Performance Degradation of TCP Vegas in Asymmetric Networks And Its Remedies

Chengpeng Fu, Ling Chi Chung, and Soung C. Liew
The Chinese University of Hong Kong,
Shatin, N.T. Hong Kong

***Abstract* - TCP Vegas employs congestion avoidance, early detection of packet loss and conservative slow-start algorithms to improve TCP performance. With its proactive congestion detection, it utilizes network bandwidth more efficiently and achieves a higher throughput than TCP Reno. This paper shows that in asymmetric networks in which the bottleneck is on the reverse path rather than on the forward path, its performance can be significantly lower than that of Reno, in contrast to the 37% throughput improvement claimed in [4][5]. In particular, Vegas may erroneously converge to an operating region in which the available bandwidth on the forward path is under-utilized by a large margin. Even worse, when connections running Vegas and Reno co-exist and compete on the same network, Vegas suffers a severe penalty. We propose an approach to improve Vegas' throughput in asymmetric networks.**

I. INTRODUCTION

TCP Reno [1][2], which makes use of slow start and congestion avoidance algorithms, is now widely deployed in the Internet. Its window is increased until packet loss is experienced, at which point the window is halved and then a linear increase algorithm takes over until further packet loss is experienced. This additive increase and multiplicative decrease leads to periodic oscillations in the congestion window, round trip delay and queue length of the bottleneck buffer in the path. Researchers [3-5] have proposed various modified TCP algorithms to eliminate the periodic oscillations of Reno [1].

In 1994, TCP Vegas [4][5], which employs a fundamentally different congestion control algorithm from that in Reno, was proposed and a claim of 37 to 71 percent throughput improvement over Reno was made. The author in [6] reproduced the claims made in [4][5] and showed that Vegas indeed offered higher throughput than Reno while reducing packet loss and the need for retransmission. Other researchers [11][12] also demonstrated that TCP Vegas outperformed Reno by means of analysis as well as simulation.

Both TCP Reno and Vegas were designed under the implicit assumption that congestion occurs on the forward path rather than the reverse path. But, in today's Internet, it is common for TCP's throughput to be limited by the congestion on the reverse path. In asymmetric networks such as ADSL and HFC, the capacity of the reverse path is significantly lower than that of the forward path.

As defined in [9], we say that a TCP connection experiences asymmetry if the ratio of the forward data rate (data packets per second) to the reverse data rate (acks per second) is larger than one [9]. TCP Reno depends on the reception of acks for pacing of the forward data-sending rate. If acks do not arrive at the sender fast enough, the data-sending rate will be throttled even if the forward path is congestion free, resulting in under-utilization of the available bandwidth [8][9]. The performance of Reno's connections on asymmetric networks was studied in [9]. Other researchers [7][8] also proposed several schemes to improve Reno's performance under asymmetry by reducing ack rate on the reverse path.

In this paper, we study TCP Vegas' performance in asymmetric networks and point out that due to Vegas' proactive congestion control mechanism, Vegas erroneously converges to an operating region in which the available bandwidth on the forward path is under-utilized by a large margin. Vegas' throughput is significantly lower than Reno's throughput in asymmetric networks. In contrast, while Reno also suffers from network asymmetry, the performance degradation is less severe. To circumvent this problem, we modify Vegas congestion avoidance mechanism to steer the algorithm to converge to an optimal point at which the forward path is fully utilized, even on asymmetric networks. We also show in this paper that Vegas' connections experience a large negative bias when competing with Reno connections for bandwidth on the bottleneck link.

This paper is organized as follows. In Section II, we give a brief review of the basic TCP Vegas algorithm. Previous work related to Vegas is presented In Section III. In Section IV, we analyze TCP Vegas in asymmetric networks and identify the cause for Vegas' performance degradation. In addition, we study the multiple-connection scenarios in which Vegas connections compete with Reno connections, and demonstrate that Vegas suffers very severe penalty when it co-exists with Reno. In Section V, a solution to the Vegas' poor performance in asymmetric networks is proposed. Section VI concludes this work.

II. TCP VEGAS MECHANISM

Reno's congestion control kicks in only when packets are lost, and this causes periodic oscillations of the congestion window size and leads to throughput degradation of the connection. Vegas, on the other hand, employs proactive

congestion detection and avoids congestion by steering the system away from packet loss before it occurs.

Intuitively, as the TCP window size (*cwnd*) increases, the throughput of a connection should also increase. If there is no congestion, the measured throughput should be close to the expected throughput; otherwise, the measured throughput will be smaller than the expected throughput. Vegas attempts to predict the onset of congestion by monitoring the difference between the measured throughput and the expected throughput, namely,

DIFF = (Expected-Actual)

In the above, *Expected = cwnd/BaseRTT*, where *BaseRTT* is the minimum of all measured *RTT* (round trip times). It is usually the *RTT* of the first segment sent by a connection; *Actual* is the measured throughput at the sender given by *cwnd/RTT*, where *RTT* is the actual round-trip time of a tagged packet. Strictly speaking, *Expected* as defined is the best possible throughput, since *BaseRTT* is the minimum of all measured *RTT*. But we shall adhere to the use of this term since previous papers on Vegas have all used it.

The essence of Vegas is to increment *cwnd* when *DIFF*BaseRTT* is smaller than a preset value, and decrement it when *DIFF *BaseRTT* is larger than another preset value, as detailed below:

```
if  ( DIFF*BaseRTT < α )
      cwnd = cwmd + 1
      /* increase congestion window size by one */
else  if  ( DIFF*BaseRTT > β )
      cwnd = cwnd – 1
      /*decrease congestion window size by one */
else  cwnd =  cwnd
       /* congestion window remains unchanged */
```

where α and β are constant values in packet unit that can be set by experimentation.

Vegas employs a new retransmission strategy [4] for detecting packet loss earlier than Reno and decreases *cwnd* multiplicatively using a factor of ¾ rather than Reno's ½ . Vegas also modifies Reno's slow-start: it halves the slow-start growth rate of Reno and doubles congestion window every other *RTT*, as opposed to Reno's every *RTT*.

III. PREVIOUS WORK ON TCP VEGAS

Since Brakmo [4][5] proposed TCP Vegas in 1994, claiming to achieve larger throughput and one-fifth to one-half the losses of TCP Reno, several papers [6][11-13] have been published on the study of Vegas performance using fluid analysis and simulation approach. In this Section, we will give a short overview of previous work on TCP Vegas.

Ahn [6] reproduced claims in [4][5] with varying background traffic and concluded that Vegas indeed offers improved throughput of at least 3-8% over Reno while reducing packet losses and subsequent retransmitted segments by a factor of 2 to 5. Ahn also pointed out that Vegas' congestion avoidance is the main source of Vegas' improved throughput, efficiency and delay statistics. Its early packet loss detection technique contributes only second-order effects on TCP performance.

Hasegawa [11] studied fairness and stability of Vegas analytically and found that Vegas can offer higher performance and stable operation, but sometimes fail to be fair due to the uncertainty of the convergence point among different *RTT* connections. Nevertheless, TCP Vegas indeed improves fairness to some extent when compared to TCP Reno.

J. La [13] observed TCP Vegas experienced two problems, which could have serious impact on its performance. One is rerouting problem. Packets routing a different path may result in the changes of variables *BaseRTT* and *RTT*, and therefore bring about inaccurate estimation of *DIFF* and thus erroneously increase or decrease Vegas' *cwnd*; another is stability problem which is still from the estimation of *BaseRTT*. If the connections overestimate *BaseRTT* due to a persistent congestion, system may be driven to a persistently congested state. Paper [13] has investigated these issues in detail.

Mo [12] used a fluid model and simulations to show that Vegas does not suffer bias from different propagation delay as TCP Reno does, and achieve better performance than Reno. At same time, he also demonstrated that TCP Vegas does not compete well when it co-exists with TCP Reno.

Positive impacts of Vegas' throughput and fairness have been fully investigated in previous work [4-6][11]. Negative impacts such as incompatibility with Reno and lack of robustness have also been pointed out in [6][12-13]. In this paper, we show that in asymmetric networks, both Vegas' throughput and compatibility with Reno can degrade by a large margin. This is fundamentally due to Vegas' proactive congestion control mechanism [4][5]. Comparatively, Reno shows less degradation over asymmetric networks. In addition, one modified congestion avoidance mechanism is proposed to improve Vegas' throughout in asymmetric networks

IV. ANALYSIS AND SIMULATION RESULTS

We introduce one simulation model in Section IV.A. In Section IV.B and IV.C, we argue that the proactive

congestion control employed by Vegas will give rise to the poor throughput performance in asymmetric networks and back up our contention with simulation results. In Section IV.D, we investigate the relative performance of TCP Vegas and Reno connections when they co-exist in asymmetric networks.

A. Network Model

We consider the network model depicted in Fig. 1. The network consists of six hosts, two intermediate routers, and links interconnecting the hosts and routers. The links are labeled with their capacities and propagation delays. The forward link between the two routers has a capacity of μ_f data packets per second and a propagation delay of τ_f seconds, together with a FIFO buffer of size B_f packets. The reverse link can transmit μ_r acks per second with propagation delay τ_r seconds and a FIFO buffer that can hold B_r acks.

Assuming the source has infinite data to send, when the receiver receives a data packet, it generates an ack that contains the sequence number of the next expected packet. Typical data packet size is 1kbyes and typical ack size is 40 bytes. The normalized asymmetric factor [9] is defined as $k = \mu_f / \mu_r$. An asymmetric network is one in which $k > 1$. For example, if the forward link has a bandwidth of 2.4Mbps and the reverse link has a bandwidth of 32kbps, and the size of the data packets and acks are respectively 1kbytes and acks of 40 bytes. Then, μ_f = 300 packets/s, μ_r = 100 acks/s, and k =3.

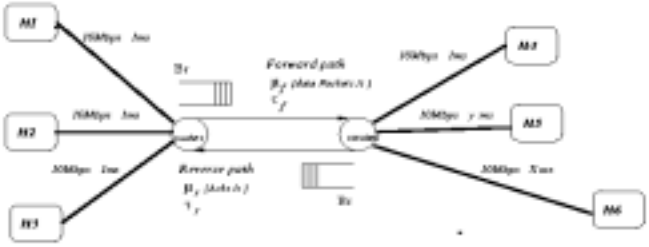


Fig. 1. Asymmetric Network Model

In Sections IV and V, we make use of the network simulator (*ns*) developed at Lawrence Berkeley Lab to investigate TCP Vegas in asymmetric networks based on the model in Figure 1.

B. Analysis on Vegas in Asymmetric Environment

Suppose that the traffic on all the links is generated by the single connection between the source and destination. With reference to Fig. 1, the *maximum* forward path throughput (in unit of packets per second) is the forward link capacity μ_f [packets/s]. The *maximum* reverse path throughput (in unit of acks per second), which is equal to the reverse link capacity μ_r [acks/s].

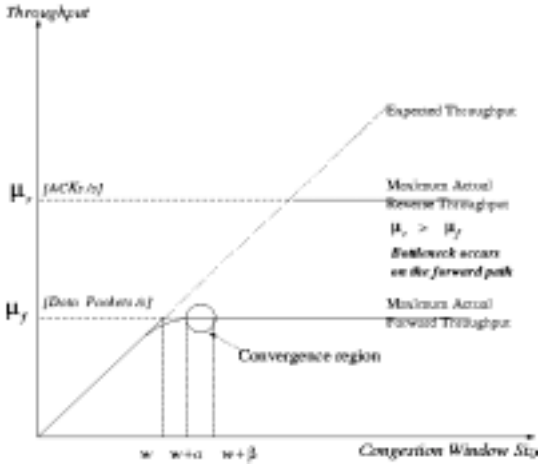


Fig. 2. Window control of TCP Vegas

Let us consider a non-asymmetric network with $k \leq 1$ (i.e., $\mu_f < \mu_r$). In Figure 2, we show conceptually the throughput versus *cwnd* curve of the TCP connection. The actual throughput *cwnd/RTT* keeps up with the expected throughput *cwnd/BaseRTT* when *cwnd* is small because every updated *RTT* is nearly equal to *BaseRTT*, which is the *RTT* of first segment sent by the connection. As *cwnd* increases, the actual throughput increases proportionally until it reaches the forward link capacity μ_f, after which the increase stops. Beyond this threshold, a queue starts to build up at the bottleneck router buffer and *RTT* starts to move up in value. If the queue is allowed to overflow, packet loss will be incurred. Vegas' main idea is to try to maintain the extra data (queue size) in the bottleneck router buffer to be between the lower threshold (α) and upper threshold (β) in order to reach high throughput and avoid packet overflow.

To see how Vegas attempts to maintain the queue size at the bottleneck link at the range between α and β, consider the following. When the queue size is zero, the RTT is *BaseRTT*. *BaseRTT* is basically the sum of the transmission delays and propagation delays throughout the forward and reverse paths (with queuing delay equal to zero). Let the backlog at the queue be denoted by *N*. Then, given the forward link between the routers is the bottleneck link with capacity μ_f, we have

$$RTT = BaseRTT + N/\mu_f \,. \qquad (1)$$

Now, μ_f is estimated by *cwnd/RTT* in Vegas. Thus, *N* can be approximated by

$$N = cwnd * (1 - BaseRTT/RTT) = DIFF * BaseRTT. \qquad (2)$$

When the estimated $N > \beta$, Vegas decrements *cwnd* to make *N* smaller; when the estimated $N < \alpha$, Vegas increases

cwnd to make *N* larger; otherwise, leave *cwnd* unchanged. Thus, we see that Vegas attempts to keep *N* between α and β.

We can rewrite (1) as

$$\mu_f * RTT = \mu_f * BaseRTT + N \quad (3)$$

In Fig. 2, we define $w = \mu_f * BaseRTT$, which is the *cwnd* just before backlog starts to build up at the bottleneck link. Substituting $\mu_f = cwnd/RTT$ on the left side of (3), we have

$$cwnd = w + N \quad (4)$$

We see that the Vegas attempts to keep *cwnd* to between $w+\alpha$ and $w+\beta$, as seen in Fig. 2.

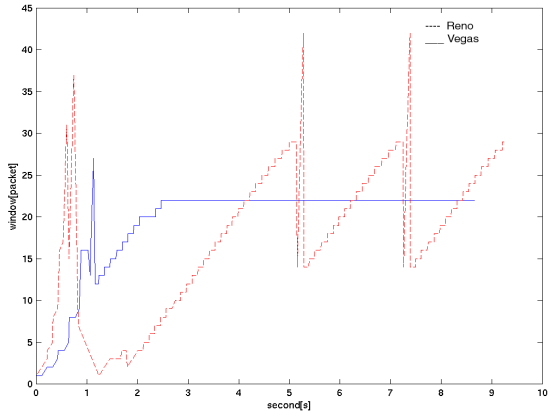
In the case of an asymmetric network with $k > 1$, the bottleneck link is the reverse link but not the forward link. Then, we have $RTT = RTTBase + N/\mu_r$, where *N* is now the ack backlog at the queue of the reverse link. Again, μ_r can be estimated by *cwnd/RTT*. Now, Vegas' algorithm will zoom in to an equilibrium state where by *N* ack fluctuates between α and β, and the data flow rate will be μ_r packets per second. Meanwhile, there is some extra capacity, $(\mu_f - \mu_r)$, at the forward link that is left unused. This under-utilization of the forward link is caused by bandwidth limitation at the reverse link and in principle the sender should be able to send more data in the forward direction. Vegas does not distinguish between bottlenecks in the forward link and reverse link and zooms into a sub-optimal solution in this case. Its proactive congestion control algorithm is actually attempting to prevent ack loss rather than packet loss.

Reno, unlike Vegas, is not a proactive congestion control scheme. It will attempt to send more packets until packet loss is detected. When the bottleneck is at the reverse link, acks will be lost rather than the data packets in the forward direction. However, intermittent ack loss will not cause Reno to throttle its data-sending rate in the forward direction significantly because of the cumulative property of acks. For instance, if ack for packet 2 and packet 3 are dropped at the bottleneck, but ack for packet 4 is received, the sender will correctly take this to mean that packets 2 and 3 have also been received. Thus, it will continue to increase its *cwnd*.

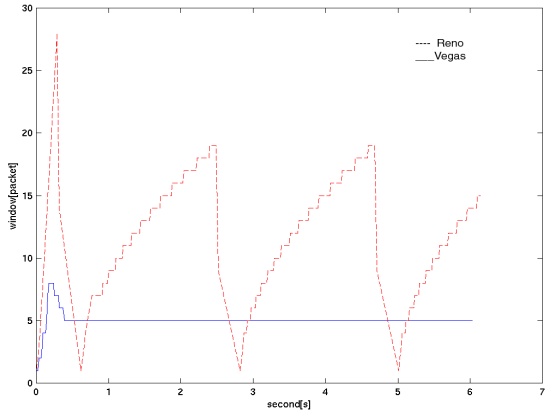
C. TCP Vegas Simulation in Asymmetric Networks

In this section, we use simulation to demonstrate the performance degradation of TCP Vegas in asymmetric networks and show how the reverse buffer size affects TCP Vegas. We point out the main reason for Vegas degradation is its proactive congestion detection mechanism. These simulation results back up our analysis above.

1) *TCP Vegas cwnd evolution and its throughput in asymmetric networks*: Fig. 3(a) plots Reno and Vegas' congestion window evolution with $\tau_r=\tau_f$ =50ms for $k \leq 1$ (i.e., when the forward path is the bottleneck). It shows that Vegas' congestion window remains rather constant while Reno's congestion window fluctuates. Although during the initial slow-start phase, Vegas performs better than Reno, the averages of the window size, however, are comparable in equilibrium. Table I shows TCP Vegas and Reno throughput and forward link utilization under the network model in Figure 1.



(a)



(b)

Fig. 3. Reno's and Vegas' congestion window evolution (a) for k=0.5 (b) k=3

Figure 3(b) plots single Reno and single Vegas' congestion window evolution with $\tau_r=\tau_f$ =50ms for $k = 3$ (now the reverse path is the bottleneck of a TCP connection). After a slow start period, Vegas' congestion window still converges to an equilibrium point within $[\, w+\alpha,\ w+\beta\,]$, where $w = \mu_r * BaseRTT$. It has been shown analytically in [9] that Reno's congestion window oscillates over a wide range

[1, $w_{renomax}$]. The low bound of the range is due to multiple packet losses, which cause a succession of window cutback. The upper bound $w_{renomax} = \mu_f * RTT + kB_r + B_f$, where $\mu_f * RTT$ is packets in flight of the connection, B_f is the forward buffer size and B_r is the reverse buffer size.

By contrast, in the asymmetric network (k>1) ack packets accumulate on the reverse buffer because of its slow link speed. As soon as Vegas detects the onset of the connection congestion, its window will converge to this sub-optimal equilibrium region without distinguishing whether actual congestion occurs on the data packet path or not. Therefore, ack packets accumulates in the reverse buffer and while the forward buffer is empty. Unlike Vegas, Reno's *cwnd* keeps increasing until data packet loss actually occurs on the forward path. The cumulative property of acks and aggressive window increase of Reno compensates somewhat for limitations of asymmetric network and provide Reno with much flexibility and adaptability in this environment. Meanwhile, TCP Vegas is constrained by its proactive congestion detection.

We see from the simulation results in Table I that although Reno's window experiences fluctuations as illustrated in Fig. 3 (b), its average is still higher than the window in Vegas. Table I shows changes of Reno's and Vegas' throughput and utilization with the different asymmetric degree. We see that in non-asymmetric network, TCP Vegas indeed outperforms Reno and its forward utilization is higher than Reno [4-6]. In the asymmetric environment, however, it is quite different. Vegas' throughput deteriorates more seriously than Reno's and its forward link utilization is lower than Reno.

TABLE I
THROUGHPUT AND UTILIZATION OF TCP VEGAS AND RENO
GIVEN μ_f=1.6Mbps, B_f=15 packets B_r= 10 acks $\tau_f = \tau_r$ =50ms

	VEGAS/RENO			
	Non-asymmetry	Asymmetry		
	k=0.5	k=3	k=5	k=7
Throughput (packet/s)	195.9/185	65.8/138.3	43.8/109.4	26.8/71.5
Throughput ratio	1.0/1.0	0.34/0.75	0.22/0.59	0.14/0.38
Forward link utilization	0.98/0.93	0.31/0.69	0.19/0.54	0.14/0.36

In order to better understand the trend of asymmetry effect, Fig. 4 shows Vegas' and Reno's throughputs as a function of μ_f for a fixed reverse link speed μ_r = 50acks/s. The system model considered is that in Fig. 1 with link delay $\tau_r = \tau_f$ =50ms. Vegas throughput is $min\{\mu_f, \mu_r\}$. When μ_f is much larger than μ_r, the forward-link capacity is heavily under-utilized. The simulation shows that for Reno, the throughput is nearly μ_f when $\mu_f < \mu_r$, but it can be larger than μ_r when $\mu_f > \mu_r$.

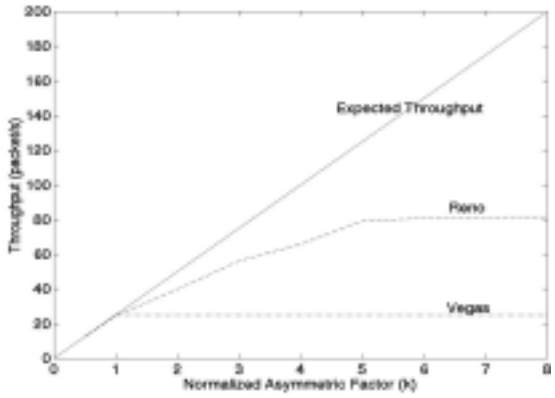


Fig. 4. TCP throughput versus degree of asymmetric network, given μ_r=50 acks/s

We note from Fig. 4 that both Reno and Vegas do not make full use of the capacity in the forward path. The higher the degree of asymmetry, the more severe the throughput penalty. However, Reno shows better adaptability over asymmetric networks than Vegas and thus suffers less.

In summary, TCP Vegas adjusts its transmission rate to match the forward-path capacity proactively to avoid data packets loss. However, in asymmetric networks, its congestion avoidance mechanism actually matches its transmission rate to the reverse-path capacity, underestimating bandwidth; in contrast, TCP Reno passively detects congestion by aggressively increasing its congestion window until data packet loss actually occurs, and the cumulative effect of acks lessens the impact of lost acks.

2) *The reverse buffer size effects on TCP Vegas in asymmetric networks*: In asymmetric networks, Vegas tries to maintain at least α but no more than β acks on the reverse buffer. Table II contains our simulation results that clearly show that TCP throughput varies with the reverse buffer size.

TABLE II
THROUGHPUT AND UTILIZATION OF TCP VEGAS AND RENO~~
REVERSE BUFFER SIZE, GIVEN μ_f= 200packets/s μ_r=50acks/s k=5
B_f=10 $\tau_f = \tau_r$=50ms

B_r (acks)	RENO/VEGAS	
	Throughput (packets/s)	Forward link Utilization
2	134.5/101.3	0.670/0.505
6	128.2/39.4	0.641/0.197
10	109.1/39.6	0.545/0.197
20	66.8/39.5	0.334/0.197
40	46.2/39.5	0.231/0.197
100	46.6/39.4	0.232/0.197

TCP Vegas and Reno performance gets worse for larger reverse buffer size when $B_r > \beta$. The intuition is as follows. For Vegas, small reverse buffer such as B_r=2 can increase the drop rate of acks (α is set to 1 and β to 3 in our simulation).

These missing acks have the effect of increasing reverse link speed. It is as if the receiver actually skipped the sending of some acks. However, when the reverse buffer size is β (usually set to 3) or more, with reference to Table II, Vegas algorithm will prevent ack drop by throttling the transmission of data packets at the sender.

In absolute term, buffer size of β is really small and one would expect most network equipment to have much larger buffer size than this. Thus, we can expect Vegas performance to be very poor in practice unless something else is done.

For comparison, we have also simulated Reno and the results are also shown in Table II. Note that Reno's throughput is also affected by the reverse buffer size. The dropping of acks will slow down the window increase of TCP Reno because Reno's window increase is based on the number of acks received rather than the amount of acks received [8-9]. However, being different from Vegas conservative window-adjusting policy, the aggressive window-increasing policy of Reno can alleviate the asymmetry effect to some extent. Therefore, Reno shows better results than Vegas. From table II, we find that Vegas' utilization of forward link is significantly lower than Reno's, especially when the reverse buffer size is relatively small.

D. Multiple Connections in Asymmetric Networks

Reference [12] argues that Vegas' throughput does not depend on its propagation delay, and unlike Reno, there is no bias in favor of connections with long delays. Paper [11] concludes that Vegas provides fair services among connections independent of different propagation delay. In asymmetric networks, Vegas keeps this fairness property although its total throughput is limited by $min\{\mu_f, \mu_r\}$.

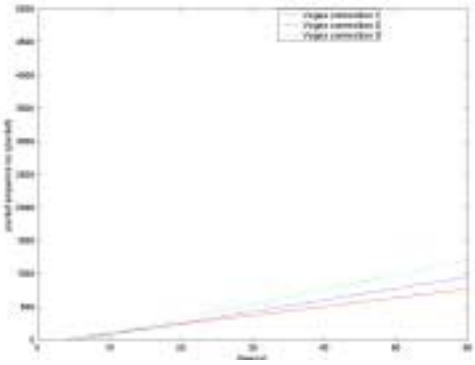
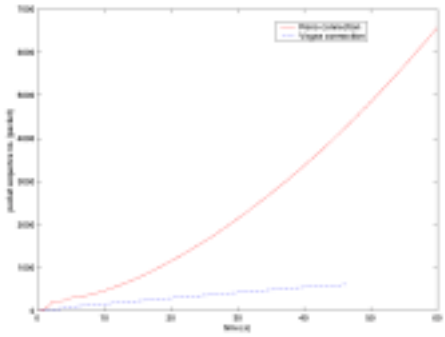


Fig. 5. Three Vegas connections with different RTT given μ_f=200packets/s, k=5 B_f=10 B_r= 10 $\tau_f = \tau_r$ =20ms

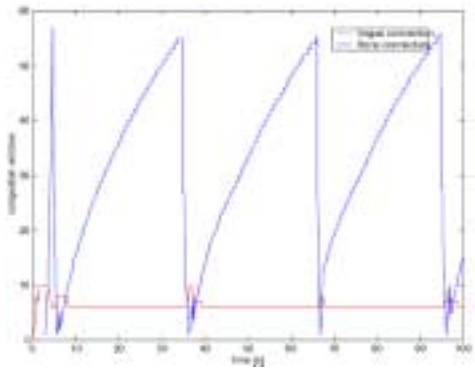
With reference to Figure 5, three connections in the network model in Fig. 1 are started up in sequential order, one from H1 to H4, one from H2 to H5 (with propagation delay y=1ms, one from H3 to H6 with propagation delay x=5ms. It can be seen that the differences of the three connections are smaller in Vegas.

However, Reno TCP has been widely deployed in the current Internet, to better evaluate Vegas evolution in practical environments, we need to study the compatibility between Reno connections and Vegas connections in addition to above compatibility among Vegas connections. Ahn [6] has observed that Vegas does not receive a fair share of bandwidth due to its conservative congestion control mechanism when competing with Reno connections. Mo [12] explained TCP Reno leaves little buffer for other connections because of its aggressive nature while TCP Vegas is conservative and tries to occupy little buffer space.

In asymmetric networks, however, when one Vegas connection is competing with Reno connection. Vegas is so much biased by Reno that it is nearly not able to transmit data packets, as seen in Fig. 6(a), which shows the vast throughput difference between a Reno and a Vegas connection when they co-exist in the same network.



(a)



(b)

Fig. 6. (a) packet sequence number (b) window evolution for co-existing one Reno and one Vegas, given μ_f=200packets/s, k=3 B_f=10 B_r= 10 $\tau_f = \tau_r$ =20ms

Vegas uses its proactive congestion control mechanism to prevent self-induced packet losses. This kind of loss prevention is completely negative in asymmetric environments because Vegas conservatively maintains a smaller ack queue size on the reverse path other than data packets queue on the forward path, seeing to Fig. 6 (b). When one Reno connection joins in, its aggressive window growth will lead to many Reno acks being accumulated on the revere buffer regardless of ack losses. This behavior is more aggressive than when competition is on the forward path (i.e., in non-asymmetric networks), in which the window is increased until overflow occurs on the forward path, at which point Reno's window is cut into halves. Intermittent ack loss in asymmetric networks will not cause Reno to throttle its data-sending rate in the forward direction because of the cumulative property of acks. It will continue to increase its *cwnd*. Of course, loss of acks of Vegas connection also improves the Vegas throughput to some extent. However, accumulating acks at the buffer cause the observed *RTT* to increase continuously and thus lead Vegas' to decrease its window quickly. With reference to Fig. 6 (b). TCP Reno connection almost uses up all of the reverse buffer space in the bottleneck node and does not signal a congestion until actual forward buffer overflow occurs or timeouts are induced by ack loss. TCP Vegas falsely regards accumulation of acks on the reverse path as a signal for the onset of congestion and adapts its window over conservatively.

In summary, from the above investigation, Reno shows more flexibility and adaptability than Vegas in asymmetric networks. This asymmetry significantly degrades Vegas throughput and causes incompatibility between TCP Vegas and Reno. Without modification of the proactive congestion control mechanism, Vegas would face large challenges if it were deployed on the Internet, which currently is quite heterogeneous.

V. SOLUTIONS

From Fig. 4, we note both Reno and Vegas do not make full use of the capacity in the forward path. The techniques (e.g. ACC, AF, SA and AF) for improving Reno's performance under asymmetry have previously been studied in [8]. In this section, we argue that these techniques are not effective for handling the Vegas' asymmetry problems. Afterwards, one new technique is proposed to improve the performance of TCP Vegas.

ACC (Ack Congestion Control), which uses gateways on the reverse link to aid congestion control, was proposed in [8]. It tries to detect the impending congestion by tracking the average queue size over a time window in the recent past and then informs the receiver to dynamically vary the delayed-ack factor to decrease the frequency of acks on the constrained reverse link. However, when a single connection of TCP Vegas is run over asymmetric networks, the ack queue size fluctuates only within the narrow range of α and β on the reverse path. Hence, ACC is not effective for Vegas because the gateway on the reverse path will not detect large accumulation of acks.

Similarly, AF (Ack Filtering) [8] is a router-based technique to remove some fraction (possibly all) of accumulated acks in the reverse buffer. Vegas' queue size on the reverse link shows little changes, unlike that seen in Reno. Therefore, AF technique is also not applicable for Vegas.

ACC and AF have reduced the asymmetric effects to some extent on TCP Reno, but they also bring about slowdown of window growth because Reno's window increase is based on the number of acks not the amount of acks. To prevent the reduced ack frequency from adversely affecting the performance of TCP Reno, SA (Sender Adaptation) and AR (Ack Reconstruction) [8] are proposed. But, such ack losses do not cause any adverse effects on TCP Vegas because its window change is based on the difference of the expected rate and measured actual rate during each round-trip time rather than the number of acks received.

We propose a non-gateway-based solution. We use the TCP option presented in RFC1323 that adds a timestamp to the TCP header. From the timestamps, we can easily compute actual flow rate ($Actual_f$) on the forward path at the sender and accurately identify the onset of congestion path on the data path. This allows us to distinguish between forward and backward path congestions. Our modified Vegas' congestion avoidance mechanism is as follows:

```
if  ( DIFF_f *BaseRTT < α )
      /* where DIFF_f = (Expected- Actual_f ) */
      cwnd = cwmd + 1
      /* increase congestion window size by one */
if  (DIFF_f *BaseRTT < β )
      cwnd = cwnd-1
else  cwnd = cwnd
      /* congestion window remains unchanged */
```

where $DIFF_f = Expected - Actual_f$. By using $DIFF_f$ instead of *DIFF*, in asymmetric networks, Vegas' *cwnd* is able to converge to the optimal point in which the available bandwidth on the forward path is fully utilized. Table III shows modified Vegas obtains greatly improved results over Vegas. Comparing to Reno's throughput in Table I, Vegas also outperforms Reno in asymmetric networks by employing such a simple modified mechanism.

TABLE III
THROUGHPUT AND UTILIZATION OF TCP VEGAS AND MODIFIED VEGAS (μ_f=1.6Mbps, B_f=15 packets B_r= 10 acks $\tau_f = \tau_r$ =50ms)

	VEGAS/ MODIFIED VEGAS			
	Non-asymmetry	Asymmetry		
	k=0.5	*k*=3	*k*=5	*k*=7
Throughput (packet/s)	195.9/194	65.8/190.4	43.8/192.8	26.8/189.2
Throughput ratio	1.0/1.0	0.336/0.98	0.224/0.99	0.137/0.97
Forward link utilization	0.98/0.97	0.312/0.95	0.198/0.96	0.144/0.95

VI. CONCLUSIONS

The Internet is characterized by heterogeneity with all sorts of network equipment and links. Network asymmetry is common with technologies such as ADSL, HFC. TCP Vegas does not deal with network asymmetry well.

Previous studies have shown that TCP Vegas outperforms TCP Reno. We show in this paper that Vegas throughput performance is significantly worse than that of Reno when the bottleneck is on the reverse path. In such a situation, the Vegas algorithm converges to the wrong operating point and fails to make full use of the available bandwidth on the forward path. Vegas congestion detection mechanism is the fundamental reason why it suffers in the asymmetric environment. In addition, Vegas also suffers a severe bias when Vegas and Reno connections compete with each other in asymmetric networks, making it difficult to deploy Vegas in practice because of the large installed base of Reno stack in the current Internet.

Techniques used to address asymmetry that are effective for Reno are not effective for Vegas. We propose a solution which makes use of the timestamp option in RFC1323 to distinguish between congestions caused by forward and reverse paths. With this method, Vegas' congestion window is able to converge to the optimal point in which the available bandwidth on the forward path is fully utilized even when the backward path is congested.

REFERENCE

[1] Van Jacobson. "Congestion avoidance and control " *ACM SIGCOMM 88*, pages 273-288, 1988

[2] Van Jacobson. "Modified TCP congestion avoidance algorithm", mailing list, end2end interest, 30 Apr 1990.

[3] R. Jain, "A delayed-Based approach for congestion avoidance in interconnected heterogeneous computer networks" *ACM Computer Communication Review*, 19(5):56-71, Oct. 1989

[4] Lawence S. Brakmo, Sean W.O'Malley, and Larry L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," Proceedings of *ACM SIGCOMM'94*, no.4 pp.24-35, October 1994

[5] Lawence S. Brakmo and Larry L. Peterson, "TCP Vegas: End to End congestion avoidance on a global internet," *IEEE Journal an Selected Areas in Communications*, vol.13, no. 8, pp.1465-1480, October 1995

[6] Jong Suk Ahn, Peter B.Danzig, Zhen Liu, and Limin Yan, "Evaluation with TCP Vegas: Emulation and experiment ," *ACM SIGCOMM Computer Communications Review*, vol.25, no. 4, pp.185-195, August 1995.

[7] Hossam Afifi, Omar Elloumi, Gerardo Rubino, "A Dynamic Delayed Acknowledgement to Improve TCP Performance for Asymmetric Links" Computers and Communications, *ISCC '98*. Proceedings. Third IEEE Symposium on , 1998

[8] H. Balakrishnan, V.N. Padmanabhan and R.H. Katz, " The effects of TCP/IP performance," Proceedings of *INFOCOM'97*

[9] T.V. Lakshman "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance." *INFOCOM '97*.

[10] Henderson, T.R.; Sahouria, E.; McCarme, S.; Katz, "On Improving the Fairness of TCP Congestion Avoidance" *GLOBECOM'98*. Volume: 1 , 1998 , Page(s): 539 -544

[11] Go Hasegawa, Masayuki Murata, and Hideo Miyahar, " Fairness and Stability of Congestion Control Mechanism of TCP" *INFOCOM '99*. Volume: 3 , 1999 , Page(s): 1329 - 1336 vol.3

[12] Mo, J.; La, R.J.; Anantharam, V.; Walrand, J., "Analysis and Comparison of TCP Reno and Vegas" *INFOCOM '99* Page(s): 1556 -1563 vol.3

[13] R. J. La, J. Walrand, and V. Anantharam, "Issues in TCP Vegas" Available at http://www.path.berkeley.edu/~hyongla, July 1998