Performance Improvement of TCP Vegas over Heterogeneous Networks

Yi-Cheng Chan, Chia-Tai Chan, andYaw-Chung Chen
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan
ycchan@csie.nctu.edu.tw, ctchan@cht.com.tw, ycchen@csie.nctu.edu.tw

Abstract

Current IP network has become the dominant paradigm for all networking environments. The significant cause of packet losses in such heterogeneous networks is no longer limited to network congestion. Traditional TCP interprets every packet loss as caused by congestion which may be not the case in the current Internet. Misinterpretation of wireless random loss as an indication of network congestion results in TCP slowing down its sending rate unnecessarily. In this paper, we propose a new variant of TCP Vegas named RedVegas. By using the innate nature of Vegas and congestion indications marked by routers, RedVegas may detect random packet losses precisely. Through the packet loss differentiation, RedVegas reacts appropriately to the losses, and therefore the throughput of connection over heterogeneous networks can be significantly improved.

1. Introduction

Owing to the great advancement in wireless networking technology and new emerging applications, providing ubiquitous mobile Internet access becomes increasingly important. The well-known problem in providing TCP (Transmission Control Protocol) congestion control over heterogeneous networks (wired/wireless environment) is that current TCP implementations rely on packet loss as an indicator of congestion. In the wired networks, a congestion is indeed a likely reason of packet loss. On the other hand, a noisy, mobile, and fading radio channel is the most likely cause of loss in the wireless networks. The effective bit error rates in wireless networks are significantly higher than that in wired networks. Since TCP does not have any mechanism to differentiate between congestion losses and wireless random losses, the latter may cause a severe throughput degradation.

The purpose of congestion control is to dynamically adapt the connection transmission rate to the currently available capacity. TCP performs at an acceptable efficiency over the traditional wired networks where packet losses are caused by network congestion. However, when TCP observes random losses, it misinterprets such losses and reduces its window size, this causes the reduction of throughput unnecessarily. Therefore, TCP’s performance drops rapidly in the presence of frequent random losses [1].

TCP Vegas [2] exhibits many superior features than the most widely deployed TCP version, namely TCP Reno [3]. Studies have demonstrated that Vegas outperforms Reno in the aspects of overall network utilization, stability, fairness, throughput, and packet loss. Even in wireless Multi-hop networks, Vegas also keeps better performance than that Reno can achieve [4]. The throughput deterioration problem of TCP over wireless networks has been addressed [5, 6, 7, 8, 9, 10, 11]. These solutions can be divided into three categories: link layer mechanisms [7, 8], end-to-end approaches [5, 6], and base station schemes [9, 10, 11]. However, part of solutions are designed especially for Reno [5, 6], Vegas has not been given equal attention.

In this paper, we propose a random error detection mechanism for TCP Vegas (abbreviated as RedVegas hereafter). By using the innate nature of Vegas and congestion indications marked by routers, RedVegas may detect random packet losses precisely. Based on the simulation results we have found that the accuracy of random loss detection is close to 100%. Through the packet loss differentiation, RedVegas reacts appropriately to the losses, and therefore the throughput of connection over heterogeneous networks can be significantly improved.

The rest of this paper is organized as follows. Section 2 describes TCP Vegas and RedVegas. Section 3 presents the simulation results. Lastly, we conclude this work in Section 4.

2. TCP Vegas and proposed mechanism

TCP Vegas detects network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. It has been demonstrated that TCP
Vegas outperforms TCP Reno in many aspects. TCP Vegas includes three improvements as compared with TCP Reno: (1) a new retransmission mechanism, (2) an improved congestion avoidance mechanism, and (3) a modified slow-start mechanism. The detailed description of Vegas can be found in [2, 12]. In this section, we first review the design principles of the TCP Vegas that is most relevant to the innovations of RedVegas and then describe RedVegas in detail.

2.1. TCP Vegas

Vegas adopts a more sophisticated bandwidth estimation scheme that tries to avoid rather than to react to congestion. To find the available bandwidth of a connection, it estimates a proper amount of extra data to be kept in the network pipe and controls the congestion window size accordingly.

Vegas records the measured round-trip time (RTT) and sets $\text{BaseRTT}$ to the minimum of ever measured RTTs. It tries to detect incipient congestion by comparing the actual throughput to the expected throughput. The amount of extra data is between two thresholds $\alpha$ and $\beta$, as shown in the following:

$$\alpha \leq (\text{Expected} - \text{Actual}) \times \text{BaseRTT} \leq \beta,$$  \hspace{1cm} (1)

where Expected throughput is the current congestion window size ($\text{CWND}$) divided by $\text{BaseRTT}$, and Actual throughput represents the $\text{CWND}$ divided by the newly measured smoothed-RTT. The $\text{CWND}$ is kept constant when the amount of extra data is between $\alpha$ and $\beta$. If the amount is greater than $\beta$, it is taken as a sign for incipient congestion, thus the $\text{CWND}$ will be reduced. On the other hand, if the amount is smaller than $\alpha$, the connection may be under utilizing the available bandwidth. Hence, the $\text{CWND}$ will be increased.

As in Reno, a triple-duplicate acknowledgement (ACK) always results in packet retransmission. However, in order to retransmit the lost packets quickly, Vegas extends Reno’s fast retransmission strategy. Vegas measures the $\text{RTT}$ for every packet sent based on fine-grained clock values. Using the fine-grained $\text{RTT}$ measurements, a timeout period for each packet is computed. When a duplicate ACK is received, Vegas will check whether the timeout period of the oldest unacknowledgement packet is expired. If so, the packet is retransmitted. This modification leads to packet retransmission after just one or two duplicate ACKs. When a non-duplicate ACK that is the first or second ACK after a fast retransmission is received, Vegas will again check for the expiration of the timer and may retransmit another packet. Note that, packet retransmission due to an expired fine-grained timer is conditioned on received certain ACKs.

After a packet retransmission was triggered by a duplicate ACK and the ACK of the lost packet is received, the congestion window size will be reduced to alleviate the network congestion. There are two cases for Vegas to set the $\text{CWND}$. If the lost packet has been transmitted just once, the $\text{CWND}$ will be three fourth of the previous congestion window size. Otherwise, it is taken as a sign for a more serious congestion, and one half of the previous congestion window size will be set into $\text{CWND}$.

If a loss episode is severe enough that no ACKs are received to trigger fast retransmit algorithm, eventually, the losses will be identified by Reno-style coarse-grained timeout. When this occurs, the slow-start threshold ($\text{SSTHRESH}$) will be set to one half of $\text{CWND}$, and then the $\text{CWND}$ will be reset to two, and finally the connection will restart from slow-start.

2.2. The proposed mechanism

The issue of packet losses differentiation can be divided into two parts: (1) how to distinguish between congestion loss and random loss, and (2) how to make use of the information to refine the congestion window adjustment process. The success of our RedVegas relies on the cooperation of the end-hosts and routers. It assumes that the routers are capable of marking packets when congestion occurs.

The key idea of RedVegas is described as follows. Since Vegas always attempts to keep the amount of extra data between two thresholds $\alpha$ and $\beta$ in the network pipe. In fact, the extra data is major part of in-flight packets that stays in the bottleneck buffer. When an arriving packet is dropped due to the congestion in the bottleneck link. At this moment, it is very likely that some preceding packets belonging to the same connection have queued in the bottleneck buffer. Specifically, the amount of these preceding packets should be between $\alpha$ and $\beta$. If the router detects congestion and marks a congestion indication bit on all packets in the current buffer, the consecutive packets of the same connection prior to the dropped packet will very likely be marked. Based on the ACKs of the marked packets, a RedVegas source may infer the cause of packet loss accordingly.

Like the Explicit Congestion Notification (ECN) mechanism [13], RedVegas uses two bits in IP header as the congestion indication field (CI) and one bit in TCP header as the CI-echo flag. The first bit of CI represents the CI-capable transport (CICT) of a packet and the second bit serves as congestion experienced flag. When a RedVegas source wants to send a packet, it sets the CICT bit. Whenever a router drops a packet due to congestion, it will mark all packets’ congestion experienced flag which queued in the buffer and the CICT bit is set. Every time a RedVegas destination acknowledges a received packet, it checks whether the congestion experienced flag in the received packet is marked or not. If so, it will set the CI-echo flag in the ACK packet. As a RedVegas source receives an ACK, it checks
the CI-echo flag. If the flag is set, the sequence number\(^1\) of the newest acknowledged packet will be recorded in the variable \(NCSEQ\).

As in Vegas, RedVegas has three ways to detect packet loss, a triple-duplicate ACK, a fine-grained timeout, and a coarse-grained timeout. Whenever a packet loss is identified by a triple-duplicate ACK or a fine-grained timeout, RedVegas will try to infer the cause of loss. When a packet loss is detected and the difference between the sequence number of the lost packet and \(NCSEQ\) is no greater than \(\beta\), RedVegas assumes the loss is a congestion loss. Otherwise, random loss is inferred. If the losses are identified by a coarse-grained timeout, RedVegas does not intend to infer the cause of losses. Since the losses are severe and the information carried by the received ACKs may be passe. RedVegas leaves this part of algorithm to be intact.

In the proposed scheme, the ACKs with CI-echo flag of both the preceding and succeeding packets of a lost packet may assist the inference. To ease the illustration, we assume the part of consecutive packets of a connection that pass through the network pipe are packet \((i-3)\) to packet \((i+3)\), as shown in Figure 1. The lost packet \((i)\) is eventually identified by a triple-duplicate ACK, the value of \(\beta\) is three, and all ACKs of the consecutive packets successfully reach the source. If a congested router marks any preceding packet from packet \((i-3)\) to packet \((i-1)\) or succeeding packet from packet \((i+1)\) to packet \((i+3)\), the loss packet \((i)\) will not be judged as a random loss. Since the difference between the sequence number of the lost packet and the \(NCSEQ\) will be equal to or smaller than \(\beta\), a congestion loss is inferred.

In some situations, the ACKs with CI-echo flag of the preceding packets may fail to reach the connection source. Using the CI information carried by the ACKs of the succeeding packets may help RedVegas to reduce the possibility of misinterpreting a congestion loss as a random loss. Since the misinterpreting a congestion loss as a random loss may result in a wrong reaction that will violate the objective of congestion control. RedVegas intends to keep a high accuracy of random loss detection.

After the lost packet is recovered, RedVegas will adjust the congestion window size according to the loss differentiation. A connection need not to reduce the sending rate if the loss was not caused by congestion. Thus, if the lost packet is detected as a random loss, the \(CWND\) will not be changed, that is, the \(CWND\) will be equal to the congestion window size when the loss is detected. However, if a congestion loss is perceived, the same window-adjustment mechanism as that in Vegas will be adopted.

3. Performance evaluation

In this section, we compare the performance among the three TCP variants (Vegas, Reno, and RedVegas) and study the effectiveness of loss distinguishing in RedVegas by using the network simulator ns-2.1b9a [14]. The FIFO service discipline is assumed. All parameter settings of both Vegas and RedVegas are the same. Especially, \(\alpha = 1\) and \(\beta = 3\).

The size of each FIFO queue used in routers is 16 packets, the sizes of data packets and ACKs are 1 Kbytes and 40 bytes respectively. To ease the comparison, we assume that the sources always have data to send.

The network configuration for the simulations is shown in Figure 2, in which the bandwidth and delay of each full duplex link are depicted. Sources, destinations, and routers are expressed as \(S_1\), \(D_1\), and \(R_1\) respectively. The link between \(R_2\) and \(D_1\) is a wireless link on which we assume all random losses occur. In wireless environments, if the bit error is uniformly distributed, the larger a packet is, the more likely the packet will be corrupted [15]. Keeping this in mind, when we apply a random loss rate to data packets, we always set the proportionable random loss rate to ACKs.

A TCP connection is established from \(S_1\) to \(D_1\), and a VBR source is used to generate cross traffic from \(S_2\) to \(D_2\). Complying with the study of Internet simulating [16], the VBR source is a Pareto distribution ON-OFF source with shape parameter 1.5. During ON periods, the VBR source sends data at 1.6 Mb/s. In the following simulations, unless stated otherwise, the execution time of each sample point is

---

\(^{1}\)For a practical TCP implementation, a sequence number identifies the byte in the stream of data. In this paper, to ease the description of the proposed mechanism, we use a sequence number to represent the packet.
3.1. Basic behavior

The design goal of RedVegas is to improve performance for TCP Vegas over heterogeneous networks. It is obvious that if the packet losses are due to congestion, the behaviors of Vegas and RedVegas should be the same. In this subsection, we examine the average goodputs among the three TCP variants with different cross traffic loads. The difference between the throughput and goodput is that the later only counts those packets effectively received once. Each TCP variant is examined separately and the results can be found in Figure 3.

With the increasing cross traffic load, the average goodputs of the three TCP variants degrade correspondingly. Note that the goodputs of Vegas and RedVegas are always identical. The results demonstrate that the behaviors of Vegas and RedVegas are the same when there are no random losses. It implies that RedVegas does not misinterpret congestion loss as random loss in such simulation scenarios. From the simulation results, Vegas always surpasses Reno in goodput with different cross traffic loads, this conforms to the previous studies [2].

3.2. Impact of random loss

In this subsection, we compare the average goodputs among the three TCP variants with different random loss rates. No cross traffic is introduced in the simulations. By observing the results shown in Figure 4, both Vegas and RedVegas can fully utilize the bottleneck link when the random loss rate is zero. However, Reno can not maintain such high goodput with the same condition. Since Reno needs to create packet losses by itself to probe the available band-

width along the path. Therefore, certain amount of goodput is lost.

With the increasing random loss rate, the goodput improvement of RedVegas becomes obvious. When the random loss rate is 6 %, the goodput of RedVegas is about 2.84 times higher than that of Reno can achieve. When the random loss rate is between 3 % and 13 %, RedVegas always keeps a goodput improvement of larger than 20 % in comparison with Vegas. Notably, with 7 % random loss rate, the goodput improvement is up to 28.9 %. Moreover, the goodput improvements are 16.7 % and 40 % as compared with Vegas and Reno respectively in a severe random loss rate (15 %).

3.3. Impact of random loss and cross traffic

TCP connections over heterogeneous networks may experience both random losses and congestion losses. In this
subsection we introduce random losses and cross traffic into the simulation to examine the goodputs of three TCP variants. The results are shown in Figure 5 and 6.

Figure 5 depicts the goodputs of the three TCP variants with the random loss rate varying from 0 to 15% and the VBR source with 800 Kb/s averaged sending rate to generate cross traffic. The simulation results demonstrate that both Vegas and RedVegas can always maintain higher goodputs than Reno. As compared with Vegas, when the random loss rate is greater than 3%, RedVegas always achieves a more than 10% goodput improvement. In particular, when the random loss rate is 8%, the goodput improvement of RedVegas reaches 26.7%.

As the random loss rate is fixed at 5% and the cross traffic load varies from 0 to 0.9, the simulation results also illustrate that the goodputs of RedVegas are higher than Vegas and Reno as shown in Figure 6. Compared with Vegas, the goodput improvement of RedVegas is kept between 6.7% and 28.1%; while it is kept between 6.7% and 185% compared with Reno.

3.4. Numeric analysis

Through the above simulation results, we have demonstrated that the goodput of RedVegas is higher than that of Vegas whenever random losses are introduced. However, the effectiveness of packet loss distinguishing scheme in RedVegas has not been verified. To this end, we change the cross traffic loads from 0 to 90% to study the loss differentiation accuracy of RedVegas with the random loss rate fixed at 1%, 5%, 10%, or 15%. The execution time is 500 seconds for each sampling statistics. Table 1 represents the simulation results.

As the random loss rate is 1% and the cross traffic load is 40%, RedVegas detects 472 random losses and 36 congestion losses. For the latter there are 20 wrong judgements. For the 472 detected random losses, there is only one loss caused by congestion. As the random loss rate is 5% and the cross traffic load is 20%, RedVegas infers 1498 random losses and 23 congestion losses. Among the 23 detected congestion losses, RedVegas has 10 correct judgements. However, all the 1498 random losses inferred are correct. Based on the results shown in Table 1, we mainly have the following three observations: (1) the number of congestion losses is quite small in most of cases, (2) the accuracy of congestion loss detection is not very high, and (3) the accuracy of random loss detection is close to 100%.

Since RedVegas adopts the congestion avoidance mechanism used in Vegas, it precisely control the amount of extra data between $\alpha$ and $\beta$ in the bottleneck buffer. Therefore, the congestion losses can be effectively avoided. That is why the number of congestion losses is quite small in most of cases. Recall that RedVegas intends to keep a high accuracy of random loss detection by using the CI information carried by the ACKs of both the preceding and succeeding packets, consequently, some random losses may be possibly misinterpreted as congestion losses. The misinterpreting random loss as congestion loss may degrade throughput, this is same as in Vegas. At most, the misdiagnosis does not bring any goodput improvement to RedVegas. Finally, the most remarkable observation is that the accuracy of random loss detection is close to 100%. The results demonstrate the goodput improvement of RedVegas is indeed based on the correct random loss detections.

4. Conclusions

In this research, we propose an improved scheme, RedVegas, for TCP Vegas. With the ability of precise random loss detection, RedVegas reacts appropriately to the loss which caused by network congestion or transmission error, and consequently enhances the goodput of a connection over heterogeneous networks. Simulation results show the effectiveness of our proposed mechanism. However, there is still room for improvement. The bandwidth of the bottleneck link is under-utilized when the random loss rate is high. Therefore, a new design of the fast recovery mechanism for RedVegas would be our future work.

References


Table 1. Numeric analysis of packet loss differentiation for RedVegas

(a) Data Packet Random Loss Rate = 1%  

<table>
<thead>
<tr>
<th>Cross Traffic Load</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Random Loss</td>
<td>862</td>
<td>742</td>
<td>668</td>
<td>554</td>
<td>471</td>
<td>402</td>
<td>328</td>
<td>224</td>
<td>132</td>
<td>52</td>
</tr>
<tr>
<td>Random Loss Detected</td>
<td>862</td>
<td>742</td>
<td>668</td>
<td>554</td>
<td>471</td>
<td>402</td>
<td>328</td>
<td>224</td>
<td>132</td>
<td>52</td>
</tr>
<tr>
<td>Actual Congestion Loss</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>16</td>
<td>23</td>
<td>37</td>
<td>40</td>
<td>61</td>
<td>66</td>
</tr>
<tr>
<td>Congestion Loss Detected</td>
<td>0</td>
<td>11</td>
<td>16</td>
<td>33</td>
<td>36</td>
<td>39</td>
<td>67</td>
<td>60</td>
<td>116</td>
<td>116</td>
</tr>
</tbody>
</table>

(b) Data Packet Random Loss Rate = 5%  

<table>
<thead>
<tr>
<th>Cross Traffic Load</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Random Loss</td>
<td>1805</td>
<td>1759</td>
<td>1498</td>
<td>1567</td>
<td>1257</td>
<td>1137</td>
<td>1027</td>
<td>615</td>
<td>436</td>
<td>118</td>
</tr>
<tr>
<td>Random Loss Detected</td>
<td>1805</td>
<td>1760</td>
<td>1498</td>
<td>1568</td>
<td>1258</td>
<td>1137</td>
<td>1027</td>
<td>615</td>
<td>436</td>
<td>118</td>
</tr>
<tr>
<td>Actual Congestion Loss</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td>18</td>
<td>23</td>
<td>34</td>
<td>48</td>
<td>62</td>
</tr>
<tr>
<td>Congestion Loss Detected</td>
<td>0</td>
<td>8</td>
<td>23</td>
<td>28</td>
<td>47</td>
<td>51</td>
<td>57</td>
<td>88</td>
<td>152</td>
<td>168</td>
</tr>
</tbody>
</table>

(c) Data Packet Random Loss Rate = 10%  

<table>
<thead>
<tr>
<th>Cross Traffic Load</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Random Loss</td>
<td>719</td>
<td>555</td>
<td>694</td>
<td>602</td>
<td>695</td>
<td>616</td>
<td>441</td>
<td>453</td>
<td>322</td>
<td>89</td>
</tr>
<tr>
<td>Random Loss Detected</td>
<td>719</td>
<td>555</td>
<td>694</td>
<td>602</td>
<td>695</td>
<td>616</td>
<td>441</td>
<td>453</td>
<td>322</td>
<td>89</td>
</tr>
<tr>
<td>Actual Congestion Loss</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>29</td>
</tr>
<tr>
<td>Congestion Loss Detected</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>16</td>
<td>15</td>
<td>31</td>
<td>73</td>
<td>102</td>
</tr>
</tbody>
</table>

(d) Data Packet Random Loss Rate = 15%  

<table>
<thead>
<tr>
<th>Cross Traffic Load</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Random Loss</td>
<td>167</td>
<td>236</td>
<td>235</td>
<td>185</td>
<td>219</td>
<td>169</td>
<td>168</td>
<td>136</td>
<td>123</td>
<td>51</td>
</tr>
<tr>
<td>Random Loss Detected</td>
<td>167</td>
<td>236</td>
<td>235</td>
<td>185</td>
<td>219</td>
<td>169</td>
<td>168</td>
<td>136</td>
<td>123</td>
<td>51</td>
</tr>
<tr>
<td>Actual Congestion Loss</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Congestion Loss Detected</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>


