TCP Vegas-A: Solving the Fairness and Rerouting Issues of TCP Vegas

K.N. Srijith, Lillykutty Jacob, A.L. Ananda
School of Computing, National University of Singapore
3, Science Drive 2, Singapore 117543

Abstract

In spite of the larger performance gain such as higher throughput and almost zero packet retransmissions compared to TCP Reno, TCP Vegas still has a few obstacles for it to be deployed in the Internet. Studies have shown unfair treatment to Vegas connections when they compete with Reno connections. Other issues identified with TCP Vegas are problems of rerouting, persistent congestion, and discrepancy in flow rate tied with starting times and link bandwidth. We reinvestigate these issues and propose modifications to the congestion avoidance mechanism of the TCP Vegas, with the slow-start and congestion recovery algorithms of Vegas remaining untouched. Unlike the solutions proposed in the recent past to deal with some of these issues, our solution is neither dependent on any critical parameter values nor on the buffer management scheme at the routers (e.g., RED). Our experiments show that the modified TCP Vegas (Vegas-A) is able to obtain a fairer share of the network bandwidth when competing with other TCP flows. We also show that Vegas-A can tackle rerouting issues and rectify Vegas's bias against higher bandwidth flows. At the same time, our experiments prove that Vegas-A preserves the properties of Vegas that have made it a noteworthy protocol.

I. Introduction

Following the proposal by Brakmo *et al*. [1] of TCP Vegas, which has a fundamentally different congestion control scheme from that of TCP Reno, several studies were made on TCP Vegas. These studies have established that Vegas does achieve higher efficiency than Reno, causes much fewer packet retransmissions, and is not biased against the connections with longer round trip times (RTTs), but does not receive a fair share of bandwidth when competing with Reno connections [2,3,4].

By decomposing TCP Vegas into its individual algorithms and addressing the effect of each of these algorithms on performance, Hengartner *et al*. [5] have shown that the reported performance gains are achieved primarily by TCP Vegas's new techniques for slow start and congestion recovery, and that the congestion avoidance mechanism of Vegas has only a minor influence on throughput while also being responsible for the fairness problems. Hasegawa *et al*. [6] and Raghavendra *et al.* [7] show that with the RED routers in place of the tail-drop routers, the fairness between Vegas and Reno can be improved to some degree. But there exists an inevitable trade-off between fairness and throughput, i.e., if the packet dropping probability of RED is set to a large value, the throughput share of Vegas can be improved, but the total throughput is degraded. In this paper, we propose modifications only to the congestion avoidance mechanism of TCP Vegas to address the fairness issues. We call our modified version 'TCP Vegas-A'.

In [8], La *et al*. identifies the problems associated with TCP Vegas's use of estimate of RTT as a decision variable. In particular, they identify the issue of rerouting, the persistent congestion problem, and the discrepancy in flow rates tied with starting times of different Vegas connections. They propose a solution to overcome the problem of rerouting and show that persistent congestion problem could also be solved using the combination of the same modification and RED gateways. However, finding appropriate threshold values for RED gateways and finding the appropriate parameter values for the proposed modification of TCP Vegas's congestion avoidance mechanism are open problems. TCP Vegas-A addresses the issues of rerouting and bias against older connections as well. Furthermore, our experiments show that TCP Vegas-A preserves the proven nice properties of TCP Vegas.

The paper is organized as follows: Section 2 briefly introduces the congestion avoidance mechanism of TCP Vegas. The issues related to the present implementation of TCP Vegas are presented in Section 3. In Section 4, we propose our modifications to TCP Vegas. Section 5 is on the simulation setup and in Section 6 we present experimental results to show that the proposed modification does perform better under the scenarios outlined in Section 3 and discuss these results. We conclude in Section 7.

II. Congestion Avoidance of TCP Vegas

TCP controls its flow rate based on its estimate of the available network bandwidth. Among many new features implemented in TCP Vegas, the most important difference between TCP Vegas and TCP Reno lies in its *bandwidth estimation* scheme. TCP Vegas dynamically increases/decreases its sending window size based on fine-grained measurement of RTTs, whereas TCP Reno continues to increase its window size until packet loss is detected. While TCP Reno views packet losses as a sign of network congestions, TCP Vegas uses a sophisticated bandwidth estimation scheme, wherein it uses the difference between the expected and achieved flow rates to estimate the available bandwidth in the network. The idea is that when the expected and actual throughputs are almost equal, the network is not congested. In other words, in a congested scenario, the actual throughput will be much smaller than the expected. Thus, based on this estimation of network congestion, TCP Vegas updates the congestion window. According to the TCP Vegas mechanism:

(a) *expected_rate = cwnd(t)/base_rtt* , where *cwnd(t)* is the current congestion window size and *base_rtt* is the minimum RTT of that connection.

(b) *actual_rate = cwnd(t)/rtt*, , where *rtt* is the actual round-trip time

(c) The source estimates the backlog in the router queue from the difference

diff = expected_rate – actual_rate

(d) Using the value of *diff* the congestion window value (*cwnd)* is adjusted as:

$$cwnd = \begin{cases} cwnd + 1 & \text{if } diff < \alpha \\ cwnd - 1 & \text{if } diff > \beta \\ \text{cwnd} & \text{otherwise} \end{cases}$$

What the above algorithm tries to do is, when the expected and the actual throughput are close to each other, the connection may not be utilizing the available network bandwidth, and hence should increase the flow rate. On the other hand, when the actual throughput is much less than the expected throughput, the network is likely to be congested and hence the connection should reduce the flow rate.

III. Issues with TCP Vegas

Although TCP Vegas has been shown to provide better performance in terms of increased throughput, reduced jitter and much reduced retransmissions, there are a number of issues associated with TCP Vegas that prevent it from being accepted and implemented on the Internet. In this section we discuss these issues.

A. Fairness

TCP Vegas uses a conservative algorithm to decide when to increase/decrease the congestion window. However, TCP Reno, in an effort to fully utilize the bandwidth, continues to increase the window size until a packet loss is detected. Thus, while TCP Vegas tries to maintain a smaller queue, TCP Reno and its variants keep a larger number of packets in the buffer on the average, and thus steal a larger bandwidth.

When TCP Vegas and TCP Reno connections share a bottleneck link, Reno uses up most of the router buffer space. TCP Vegas, interpreting this as a sign of congestion, decreases the congestion window, which leads to an unfair share of bandwidth for TCP Vegas. Furthermore, when the router buffer size is large, the average window size of TCP Reno connections will be large, while the window size of TCP Vegas connections will remain unchanged, as it does not inflate the window size larger than β. This means, the larger the router buffer size, the worse the fairness between Reno and Vegas connections.

Hasegawa *et al*. [6] proposed TCP Vegas$^+$ as a method to tackle TCP Vegas's fairness issue. Vegas$^+$ has two modes. In the *moderate mode*, Vegas$^+$ behaves identically to the normal TCP Vegas, but it enters the *aggressive mode* to increase its window size aggressively (identical to TCP Reno) when it detects the presence of competing Reno traffic. However, Vegas$^+$ assumes that an increase in the RTT value is always due to the presence of competing traffic and rules out other possibilities like rerouting. Furthermore, performance of Vegas$^+$ depends on the choice of optimal value for $count_{max}$ in order to switch between the two modes.

B. Rerouting

In Vegas, the parameter *baseRTT* denotes the smallest round-trip delay and is used to measure the expected throughput. When a route is changed, the RTT of a connection can change. Vegas is not usually affected if the new route has a smaller RTT, as *baseRTT* will be updated. But when the new route has a longer RTT, the connection will not be able to deduce whether the longer RTTs experienced are because of congestion or route change. Without this knowledge, TCP Vegas assumes that the increase in RTT is because of congestion along the network path and hence decreases the congestion window size.

This is exactly opposite of what the connection should be doing: When the propagation delay increases, the bandwidth-delay product (*bw*d*), which is an estimate of the number of packets that can be held in the network, increases. For any connection, the expression (*cwnd – bw*d*) gives the number of packets in the buffers of the routers. Since the aim of TCP Vegas is to keep the number of packets in the router between α and β, it should increase the congestion window to keep the same number of packets in the buffer when the propagations delay increases.

La *et al.* [8] proposed a modification to the Vegas to counteract the rerouting problem. Their modification uses any lasting increase in RTT as a sign of rerouting. For the first K packets, the mechanism is the same as TCP Vegas. However, subsequently, the source keeps track of the minimum RTT of every N packets. If this minimum RTT is much larger than the *baseRTT* for L consecutive times, the source updates its *baseRTT* to the minimum RTT of the last N packets and resets the *cwnd* based on this new *baseRTT*. Their reasoning for this modification is that since the increase in RTT forces the source to decrease the *cwnd*, the increase in RTT comes mostly from the propagation delay of the new route and not from the congestion. However, there are two more parameters δ and γ in this scheme that update the *baseRTT*, and finding the appropriate values for K, N, L, δ and γ remains an open problem.

C. Unfair treatment of 'old' connection

As pointed out in [5], TCP Vegas's congestion control mechanism is inherently unfair to older connections. When a Vegas connection is established in an uncongested network, the baseRTT is likely to be close to the minimal RTT possible. When, later on, the network gets congested, the measured RTT rtt_1 increases and the ratio $baseRTT_1/rtt_1$ decreases. Consider a new connection that is initiated after the network becomes congested. The value of $baseRTT_2$ measured for this connection will be larger than $baseRTT_1$. Hence $baseRTT_2/rtt_2$ will be larger than $baseRTT_1/rtt_1$ since rtt_1 and rtt_2 are nearly equal. As derived in [5], window size that would trigger a reduction in congestion window size is given by:

$$cwnd > \frac{\beta}{1 - \frac{baseRTT}{rtt}}$$

Thus, we see that, for the older connection, the critical value of *cwnd* is smaller than that of the newer connection. Hence, the second connection can achieve higher bandwidth than the first (older) connection.

Similarly, the critical value of congestion window that triggers an increase in congestion window is given by:

$$cwnd < \frac{\alpha}{1 - \frac{baseRTT}{rtt}}$$

Since the right side hand side term is larger for the newer connection, it is more likely to be able to increase its congestion window than the older connection, and thus again get a more than fair share of the network bandwidth.

A closely related problem is the *persistent congestion* problem, where the connections may overestimate the propagation delays and possibly drive the network to a persistently congested state. Consider a connection that starts when the network is congested. Due to the queuing delay caused by the back log from other connections, the packets from the new connection experiences RTTs that are considerably larger than the actual propagation delay of the path. Due to this inaccurate estimation of the *baseRTT*, it will overestimate the possible window size (as pointed out by the critical window sizes that trigger increase/decrease of the *cwnd*). This scenario will repeat for each new connection, leading to persistent congestion. La *et al.* [8] suggested that the same solution as the one they proposed for rerouting problem, together with RED routers could solve the persistent congestion problem. Low *et al.* [9] also proposed augmenting Vegas with appropriate active queue management (AQM) such as Random Exponential Marking (REM) for avoiding the persistent congestion problem.

IV. Our Modification – 'Vegas-A'

In this section we present our modification to Vegas, referenced to as Vegas-A, wherein 'A' stands for adaptive. TCP Vegas has fixed values for α and β, set to 1 and 3 usually. Recall that Vegas's strategy is to adjust the source's sending rate (congestion window) in an attempt to keep a small number of packets buffered in the routers along the path. Thus, when Vegas connections compete with Reno connections, the fairness problem is worse with larger router buffers. Since the average number of packets in the router buffer is to be kept within α and β, the main idea in Vegas-A is that, rather than fixing α and β they be made adaptive. At the start of a connection, α is set to 1 and β to 3. These values however change dynamically depending on the network conditions. The slow start and congestion recovery algorithms of Vegas-A are the same as that of Vegas. However, Vegas-A uses the following congestion avoidance mechanism:

Terms used:

$Th_{(t)}$ = actual throughout at time t; $Th_{(t-rtt)}$ = actual throughput at previous rtt. The definitions of *expected_rate*, *actual_rate* and *diff* are the same as those in Vegas.

if β > *diff* > α {
  if $Th_{(t)} > Th_{(t-rtt)}$ {
    cwnd = cwnd +1
    / The reasoning is that, even though diff > α, the throughput has been increasing. This indicates that the network is not fully utilized and that network bandwidth is still available. Hence, the sending rate can be increased, to probe the network */*

    Change α= α+1, β= β+1
    */*Since throughout is increasing over time, diff is decreasing. The existing small values of α and β prevent the connection from making use of the available bandwidth. Hence we increase α and β to help congestion window grow. */*
  }
  else if $Th_{(t)} <= Th_{(t-rtt)}$
  {no update of cwnd, α, β}
}
else if *diff* < α {
  if α >1 and $Th_{(t)} > Th_{(t-rtt)}$ {
  { cwnd = cwnd +1 }
  else if α >1 and $Th_{(t)} < Th_{(t-rtt)}$ {
    cwnd = cwnd –1, α= α-1, β= β-1
    */*In Vegas-A, a small value of diff needs not necessarily imply that bandwidth utilization is low. It might be that α has grown to a large value and when congestion occurs, even a small throughput can still make diff less than α. Hence we need to decrease cwnd and the inflated α and β. */*
  }
  else if α =1
    {cwnd = cwnd+1.}
}
else if *diff* > β
  {cwnd= cwnd-1, α= α-1, β= β-1}
else
  {no update of cwnd, α, β}

V. Simulation Setup

Simulations were done using the Network simulator (ns) [10]. The generic network topology simulated is as shown below.

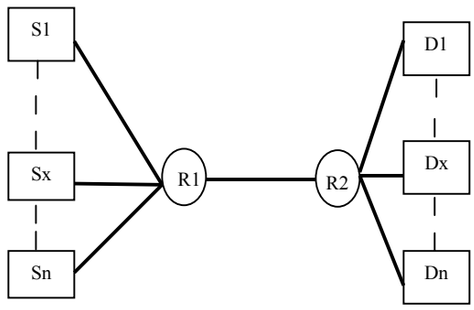

Fig. 1. Simulated wired network topology

The data rates and propagation delays of the various links - source-to-router R1, R1-to-R2 (the bottleneck link), and R2-to-destination - are specified in the context of each experiment. Routers use tail-drop policy with a buffer size of 50 packets, unless stated otherwise.

The TCP/Vegas agent of NS, which is based on the USC's NetBSD Vegas implementation, was modified to implement Vegas-A. We compare the performance of Vegas-A with that of Vegas and New Reno. Unlike Reno, New Reno avoids timeouts (and hence slow starts) when there are multiple packet losses from a window, thus performing better than Reno.

VI. Results and Discussions

We conducted extensive simulations to compare the performance of Vegas-A with original Vegas implementation under various network conditions. This section details the results obtained.

A. Rerouting

In this scenario, we have one Vegas connection (S1-D1). We simulate a change in the route by changing the RTT of the link connecting S1 to R1. The link S1-R1 has a bandwidth of 1Mbps and initial RTT of 20ms. After 20 seconds of file transfer over the connection, the RTT of the link is changed to 200ms. The links R1-R2 and R2-D1 have bandwidth of 1Mbps each and RTT of 10ms for the entire duration of the connection. The simulation was run for 200 seconds so as to give the connection time to stabilise. Table 1 shows the comparison of average throughputs (in bits/sec) obtained in the two cases.

Table 1 Throughputs for rerouting conditions

	TCP Vegas	TCP Vegas-A	Difference	% Increase
Throughput	217320	940240	772920	333 %

Figures 2 and 3 show the variation of throughputs for the full run. The dotted curve shows the instantaneous throughput, averaged over 0.1second intervals, and the solid one shows the average throughput. As we can see, as soon as the RTT changes at 20 seconds, the instantaneous throughputs of Vegas and Vegas-A start to decrease. However, the throughput of Vegas (see Figure 2) went all the way down to 0.12Mbps. As seen in Figure 3, the throughput of Vegas-A does suffer initially when the RTT changes at 20 seconds.

The "actual throughput" decreases, *diff* grows large, and hence *cwnd* is decreased. But when *cwnd* decreases, expected throughput also decreases. Finally *cwnd* decreases so much that *diff* falls between α and β. Then, when there is a slight improvement in the throughput for any particular RTT cycle, the Vegas-A connection increases the value of *cwnd*, α and β. This increase in *cwnd* in turn increases the throughput further, which will then trigger an increase in *cwnd*, α and β again. This process continues until *diff* becomes less than α. Then *cwnd* alone increases. This growth of the congestion window is shown in Figure 5. The end result is that the bandwidth available is fully utilized and the throughput goes back to the same large value prior to the rerouting. Figure 4 shows the throughput variation for TCP New Reno under similar conditions.

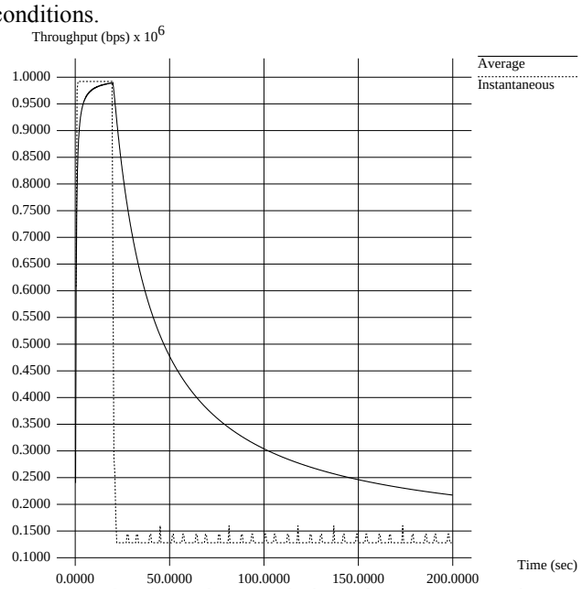

Fig. 2. Throughput variation of Vegas connection due to RTT change

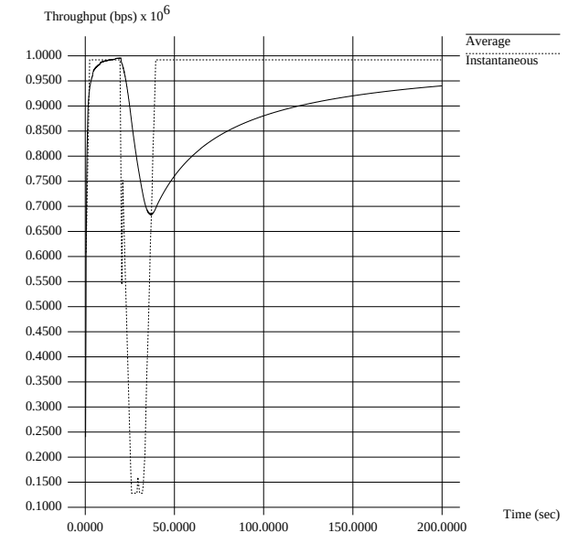

Fig. 3. Throughput variation of Vegas-A connection due to RTT change

B. Bandwidth sharing with New Reno

Next, the performance of Vegas-A when it shares a link with TCP New Reno is considered. There are two connections, S1-D1 and S2-D2. Both the sources are transferring files, but former one uses TCP Vegas-A or Vegas and the latter one uses TCP New Reno. The links S1-R1 and S2-R1 are both 8Mbps with RTT of 20ms. The bottleneck link R1-R2 is of bandwidth 800Kbps and RTT of 80ms. R2-D1 and R2-D2 links are both 8Mbps and RTT 20 ms. S1 was started first and then S2's New Reno traffic was introduced after 10 seconds. Figure 6 shows the instantaneous and average throughputs of the TCP New Reno and TCP Vegas

connections as they are competing with each other for a share of the bandwidth. As can be seen from the figure, when TCP New Reno starts at 10 seconds, it starts to consume Vegas's share of bandwidth.

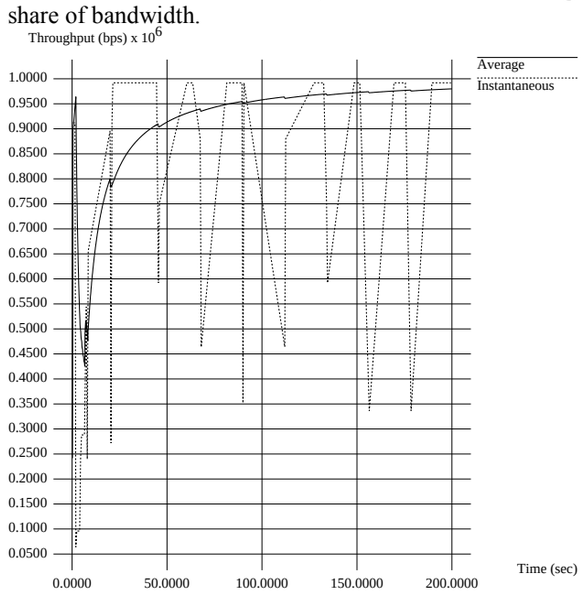

Fig. 4. Throughput variation of New Reno connection due to RTT change

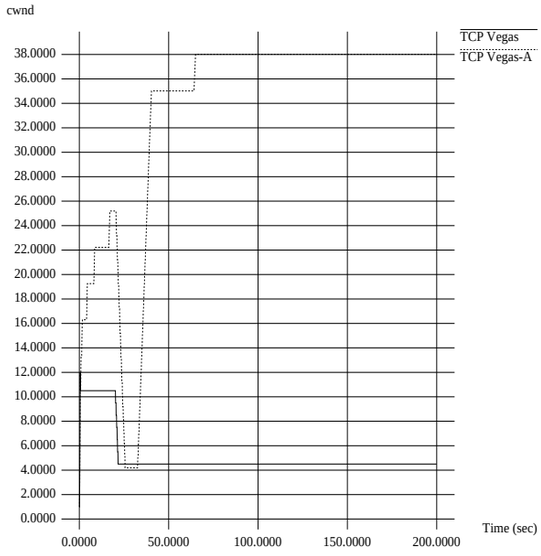

Fig. 5. *cwnd* variation for Vegas and Vegas-A connection due to RTT change

However, when TCP Vegas is replaced with Vegas-A, the results are very different, as seen in Figure 7. Even though TCP New Reno gets a more than fair share of the bandwidth, the unfairness is not as much as in the case of TCP Vegas. Table 2 gives numerical value for the throughput of each connection. The use of Vegas-A reduced the 'unfairness' from a ratio of 5.3:1 to 3.2:1.

Table 2. Throughput ratios for New Reno-Vegas and New Reno-Vegas-A connections

New Reno/Vegas	New Reno/Vegas-A
703875/132040=5.33	633307/199320=3.17

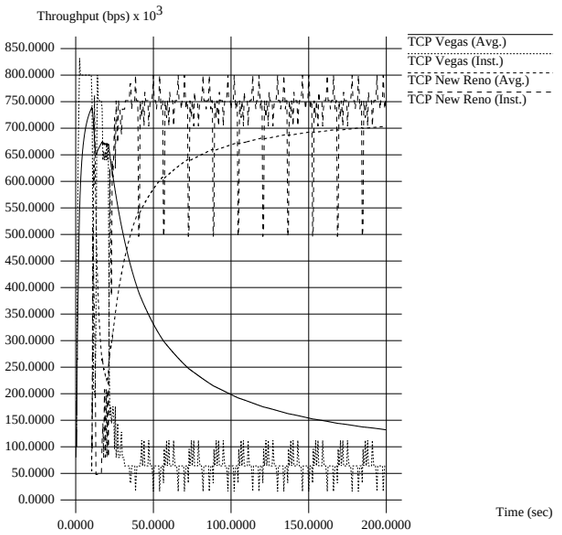

Fig. 6. Throughput of TCP New Reno and Vegas connections over congested link

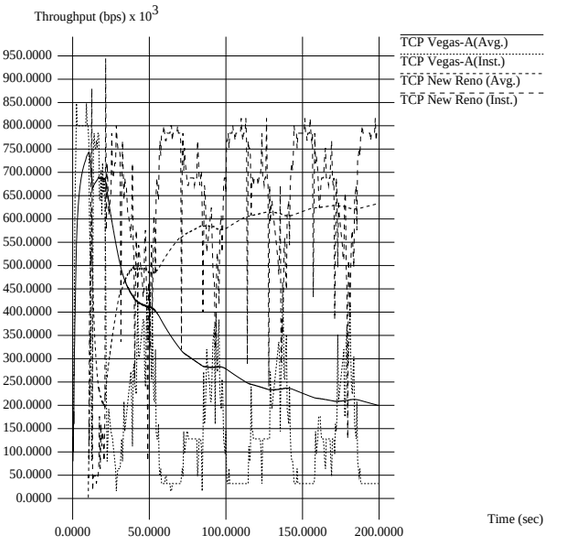

Fig. 7. Throughputs of TCP New Reno and Vegas-A connections over congested link

When the above experiment was repeated with 3 New Reno sources and 3 Vegas/Vegas-A sources, TCP Vegas-A was found getting a more than fair share of the congested link bandwidth. Each source-to-R1 link was 1Mbps and RTT of 20ms. R1-R2 link was 1Mbps and RTT of 80ms. R2-to-destination link was 1Mbps and had RTT of 20ms.

Figures 8 and 9 show the average throughputs obtained. The Vegas/Vegas-A sources (S1, S2, S3) were started at 0,10,20 seconds while the New Reno sources (S4, S5, S6) were started at 30,40,50 seconds. The average throughput obtained by each flow is given in Table 3.

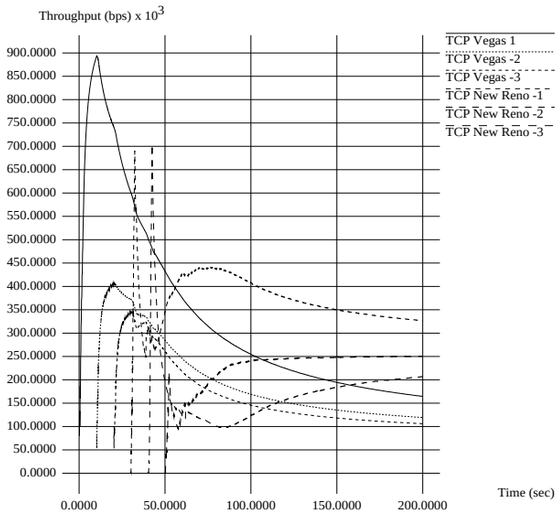

Fig. 8. Throughput of 3 TCP New Reno and 3 Vegas connections over congested link

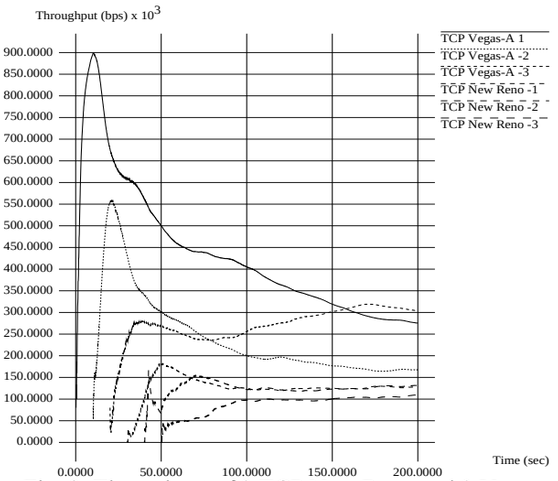

Fig. 9. Throughput of 3 TCP New Reno and 3 Vegas-A connections over congested link

Table 3. Throughputs of New Reno/Vegas and New Reno/Vegas-A connections

Sources	Vegas & New Reno	Vegas-A and New Reno
S1	164480	275480
S2	119242	167789
S3	106044	304178
S4	326590	127908
S5	207002	131402
S6	250828	109336

From the values in Table 3, the *Interprotocol fairness ratio* [11] for the Vegas & New Reno connections can be calculated as 0.497 and that of the Vegas-A & New Reno connections as 2.028. *Interprotocol fairness ratio* is the ratio of the average bandwidth shares between the two protocols, i.e., it is the ratio of average TCP Vegas/Vegas-A bandwidth calculated across all the Vegas/Vegas-A flows to the average TCP New Reno bandwidth calculated across all the New Reno flows. The obtained values show that Vegas-A actually outperforms New Reno.

D. Fairness between old and new connections

As mentioned in Section 3.C, TCP Vegas has the disadvantage that newer connection of Vegas enjoys a larger throughput because of the discrepancy in the estimation of *baseRTT*. Vegas-A's modified congestion control mechanism overcomes this shortcoming. To illustrate this, we simulated a scenario where 5 TCP Vegas/Vegas-A connections are sharing a link. The source-to-router links were set to 1Mbps and RTT of 20ms, while the router-to-destination links had a bandwidth of 1Mbps and RTT of 100ms. The sources were started at intervals of 50 seconds each. Table 4 shows the average throughputs obtained by the connections for the simulation lasting 900 seconds. Figure 10 shows the average throughputs of the Vegas connections, while Figure 11 shows that of the Vegas-A connections.

Table 4. Throughputs of five Vegas and Vegas-A connections

Sources	Vegas	Vegas-A
S1	218531	221447
S2	191533	199760
S3	206176	247431
S4	247585	229577
S5	266913	234662
Std. Deviation	33217.1	17711.1

It is obvious from Figure 10 that new connections enjoy larger throughputs compared to old connections in the case of Vegas. On the other hand, with Vegas-A, this problem is reduced, as seen in Figure 11. Table 4 shows that the standard deviation of the average throughput of Vegas-A connections is less than that of Vegas connections. This means that Vegas-A connections share the bandwidth in a fairer manner compared to the Vegas connections.

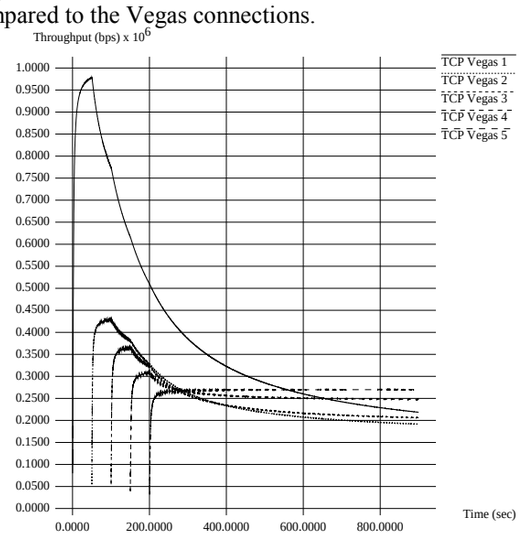

Fig. 10. Throughput of 5 Vegas connections over congested link

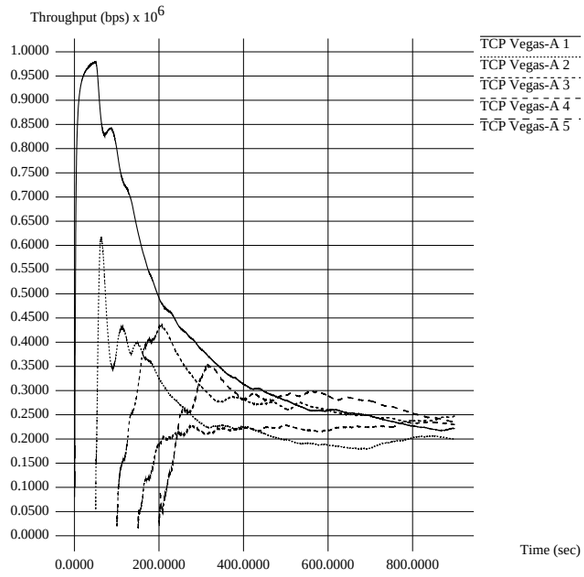

Fig. 11. Throughput of 5 Vegas-A connections over congested link

E. Bias against high bandwidth flows

Hasegawa *et al.* [12] showed that TCP Vegas, like Reno, has the fairness bias against connections with higher bandwidth. To test whether Vegas-A can perform better than Vegas in terms of this fairness, simulations were conducted with 3 connections S1-D1, S2-D2, and S3-D3, and with the S1-R1 ink bandwidth limited to 128Kbps, S2-R1 to 256Kbps, and S3-R1 to 512Kbps. The RTT of each source-to-R1 link was set to 5ms. R1-R2 link was bandwidth limited to 400Kbps and the RTT was set to 10ms. R2-to-destination link had a bandwidth of 1Mbps and RTT of 5ms. The simulations were carried out for a period of 1800 seconds. Table 5 summarizes the throughputs (in Kbps) obtained and compares it with the expected values. With the Si-R1 link bandwidth represented by bw_i, the expected value for connection Si-Di is calculated as $bw_i * R / \sum_1^3 bw_i$ where R is the bottleneck link bandwidth.

Table 5. Comparison of Vegas and Vegas-A bias against high bandwidth connection for 1800 seconds

	S1	S2	S3
Expected	57.14	114.29	228.57
Vegas	123.34	146.85	120.46
Vegas-A	98.90	134.54	158.25

Ideally, S3-D3 connection should have received 228Kbps of the bottleneck bandwidth, but when Vegas is used, the connection ends up with just 120.5Kbps bandwidth. However, when the source is switched to Vegas-A, the three connections enjoy fairer shares of the bandwidth (closer to the expected values). As we have observed, in the case of Vegas, the *cwnd* attains a steady value, such that $\alpha < diff < \beta$, in the very early stage of the connection, and remains like that for the entire duration of the connection. In the case of Vegas-A, α and β can vary dynamically, allowing *cwnd* to probe for more bandwidth share.

F. Retaining the properties of Vegas

TCP Vegas-A tries to improve upon the congestion control mechanism of TCP Vegas by trying to adapt to the availability of the network bandwidth. However, it is essential that in doing so, Vegas-A should not lose the properties of Vegas. To ensure that Vegas-A does inherit the properties of Vegas, simulations were conducted with only one Vegas/Vegas-A/New Reno source connected through the routers to the destination. The S1-R1 link bandwidth was set to 1Mbps and the RTT of the link to 5ms and 45ms, respectively, for two sets of experiments. The R1-R2 bandwidth was restricted to 250Kbps and the RTT to 5ms and 45ms, respectively, for the two sets of experiments. The R2-D2 link bandwidth was set to 1Mbps and RTT to 10ms. File sizes of 10MB and 5MB were sent from source to the destination. Tables 6 and 7 show the values recorded for the time taken for the complete transfer, the average queue length at the router (in packets), and the number of retransmitted packets.

As the tables show, Vegas and Vegas-A outperform New Reno in all performance measures, as expected. The only difference between Vegas and Vegas-A is that the average buffer occupancy at the routers is larger for Vegas-A compared to Vegas. This is expected since Vegas-A adapts α and β to values larger than 1 and 3, which in turn let Vegas-A increase the congestion window to a larger value and thus pushing in more data into the network. However, note that this larger average buffer occupancy does not seem to increase the time required to complete the transfer of the files, thus showing that the queuing delay is not increased much.

Table 6. Comparison of New Reno, Vegas and Vegas-A connection on a 20ms RTT link

Source	5MB file			10MB file		
	Time	Avg. Queue	Retx. Pkts.	Time	Avg. Queue	Retx. Pkts.
New Reno	162.353	11.46	58	322.353	23.27	62
Vegas	160.253	0.65	0	320.253	1.3	0
Vegas-A	160.253	5.62	0	320.256	12.4	0

Table 7. Comparison of New Reno, Vegas and Vegas-A connections on a 100ms RTT connection

Source	5MB file			10MB file		
	Time	Avg. Queue	Retx. Pkts.	Time	Avg. Queue	Retx. Pkts.
New Reno	166.505	11.29	59	326.505	23.0	62
Vegas	160.651	0.82	0	320.651	1.63	0
Vegas-A	160.654	4.85	0	320.651	10.84	0

Next we studied the performance of Vegas, Vegas-A and New Reno when the router queue sizes were varied. The RTT of the source to destination links were fixed at 40ms. S1-R1 and R2-D1 link bandwidths were set at 1Mbps, while the R1-R2 link bandwidth was set at 500Kbps. Files of size 10MB was sent from S1 to D1. Table 8 shows the values obtained. The unit of time is seconds and the retransmissions (Retx.) records the number of packets retransmitted.

Table 8. Comparison of New Reno, Vegas and Vegas-A connections with different router buffer queue size

Buffer size →	10		15		20		25		30	
Source	Time	Retx.	Time	Retx.	Time	Retx.	Time	Retx.	Time	Retx.
New Reno	161.3	106	161.6	74	161.9	61	162.2	56	162.5	55
Vegas	160.4	0	160.4	0	160.4	0	160.4	0	160.4	0
Vegas-A	160.5	2	160.6	1	160.6	1	160.4	0	160.4	0

The results show that when the buffer size is as small as 10,15 or 20, Vegas-A retransmits at the most, 1 or 2 packets, while Vegas does not loose any packets at all. This number is extremely small compared to the number of packets dropped and retransmitted when New Reno is used. Furthermore, when the queue size is increased to 25 and 30, the behaviour of Vegas-A approaches that of Vegas.

VII. Conclusion

In this paper, we introduced a modified version of TCP Vegas, TCP Vegas-A, which is able to mitigate some of the limitations of Vegas-A: Vegas-A performs better when competing with other TCP connections like New Reno for shared bandwidth; Vegas-A overcomes the rerouting limitations of Vegas and is able to adapt to the changes in RTT and routes faster; Vegas-A connections do not suffer from the 'unfairness towards old connection' and 'unfairness against higher bandwidth connections' problems of Vegas. It was also shown that even though Vegas-A is different from Vegas, the basic congestion control algorithm still remains as effective as Vegas in decreasing the average queue occupancy and packet retransmissions.

REFERENCES

[1] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. of ACM SIGCOMM'94*, pp. 24-35, October 1994.

[2] L.S. Brakmo and L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, 13(8): 1465-80, October 1995.

[3] J.S. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment," in *Proc. of ACM SIGCOMM'95*, pp. 185-195, August 1995.

[4] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proc. of IEEE INFOCOMM'99*, pp. 1556-1563, March 1999.

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

[6] Hasegawa, K. Kurata, and M. Murata, "Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet," in *Proc. of the IEEE International Conference on Network Protocols (ICNP 2000)*, November 2000.

[7] Raghavendra and R.R. Kinicki, "A Simulation Performance study of TCP Vegas and Random Early Detection," in *Proc. of IPCCC'99*, pp. 169-176, February 1999.

[8] Richard J. La, Jean Walrand, and Venkat Anantharam, "Issues in TCP Vegas," available at *http://www.path.berkeley.edu/~hyongla*, July 1998.

[9] Steven Low, Larry Peterson, and Limin Wang, "Understanding TCP Vegas: A Duality Model," in *Proc of ACM SIGMETRICS 2001*, pp. 226-235, June 2001.

[10] Network Simulator (NS), *http://www.isi.edu/nsnam/ns*.

[11] Reza Rejaie, Mark Handley, and Deborah Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet," in *Proc. of IEEE INFOCOM*'99, pp.1337- 1345, March 1999.

[12] Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Comparisons of Packet Scheduling Algorithms for Fair Service among Connections on the Internet," in *Proc. of IEEE INFOCOM 2000*, pp. 1253-1262, 2000.