TCP New Vegas: Performance Evaluation and Validation

Joel Sing and Ben Soh
Department of Computer Science and Computer Engineering
La Trobe University
Bundoora VIC 3083, Australia
Email: joel@ionix.com.au, ben@cs.latrobe.edu.au
Phone: +61 3 5444 7326, +61 3 9479 1054

Abstract

The performance of TCP Vegas degrades significantly when used over networks with a large bandwidth delay product or when the receiver implements delayed acknowledgments. Three sender-side modifications have previously been proposed, aiming to overcome these problems. We have demonstrated that these modifications, collectively known as TCP New Vegas, provide significant performance gains (in excess of 300% in certain cases) whilst maintaining the primary benefits provided by TCP Vegas' design. This paper details further research undertaken to validate these modifications and to evaluate the performance of TCP New Vegas when used over networks with limited buffers, multiple hops, error-prone links and competing flows.

1. Introduction

In [11] we presented three sender-side modifications to TCP Vegas’ [4] congestion control algorithms. We have demonstrated that these modifications can significantly increase achieved throughput, especially when the flow traverses a network with a large Bandwidth Delay Product (BDP). The impact of delayed acknowledgments [3, 2] also becomes less noticeable, especially during the slow-start phase. In the spirit of TCP naming, we refer to a TCP Vegas implementation that includes these modifications as TCP New Vegas.

This paper1 details research undertaken to validate these protocol modifications and to evaluate TCP New Vegas’ performance when used over networks with different characteristics. In section 2 we provide a summary of the three TCP variants compared throughout this paper. The method used for protocol validation and performance evaluation is described in section 3. The configuration used for each set of simulations is detailed in section 4, with the results also being presented and discussed. Finally, future research is presented and conclusions are drawn.

2. Background

The performance evaluation and validation detailed in this paper compares TCP New Vegas against two other TCP variants - namely TCP SACK and TCP Vegas. The design and implementation of these variants, along with their core algorithms, are outlined in the following sections.

2.1. TCP SACK

The use of Selective Acknowledgments (SACK) [8] provides increased feedback to the sender in regard to segments that have been received and queued by the receiver, but are not yet acknowledged due to one or more segments that are missing from the window. This information allows the sender to selectively retransmit the packets that have not been received, reducing the occurrence of costly timeouts. This is especially critical where packets have been lost due to corruption, as it allows for these packets to be simply retransmitted without retransmitting packets that have already been successfully received but not yet fully acknowledged. The base congestion control algorithms used for a TCP SACK implementation are the same as the TCP Reno [2] congestion control algorithms, with minor modifications to allow for the use of the SACK information during retransmissions.

2.2. TCP Vegas

Unlike most TCP variants, TCP Vegas does not rely on packet loss to signal congestion, instead it uses Round Trip Time (RTT) measurements to determine available network

---

1Due to space constraints, some content and graphs have been omitted. For a full copy of this paper please contact the authors via email.
capacity. The congestion control algorithms within TCP Vegas calculate the expected throughput rate and the actual throughput rate once per RTT. The difference is then calculated, effectively indicating the number of packets being queued within the network. Once this difference exceeds a certain threshold (typically one packet), slow-start is terminated and congestion-avoidance commences. Upon exiting slow-start, TCP Vegas decreases the congestion window by one-eighth of its current size in order to ensure that the network does not remain congested. During slow-start, \( cwnd \) is increased by one segment per acknowledgment during every second RTT. During congestion-avoidance, \( cwnd \) will be increased by one segment, decreased by one segment or left unchanged, with this decision being made once per RTT.

2.3. TCP New Vegas

TCP New Vegas is based on TCP Vegas and implements three sender-side changes. Firstly, the addition of packet pacing is used to spread the transmission of packets over the entire RTT. This prevents bursts of packets from being sent, avoiding skewed RTT measurements caused by transient queues.

Secondly, packet pairing is implemented by sending two packets at a time and increasing the congestion window during slow-start by two segments on every second acknowledgment. Providing an even number is used for the initial window size, this process ensures that no packet is sent without having a second available to immediately follow it. Unless packet loss occurs, the receiver will immediately respond with one or more acknowledgments, preventing the RTT from being skewed via the addition of the receiver delay resulting from the use of delayed acknowledgments.

Thirdly, a technique known as Rapid Window Convergence (RWC) is implemented in order to increase the congestion window more rapidly during the first part of the congestion-avoidance phase. RWC uses a linear increase based on the value of \( cwnd \) at the termination of slow-start, allowing the congestion window to grow and converge far more rapidly than it would normally, whilst working to avoid over-subscription of network resources. The complete design details of TCP New Vegas are available in [11].

3. Validation method

In order to validate TCP New Vegas, we have undertaken a range of simulations using Network Simulator 2 (ns2) [1]. Using a base simulation, we have varied a number of parameters in order to implement properties exhibited by real world networks. These simulations have been executed for each of the three TCP variants. Network latencies of 10ms and 230ms have been used for each simulation, resulting in RTTs of 20ms and 560ms. The first of these is a minimal delay, similar to that experienced by users with broadband Internet access. The second is typical of that experienced by users of geostationary satellite links.

Unless otherwise stated, all simulations have emulated a 30 second FTP transfer between two hosts (A and B) over a single hop (R), with each link being implemented as error-free with a transmission speed of 2Mbps (see figure 1). All links utilise droptail queues with the buffer size being set to 1000 packets. A Maximum Segment Size (MSS) of 536-bytes has been used (resulting in a packet size of 576-bytes assuming minimal TCP and IP headers), along with an initial window size of two segments. Delayed acknowledgments have also been implemented at the receiver.

4. Comparing the protocols

We have compared TCP New Vegas via simulation against TCP SACK and TCP Vegas under numerous configurations. These include networks with limited buffers, multiple hops, error-prone links and competing flows. The results of these simulations are presented and discussed in the following sections.

4.1. Limited buffers

In order to evaluate each variant’s ability to cope with limited buffers, simulations were run with all links implementing droptail queues. For each network latency, simulations were run with the maximum queue size at the router being set to \( n \cdot \lceil 2^{\log_2 c} \rceil \), where \( c \) is the optimal congestion window size for the network and \( n \) is 1.0, 0.5 or 0.25. This gives queue sizes of 16, eight and four for the 20ms RTT network and 256, 128 and 64 for the 560ms RTT network.

When a network exhibits a 20ms RTT, a queue size of 16 packets results in near identical performance for all three TCP variants. Both TCP Vegas and TCP New Vegas converge very quickly and maintain a consistent congestion window size for the duration of the connection. TCP SACK on the other hand quickly overshoots the available queue size, with \( cwnd \) reaching 39 segments before being reduced to 19 segments after packet loss occurs. For the remainder of the connection’s duration, the congestion window is increased until it exceeds the combined capacity of the network and queue, resulting in a loss event. At this point
the congestion window is halved from 26 to 13 segments and the linear increase is resumed. As a result, TCP SACK forces the network to drop 36 packets during transmission. In comparison, neither TCP Vegas nor TCP New Vegas force the network to drop any packets during transmission. Halving the queue size from 16 to eight segments only serves to amplify the effect of TCP SACK’s oscillating congestion window size, with 52 segments being dropped by the network, nine of which are due to slow-start overshoot. At 2.36 seconds, the throughput of TCP SACK is surpassed by TCP New Vegas, due to a packet that was dropped at 2.25 seconds being retransmitted. TCP Vegas also surpasses TCP SACK just prior to 3.09 seconds, also because of the retransmission of lost data, this time due to a loss that occurred at 2.93 seconds.

Decreasing the queue size to four segments effectively cuts the slow-start exponential growth short of the optimal congestion window size, leaving the congestion-avoidance phase to grow the window to the optimal value. This has little impact on TCP Vegas, however both TCP SACK and TCP New Vegas are impacted significantly. TCP SACK causes the network to drop 61 packets, only two of which are now due from slow-start. The remaining 59 are dropped as the window oscillates and oversubscribes the network’s capacity. The packet pairing and RWC components of TCP New Vegas force the network to drop three segments, the first of which results in the congestion window size being halved, the second and third force the slow-start phase to be restarted. The second time in slow-start results in the congestion window increasing to four segments before the congestion-avoidance phase is entered, which quickly increases the window to just under 14 segments. The window is then held for the remainder of the connection’s duration. This process results in no data transfer occurring for the first 2.5 seconds, at which point normal data transfer begins, surpassing TCP SACK’s throughput at 8.7 seconds. TCP Vegas also surpasses TCP SACK, this time just before 1.61 seconds. The continual loss events and retransmissions severely impact TCP SACK, achieving 18% less throughput than TCP Vegas and 14% less throughput than TCP New Vegas.

For a network that exhibits a latency of 560ms, a queue size of 256 has no impact on TCP Vegas and TCP New Vegas, as neither variant consumes all available queue space, resulting in zero packet loss. Due to the design of TCP SACK, the network is forced to drop packets when the congestion window grows to a size that exceeds the combined capacity of the network and the queue (around 500 packets in this configuration). As a result, 251 packets are dropped between 9.05 and 10.21 seconds, with the congestion window being halved from 750 to 375 segments as slow-start is terminated. Recovery and retransmission of these lost packets causes a dip in throughput between 11.05 and 12.22 seconds. Due to the large BDP no further losses occur before the connection is terminated. TCP Vegas’ performance is limited by its early termination of slow-start and slow-window growth. TCP New Vegas’ performance is also lower than that of TCP SACK, primarily due the congestion window taking longer to reach an optimal value.

Lowering the queue size to 128 packets has no impact on TCP Vegas and TCP New Vegas, however it does impact TCP SACK slightly. As for the previous queue size, TCP SACK forces the network to drop packets, however since the combined network and queue size is around 372 packets, the packets are dropped at an earlier point in time. With this configuration, 187 packets are dropped by the network between 8.46 and 9.33 seconds, with recovery and retransmission occurring between 9.90 and 11.39 seconds. The congestion window is reduced from 550 segments to 275 segments at the termination of slow-start, still being larger than the optimal size.

Decreasing the queue size further to 64 packets still has no impact on TCP Vegas and TCP New Vegas, however TCP SACK is now impacted significantly. With this configuration, TCP SACK only forces the network to drop 37 packets between 7.36 and 8.07 seconds. However, since the congestion window only reached 304 segments prior to the termination of slow-start, halving this results in a congestion window of 152 segments, 92 segments less than the optimal value. Given the large RTT of the network combined with the impact of delayed acknowledgments, congestion-avoidance only manages to increase the congestion window to 170 segments prior to connection termination. As a result, TCP SACK is surpassed by TCP New Vegas at the 24.6 seconds mark, achieving 8.2% less throughput than TCP New Vegas at the 30 second mark.

4.2. Multiple hops

Most real world networks exhibit multiple hops between end nodes, with 10 to 20 hops not being uncommon. The distance as measured in hops from the La Trobe University network to a number of common websites is given in table 1 along with the corresponding network RTT. It is interesting to note that the number of hops to websites hosted overseas is very similar to that of websites hosted locally, however the RTT is significantly higher due to the utilisation of a transcontinental link which exhibits a latency of around 70ms, adding 140ms to the RTT.

In order to evaluate performance over multiple hops, simulations were performed over 16 and 20 hops with a 2Mbps, 1ms latency link being implemented between each router. In order to emulate a connection to a transcontinental website, the simulations were run a second time with a latency of 70ms being used for the link between the eighth and ninth routers.
Table 1. Hop count and RTT measurements

<table>
<thead>
<tr>
<th>Site</th>
<th>Hops</th>
<th>RTT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
<td>20</td>
<td>212</td>
</tr>
<tr>
<td><a href="http://www.slashdot.org">www.slashdot.org</a></td>
<td>20</td>
<td>180</td>
</tr>
<tr>
<td><a href="http://www.kernel.org">www.kernel.org</a></td>
<td>12</td>
<td>175</td>
</tr>
<tr>
<td><a href="http://www.microsoft.com.au">www.microsoft.com.au</a></td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td><a href="http://www.whirlpool.net.au">www.whirlpool.net.au</a></td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td><a href="http://www.telstra.com.au">www.telstra.com.au</a></td>
<td>13</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 2. Difference between 16 and 20 hops

<table>
<thead>
<tr>
<th>Variant</th>
<th>34ms vs 42ms</th>
<th>172ms vs 180ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACK</td>
<td>99.4%</td>
<td>99.2%</td>
</tr>
<tr>
<td>Vegas</td>
<td>98.6%</td>
<td>93.9%</td>
</tr>
<tr>
<td>New Vegas</td>
<td>98.8%</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

The change in performance when the hop count is varied from 16 to 20 hops is negligible, with most of the impact resulting from the increased network RTT. Overall, TCP SACK outperforms both of the other variants, although TCP New Vegas is always a close second, spending slightly more time in slow-start due to its window increase algorithm. TCP Vegas suffers a decrease in performance as hop count is increased, more so due to the increased RTT time than the number of hops. A summary of the performance difference between 16 and 20 hops is provided in table 2.

4.3. Error-prone links

Wireless and satellite connections typically exhibit a high Bit Error Rate (BER) due to loss of signal strength, channel fading, noise and interference - all of which are undesired side-effects caused by radio based transmission. As a result, TCP connections that are routed over wireless or satellite links need to be able to cope with non-congestion based packet loss. In order to evaluate the ability of each variant to function over error-prone links, the base simulation was run with a uniform distributed BER being implemented on the link between nodes R and B. Error rates of 1 bit in 10^9, 1 bit in 10^7 and 1 bit in 10^6 were simulated.

For a network exhibiting a RTT of 20ms, each of the three variants perform near equivalent for an error rate of 1 bit in 10^9 or lower, as they do for an error-free link. However, when the error rate is increased to 1 bit in 10^7 the situation changes dramatically (see figure 2). Out of the three variants, TCP New Vegas is the only one which manages to achieve a congestion window size that approaches the optimal size, reaching 180 segments before slow-start is terminated. Both TCP Vegas and TCP SACK terminate much sooner, at 31 and 34 segments respectively. This allows TCP New Vegas to significantly outperform the other variants, achieving 337% more throughput than TCP SACK and 441% more throughput than TCP Vegas at the 30 second mark. When the error rate is further increased to 1 bit in 10^6, the three variants struggle with congestion window growth and the window size oscillates, growing during the error-free period before being decreased suddenly when a packet is lost due to corruption. TCP SACK suffers more than the Vegas variants, with the congestion window baseline continually decreasing during the 30 second period.

4.4. Competing flows

In order to evaluate TCP fairness, simulations were run with multiple flows competing for a share of a bottleneck link. Nodes A and B were connected to router R1 via separate 10Mbps duplex links with a latency of 2ms. Nodes
C and D were connected to router R2 in the same manner. Routers R1 and R2 were then connected using a 2Mbps duplex link with a 6ms or 276ms latency, resulting in a network RTT of 20ms and 560ms respectively. In order to prevent large transient queues, the queue size for each network interface was set to \(2^{ceil(\log_2 c)}\) where \(c\) is the number of packets required to fully utilise the BDP of the bottleneck link. A diagram of the simulated network is provided in figure 3. An FTP session is then simulated from A to C having a five minute duration. At 60 seconds, a second FTP session is established between nodes B and D, before being terminated at 240 seconds.

For a network exhibiting a RTT of 20ms, TCP Vegas is the only variant which is able to achieve a continuous fair allocation of bandwidth, rapidly converging to 1Mbps for each flow. Whilst TCP SACK passes the friendliness test, the amount of bandwidth achieved by each flow continually oscillated between 0.5Mbps and 1.5Mbps, at a frequency of between four and five seconds. The use of TCP SACK results in the network dropping 740 packets over the five minute period, compared to one drop for TCP New Vegas and zero for TCP Vegas. With this configuration TCP New Vegas fails to achieve fairness, with the second flow consuming a continual 1.3Mbps. This is possibly caused by the implementation of a more aggressive congestion-avoidance algorithm combined with Vegas’ tendency to yield bandwidth to competing flows.

For a network exhibiting a RTT of 560ms, TCP New Vegas is the only variant which achieves fairness (see figure 4), with the two flows converging to approximately 1Mbps around 100 seconds after the start of the second flow. Using TCP SACK, the first flow manages to retain most of the bandwidth, yielding only 0.7Mbps at the end of the 180 seconds. With TCP Vegas the second flow gains a mere 0.6Mbps of the bandwidth, remaining constant from 40 seconds through to termination. It is also worth noting that with TCP Vegas, the first flow never consumes all of the available bandwidth before and after the second flow.

Severe unfriendliness occurs when either TCP Vegas or TCP New Vegas attempt to compete with TCP SACK. This problem has been previously documented [9, 5] and largely relates to the fact that TCP SACK attempts to consume the available network capacity as well as all available buffer space within the network, resulting in large transient queues. In our simulations, when a TCP SACK flow already exists in a network exhibiting a 20ms RTT a TCP New Vegas flow only manages to acquire 0.6Mbps of the bandwidth. It is interesting to note that due to the short feedback delay, TCP New Vegas is able to increase its congestion window when TCP SACK decreases its congestion window due to packet loss. Once TCP SACK starts to increase its congestion window again, TCP New Vegas begins to decrease its window. This process results in both variants having a continually oscillating congestion window. When TCP New Vegas is used for the first flow and TCP SACK for the second, the aggressiveness of TCP SACK results in it gaining 1.2Mbps of the available bandwidth, once again remaining consistent for the duration of the connection. It would appear that TCP New Vegas is more capable of withstanding this aggressiveness than TCP Vegas which yields 1.4Mbps of the available bandwidth to TCP SACK.

When the network RTT is increased to 560ms, the unfriendliness becomes even worse due to the increased BDP, longer feedback delay and larger queue sizes. When TCP New Vegas is used for the second flow and TCP SACK for the first flow, the second flow is unable to gain more than 0.2Mbps, hovering around 0.1Mbps for most of the connection lifetime. TCP SACK yields slightly when the second flow starts, only to continue increasing its congestion window size, utilising more of the link bandwidth before backing off slightly when a loss event occurs. Using the variants in reverse TCP New Vegas quickly yields to TCP SACK, allowing the second flow to utilise half of the bandwidth within a very short period of time (see figure 4). Instead of only using its share of the bandwidth, TCP SACK continues to increase its congestion window which results in TCP New Vegas decreasing its congestion window. Within 140 seconds, TCP SACK has gained nearly 1.9Mbps of the available bandwidth, leaving just 0.1Mbps for the original flow. Once again TCP Vegas yields even more to this aggressiveness, gaining only 0.08Mbps of the available bandwidth after 105 seconds.

5. Further research and conclusion

This research has identified a number of areas for further research. Firstly, the RWC algorithm could be refined to enhance performance. Secondly, the implementation of some form of damping may allow for TCP New Vegas’ congestion window to remain more stable, in turn increasing achieved throughput over error-prone links and potentially preventing loss of bandwidth caused by yielding to TCP SACK flows. Thirdly, more research needs to be undertaken to identify ways that TCP New Vegas can compete more fairly with TCP SACK, whilst retaining its current perfor-
In this paper we have detailed research undertaken to validate and evaluate TCP New Vegas by exposing it via simulation to networks with various real-world properties. In most cases, TCP New Vegas has outperformed TCP Vegas and achieved increased throughput, primarily due to better management of congestion window growth. TCP New Vegas is a significant protocol for users of mobile, wireless and satellite networks, where packets are often lost due to corruption instead of congestion. Furthermore, increased throughput is achieved by maintaining a stable congestion window, unlike TCP SACK which forces the network to drop packets and maintain large transient queues.

6. Acknowledgment

This work was supported by the Australian Research Council (ARC) along with industry partners, Telstra Corporation and Ursys, under a research grant. We thank the ARC and our industry partners for their support and assistance. This project would not have been possible without the hard work, dedication and support of numerous people. In particular, we would like to thank Jean Armstrong, David Ensor, Michael Feremez, Mike Flavel, Peter Goddard, Robert Hinnenichs, Jennifer Jones, Tony Knox, Mary Martin, Phil Scott, Nikki Sing and Jeremy Swift.

References


