCP Tahoe, a source continuously increases its window size until a packet loss is encountered, then it resets the window size. The periodic packet loss, the window size resetting, and the rough threshold setting put quite a bit of restrictions on the throughput of TCP Tahoe.

In TCP Reno the throughput is improved through the so-called fast recovery algorithm [5], which modifies the window size setting after a packet loss. Its window size after congestion is set to the threshold value. Both TCP Tahoe and TCP Reno use the segment loss to signal the network congestion. Since the new window size reduces to half of that before the packet loss, it concerns more the safety than the adequacy. Here the window size after congestion may not properly reflect the network status information, and whether to increment window exponentially or linearly is based upon the roughly estimated threshold value. Without restricting the window size, the transmission rate always tends to exceed the available bandwidth, which then can be detected through the packet loss. Since each source adjusts its window size every round-trip time, sources with shorter round-trip times can acquire more bandwidth and achieve higher throughput than those with longer round-trip times. Therefore, the periodically occurring congestion and the roughly estimated window size are two factors that restrict both throughput and fairness.

2.2. Improved congestion avoidance schemes

Both TCP Tahoe and TCP Reno do not keep any information regarding the network status, such as congestion. Thus, to detect the available bandwidth, they continuously increase the window size until congestion occurs. Although such window size adjustment schemes avoid the occurrence of further congestion, both schemes are unable to fully utilize the available bandwidth. To overcome the periodic congestion and inadequate window size selection, several schemes based on the network status information have been proposed.

When queue occupancies in intermediate nodes grow, the round-trip times of the successfully acknowledged packets increase as well. DUAL algorithm [6] is based on reacting to such round-trip-time increment; the CARD [7] approach uses the analytic derivation of an optimum window size for a deterministic network; while the Tri-S scheme [8] takes advantage of the fact that a network approaches congestion through flattening the service rate.

TCP Vegas [9] is similar to Tri-S in dealing with changes of the sending rate. TCP Vegas compares the sending rate with the expected rate, which is calculated based on the current window size. While the window size is adjusted linearly when the difference between the expected rate and the actual sending rate exceeds a defined threshold. Besides, TCP Vegas increases its window size exponentially every other round-trip time during the slow-start phase, this is to ensure that the previous sending rate is allowed before it continues the process.

2.3. Deficiency of TCP Vegas

Although TCP Vegas achieves pretty good performance, its bandwidth allocation among sources with different round-trip times are indeed not fair enough. The exponential increase of window size every other round-trip time during the slow-start phase restricts the throughput, so does the linear increase during the congestion avoidance phase. Further, the linear window size adjustment makes TCP Vegas unable to quickly change the window size to a proper value during the congestion avoidance phase.

TCP Vegas calculates the expected throughput based on the previous window size, thus, the faster the window size increases, the higher the throughput will be. Since a source with shorter round-trip time always increases its window size faster, and the window is stable when the actual rate is close to the expected rate, it would experience a higher throughput than sources with longer round-trip times.

3. Pseudo-rate TCP

Although TCP Vegas provides a good mechanism to avoid periodic congestion, its throughput and fairness can still be improved. Using the concept of rate-based flow control, we design an advanced congestion avoidance scheme called the pseudo-rate TCP to provide even higher throughput and better fairness.

3.1. Throughput improvement

Under the same round-trip time, a source that acquires bandwidth faster will eventually obtain a higher throughput. In a window-based congestion avoidance scheme, more bandwidth can be obtained through window size increment, which should be as fast as possible for the sake of efficiency. This is why most of TCP congestion avoidance schemes use exponential window adjustment when the network is far from congestion.

Tahoe, Reno and New-Reno try to avoid the frequent periodic congestion by dividing the window increment into the slow start phase and the congestion avoidance phase. This process is performed through setting the threshold value. However, it is only a rough threshold value,

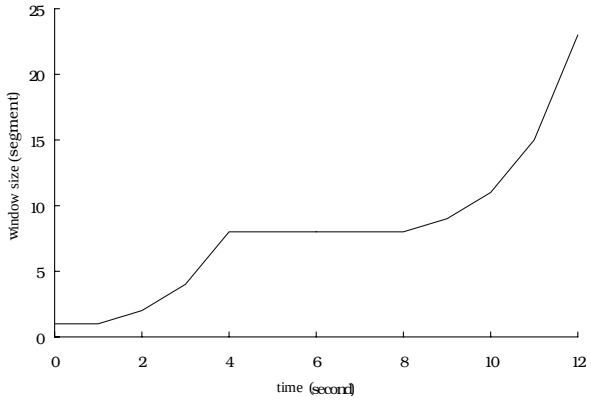

Fig. 1. Window increment behavior in pseudo-rate TCP.

which together with the continuous window increment is unable to achieve the high throughput satisfactorily. Since a proper and stable window size can avoid the potential self-generated congestion, our main idea for congestion avoidance is to approach a proper window size, then keep it stable. There are two conditions in which we must adjust the window size even if it is stable, these two conditions are congestion caused by other sources and increment of the available bandwidth. When the network becomes congested, quickly decreasing the window reduces the possibility of packet loss. In contrast, if the available bandwidth increases, quickly enlarging the window can result in better bandwidth utilization.

The behavior of window increment in pseudo-rate TCP is shown in Fig. 1, in which the window bound is eight segments during the first 8 s. The window size increases exponentially from one segment to eight segments and then constrained by the window bound. When the window bound changes, the window increment portion changes exponentially. In our example, the window bound changes from eight segments at the eighth second to larger than 23 segments at 12th second. It is clear that the window increases exponentially starting from the previous stable point. As a sequel, in pseudo-rate TCP a stable window size may change due to changing the window bound.

In pseudo-rate TCP, a window bound is set to restrict the window increment. Therefore a stable window size will not change unless the window bound is increased. This is different from the threshold mechanism employed in TCP Tahoe and Reno, in which a threshold is used to decide whether to increase the window size or not. If a congestion avoidance scheme does not increase its window size unrestrictedly, it no longer needs the threshold setting. Thus pseudo-rate TCP uses the queuing delay information for window increment, and uses the window bound to keep the window stable.

An efficient way for window increment would be a compromise between the linear and the exponential approach. The window decrement would be a direct drop in the measured window bound, because congestion avoidance is required more than high throughput. The robustness of our approach can be achieved by the window increment policy, which is based on the window gap. The current window increment portion will grow exponentially up to the window gap. Therefore, only the gap is filled up exponentially, while the window size variation is restricted within several segments during the first several round-trip times. This avoids both the linear increment of window, which is slow, and the effect of large window variation after a window bound change.

To exponentially fill-up the window gap, a window is divided into two parts, the Base and the Offset. The window size used to control the packet transmission, called CONGESTION WINDOW (cwnd), is the sum of the Base and the Offset. The former is calculated as the previous cwnd minus one segment when cwnd equals the window bound. The latter is set to at least one segment and it increases exponentially every round-trip time.

The implementation, which adjusts the Offset exponentially every round-trip time, is accomplished with an additional variable, the Offset Bound. This variable is reset to zero periodically at every round-trip time. Whenever an ACK packet is received, the first step is to check whether the Offset is larger than the Offset Bound. If yes, the former will be increased by one segment and the latter increased by two segments. Following this way, the Offset value can be kept at twice that of the previous value at every round-trip time. Further, this method can avoid the possibility that the number of received ACKs is larger than the Offset value.

Table 1 is an example that illustrates the behavior of window size increment. Two parts of the window are called Base and Offset respectively. The example in Fig. 1 is a special case in which k is set to 4. The window bound is shown in the row named “Bound”.

The throughput improvement is achieved through traditional exponential window increment and stable window size. This window adjustment can improve both the bandwidth utilization and the robustness of congestion avoidance scheme. As a sequel, the rest of the work is setting the window bound and offset bound to improve the throughput.

Table 1
The window variation in pseudo-rate TCP

Round	0	1	2	⋯	k	$k+1$	$k+2$	$k+3$	⋯	$2k+1$	$2k+2$	$2k+2$	⋯
Base	0	0	0	⋯	0	2^k-1	2^k-1	2^k-1	⋯	2^k-1	2^k-1	2^k-1	⋯
Offset	1	2	4	⋯	2^k	1	1	1	⋯	2	4	8	⋯
Bound	2^k	2^k	2^k	⋯	2^k	2^k	2^k	2^k		2^{k+1}	2^{k+1}	2^{k+1}	⋯
Window	1	2	4	⋯	2^k	2^k	2^k	2^k	⋯	2^k+1	2^k+3	2^k+7	⋯

3.2. Fairness improvement

Since a proper window size is determined through the implicit rate calculation, our TCP congestion avoidance scheme is therefore called pseudo-rate TCP. The fairness of the pseudo-rate TCP can also be achieved through properly setting the window bound. The bandwidth sharing in a common FIFO environment depends on the queue occupancy ratio of each source, when every source keeps nearly equal queue occupancy in the switch, the bandwidth sharing will be fair. Also, the queue occupancy can be derived from both queuing delay and service rate, these information are used to calculate the proper window size, which is then used to keep the queue occupancy at a predefined level, so that a fair bandwidth sharing can be achieved.

Similar to TCP Vegas, which adjusts the window size every round-trip time, pseudo-rate TCP adjusts its window bound every round-trip time, and implements its congestion avoidance mechanism at the source. Two variables are required to calculate the window bound. The first one is the network service rate. Since TCP uses a retransmission mechanism, a source always keeps the packet size information before it is acknowledged. Thus, upon receiving an acknowledgement packet, the source is able to count the total amount of data serviced by the network, and the network service rate can be calculated based on the measuring interval.

The second variable is the fixed delay, which contains the propagation delay, the transmission delay, the store-and-forward delay and the processing time. The processing time in the destination is assumed fixed. Although the fixed delay is hard to calculate, but the round-trip time is detectable. Since the end-to-end delay consists of both fixed delay and variable queuing delay in the network, we use the lowest round-trip delay detected as the fixed delay. This assumption has also been used in Refs. [4,9].

When both the network service rate and the fixed delay are known, the allowed window size can be calculated as follows:

1. Let RTT be the measured round-trip time.
2. N_{data} is the number of segments serviced by the network within the measured round-trip time.
3. T_{segment} is the time to service a segment, the value is RTT divided by N_{data}.
4. μ is the network service rate, which is the reciprocal of T_{segment}.
5. T is the fixed delay, which is the lowest RTT.
6. N_{queue} is the control parameter setting. It represents the proper number of packets stored in the switch.
7. The allowed round-trip time is $T + N_{\text{queue}}T_{\text{segment}}$.
8. The bound of the window size is derived from both the allowed round-trip time and the network service rate, it is $\mu(T + N_{\text{queue}}T_{\text{segment}})$.

Since the window bound is set based on the fairness consideration, the quick window adjustment proposed in the previous section can reduce the time required to adjust the actual window to the fairness window. Further, such adjustment allows a few round-trip times during which large window size variation as window bound re-calculation will not occur.

3.3. Error recovery

Since the design goal of pseudo-rate TCP is to avoid the self-generated packet loss and keep the fair bandwidth sharing, the error recovery from the packet loss can follow the matured fast recovery algorithm, New-Reno [10]. Unlike the Reno algorithm used in TCP Vegas, the New-Reno improvement successfully avoids the weakness of Reno under the multiple-packet-loss situation.

The pseudo-rate TCP can avoid the periodic congestion and the packet loss caused by this periodic congestion. Therefore, the reason for packet loss is constrained to be the channel error and other flow generated congestion. When the packet loss is caused by the channel error, New-Reno improvement can retransmit lost packet without bandwidth waste. In contrast, if the packet loss is due to congestion generated by other data flow, the New-Reno improvement is able to recover the packet loss without having to re-enter the fast recovery phase.

After the error recovery phase, the pseudo-rate TCP maintains its window the same way as New-Reno does and restarts the window bound calculation. Once a new window bound is obtained, the window size adjustment mechanism starts to grab the available bandwidth quickly, as described in Section 3.1.

3.4. Pseudo-rate TCP mechanism

Pseudo-rate TCP consists of two processes, the window size adjustment and the window bound setting. The former is performed once every acknowledgement packet is received. The window bound value is calculated at the end of each cycle period, which starts from a packet transmission and ends upon the reception of an acknowledgement with sequence number larger than that of the first packet in the same cycle. This is similar to TCP Vegas that adjusts the window size every round-trip time.

To detect the cycle period, it must keep a packet under monitoring. If there is no such packet, the packet to be transmitted next will be used. Once a cycle period starts, its transmission time can be recorded. Upon receiving the acknowledgement of the monitored packet, the shortest round-trip time will be updated, also the network service rate during the past cycle period will be calculated as the total acknowledged packet size divided by the cycle period.

There are two bound values for controlling the window size, the window bound and the offset bound. The former is the acceptable window size, which does not change upon the reception of packet acknowledgement. The latter is for emulating the exponential adjustment of the window gap. The exponential adjustment is similar to the TCP exponential

increment. It doubles the window size smoothly every cycle period. As a result, the undesired traffic burst, which is usually caused by sudden increase of the window size, can be avoided.

The window bound is set every cycle period and its value is calculated using the formula $\mu(T + N_{\text{queue}}T_{\text{segment}})$, as described previously. The window offset will be allowed to increase only under two conditions. The first condition is that the sum of base window and window offset is smaller than the window bound. The second condition is that the window offset is larger than the offset bound. Otherwise, the base window is set to the window bound and the window offset is set to zero.

We use both the window offset and the offset bound to approximate the exponential increase of window offset. The window offset will be increased by one segment and the offset bound increased by two segments whenever the condition allows and an acknowledgement packet arrives. Therefore, the window offset increment is equal to the window offset after the offset bound was set to zero. Since the offset bound is set to zero in every cycle period, the window offset can be doubled in every cycle period. As a result, the window offset increases exponentially and smoothly.

The algorithm of pseudo-rate TCP is presented as follows:

1. *Packet transmission:* When the total size of unacknowledged packets is smaller than the sum of the base window and the window offset, more packets are allowed to transmit. If there is no monitor packet for deciding the cycle period, the next packet to be transmitted will be the monitor packet and its transmission time will be recorded.
2. *Packet reception:* Once the acknowledgement of a monitor packet is received, the current cycle period ends, and the following procedures are performed:
   - The window bound is updated using the formula $\mu(T + N_{\text{queue}}T_{\text{segment}})$.
   - The offset bound is set to zero.
   - If the sum of the base window and the window offset is larger than the window bound updated previously, the base window and the window offset are set to window bound and zero, respectively.
   - If the sum of the base window and the window offset is smaller than the updated bound, and also the window offset is zero, the window offset will be set to one segment. In case the offset bound is smaller than the current window offset, the window offset and its bound will be increased by one segment and two segments, respectively.

Once the non-duplicate acknowledgement is received and the window offset is smaller than the offset bound, the window offset is increased by one segment and the offset bound by two segments.

Once the duplicate acknowledgement is received the error recovery procedure described in Section 3.3 is initiated.

3.5. Comparison between TCP Vegas and pseudo-rate TCP

As TCP Vegas features higher performance than TCP Reno [9], we compare the difference between TCP Vegas and Pseudo-Rate TCP:

1. In the slow-start phase: As TCP Vegas increases its window size every other round-trip time and pseudo-rate TCP does that every round-trip time, pseudo-rate TCP achieves better throughput in the slow-start phase. Therefore, although the window size increment of TCP Vegas and pseudo-rate TCP are all exponentially, TCP Vegas requires twice the time period to reach the same window size.
2. Increase of available bandwidth under a stable window: As TCP Vegas adjusts its window linearly and pseudo-rate TCP approaches the allowed window bound exponentially, the latter features better throughput under a stable window.
3. Decrease of available bandwidth under a stable window: TCP Vegas still adjusts its window linearly and pseudo-rate TCP reduces its window to a specified level directly, pseudo-rate TCP features better chance to avoid the occurrence of congestion.
4. Robustness: As the decision for window adjustment is made every round-trip time, TCP Vegas and pseudo-rate TCP both feature the same level of robustness in dealing with the network congestion.
5. Fairness: Although the fairness has various definitions, in this work it is the max–min fairness. When the transmission rates of all sources are limited by the congestion control schemes, each source should achieve equal bandwidth sharing. A source in TCP Vegas adjusts its window size according to the previous transmission rate, and this is a relative adjustment that cannot guarantee a correct fair-share rate. While pseudo-rate TCP adjusts its window according to the queuing delay, this is an absolute adjustment which uses the queue occupancy ratio to guarantee a fair bandwidth share. Therefore, pseudo-rate TCP achieves better fairness in bandwidth sharing for sources with different round-trip times.

According to the above comparison, it is obvious that the proposed pseudo-rate TCP features higher throughput and better fairness than TCP Vegas, the simulation result in the following will verify our claims mentioned above.

4. Simulation and numerical results

4.1. Performance metrics

The performance of pseudo-rate TCP is measured using two well-known performance metrics based on the

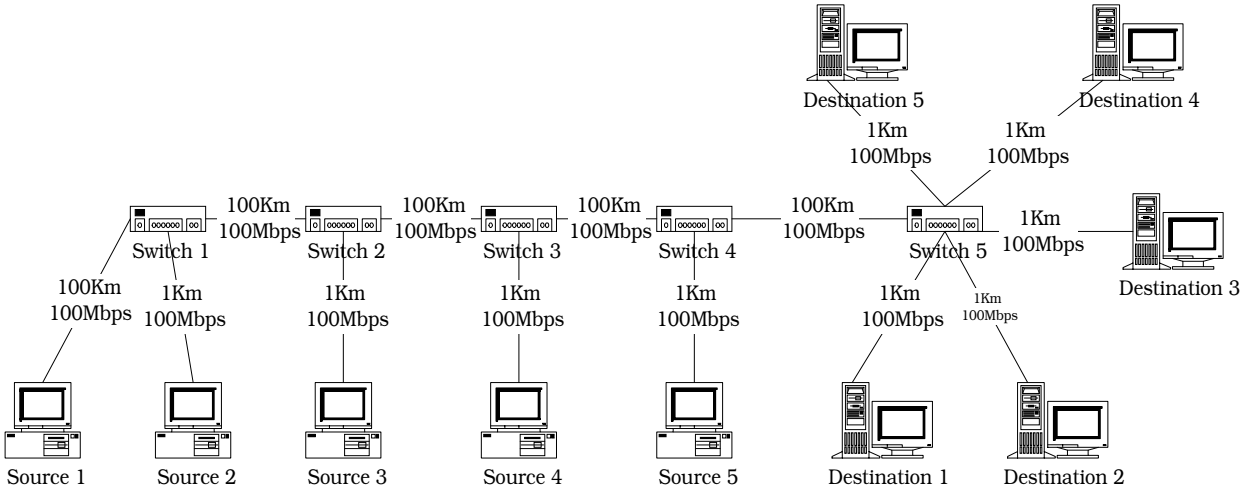

Fig. 2. Network configuration for the simulation.

throughput and the fairness index. Jain's fairness index [11] is used to evaluate the fairness among sources with different round-trip times, it is described as follows: given a set of throughputs $(x_1, x_2, \ldots, x_n)$, the function below assigns a fairness index to the set:

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

As the throughput is non-negative, the fairness index always results in values between 0 and 1. If the fairness index has value 1, it represents that all links share the bandwidth equally.

4.2. Network configuration for the simulation

The network topology for the simulation is a parking-lot configuration, which is commonly used in literature to illustrate the fairness problem. The configuration used is shown in Fig. 2. The intermediate switch uses a common-FIFO to service all sources.

The transmission rates of all sources are only constrained by the congestion avoidance schemes. The simulation time is 1 s. Since we want to see the performance of the congestion avoidance scheme during the ramp-up time, as well as during the steady state, after 500 ms, sources 1 and 3 will retransmit unacknowledged packets only. Following this way, the allowed service rate for the remaining sources increases, also the performance to acquire more bandwidth and to avoid unfairness among sources with different round-trip times are shown. The link error rate is assumed to be ignored since the current transmission technology is able to provide very low channel error rate.

We only compare pseudo-rate TCP with TCP Vegas because the performance of TCP Vegas is much better than TCP Reno [9]. The simulation parameters are shown in Table 2.

The threshold in pseudo-rate TCP is the allowed queue occupancy in the switch, it is used to calculate the window bound.

Table 2
Simulation parameter values

Parameter	Value
Segment	512 byte
TCP Vegas α	1
TCP Vegas β	3
Pseudo-rate TCP threshold	3
Switch per port buffer (packet)	128

4.3. Numerical results

The window size corresponds to the amount of data allowed to transmit into the network without acknowledgement. Therefore, the window size represents the product of the allocated bandwidth and the round-trip time for a source. Since Sources 1 and 3 cease the transmission of new packets after 500 ms, the window size variation of Sources 2, 4 and 5 throughout the simulation period can be observed in Figs. 3 and 4.

Fig. 3 shows the window size variation of TCP Vegas, which avoids periodic congestion effectively, thus TCP Vegas performs better than TCP Reno. Since the window size increases exponentially every other round-trip time in slow start phase, the ramp-up time would be longer than that in pure exponential increment. The gradual window adjustment of TCP Vegas after steady state occurs at around 500 ms, this is because the window size changes linearly after it enters the congestion avoidance phase. The gradual window adjustment restricts the throughput slightly. Since sources with shorter round-trip times still uses larger

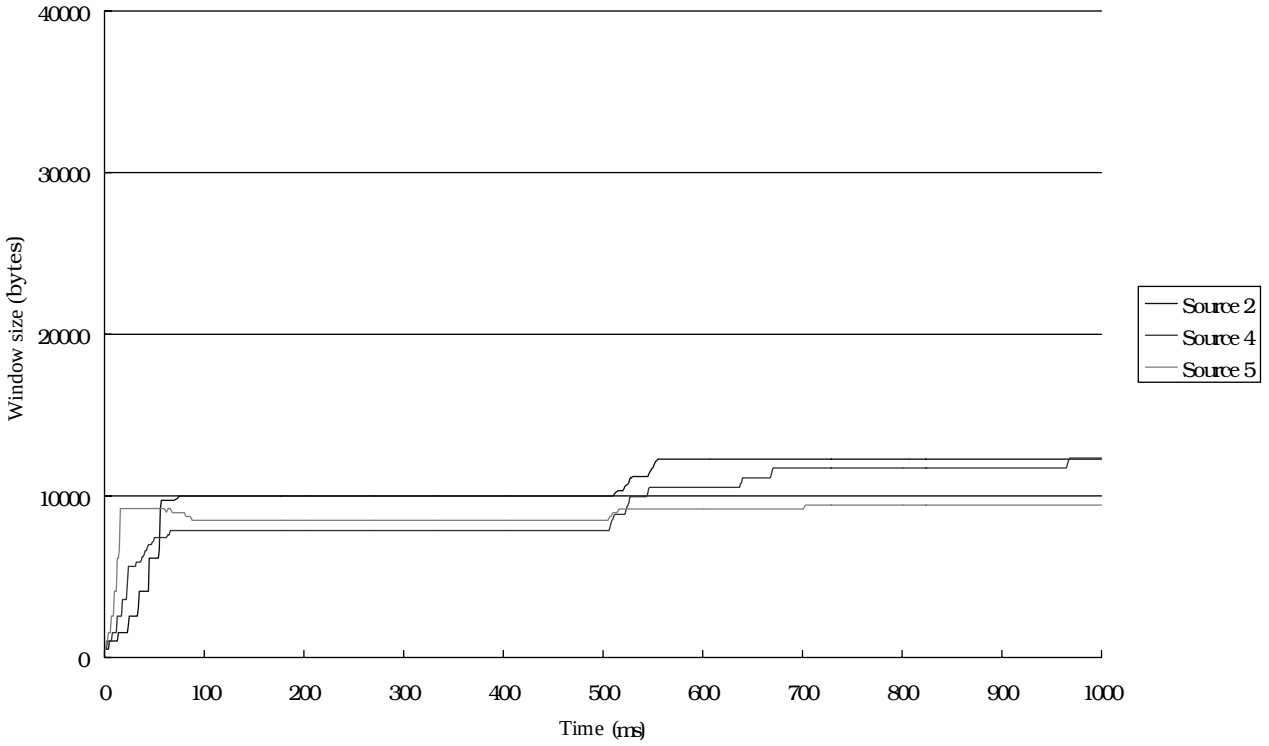

Fig. 3. Window size variation of TCP Vegas.

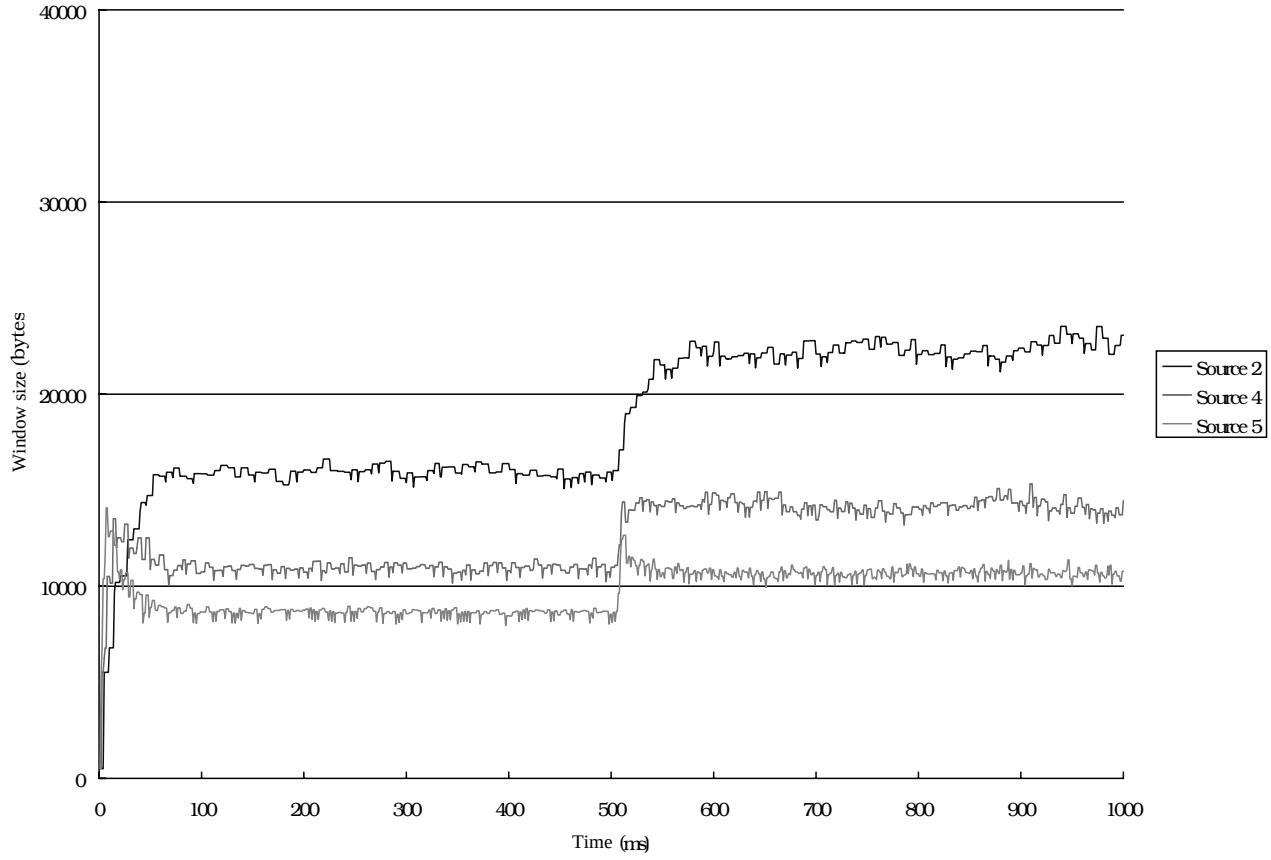

Fig. 4. Window size variation of pseudo-rate TCP.

Table 3
Performance comparison

	Throughput (%)	Fairness index
TCP Vegas	95	0.889
Pseudo-rate TCP	98	0.993

window sizes than that with longer round-trip times, the fairness of TCP Vegas is still not optimized.

The window size variation of pseudo-rate TCP is shown in Fig. 4. Pseudo-rate TCP does not feature the periodic drop of window size caused by self-generated congestion. Since it increases the window size exponentially every round-trip time, its ramp-up time is shorter than that in TCP Vegas. Further, it exponentially fills up the window gap whenever the window bound increases. Therefore, the throughput of pseudo-rate TCP is better than TCP Vegas. Since the window bound is affected by the network service rate, and the switch provides a common-FIFO to all sources, the window size fluctuates slightly around the window bound. The fluctuation range is about 5% of the window size.

Observing the period between 0–20 and 500–520 ms, it is clear that the window size increases exponentially regardless of the size of the previous stable window. After these periods, sources with shorter round-trip times start to reduce their window size, while other sources increase their window size. Since the window size adjustment is a result of compromise instead of greediness, the window size increment is no longer exponential.

Although under pseudo-rate TCP, sources with shorter round-trip times increase their windows faster, the window bound will restrict their window increment. While sources with longer round-trip times increase their window size, sources with shorter round-trip times reduce both their window bound and their window sizes. Therefore, all of the active sources can share the bandwidth much fairly than TCP Vegas.

The performance of two congestion avoidance schemes is discussed as follows. At the end of simulation, the throughput is calculated based on the total throughput of five sources, and the fairness index is based on the fairness among three sources with same simulation condition. The performance comparison is shown in Table 3.

Under our simulation environment, the throughput of pseudo-rate TCP is 3% higher, and the fairness is 10% better than TCP Vegas.

The TCP throughput in the simulation is shown in Fig. 5. Since TCP Vegas successfully avoids the periodic congestion, its performance is much better than TCP Reno [9], thus there is not much space left for us to improve the total throughput. However, pseudo-rate TCP not only successfully avoids the periodic congestion but also utilizes the unused window with exponential growth, its throughput is even better than TCP Vegas according to our simulation.

Fig. 6 shows the fairness index during the simulation. TCP Vegas does not restrict the window increment for sources with shorter round-trip times. Therefore, the fairness among sources with different round-trip times is still low because sources with shorter round-trip times increase their window size faster. Thus, the fairness of TCP Vegas is

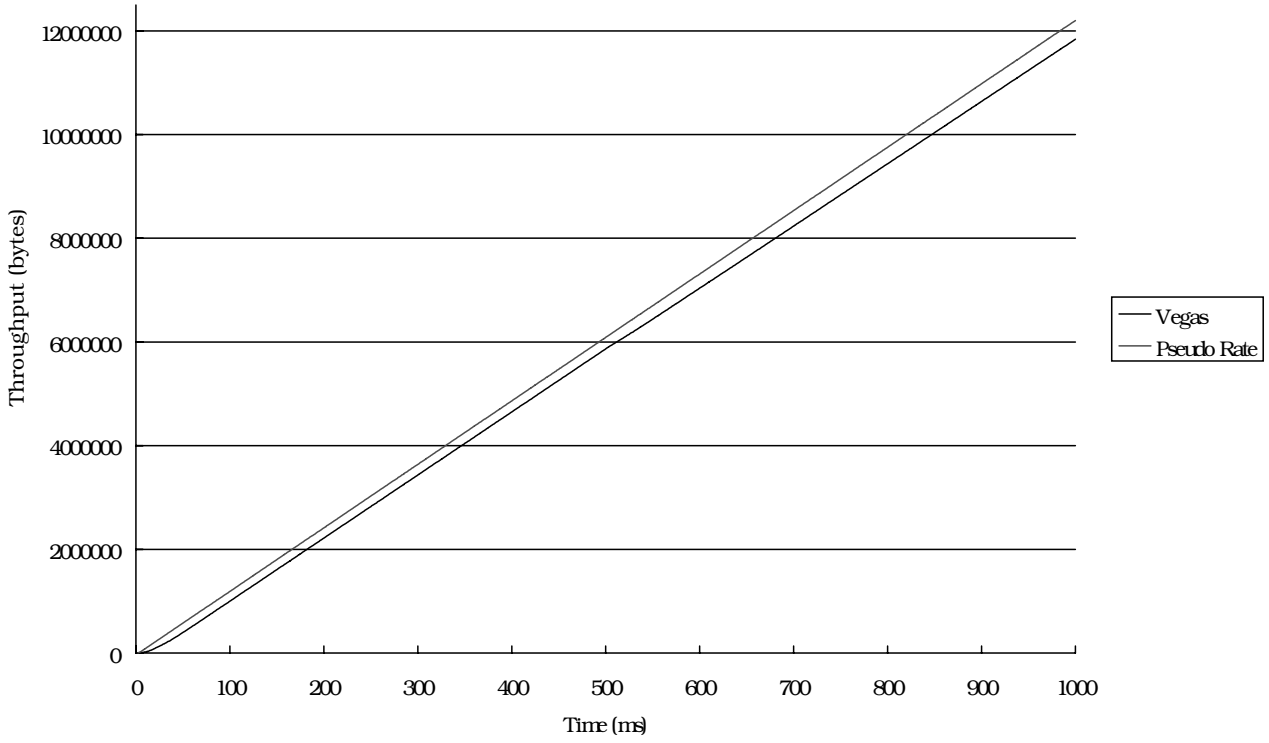

Fig. 5. Throughput comparison of congestion avoidance schemes.

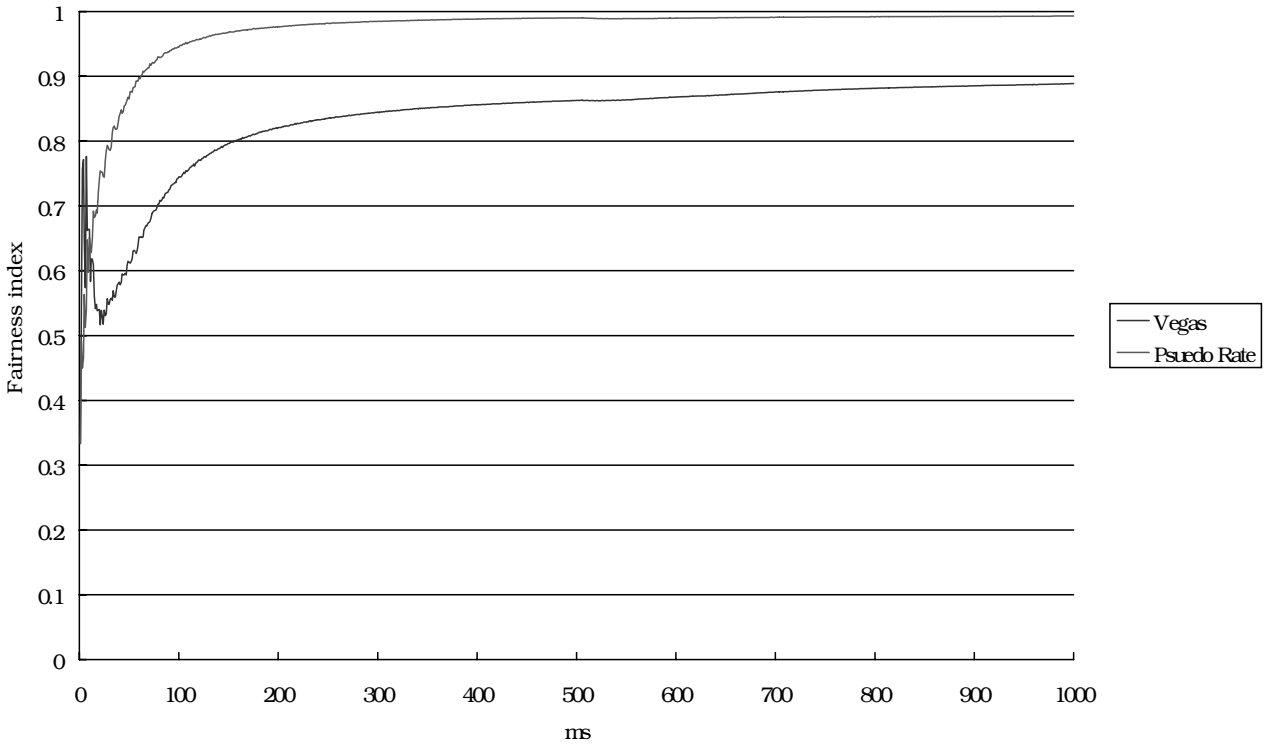

Fig. 6. Fairness index of congestion avoidance schemes.

not good enough. In contrast, the fairness in pseudo-rate TCP is much better than TCP Vegas because its mechanism successfully constrains the window size used by sources with shorter round-trip times.

5. Conclusion

Pseudo-rate TCP is designed based on the rate-based congestion avoidance scheme, with conversion of the window size into the rate information. Therefore, neither precise timing information nor rate-based transmission is required. Further, it retains the advantage of self-clocked characteristic of acknowledgement and achieves a stable and fair window size. The exponential window adjustment shortens the time to reach the allowed window size, this increases the throughput of pseudo-rate TCP, while the fair window-bound setting guarantees the fairness among sources with different round-trip times. Overall, it features simpler implementation with better performance compared to TCP Vegas.

References

[1] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, Sun Microsystems, TCP Selective Acknowledgement Options, RFC2018, October 1996.

[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, pp. 186–197.

[3] J.R. Chen, Y.C. Chen, C.T. Chan, A distributed end-to-end rate control scheme for ABR service, IEEE GLOBECOM98, 1998, p. s1008.

[4] S. Keshav, A control-theoretic approach to flow control, ACM SIGCOMM, August 1991, pp. 189–201.

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

[6] 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.

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

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

[9] L.S. Brakmo, L.L. Peterson, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE Journal on Selected Areas in Communications 13 (8) (1995) 1465–1480.

[10] S. Floyd, T. Henderson, The NewReno modification to TCP's fast recovery algorithm, RFC2582, April 1999.

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

[12] 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, pp. 300–307.