An Enhanced Slow-Start Mechanism for TCP Vegas*

Cheng-Yuan Ho[a], Yi-Cheng Chan[b], and Yaw-Chung Chen[a]
[a] Department of Computer Science and Information Engineering
National Chiao Tung University
[b] Department of Computer Science and Information Engineering
National Changhua University of Education
cyho@csie.nctu.edu.tw, ycchan@cc.ncue.edu.tw, ycchen@csie.nctu.edu.tw

Abstract

In this article, we present a new slow-start variant, which improves the throughput of TCP Vegas, we call this new mechanism Gallop-Vegas which quickly ramps up to the available bandwidth and reduces the burstiness during the slow-start phase. Since TCP (Transmission Control Protocol) is known to send bursts of packets during its slow-start phase due to the fast increase of window size and the ACK-clock based transmission. This phenomenon causes TCP Vegas to change from slow-start phase to congestion-avoidance phase too early in the large BDP (bandwidth-delay product) links. Therefore, in Gallop-Vegas, we increase the congestion window size with a rate between exponential growth and linear growth during slow-start phase. Our extensive simulation results show that Gallop-Vegas significantly improves the performance during the slow-start phase. Furthermore, it is implementation feasible because only sending part needs to be modified.

1 Introduction

With the rapid growth of Internet population and the intensive usage of TCP/IP protocol suite, the TCP congestion control algorithm has become a key factor influencing the performance and behavior of the Internet. Several studies have reported that TCP Vegas [1, 2] provides better performance than TCP Reno with respect to overall network utilization, stability, fairness, throughput, packet loss, and burstiness. Since TCP Vegas uses the difference between the expected and actual flow rates to infer the congestion window adjustment policy from throughput measurements, it usually reduces the sending rate before the connection actually experiences a packet loss.

Although TCP Vegas successfully detects network congestion in the early stage, however, the burstiness during slow-start phase causes Vegas to change from slow-start phase to congestion-avoidance phase too early, especially in a large-bandwidth link with long-delay. Since the sender has no prior knowledge regarding the available bandwidth on the networks, it leads to the abrupt transition of congestion window with exponential growth and transmission of highly bursty traffic from the source, and it in turn would cause buffer overflow at the bottleneck link during the slow-start phase [3, 4, 5, 6, 7, 8]. To solve this problem, certain new techniques, schemes, or refinements of slow-start phase of TCP Vegas have been proposed. In these earlier works, there are three ways to smooth out the burstiness or postpone the time instant of changing from slow-start phase to congestion-avoidance phase. The first approach is selecting γ dynamically to suit various kinds of BDP networks [3], but it needs to estimate the available bandwidth of the network at the steady state. However, to estimate the available bandwidth based on end-to-end congestion avoidance mechanism on a global internet is difficult. Another way is to set the maximum slow-start threshold to avoid buffer overflow and limit the sending rate [4, 5], this not only reduces the throughput of sender but also sets the maximum slow-start threshold to 64 Kbytes. In a large BDP network, this value may be too small and causes Vegas switching to congestion-avoidance phase early. On the other hand, this fixed slow-start threshold may be of no use in a small BDP network. The last method uses a smooth slow-start algorithm to reduce burst data transfer [6]. However, it uses 200 msec timer interrupt to control data transfer and only fits some network topology. Also, using timer interrupt increases the overhead of the operating system. In addition, they consider some network models and prove mathematically to show burstiness in the communication networks [7, 8]. Overall, these works just describe the problems and

*This work was sponsored in part by National Science Council under grant no. NSC 91-2219-E009-036

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

try to characterize them into models,the burstiness problem is still not solved.

In this paper, we propose a modification to the slow-start phase, which changes the increase manner of congestion window to between exponential growth and linear growth, so that a smooth transmission during the slow-start phase can be achieved. We call it Gallop-Vegas. It tries to detect incipient congestion by comparing the measured throughput to the notion of expected throughput. This congestion detection mechanism is same as that in TCP Vegas, except that it calculates every RTT (Round-Trip Time) instead of every other RTT. The congestion window is increased only if these two values are close enough, and the increment of congestion window varies according to the current status of the network.

When TCP Vegas changes from slow-start phase to congestion-avoidance phase, it decreases the congestion window by one-eighth. This may be suitable for small-bandwidth links, but in large-bandwidth links it will slowly reach the available bandwidth. We make a little modification for slow-start phase while similar congestion detection mechanism is still applied.

The implementation of Gallop-Vegas is simple. Only the sending part requires modifications, thus it facilitates incremental deployment in today's Internet. Furthermore, the simulation results reveal that Gallop-Vegas is more efficient than TCP Vegas.

The remainder of this paper is organized as follows. Section 2 describes the details in slow-start phase of TCP Vegas. Section 3 expresses the algorithm and ideas of Gallop-Vegas as well as the possible probe strategies. Section 4 demonstrates the simulation results. Finally, Section 5 concludes the paper.

2 Slow-Start of TCP Vegas

The bandwidth estimation scheme in TCP Vegas [1] is proactive because it tries to avoid rather than react to congestion. Vegas uses the difference in the expected and actual flow rates to estimate the available bandwidth of the network. When the network is not congested, the actual rate is close to the expected flow rate. However, if the actual rate was much smaller than the expected rate, it indicates that the buffer space in the network is going to be filled up and the network is approaching a congested state. This difference in flow rates can be calculated as *Diff = Expected - Actual*, where *Expected* and *Actual* are the expected and actual rates, respectively. If *d* denotes the minimum-observed round-trip time (also known as *BaseRTT*), *D* denotes the actual round-trip time (*RTT*) of a packet, and *W* denotes the congestion window size, then *Expected = W/d* and *Actual = W/D*. The estimated backlog of packets in the network queues can then be computed as

$$\Delta = (Expected - Actual) \times BaseRTT = W \times \frac{(D-d)}{D}. \quad (1)$$

Similar to Reno, Vegas uses a slow-start mechanism that allows a connection to quickly ramp up to the available bandwidth. However, unlike Reno, to ensure that the sending rate does not increase too fast to congest the network during the slow start, Vegas doubles the congestion window size only *every other RTT*. In addition, every other RTT, Vegas calculates the difference in the flow rates (*Diff*) and Δ as given in (1). When $\Delta > \gamma$ (whose default is 1), Vegas leaves the slow-start phase, decreases its congestion window size by 1/8 and enters the congestion-avoidance phase.

The fundamental problem in the slow-start algorithm of Vegas is that doubling its sending rate in short interval causes Δ bias. This characteristic of slow-start may lead to early transition to congestion-avoidance phase and cause severe performance degradation. Under the TCP/IP architecture, it is difficult to estimate the exact available bandwidth of bottleneck link along the end-to-end path. Although Vegas has burst avoidance mechanism that limits the number of segments to be sent at one time (that is, back-to-back) to three segments [2], it still causes burstiness in sending packets. According to the work [6], in short-delay networks, the optimum window size is small and there is no significant difference between RTT and actual data transfer time. Thus, burst data transfer occurs un-apparently in short-delay networks. But in long-delay networks, slow-start leads to burst data transfer which causes congestion and leads to much smaller window size than optimum one. This in turn results in shorter slow-start phase and longer congestion-avoidance phase.

3 Gallop-Vegas

3.1 Motivation

The fundamental problem in the slow-start phase is that the congestion window size increases too quickly. This causes bias of Δ (given in (1)) and performance degradation, as well as leads to long congestion-avoidance phase. In previous works, it either avoids burstiness with some limiting factors or addresses the problems in the slow-start phase, but it did not provide a valid method for various communication networks. Shortening the duration of raising the transmission rate to the available bandwidth improves sender's throughput for different communication network, especially for a short life-cycle connection. For this reason, we propose a modification to the slow-start phase, called Gallop-Vegas.

3.2 The Scheme of Gallop-Vegas

In Gallop-Vegas, we do not increase congestion window size in the first RTT, which is either the beginning of the

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

start or after a retransmission timeout, because we have no idea about this connection. After the second RTT, we start to increase the congestion window with a rate between exponential growth and linear growth, as shown in Fig. 1, we call it *Stable growth*.

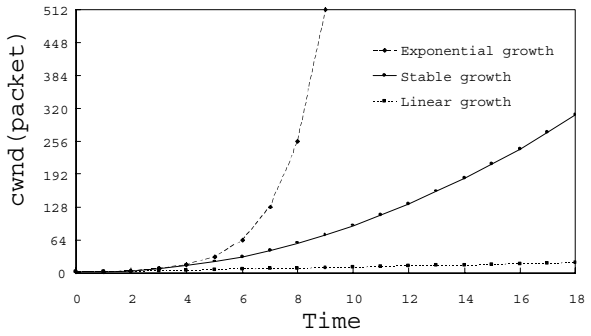

Figure 1. The growth of congestion window

When a sending source increases (or decreases) its congestion window at the n^{th} RTT, the influence to the network can be detected at the $(n+2)^{th}$ RTT. As a result, Vegas calculates the Δ and doubles the congestion window (if it is possible) every other RTT. However, doubling the congestion window size may cause the traffic burstiness in the network, and therefore leads to the congestion-avoidance phase too early. To avoid this phenomenon, we calculate the Δ and increase the congestion window with *Stable growth* (if it is possible). This approach for increasing congestion window is more efficient comparing with Vegas, and the effect can be observed by both simulation and analysis.

Let *'maxincr'* be a dynamic value representing the maximum value of the congestion window increment, and *'incr'* be the current window increment, with value 0 at the beginning of the start or after a retransmission timeout, and no bigger than *'maxincr'*. In order to record the comparison result of Δ with γ, or with β (if Δ is not smaller than γ) at the last RTT, we create a parameter *'status'*. Table 1 shows three sets of value, state, and corresponding motivation at last RTT of *'status'*, whose default value is 0. We choose β (whose default is 3) to compare with Δ because the router is allowed to queue a number, which is between α (whose default is 1) and β, of ackets in Vegas.

In slow-start phase, we let *'maxincr'* be the current congestion window size first, and then compare Δ with γ. When Δ is smaller than γ, we add the value of *'incr'* to the current congestion window size, then *'incr'* is increased by one until it is no smaller than *'max-incr'*. At last we set *'status'* to zero to represent this state. The parameter *'incr'* is increased step by step as long as there is enough bandwidth in the network. While Δ is no smaller than γ, we compare Δ with β to adjust those parameters.

Table 1. Value, state, and corresponding motivation at last RTT

value	state	corresponding motivation at last RTT
0	(1) $\Delta < \gamma$	*'incr'* is increased by one.
	(2) $\gamma \leq \Delta < \beta$	Do not do any action.
1	$\gamma \leq \Delta < \beta$	*'incr'* is decreased by one half
2	$\beta < \Delta$	The congestion window size is decreased by the sum of *'incr'* and surplus of queue (Δ - β).

While Δ is smaller than β, the addition of congestion window size is *'incr'*. Nevertheless, we modify *'incr'* according to the state, which is represented by the parameter *'status'*, at last RTT before increasing the congestion window. Since it exceeds the lower bound of packets queued in the router(s) for a connection, it should slow down the increment. For another reason, it may exceed the available bandwidth without slowing down the increment. If the *'status'* is zero, we have to do three steps. First, *'incr'* is decreased by one half in order to slow down the growth. Second, if *'incr'* is no bigger than one, set it to one, and *ssthresh* (slow-start threshold) to two in order to transit to the congestion-avoidance phase at next RTT. At last, *'status'* is marked as one to represent entering this state at either the first time or the odd number of times. On the contrary, if *'status'* is one, we just change it to zero when continuously getting into this state at the even number of times. Since we decreased the *'incr'* by one half at last RTT, we do not know the influence of network yet. If we still decreased the *'incr'* by one half at this RTT again, we may get just the opposite. In other words, it may be too quickly to decrease *'incr'* by one half while Δ is between γ and β.

If Δ is no smaller than β, we cut the congestion window size down by the sum of the increment at last RTT and surplus packets of queue, which is Δ - β. Then we set *'status'* to two for avoiding repeatedly decreasing the congestion window size when changing to the congestion-avoidance phase at the first time. We call the increasing of congestion window (the increment of *'incr'*) *Stable growth* here after.

We only change one action of Vegas when it gets into congestion-avoidance phase at the first time. If *'status'* is two, we just set *'status'* to zero without changing the congestion window because we don't know the influence of network at last RTT yet. Otherwise, we perform the same action as in Vegas. In retransmission-timeout phase, we reset these three parameters to their default values.

In summary, Vegas sends two packets back-to-back

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

when it receives one ACK (acknowledgement). This may cause bursty traffic if a lot of ACKs return to the sender consecutively. This may cause Vegas turning to congestion-avoidance phase early and then congestion window grows up slowly. On the other hand, Gallop-Vegas transmits one packets when receiving one ACK, and it sends an extra packet to increase the congestion window size after getting two or more ACKs. Through this method, we smooth out the burstiness without using timer. Since comparing with congestion window, the increment is always smaller, therefore Gallop-Vegas reduces the burstiness in transmission and achieves a long slow-start phase. Thus the throughput of Gallop-Vegas grows up faster and is much larger than Vegas.

3.3 Pseudo Code of Gallop-Vegas

The following pseudo code represents the aforementioned statements regarding the *Stable growth*.

```
In Slow-Start phase
  maxincr = cwnd;
  if (Δ > γ)
    if (Δ >= β)
        cwnd -= (last increment of cwnd + Δ - β);
        ssthresh = 2;
        if (cwnd < 2) cwnd = 2;
        status = 2;
    else
        if (status == 0)
          incr = int(incr/2);
          if (incr <= 1)
            ssthresh = 2;
            incr = 1;
        else status = 0;
                cwnd += incr;
  else
    cwnd += incr;
    if (incr < maxincr) incr += 1;
    status = 0;
```

where cwnd is the congestion window size, and ssthresh is the slow-start threshold.

```
In Congestion-Avoidance phase
  if (status != 2)
    Do the motion of original Vegas
  else
    status = 0;

In Retransmission-Timeout phase
  maxincr = 2; (initial congestion window size)
  incr = 0;
  status = 0;
```

4 Performance Evaluation

4.1 The Simulation Setup

The simulation experiments are conducted using *ns2* [9], version 2.26, developed at Lawrence Berkeley National Laboratories (LBNL). Suppose that there is no packet loss in the simulation. The simulation network topology of one single link is shown in Fig. 2, where S1 represents a sender host, whose algorithm is either Vegas or Gallop-Vegas. The type of service used in our simulation is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets, which are 1000 bytes. R1 and R2 represent two finite-buffer gateways. The buffer size at each gateway is set to 100 packets. For the constant-load experiment, drop-tail gateways with FIFO service are assumed. The bandwidth of access links are 1Gbps, and propagation delays are 1ms.

The bandwidth of connection link is X-Mbps, where X is 1.5, 5, 10, 25 or 50, and propagation delay is Y-ms, where Y is 3, 8, 23, 28 or 48. The combinations of X and Y generate 25 networks. Although we have used different bandwidths and propagation delays of connection links, only the simulation results of X = 50, and Y = 48 are presented here. Other values of X and Y will be shown in the figure of convergence time. We choose these values of communication network to represent small-bandwidth, large-bandwidth, short-delay, and long-delay, respectively. The sender uses the slow-start at the start of a connection, and/or after a retransmission timeout, and hence it features similar behavior during slow-start phase.

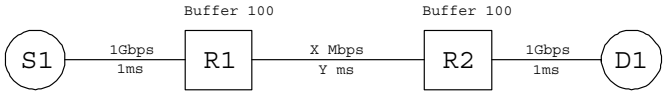

Figure 2. Simulation topology used for Vegas or Gallop-Vegas experiments

In following sections, we will show the simulation result, convergence time with different BDPs of communication networks, and ten senders with the same algorithm sharing a common bottleneck of 100 Mbps bandwidth and 48 ms propagation delay.

4.2 Simulation Result

We compare Gallop-Vegas with Vegas which uses two different parameter values, one with γ one, and the other with γ three (as β). Since Gallop-Vegas changes from slow-start phase to congestion-avoidance phase when Δ is no

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

smaller than β. In Fig. 2, X is 50 and Y is 48, it means that the bandwidth of bottleneck link is 50 Mbps, and the end-to-end propagation delay is 48 ms. Fig. 3 and Fig. 4 show the congestion window size and throughput between Vegas and Gallop-Vegas, respectively. We can observe that the performance of Gallop-Vegas is better than Vegas. Both varieties of Vegas turn to congestion-avoidance phase early, one is at 1.2 seconds and the other is at 1.5 seconds, and they increase congestion window through linear growth. They spend more than 50 seconds (which almost equals 500 RTTs) to reach the available bandwidth. However, Gallop-Vegas switches to congestion-avoidance when it reaches the available bandwidth at 11.4 seconds, and only spends 0.4 seconds (which approximately equals to 4 RTTs) to reach the real available bandwidth. The maximum number of queueing packets in these algorithms are almost the same.

An interesting phenomenon in the simulation was observed. Both of Vegas lose packets but Gallop-Vegas does not when the buffer size of the router is decreased. There is an example with the router buffer size 30. The congestion window size and throughput between Vegas and Gallop-Vegas are described in Fig. 5 and Fig. 6, respectively. In this environment, the router R1 drops three packets of Vegas at 1.1 seconds because both of Vegas double the congestion window size from 32 to 64. This causes a bursty traffic to a router, which could not handle these packets in time. The same situation happens at 1.6 seconds, where router R1 drops sixteen packets. Since Vegas starts the fast retransmission procedure to redeem the lost packets at 1.1 seconds, after that, Vegas doubles the current window size contin-

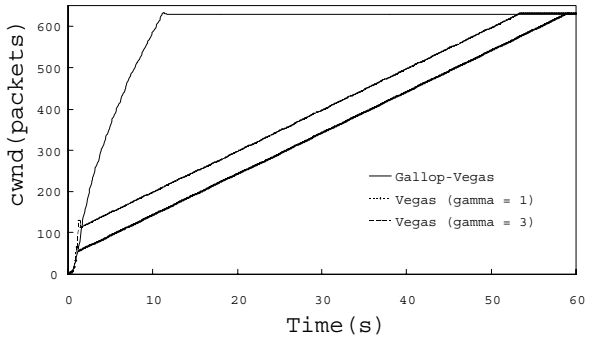

Figure 3. Congestion window size comparison between Vegas and Gallop-Vegas with 50 Mbps bottleneck bandwidth, and 48 ms link propagation delay.

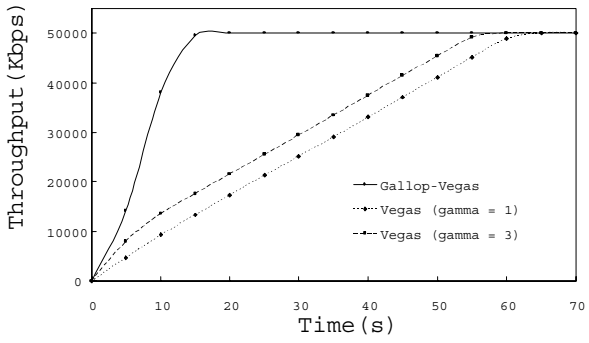

Figure 4. Throughput comparison between Vegas and Gallop-Vegas with 50 Mbps bottleneck bandwidth, and 48 ms link propagation delay.

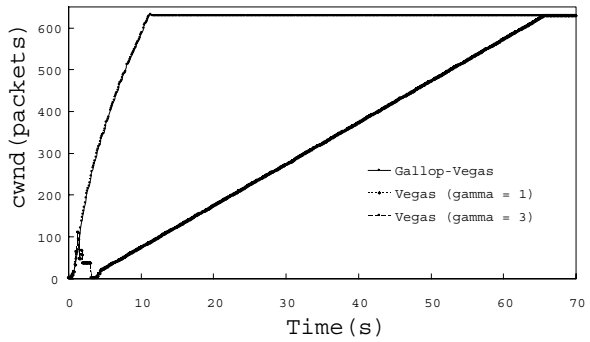

Figure 5. Congestion window size of Vegas and Gallop-Vegas. There are packet lost in both of Vegas. (Buffer size = 30)

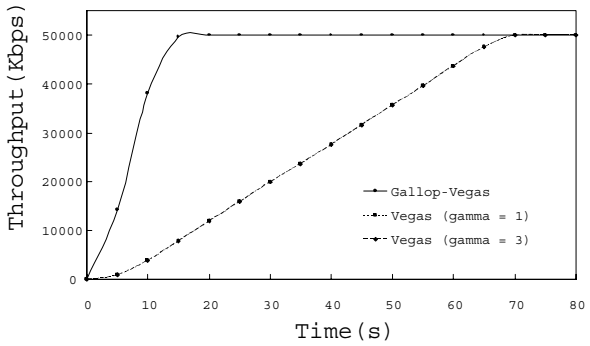

Figure 6. Throughput of Vegas and Gallop-Vegas. There are packet lost in both of Vegas. (Buffer size = 30)

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

uously, this action causes the packet loss because of the burstiness again. Since there are too many packets losses at this time, Vegas has to wait a retransmission timeout for a long time. On the other hand, Gallop-Vegas increases the congestion window size with *Stable growth*, so it does not cause a large burstiness. It could increase congestion window size steadily during the slow-start phase.

Now, we use the convergence time [3] with different BDPs of communication networks to compare Gallop-Vegas with two varieties of Vegas. The result is shown in Fig. 7. We can see that the convergence time of Gallop-Vegas grows slowly (or linearly) while BDP increases quickly. However, the convergence time of both Vegas varieties climbs very fast. The convergence time of Vegas is five times more than that of Gallop-Vegas at 5000Kb. We could conclude that Gallop-Vegas is as good as Vegas in the small BDP and much better than Vegas in the large BDP with the demonstration in Fig. 7.

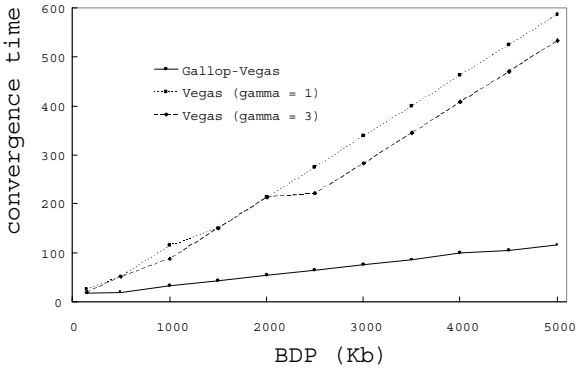

Figure 7. The convergence time with different BDPs of communication networks.

4.3 Multiple Senders in One Network

After comparing one sender in the same network topology, we compare the cases of multiple senders with the same algorithm of Gallop-Vegas and Vegas. The simulation network topology used is shown in Fig. 8, and the whole skeleton is same as that in Fig. 2. The difference between Fig. 8 and Fig. 2 is that there are more senders, bigger buffer size, and larger bottleneck bandwidth in Fig. 8.

The utilization of the bottleneck link is shown in Fig. 9. As seen in this figure, Gallop-Vegas utilizes the bandwidth of bottleneck link more efficiently than Vegas. One interesting observation is that doubling congestion window causes bursty traffic and makes all senders turn into congestion-avoidance phase at the same time when Vegas's γ is three. This phenomenon is fair to all senders, however, the utilization of the available bandwidth is inefficient.

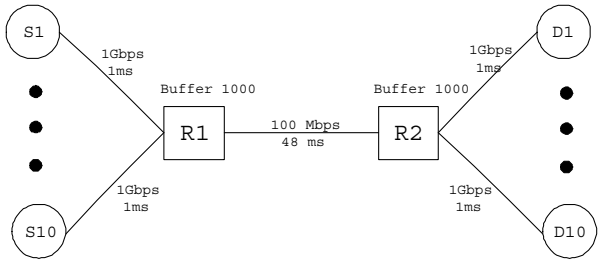

Figure 8. Simulation topology with multiple sender used for Vegas or Gallop-Vegas experiments.

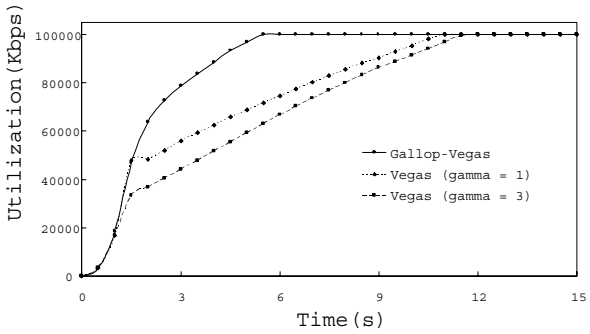

Figure 9. The utilization of bottleneck link between Vegas and Gallop-Vegas.

5 Conclusion

We propose and evaluate a new variant of the slow-start mechanism in TCP Vegas, called Gallop-Vegas, to reduce the burstiness, to raise the rate to the available bandwidth in shorter time, and to improve the start-up performance. In this work, we achieve more efficient throughput in slow-start phase comparing with original TCP Vegas. Although Gallop-Vegas is more suitable for large bandwidth or long-delay networks, it still increases transmit performance in small bandwidth or short-delay networks. The design of Gallop-Vegas is simple and implementation feasible on existing operating systems. We will combine the improvement of congestion-avoidance phase and show the fairness in the future.

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

References

[1] L. Brakmo and L. Peterson, 'TCP Vegas: End-to-end Congestion Avoidance on a Global Internet', *IEEE J. Sel. Areas Commun., VOL. 13, NO. 8, pp. 1465-1480, October 1995*

[2] U. Hengartner, J. Bolliger, and Th. Gross, 'TCP Vegas Revisited', *in Proc. of IEEE Infocom 2000, pp. 1546-1555, March 2000*

[3] S. Vanichpun and W. Feng, 'On the Transient Behavior of TCP Vegas', *11th IEEE International Conference on Computer Communications and Networks (ICCCN'02), Miami, Florida, pp. 504-508, October 2002*

[4] H. Wang, H. Xin, D. S. Reeves, and K. G. Shin, 'A Simple Refinement of Slow-start of TCP Congestion Control', *IEEE Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on, pp. 98-105, 3-6 July 2000*

[5] H. Wang and C. Williamson, 'A New Scheme for TCP Congestion Control: Smooth-Start and Dynamic Recovery', *Proceedings of MASCOTS'98, Montreal, Canada, pp. 69-75, July 1998*

[6] Y. Nishida, 'Smooth Slow-Start: Refining TCP slow-start for Large-Bandwidth with Long-Delay Networks', *23rd. Annual Conference on Local Computer Networks October 11 - 14, 1998 Boston, Massachusetts, pp. 52-60*

[7] D. Starobinski and M. Sidi, 'Stochastically Bounded Burstiness for Communication Networks', *Information Theory, IEEE Transactions on, Vol. 46, pp. 206-212, January 2000*

[8] T. Konstantopoulos and V. Anantharam, 'Optimal Flow Control Schemes that Regulate the Burstiness of Traffic', *Networking, IEEE/ACM Transactions on, VOL. 3, pp. 423-432, August 1995*

[9] http://www.isi.edu/nsnam/ns/

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE