Modeling of Packet Switched Network over TCP Veno: TCP Improvement for Transmission over Wireless Access Networks.

Baswaraj
Department of MCA, CMR Institute of Technology , Bangalore-560037
Email: basawarajnb@gmail.com
Dr.D.C. Tomar
Department of Computer Science, MNM Jain Engineering College, Chennai-600020
Email: dctomar@gmail.com
Dr.K.Prasad Rao
Department of MCA, CMR Institute of Technology, Bangalore-560037
Email: prasadarao@gmail.com

ABSTRACT :

Computer networks form an essential substrate for the multitude of distributed application which is now an essential part of modern business and personal life. It is important to optimize the performance of computer networks, so that users can derive optimum utility from the expertise in network infrastructure. Most networks perform well when lightly used, but problems appear when network load increases. This loss of network performance when a network is heavily loaded is called congestion. Wired networks are becoming an integral part of the Internet. Unlike wireless networks, random packet loss due to bit errors is not negligible in wired networks, and this causes significant performance degradation of transmission control protocol (TCP). We propose and study a novel end-to-end congestion control mechanism called TCP Veno that is simple and effective for dealing with random packet loss. A key ingredient of Veno is that it monitors the network congestion level and uses that information to decide whether packet losses are likely to be due to congestion or random bit errors. Specifically: A). it refines the multiplicative decrease algorithm of TCP Reno—the most widely deployed TCP version in practice—by adjusting the slow-start threshold according to the perceived network congestion level rather than a fixed drop factor and B) it refines the linear increase algorithm so that the connection can stay longer in an operating region in which the network bandwidth is fully utilized, based on extensive network testbed experiments and live Internet measurements.

KEYWORDS :

Congestion control, congestion loss, random loss, transmission control protocol (TCP) Reno, TCP Vegas, TCP Veno wireless access networks

1.INTRODUCTION

Random packet loss in the wireless access networks can cause performance degradation in an end-to-end TCP connection. We have proposed and demonstrated that a novel TCP version called TCP Veno can deal with random packet loss effectively on an end-to-end basis. Numerous experiments [1]have been conducted in experimental networks and live Internet

DOI : 10.5121/acij.2012.3111

straddling across different areas in Hong Kong, and Mainland China. The detailed investigations clearly indicate that TCP Veno can achieve significant improvement without *adversely* affecting other concurrent TCP connections in the same network. Veno is desirable from the following three standpoints.

1).Deploy ability: For any improved TCP to be worthwhile, it must be easy to deploy over the existing Internet. Therefore, before modifying the legacy TCP design, we must ask the question: "Is this modification or design amenable to easy deployment in real networks?" To realize this goal, ideally, there should little or preferably no changes required at intermediate routers. For the two ends of the communicating terminals, it is preferred that only one side requires changes, so that the party interested in the enhanced performance can incorporate the new algorithm without requiring or forcing all its peers to adopt the changes. Generally, it is preferred that the sender that sends large volumes of data to many receivers to modify its algorithm (e.g., the Web server) rather than the receivers. The simple modification at the TCP sender stack makes Veno easily deployable in real networks. Any party who is interested in the enhanced performance can single-handedly do that by installing the Veno stack.

2) **Compatibility:** Compatibility refers to whether a newly introduced TCP is compatible with
the legacy TCP in the sense that it does not cause any detrimental effects to the legacy TCP, and vice versa. When they are running concurrently in the same network, Veno coexists harmoniously with Reno without" stealing" bandwidth from Reno. Its Improvement is attributed to its efficient utilization of the available bandwidth.

3) **Flexibility:** Flexibility refers to whether the TCP can deal with a range of different environments effectively. It is difficult to categorically declare one TCP version to be flexible or not flexible. We can say, however, Veno is more flexible than Reno in that it can deal with random loss in wireless networks better, alleviate the suffering in asymmetric networks [1], and has comparable performance in wired networks. Although the idea of Veno is straightforward on hindsight, it was not that obvious in the beginning when this work was first started. It combines elements from two opposing TCP camps:

- **Reno which uses reactive congestion control and**
- **Vegas which use proactive congestion control.**

Veno can still be regarded as a reactive algorithm because it certainly does not aim to eliminate packet loss entirely: it makes use of the idea of state estimation in Vegas to formulate better strategies to deal with packet loss and to stay in the "optimal" operating region longer. This paper has not addressed the issue of bursty packet loss in wireless networks. SACK [14] option has been proposed and shown to be effective in dealing with multiple packet losses within a congestion window. Veno by itself does not solve the problem of multiple packet losses.

However, Veno and SACK can be combined easily to yield an enhanced TCP implementation, SACK Veno. Reference [2] has investigated the performance of SACK Veno under different network conditions. The results show that SACK Veno can increase the throughput of pure SACK by up to 60% at packet loss rate of 0.01. As with comparison between Veno and Reno, experiments show that the improvement of SACK Veno can be attributed to its better efficiency rather than aggressiveness in grabbing bandwidth from other connections. Many TCP protocol stacks are now providing the SACK option. If SACK is already available on the receiver side, SACK Veno requires only the modification of the sender stack.

Generally speaking, we can see TCP Veno borrows the idea of congestion detection scheme in Vegas and intelligently integrates it into Reno's additive increase phase. Its state estimation is used as supplementary reference information to decide how to refine additive increase at the next step and how much the window is to be reduced once fast retransmit is triggered. Of course, principally, we could also use other predictive congestion detections (e.g., PBM [7], Packet Pair [25]) or/and the other better predictive congestion detection schemes, or combinations of these schemes to refine Reno evolution.

What TCP Veno proposes is to refine Reno's AIMD evolution over heterogeneous networks by using the complete judgment of network state estimation – congestive state or non-congestive state, rather than merely depending on packet loss occurrence.

To date, much work has been explored to center on accurately making out which of packet losses are due to congestion and which are due to bit-errors or other no congestion reasons, but, limited to too many uncertain factors (i.e., background traffic changing along the connection path) in real networks, progress in this kind of judging looks very slow. Perhaps, it may be more meaningful and more practical for a TCP connection to differentiate between congestive drops (occurring in congestive state) and no congestive drops (occurring in non congestive estate). Veno adopts such differentiation to circumvent the packet-loss-type-distinguishing

1.CONGESTION AVOIDANCE:

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid congestion at common network bottlenecks. Congestion avoidance is achieved through packet dropping. Among the more commonly used congestion avoidance mechanisms is Random Early Detection (RED), which is optimum for high-speed transit networks. Cisco IOS QoS includes an implementation of RED that, when configured, controls when the router drops packets. If you do not configure Weighted Random Early Detection (WRED), the router uses the cruder default packet drop mechanism called tail drop for an explanation of network congestion. This is the default congestion avoidance behavior when WRED is not configured.

1.TCP VEGAS

Following a series of congestion collapses starting in October of '86, Jacobson and Karels developed a congestion control mechanism, which was later named TCP Tahoe [5]. Since then, many modifications have been made to TCP, and different versions have been implemented such as TCP Tahoe and Reno. TCP Vegas was first introduced by Brakmo et al. in [2]. There are several changes made in TCP Vegas. First, the congestion avoidance mechanism that TCP Vegas uses is quite different from that of TCP Tahoe or Reno. TCP Reno uses the loss of packets as a signal that there is congestion in the network and has no way of detecting any incipient congestion before packet losses occur. Thus, TCP Reno reacts to congestion rather than attempts to prevent the congestion. TCP Vegas, on the other hand, uses the difference between the estimated throughput and the measured throughput as a way of estimating the congestion state of the network. We describe the algorithm briefly here. For more details on TCP Vegas, refer to [2]. First, Vegas sets Base RTT to the smallest measured round trip time, and the expected throughput is computed according to

$$\text{Expected} = \frac{\text{Window Size}}{\text{Base RTT}}$$

Where Window Size is the current window size. Second, Vegas calculate the current Actual Throughput as follows. With each packet being sent, Vegas records the sending time of the packet by checking the system clock and computes the round trip time (RTT) by computing the elapsed time before the ACK comes back. It then computes Actual throughput using this estimated RTT according to

$$\text{Actual} = \frac{\text{Window Size}}{\text{RTT}}$$

Then, Vegas compares Actual to Expected and computes the difference

$$\text{Diff} = \text{Expected} - \text{Actual};$$

This is used to adjust the window size. Note that Diff is non-negative by definition. Define two Threshold values, $0<\alpha<\beta$. If $\text{Diff}<\alpha$, Vegas increases the window size linearly during the next RTT. If $\text{Diff} > \beta$, then Vegas decreases the window size linearly during the next RTT. Otherwise, it leaves the window size unchanged. What TCP Vegas attempts to do is as follows. If the actual throughput is much smaller than the expected throughput, then it suggests that it is likely that the network is congested. Thus, the source should reduce the flow rate. On the other hand, if the actual throughput is too close to the expected throughput, then the connection may not be utilizing the available flow rate, and hence should increase the flow rate. Therefore, the goal of TCP Vegas is to keep a certain number of packets or bytes in the queues of the network [2, 9]. The threshold values, α and β, can be specified in terms of number of packets rather than flow rate. Now note that this mechanism used in Vegas to estimate the available bandwidth is fundamentally different from that of Reno, and does not purposely cause any packet loss. Consequently this mechanism removes the oscillatory behavior from Vegas, and Vegas achieve higher average throughput and efficiency. Moreover, since each connection keeps only a few packets in the switch buffers, the average delay and jitter tend to be much smaller. Another improvement added to Vegas over Reno is the retransmission mechanism. In TCP Reno, a

rather coarse grained timer is used to estimate the RTT and the variance, which results in poor estimates. Vegas extend Reno's retransmission mechanism as follows. As mentioned before, Vegas record the system clock each time a packet is sent. When an ACK is received, Vegas calculates the RTT and uses this more accurate estimate to decide to retransmit in the following two situations [2] When it receives a duplicate ACK, Vegas checks to see if the RTT is greater than timeout. If it is, then without waiting for the third duplicate ACK, it immediately retransmits the packet. When a non-duplicate ACK is received, if it is the first or second ACK after a retransmission, Vegas again checks to see if the RTT is greater than timeout. If it is, then Vegas retransmit the packet.

1.REROUTING

TCP Vegas that was first suggested by Brakmo et al. [2] does not have any mechanism that handles
Rerouting of the connection. In this section, we will show that this could lead to strange behavior
for TCP Vegas connections. Recall that in TCP Vegas BaseRTT is the smallest round trip delay,
Which is an estimate of the propagation delay of the path . First, if the route of a connection is changed by a switch, then without an explicit signal from the switch, the end host cannot directly detect it. If the new route has a shorter propagation delay, this does not cause any serious problem for TCP Vegas because most likely some packets will experience shorter round trip delay and Base RTT will be updated. On the other hand, if the new route for the connection has a longer propagation delay, the connection will not be able to tell whether the increase in the round trip delay is due to congestion in the network or a change in the route. Without this knowledge the end host will interpret the increase in the round trip delay as a sign of congestion in the network and decrease the window size. This is, however, the opposite of what the source should do. When the propagation delay of connection **i** is d_i, the expected number of backlogged packets of the connection is $w_i - r_i d_i$, where w_i is connection **i**'s window size and r_i is the flow rate. Since TCP Vegas attempts to keep between α and β packets in the switch buffers, if the propagation delay increases, then it should increase the window size to keep the same number of packets in the buffer. Because TCP Vegas relies upon delay estimation, this could impact the performance substantially. We have simulated a simple network in order to see how TCP Vegas handles changes in propagation delay.

Connection #	Propagation Delay (ms)	Starting Time (sec)
1	60	0.0
2	54	2.0
3	48	5.0
4	64	7.0
5	56	10.0
6	42	15.0
7	74	20.0
8	44	22.0
9	50	25.0
10	52	40.0

Table 1: Propagation delays and Starting times

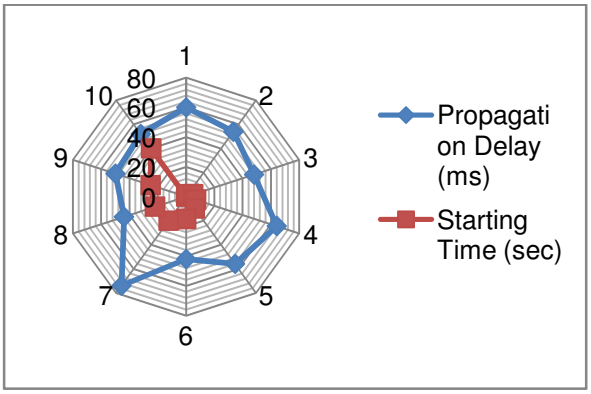

Figure 1. The graph shows the propagation delays & starting time

The queue size increases with the number of active connections. Even though the existing connections reduce their window sizes, the incorrect measurement of the true propagation delay causes the new connections to set the window sizes to larger values than they should be. At t = 50 sec, the average queue size is almost 60 while there are only ten active connections. If each connection had an accurate measurement for baseRTT, the average queue size should not exceed 30.

1. CONCLUSION AND FEATURE ENHANCEMENT

Attaching a service policy configured to use WRED to an interface disables WRED on that interface. If any of the traffic classes that you configure in a policy map use WRED for packet drop instead of tail drop, you must ensure that WRED is not configured on the interface to which you intend to attach that service policy.

RED reduces the chances of tail drop by selectively dropping packets when the output interface begins to show signs of congestion. By dropping some packets early rather than waiting until the buffer is full, RED avoids dropping large numbers of packets at once and minimizes the chances of global synchronization. Thus, RED allows the transmission line to be used fully at all times.

In addition, RED statistically drops more packets from large users than small. Therefore, traffic sources that generate the most traffic are more likely to be slowed down than traffic sources that generate little traffic.

2. REFERENCES

[1]. C. P. Fu, “TCP Veno: End-to-End Congestion Control Over Heterogeneous Networks,” Ph.D. Dissertation, The Chinese Univ. Hong Kong,, Hong Kong,, 2010
[2]. K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno, and SACK TCP,” Comput. Commun. Rev., vol. 26, no. 3, pp. 5–21, July 2007.
[3]. S. Keshav, “A Control-Theoretic Approach to Flow Control,” in Proc. SIGCOMM’91, Zurich, Switzerland, Sept. 1999, pp 3–15
[4]. Cheng Peng Fu, Associate Member, IEEE, and Soung C. Liew, Senior Member, IEEE“TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks” IEEE Journal On Selected Areas In Communications, Vol. 21, No. 2, February 2003,Pp-216-228
[5].Computer Networks: Andrew S. Tenenbaum
[6].T.Bonals. “Comparison of TCP Reno and TCP Vegas via Fluid Approximation.”
[7]. J. Mo, R .La, V . Ananthram, J . Walrand. “Analysis and comparison of TCP Reno and Vegas.” INFOCOM99.
[8] J. Ahn, P. Danzig, Z. Liu, and L. Yan, Evaluation of TCP Vegas: emulation and experimen",Computer Communication Review, Vol. 25, No. 4, pp. 185-95, Oct. 1995.
[10] L.S. Brakmo, S. O'Malley, and L.L. Peterson. \TCP Vegas: New techniques for congestion Detection and avoidance", Computer Communication Review, Vol. 24, No. 4, pp. 24-35, Oct. 1994.
[11] 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, Vol. 13, No. 8, pp. 1465-80, Oct. 1995.
[12] S. Floyd and V. Jacobson, \Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, Vol. 1, No, 4, pp. 397-413, August 1993.
[13] V. Jacobson, \Congestion avoidance and control.", Computer Communication Review, Vol. 18,No. 4, pp. 314-29, August 1988.

[14] J. Mo, R.J. La, V. Anantharam, and J. Walrand, Analysis and Comparison of TCP Vegas.", Available at http://www.path.berkeley.edu/ hyongla, June 1998.
[15] J. Mo and J. Walrand, \Fair End-to-end Window-based Congestion Control", SPIE '98 International Symposium on Voice, Video, and Data Communications, Nov. 1998.
[16] W.R. Stevens, TCP/IP Illustrated, Vol. 1, Addison-Wesley Pub. Co., 1994.
[17] J. Walrand Communication networks: arts course. WCB/McGraw-Hill, Boston, MA, 1998