Research Inventy: International Journal Of Engineering And Science
Vol.2, Issue 8 (March 2013), Pp 33-37
Issn(e): 2278-4721, Issn(p):2319-6483, Www.Researchinventy.Com

Analysis of Tcp Vegas in Linux 2.6.1

[1,]Ashish Vikram Singh, [2,]Anjali Chauhan [3,]Dr. Sarika Agarwal
[1,2,3,4,]Dronacharya College of Engineering,Department of Computer Science

Abstract-*This paper investigates the congestion control capabilities of TCP-CReno and Vegas introduced in Linux Kernel 2.6 for streamlining the kernel deploying CUBIC Congestion Control Protocol In this paper we use multilink multisource model which would consist of a distributed optimized algorithm and will provide a fundamental layout of delay, fairness and loss properties of TCP Vegas. It implies that Vegas stabilizes proportionally for allocation of network capacity when there is sufficient buffering in the network i.e. it clarifies the mechanism through which any kind of persistent congestion may arise and also its consequences. Thereafter, suggests how we might use active queue management to prevent it.*

Keywords*- TCP-Reno, TCP-Vegas, AIMD*

I. INTRODUCTION

The computer network, since the birth, specially the recent several years, obtained the fast development. The Internet has already been government, Business Company, social organization and individual a necessary fraction. Continuous development along with the network scale, the network congestion becomes more and more serious. The primary congestion reason in network is that the users need the transmission capacity and handling ability are much more than the limit which the actual network can provide, thus causes that the data loss rate rises as well as network system application performance drops and other problems occur.The TCP protocols are the most dominated protocols in the Internet [2], their success lies in the good congestion control strategy. TCP congestion control generally adjust the congestion window size according to loss rate, this includes Reno[7], perhaps the most well-known TCP congestion control algorithm TCP Vegas was introduced in 1994 as an alternative source based congestion control mechanism for the internet. A Vegas source anticipates the onset of congestion by monitoring the difference between the rate it is expecting to see and the rate it is actually realizing. Vegas strategy is to adjust the source's sending rate in an attempt to keep a small number of packets buffered in the routers along the path. The global objective of Vegas is to maximize the aggregate utility of all resource, subject to the capacity constraints of the network resources.

II. PRE-REQUISITE

Vegas has been interpreted in the following manner- A source monitors the difference between its expected rate and its actual rate, and increments or decrements its window by one in the next round trip time according to whether the difference is less or greater than a parameter α.

A. Vegas Algorithm

The major drawback in TCP Reno is that it does not receive fair share of bandwidth[3] . In TCP Reno while a source does not detect any congestion, it continues to increase its window size by one during one round trip time obviously the connections with shorter delays can update the connections with shorter delay can update their window sizes faster than those with longer delays and thus faster than those with longer delays and thus steal higher bandwidth. It is harmful to the other version of TCP Connection with longer delays. It takes the difference between expected and actual flow rates to estimate the available bandwidth in the network.

Difference = (Expected Rate – Actual Rate) * Base RTT.
Base RTT = Min. Round Trip Time.
Expected Rate = CWND/Base RTT.
Actual Rate = CWND/RTT
CWND = Current Congestion window
RTT = Actual RTT
When network is not congested, the actual flow rate will be very close to expected flow rate otherwise Actual rate is smaller than Expected.

B . CP Vegas takes the difference and on this basis of this difference in flow rate [8], estimates congestion level and adjust/update the window size accordingly.

[1] First, the source computes the expected flow rate
Expected = CWND/BaseRTT,

[2] where CWND is the Current window size and BaseRTT is the minimum round trip time

[3] Second, the source estimates the current flow rate by using the actual round trip time according to
Actual = CWND/RTT
where RTT is the actual round trip time of a segment
The source, using the expected and actual flow rates, computes the estimated backlog in the queue from
Difference = (Expected Rate - Actual Rate) * BaseRTT.
Based on Difference, the source updates its window size

III. DELAY, FAIRNESS AND LOSS

A. Delay

The previous section developed two equivalent interpretations of the Vegas algorithm. The first is that a Vegas source adjusts its rate so as to maintain its actual rate to be between αs and βs KB/s lower than its expected rate, where αs (typically 1/ds) and βs (typically 3/ds) are parameters of the Vegas algorithm. The expected rate is the maximum possible for the current window size, realized if and only if there is no queuing in the path. The rationale is that a rate that is too close to the maximum underutilizes the network, and one that is too far indicates congestion. The second interpretation is that a Vegas source adjusts its rate so as to maintain between αsds (typically 1) and βsds (typically 3) number of packets buffered in its path, so as to take advantage of extra capacity when it becomes available.

B. Fairness

Although we did not recognize it at the time, there are two equally valid implementations of Vegas, each springing from a different interpretation of an ambiguity in the algorithm. The first, which corresponds to the actual code, defines the αs and βs parameters in terms of bytes (packets) per *round trip time*, while the second, which corresponds to the prose in [1], defines αs and βs in terms of bytes (or packets) per *second*. These two implementations have an obvious impact on fairness: the second favors sources with a largepropagation delay,In terms of our model, Theorem 1 implies that the equilibrium rates x∗are *weighted proportionally fair*. The first implementation has αs = α/ds inversely proportional to the source's propagation delay. Then the utility functions Us(xs) = αsds log xs = α log xs are identical for all sources, and the equilibrium rates are *proportionally fair* and are independent of *propagation* delays. We call this implementation *proportionally fair* (PF). The second implementation has identical αs = α for all sources. The the utility functions and the equilibrium rates are weighted proportional fair, with weights proportional to sources' propagation delays. it implies that if two sources r and s face the same path price, e.g., in a network with a single congested link, then their equilibrium rates are proportional to their propagation delays:
x∗rdr=x∗sds

In a network with multiple congested links, weighting the utility by propagation delay has a balancing effect to the 'beat down' phenomenon, if the propagation delay is proportional to the number of congested links in a source's path. We call the second implementation *weighted proportionally fair* (WPF).

C. Loss

Provided that buffers at links l are large enough to accommodate the equilibrium backlog, a Vegas source will not suffer any loss in equilibrium owing to the feasibility condition (2). This is in contrast to TCPReno which constantly probes the network for spare capacity by linearly increasing its window until packets are lost, upon which the window is multiplicatively decreased. Thus, by carefully extracting congestion in formation from observed round trip time and intelligently reacting to it, Vegas avoids the perpetual cycle of sinking into and recovering from congestion. This is confirmed by the experimental results.The study observes that the loss recovery mechanism, not the congestion avoidance mechanism, of Vegas makes the greatest contribution. This is exactly what should be expected if the buffers are so small as to prevent Vegas from reaching equilibrium.In [3], the router buffer size is 10 segments; with background traffic, it can be easily filled up, leaving little space for Vegas' backlog. The effect of buffer size on the throughput and retransmission of Vegas is illustrated through simulations in [4].

ANALYSIS OF TCP VEGAS

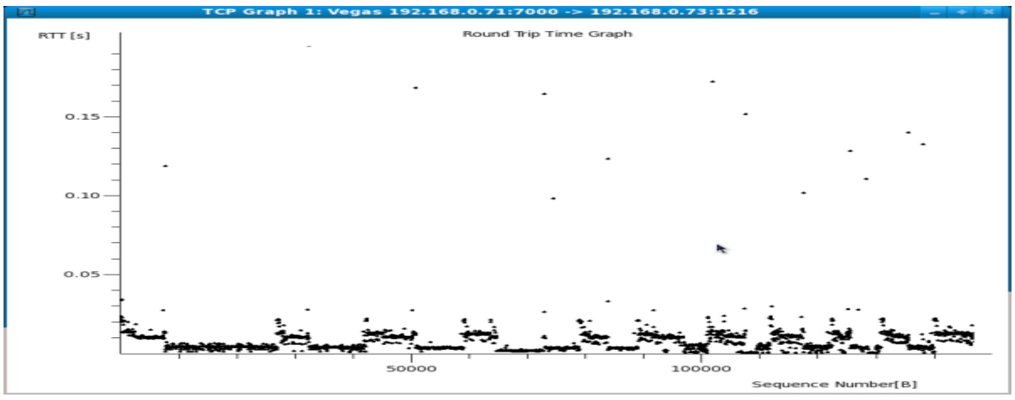

Fig : RTT Graph for TCP Vegas

In the analysis of TCP Vegas, we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB. As per the graph shown, the manimum RTT was around 0.001 sec and maximum RTT was around 0.175 We can calculate the capacity of the pipe as

capacity (bits) = bandwidth (bits/sec) × round-trip time (sec)
This is normally called the *bandwidth-delay product.* This value can vary widely, depending on the network speed and the RTT between the two ends.
On similar pattern, TCP Vegas Congestion Control is based on two parameters representing ‘expected’ and ‘actual’ rate calculated as –
Difference = (Expected Rate – Actual Rate) * Base RTT.
Base RTT = Min. Round Trip Time.
Expected Rate = CWND/Base RTT.
Actual Rate = CWND/RTT
CWND = Current Congestion window
RTT = Actual RTT
If we analyze our graph that show following parameters.
Here,
Base RTT = 0.001
Current Window size = 65535 bits.
Expected Rate=65535/0.001=65535000
Actual Rate=65535/0.005=13107000
Difference=(65535000-13107000)*0.001=51828
The congestion window is adjusted depending upon the difference between expected and actual sending rates.

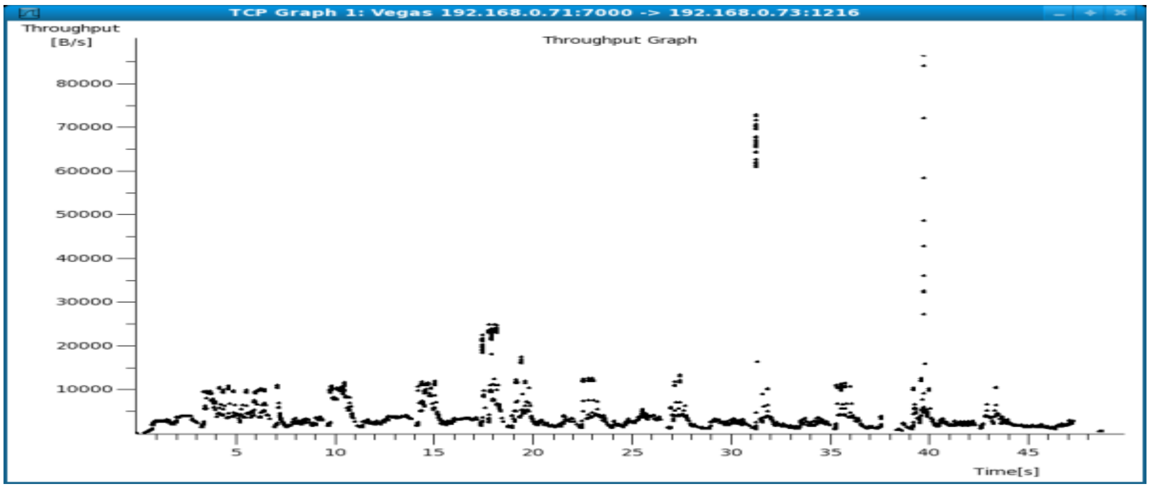

Fig : Throughput Graph for TCP Vegas

In the analysis of TCP Vegas Throughput, we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB. The random bursts of data attack on the socket receive buffers and TCP enters into congestion avoidance mode as per –

ssthresh = CWND/2

CWND = ssthresh + 3

When TCP Vegas source receives three duplicate *ack*s, it performs fast retransmit and fast recovery as TCP Reno. Actually, TCP Vegas develops a more refined fast retransmit mechanism based on a fine-grain clock. After fast retransmit TCP Vegas sets the congestion window to ¾ of the current congestion window and performs again the congestion avoidance algorithm. The graph shows that there is a cautious increase and successive decrease in the throughput due to Vegas algorithm. There are a large number of apparent traces showing the congestion window to become ¾ of the current congestion window. The throughput touches the peak of 679.6875 kbps with an immediate corrective congestion window size afterwards. The nature of the traffic is wedge shaped as there are no sharp increases like Reno and it clearly matches the objectives of the congestion control algorithm. The nature is also self similar since the peak of 78 kbps is repeated at regular intervals of around 5 sec. The peak of 600 kbps and above is repeated after almost 30 seconds.TCP Vegas in Linux has been extremely cautious and is able to deal with excessive traffic using AIMD pretty well. The sizing constraints of CWND and ssthresh recursively cut short the possibilities of congestion. On the downside, the throughput does not seem to follow sharp 'additive increase' in congestion window like Reno. This can be less useful in case of high speed networks. The congestion window has become more sensitive to congestion but less aggressive in following 'additive increase'. The graph clearly shows that the congestion happened after 50 sec and the congestion window cautiously dealed with it, finally showing a 'multiplicative decrease'.

IV. COUPLING BACKLOG AND PRICE

Vegas rely on the buffer process to compute its price. The *equilibrium* prices depend not on the congestion control algorithm but *solely* on the state of the network: topology, link capacities, number of sources, and their utility functions. As the number of sources increases the equilibrium prices, and hence the equilibrium backlog, increases. This not only necessitates large buffers in the network, but worse still, it leads to large feedback delay and possibly oscillation. Indeed, if every source keeps αsds= αpackets buffered in the network, the equilibrium backlog will be αN packets, linear in the number N of sources.

V. PROPAGATION DELAY ESTIMATION

We have been assuming in our model that a source knows its round trip propagation delay ds. In practice it sets this value to the minimum round trip time observed so far. Error may arise when there is route change, or when a new connection starts [5]. First, when the route is changed to one that has a longer propagation delay than the current route, the new propagation delay will be taken as increased round trip time, an indication of congestion. The source then reduces its window, while it should have increased it. Second, when a source starts, its observed round trip time includes queuing delay due to packets in its path from existing sources. It hence overestimates its propagation delay do and attempts to put more than αsds packets in its path, leading to persistent congestion.3 We now look at the effect of estimation error on stability and fairness.

A. REMARKS

We did not see persistent congestion in our original simulations of Vegas. This is most likely due to three factors. First, Vegas reverts to Reno-like behavior when there is insufficient buffer capacity in the network.Second, our simulations did not take the possibility of route changes into consideration, but on the other hand, evidence suggests that route changes are not likely to be a problem in practice [6]. Finally, the situation of connections starting up serially is pathological. In practice, connections continually come and go; hence all sources are likely to measure a base RTT that represents the propagation delay plus the average queuing delay.

REFERENCES

[1] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. IEEE Journal on Selected Areas in Communications, 13(8), October 1995. Available at http://netweb.usc.edu/yaxu/Vegas/Reference/brakmo.ps.

[2] Sanjeewa Athuraliya and Steven H. Low.Optimization flow control with Newton–like algorithm, Journal of Telecommunication Systems, 15(3/4):345–358, 2000.

[3] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas revisited. In Proceedings of IEEE Infocom, March 2000.

[4] Steven H. Low, Larry Peterson, and Limin Wang.Understanding Vegas: a duality model. In Proceedings of ACM Sigmetrics, June 2001.

[5] J. Mo, R. La, V. Anantharam, and J.Walrand. Analysis and comparison of TCP Reno and Vegas. In Proceedings of IEEE Infocom, March 1999.

[6] V. Paxson. End-to-end routing behaviour in the Internet. In Proceedings of SIGCOMM'96, ACM, August 1996.

[7] Feng W, Vanichpun S (2003). Enabling Compatibility Between TCP Reno and TCP Vegas. Proceeding of Symposium on Applications and the Internet (SAINT'03), Florida, USA.

[8] Hasegawa G, Murata M, Miyahara H (1999). Fairness and Stability of Congestion Control Mechanism of TCP. Proceeding of IEEE Conference on Computer Communications (INFOCOM'99), New York, USA.

[9] Hong Z, Amoakoh G, Zhongwei Z (2006). Performance of STT-Vegas in Heterogeneous Wired and Wireless Networks. Proceeding of 3rdInternational Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, Waterloo, Canada.

[10] Network Simulator NS-2 (2006). Information Sciences Institute, University of Southern California, USA. http://www.isi.edu/nsnam/ns/.

[11] Podlesny M, Williamson C (2010). Providing Fairness Between TCP NewReno and TCP Vegas with RD Network Services. Proceedings of 18th Workshop on Quality of Service (IWQoS), Beijing, China.

[12] Sing J, Soh B (2005). TCP New Vegas: Improving the Performance of TCP Vegas over High Latency Links. Proceeding of 4th IEEE International Symposium on Network Computing and Applications, NCA.

[13] Tsang ECM, Chang RKC (2001). A Simulation Study on the Throughput Fairness of TCP Vegas. Proceeding of 9th IEEE International Conference on Networks, Bangkok, Thailand.

[14] Wei DX, Jen C, Low SH, Hedge S (2006). Fast TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Transactions on Networking, 14(6): 1246-1259.