Scientific Research and Essays Vol. 6(9), pp. 2003-2010, 4 May, 2011
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE11.252
ISSN 1992-2248 ©2011 Academic Journals

Full Length Research Paper

Characterization and observation of (transmission control protocol) TCP-Vegas performance with different parameters over (Long term evolution) LTE networks

Ghassan A. Abed*, Mahamod Ismail and Kasmiran Jumari

Department of Electrical, Electronic and Systems Engineering, Faculty of Engineering and Built Environment, Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor Darul Ehsan, Malaysia.

Accepted 28 April, 2011

Despite the larger performance and higher throughput compared to transmission control protocol (TCP) Reno, TCP Vegas still has a few obstacles to be deployed in new networks, such as 4G LTE systems (4th Generation, Long term evolution). One of these obstacles is the Vegas congestion control that is not used in all available bandwidth capacity of the network path and which causes minimization in packet quantity transferred from source to destination. However, many researches have shown unfair treatment of TCP Vegas connections when they compete with Reno connections. So, Vegas need more developments and more modifications to be efficient over bidirectional links with unbalanced traffic, and on wireless links. TCP-Vegas is a congestion control algorithm that reduces queuing and packet loss, and thus reduces latency and increases overall throughput, by carefully matching the sending rate to the rate at which packets are successfully being drained by the network. This paper presents results from a series of simulation experiments designed to study TCP Vegas performance in LTE network model using NS-2 network simulator. The characterization and analysis of Vegas behavior performed using the main two parameters, alpha and beta, to configure the congestion window (*cwnd*) phases. After used multiple values, the configuration results show that TCP Vegas perform better than TCP Reno.

Key words: Transmission control protocol; TCP-Vegas, *cwnd*, long term evolution (LTE), NS-2.

INTRODUCTION

TCP Vegas is a modification of TCP Reno introduced by Brakmo and Peterson (1994). Vegas employs three techniques to increase throughput and decreases the packet loss, which are new retransmission mechanism, a congestion avoidance mechanism and a modified slow-start mechanism. TCP-Vegas is a proactive congestion control mechanism proposed to improve TCP performance by using round trip time (RTT) as a main parameter to monitor traffic condition and avoid congestion (Hong et al., 2006). TCP-Vegas detects congestion at an incipient stage based on increasing RTT values of the packets in the connection unlike other flavors like Reno, which detect congestion only after it has actually happened via packet drops. The algorithm depends heavily on accurate calculation of the base RTT value.

In more detail, Vegas adopts a more sophisticated bandwidth estimation scheme because it uses the difference between expected and actual flow rates to estimate the available bandwidth in the network (Zhang and Bian, 2002; Sing and Soh, 2005). The idea is that when the network is not congested, the actual flow rate will be close to the expected flow rate. Otherwise, the actual flow rate will be smaller than the expected flow rate, and Vegas using this difference in flow rates to estimate the congestion level in the network and updates

*Corresponding author. E-mail: ghassan@eng.ukm.my.

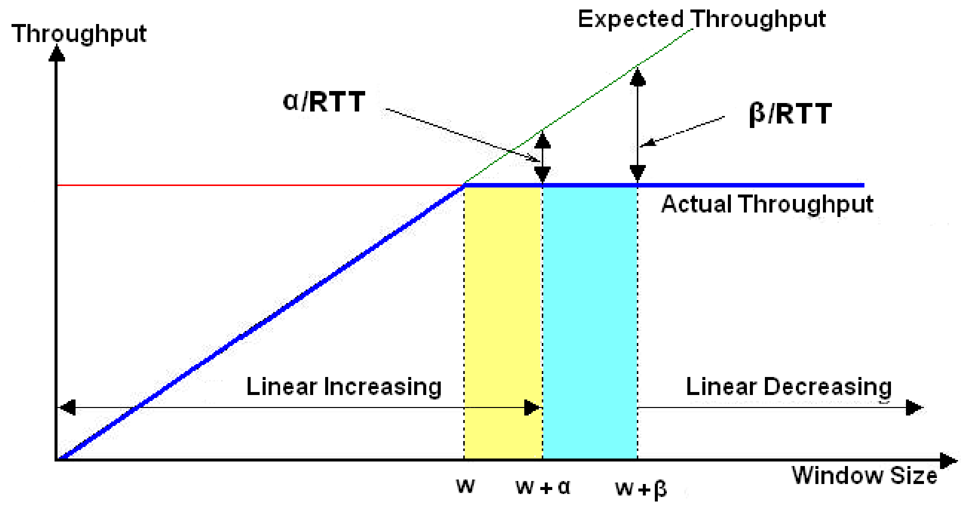

Figure 1. Phases and parameters effect of TCP Vegas.

the window size accordingly. To adjust the size of the *cwnd* appropriately, Vegas uses two threshold values, *α* and *β* to control the adjustment of *cwnd* at the source host as follow: During the congestion avoidance phase, Vegas sender does (Tsang and Chang, 2001):

cwnd = *cwnd* + 1 if Diff < (*α*/baseRTT)
cwnd = *cwnd* if (*α*/baseRTT) ≤ Diff ≤ (*β*/baseRTT)
cwnd = *cwnd* – 1 if (*β*/baseRTT) < Diff

Where:
Diff = expected rate – actual rate ≥ 0, by definition
Expected rate = data in transit/base RTT
Base RTT = the minimum of all measured RTT's
Actual rate = (next send sequence number – segment timed)/average RTT,
RTT = observed or actual round trip time (in seconds),
α, *β* = some constant thresholds.

In other words, Vegas increases *cwnd* by one packet if the per-flow queue at a bottleneck link is smaller than α, decreases *cwnd* by one packet if the per-flow queue is larger than β, and keeps *cwnd* unchanged otherwise (Podlesny and Williamson, 2010). Conceptually, Vegas tries to keep at last α packets but no more than β packets queued in the network. Thus, with only one Vegas connection, the window size of Vegas converges to a point that lies between (window + α) and (window + β) where window is the maximum window size that does not cause any queuing (Feng et al., 2003). Figure 1, shows the effects of different parameters on the behavior of Vegas throughput, where α and β represents the main controller to slow-start phase and congestion avoidance phase. It also shows how to control the throughput value for reducing the difference and contrast between the expected and the actual throughput.

Selecting *α* and *β* holds an implicit tradeoff between network utilization Good put, and fairness. By using the default setting for those parameters, that is, *α* and *β*, prior research inadvertently favored some other TCP variants over Vegas. The main aim of congestion avoidance algorithm of TCP Vegas is to measure and control the extra packets in the network. So, the congestion avoidance provides a monitoring to the changes in sending rate and to RTT's to predict congestion before losses occur. If we suppose that Vegas and Reno share a bottleneck link, Reno uses up most of the router buffer space. Vegas, interpret this as a sign of congestion, and will decrease the congestion window, which leads to an unfair share of bandwidth for Vegas.

Furthermore, when the router buffer size is large, the average of window size of Reno connections will be large, while the window size of Vegas connections will remain unchanged, as it does not inflate the window size larger than *β*.

Similar to Reno, Vegas uses a slow-start mechanism that allows a connection to quickly ramp up to the available bandwidth, but unlike Reno, to ensure that the sending rate will not increase too fast to congest the network during the slow start, Vegas doubles its congestion window size only every other RTT, and calculates the difference between the flow rates (Diff). Since Vegas estimates the baseRTT to compute the expected flow rate and adjust congestion window size, it is important to get an accurate baseRTT. Sometimes the sender may reduce its window size and it may cause inefficient throughput. In short, while Vegas assumes that packet delay or loss is due to congestion, in wireless networks, packet delays can also be due to changes in configuration of user equipment connected to the network. In this paper, a study of TCP Vegas is done and some features are simulated using the NS-2 simulator

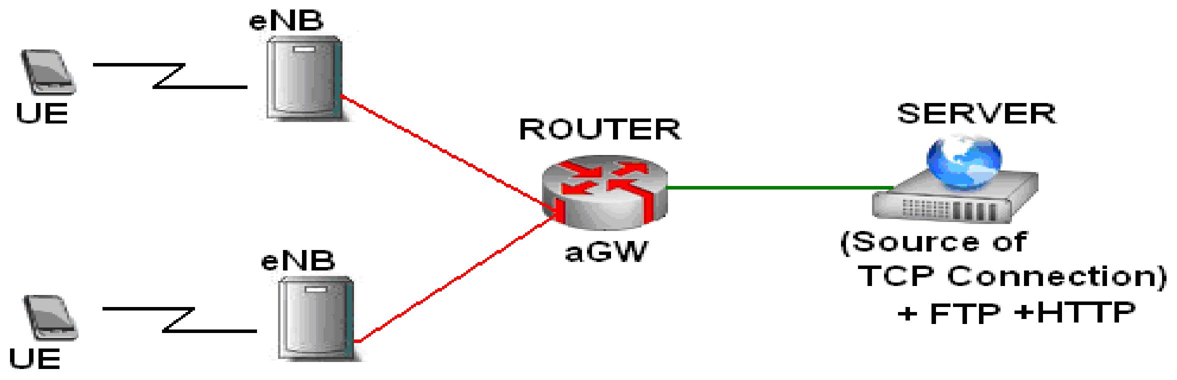

Figure 2. Simple LTE representation.

depending on change some parameters of Vegas algorithm congestion window over a model of LTE network and the results were discussed. The main objective of this research is to reach the most suitable model with best possible performance obtained from using TCP Vegas on LTE network.

The remainder of the paper is organized as follow. Subsequently, we describe the possibility and problems of implementation of Vegas using NS-2. The proposed network simulation model developed for this research was explained. Then, we illustrate the obtained results and analyze the effects of the parameters variances into the system. Further we present our conclusions and plans for future work.

TCP-VEGAS IMPLEMENTATION IN NS-2

In our research, we use NS-2 simulator, an object-oriented discrete event simulator targeted at networking research. It is developed at University of California and is written in C++ and Otcl. NS-2 supports a wide variety of network protocols offering simulation results for wired and wireless networks alike. NS-2 is a discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks (Network Simulator NS-2, 2006). In our research, we used NS-2 version 2.34. This version can be installed over Windows XP by using Cygwin, where Cygwin provides a Linux-like environment under Windows.

The tcp_vegas_alpha and tcp_vegas_beta entries control the number of packets that Vegas attempts to keep queued in the network in steady state. They are expressed in half-packet units. In default configuration Vegas tries to keep between 1 and 3 packets queued in the network and usually succeeds in stabilizes *cwnd* at a value that satisfies this constraint. This is the initial configuration, and we tried to change these parameters. In this configuration Vegas oscillates, keeping around 0 to 2 packets in the network. This is good because there will often be zero queuing delay, so that new Vegas flows will get an accurate notion of baseRTT, this will improve fairness between Vegas flows (Hasegawa et al., 1999). We have conducted considerable experimentation with TCP Vegas under NS-2, but this implementation faced some problems, because Vegas tries to sense queuing delays by observing changes in throughput each RTT and modulating *cwnd* accordingly to avoid loss. The Vegas version used does not use gamma parameter during the initial slow-start, we are considering it. But we used a large value of alpha and beta because they are required for high-speed links. There are two main known problems with TCP Vegas implementation in NS-2, setting of Slow-Start-Threshold *ssthresh*, and setting of alpha. The Vegas code works perfectly if there is no loss, but if there is a loss before delay is detected, *ssthresh* will be set by the Reno loss recovery algorithm to be *cwnd*/2. Then the flow exits slow start with a relatively large *ssthresh* (larger than 2). If *ssthresh* is higher than what the fair *cwnd* should be, this flow becomes lucky (Wei et al., 2006):

1) In the middle of an RTT, *cwnd* is increased by tcp_slow_start to *ssthresh*+1.
2) At the end of an RTT, since *cwnd*=*ssthresh*+1, the code goes to congestion avoidance, which reduces *cwnd* by 1.
3) But in the middle of the next RTT, *cwnd* is increased again to be *ssthresh*+1.

This loop keeps the *cwnd* to be *ssthresh*+1.

Since alpha equals to 1 in Vegas, this one packet worth of queuing delay stops the congestion window to grow in some points before there is really persistent queuing delay in the bottleneck. Setting alpha to be 2 (and change gamma and beta correspondingly) solves the problem.

PROPOSED TOPOLOGY AND SIMULATION

A simple LTE architecture has shown in Figure 2, it consists of one server for serving FTP, HTTP, and providesa source connection for the TCP link over the topology. In LTE system, the main job of aGW router is to control the flow rate of the streaming data from server to user equipment (UE) called evolved-NodeB (eNB), where these nodes responsible for buffering the data packets for UE over the network. Each eNB, connected to the corresponding aGW through wired simplex link of 5 Mbps bandwidth and 2 ms delay.

The proposed topology has shown in Figure 3, where six UE's are used, and connected directly to eNB within constant bandwidth and delay of 1 Mbps and 2 ms respectively. It means that we use 12 UE's nodes with two eNB's. The data stream from all UE's transferred through a main bottleneck. In fact it is not real bottleneck link, but we supposed that, here to get a bandwidth of 100 Mbps.

The parameters of modeling and simulation are presented in Table 1, so we can note that all link kept for one propagation delay

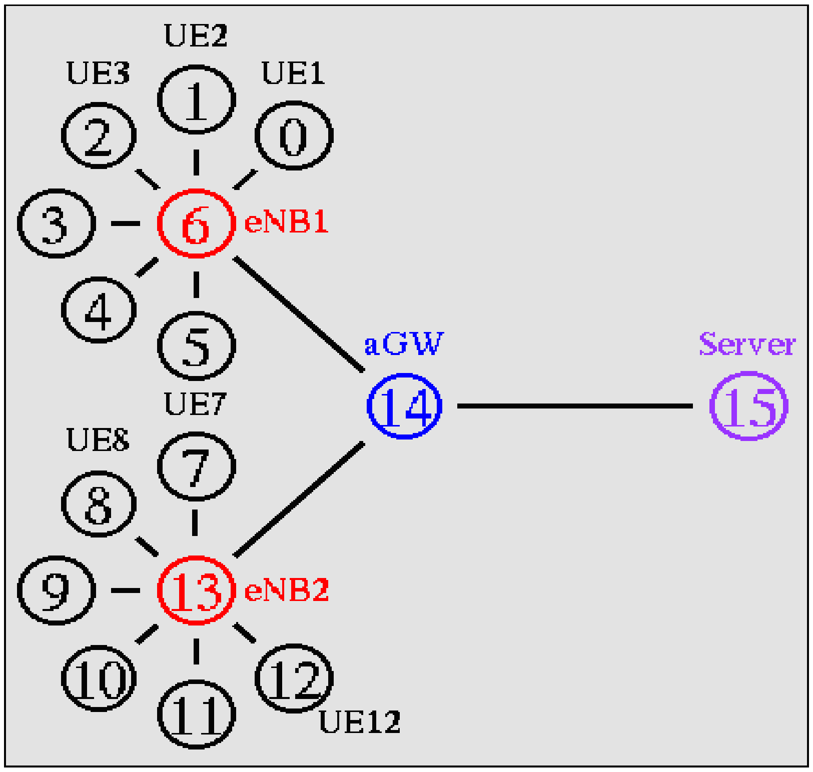

Figure 3. Real network animator (NAM).

Table 1. Simulation parameters.

Parameter	Value
TCP protocols	TCP Vegas
Propagation delay of all links	2 ms
Bandwidth of aGW link	5 Mbps
Bandwidth of server link	100 Mbps
Bandwidth of UE link	1 Mbps
Packet size	1500 Bytes
Window size	128 Kbytes
Simulation time	50 s

of 2 ms, and the maximum packet size of Vegas was set to 1500 Bytes, with minimum window size of 100 Kbytes. The router aGW, connected to the server with duplex link with bandwidth of 100 Mbps, and propagation delay of 2 ms.

As explained, the six UE's nodes, linked to the corresponding eNB, through wired link, but really these nodes must be wireless but we ignored mobility features, because they do not move yet, and if we support the movement of these nodes we must add a Handover scenario to the topology, and this is not our goal in this research, thus we ignore the interface between eNB's.

RESULTS ANALYSIS AND DISCUSSIONS

The goal of our experiments is to understand the performance of TCP Vegas over a network topology based on LTE system.

In the simulation model shown in Figure 3, the nodes 0 to 5 and 7 to 12 established with same parameters and behavior, where all of them use either Reno or Vegas. The NS-2 codes used in our experiments use TCP Vegas to change the default and individual parameters of congestion control algorithm over LTE connections. Figures 4 to 10, represent the comparison of *cwnd* for Reno and Vegas under similar network conditions, where the bandwidth, propagation delay, packet size, window size, and all other link parameters are kept the same, only Vegas parameters changed every in experiment. In Figure 4, we assumed alpha and beta have the same value, $\alpha = \beta = 20$. Actually the default vales of alpha and beta are set to 40, as shown in Figure 5. If we compare the window size between Figures 4 and 5, it is easy to conclude that when we duplicate the values of α and β from 20 to 40, the window size also duplicate from 24 Kbytes to 48 Kbytes.

As shown in Figures 4, 5, 6, 7 and 8, the size of *cwnd* increase when we increase the values of alpha and beta, so we note that the best performance is not supported by the default values, and we got a maximum window size when $\alpha = \beta = 120$, where the window size become 118 Kbytes.

This change in Vegas parameters permitting to get a

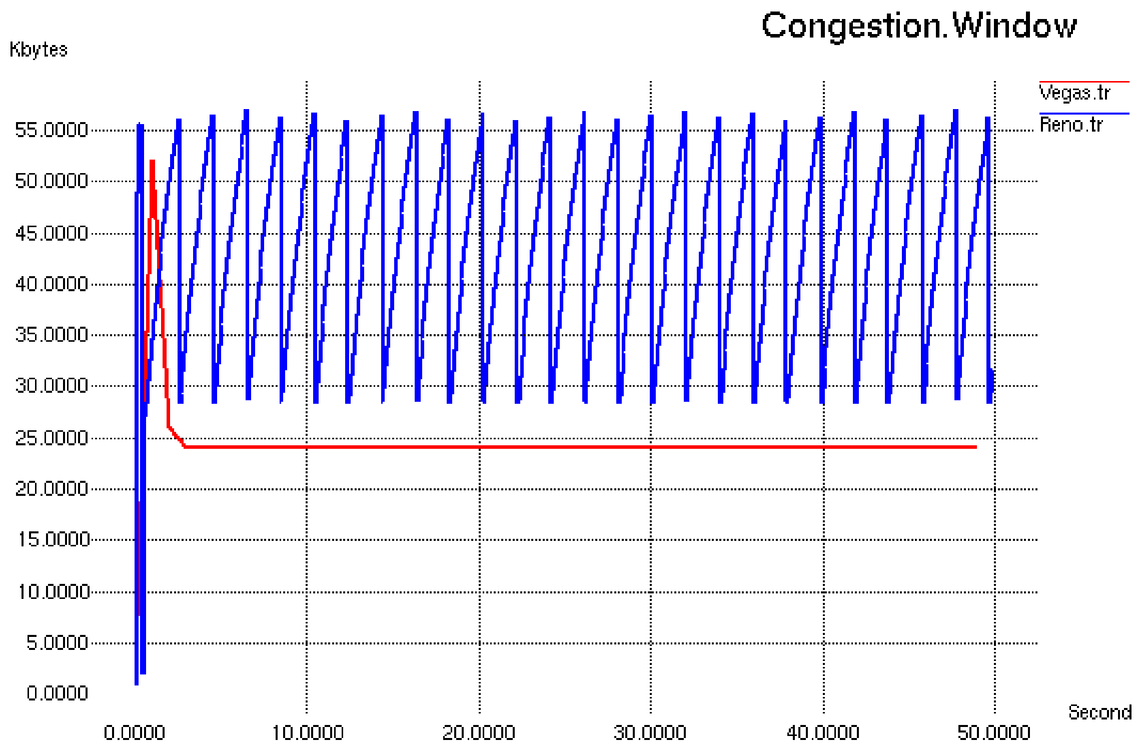

Figure 4. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = \beta = 20$).

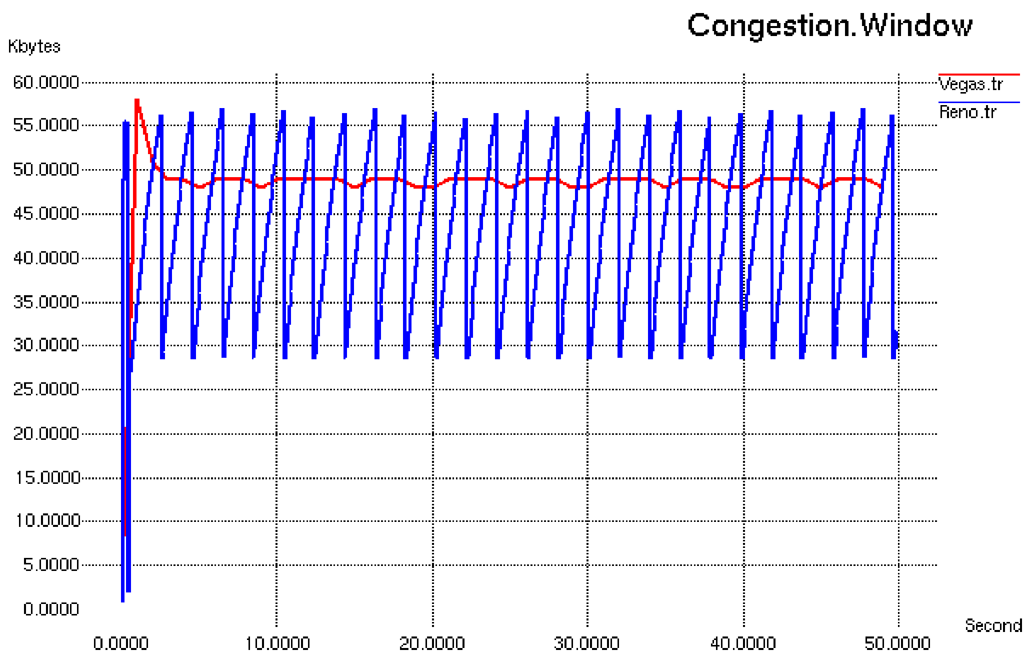

Figure 5. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = \beta = 40$).

new *cwnd* for same congestion window algorithm but with better performance, and if we compare the performance of Reno over the same topology, we can recognize that when we exchange the values of α and β, the window size improves.

In Figures 4 and 5, TCP Reno already have a better

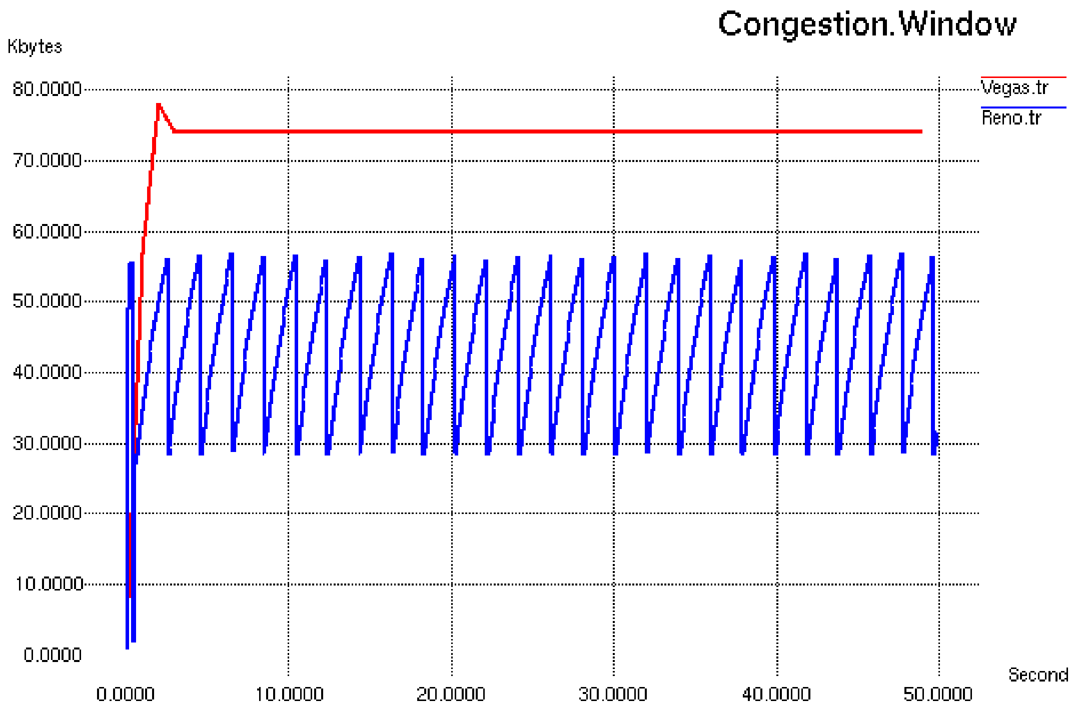

Figure 6. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = \beta = 60$).

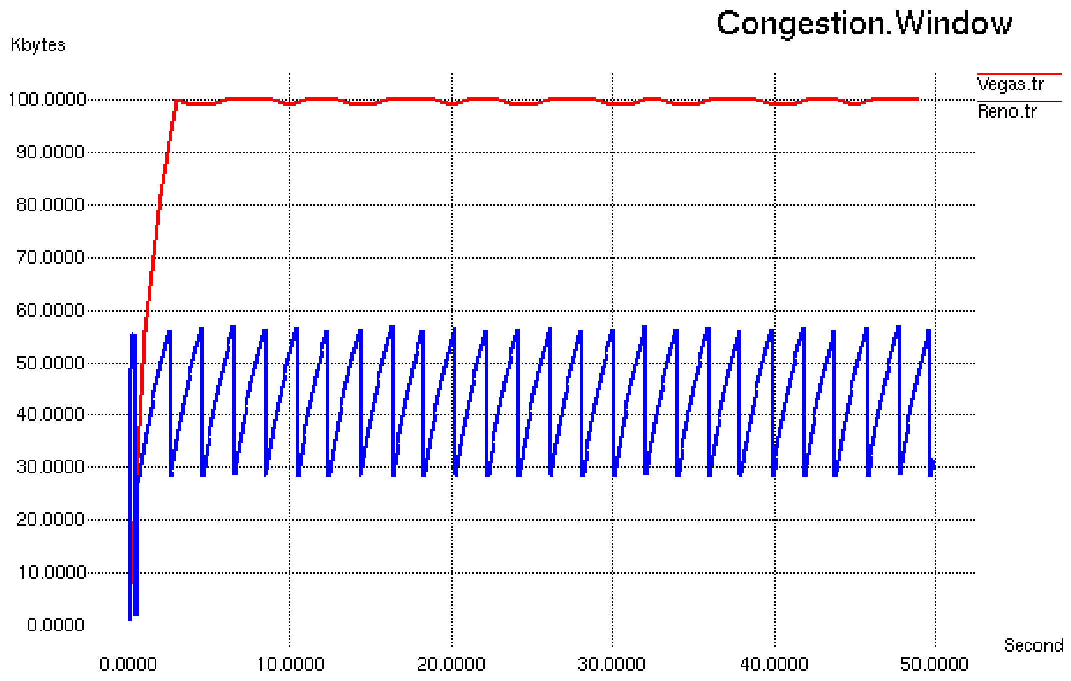

Figure 7. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = \beta = 80$).

cwnd than Vegas, where Vegas with 20 and 40 for α and β, but when α and β increased, Vegas start give a better window size than Reno, and that is appearing clearly in Figure 8, when the window size of Vegas become have twice the size that introduced by Reno.

Actually, Figures 4, 5, 6, 7 and 8, assumed that α and

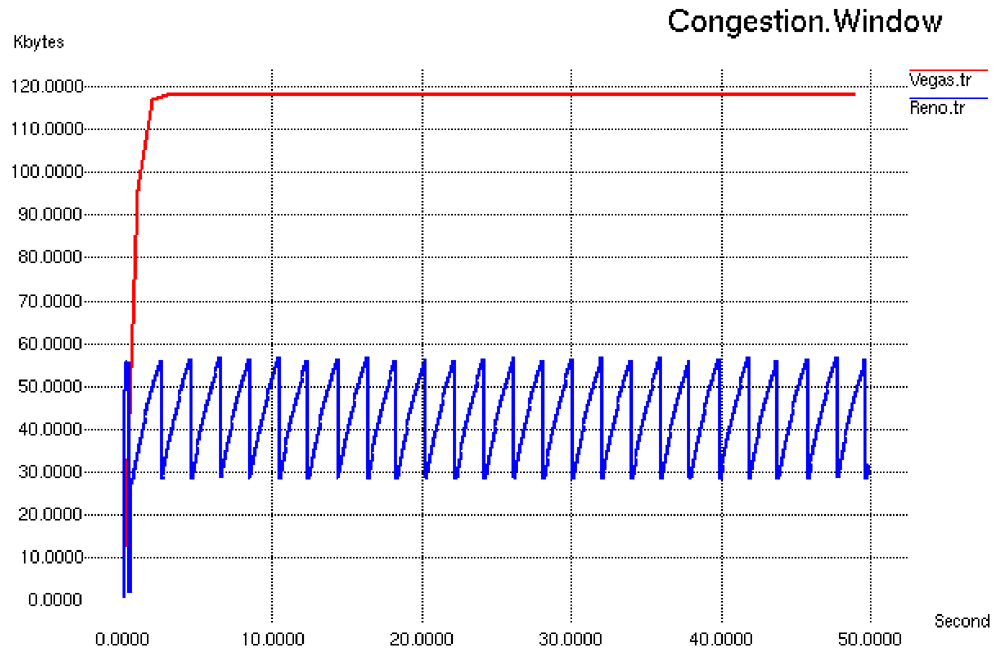

Figure 8. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = \beta = 120$).

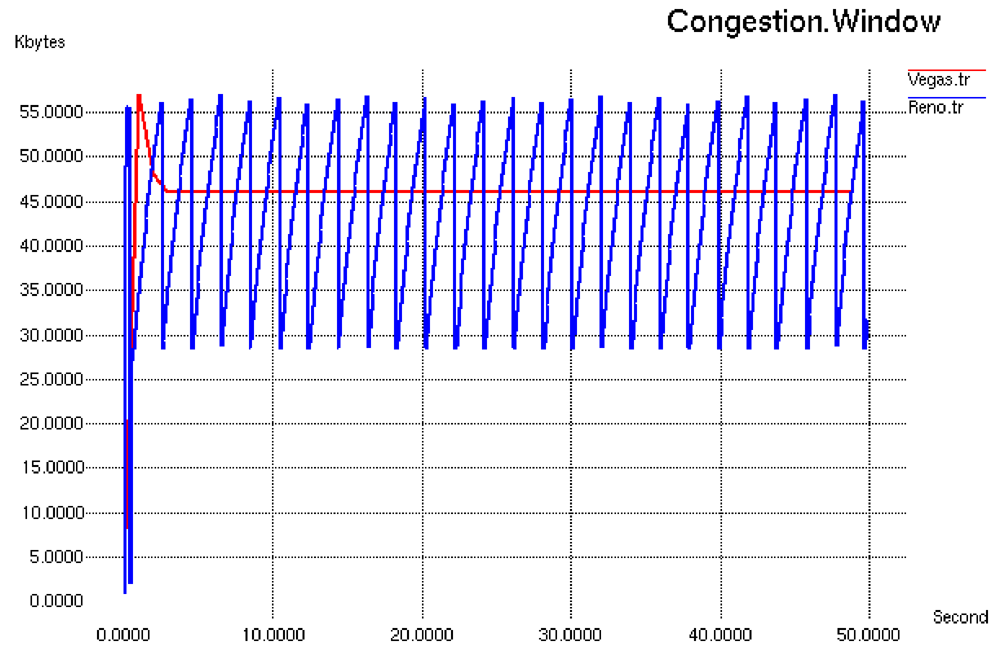

Figure 9. Comparison of *cwnd* for TCP Reno and Vegas ($\alpha = 20$, $\beta = 40$).

β, are equal or have a similar value when proceeded the experiments, for that we tried to use a different values of α and β, that is, $\alpha \neq \beta$, to see the real effects of α and β independently and to achieve which one have the bigger effect on window size.

Figures 9 and 10 represent the *cwnd* when $\alpha \neq \beta$. Here, we suppose that $\alpha = 20$ and $\beta = 40$, as shown in Figure 9, thus to see if *cwnd* is nearest when $\alpha = \beta = 20$, or when $\alpha = \beta = 40$, and that applied when $\alpha = 40$ and $\beta = 80$, as illustrated in Figure 10.

Generally, each α and β shared or combined the size of window over the link, and neither α nor β are responsible

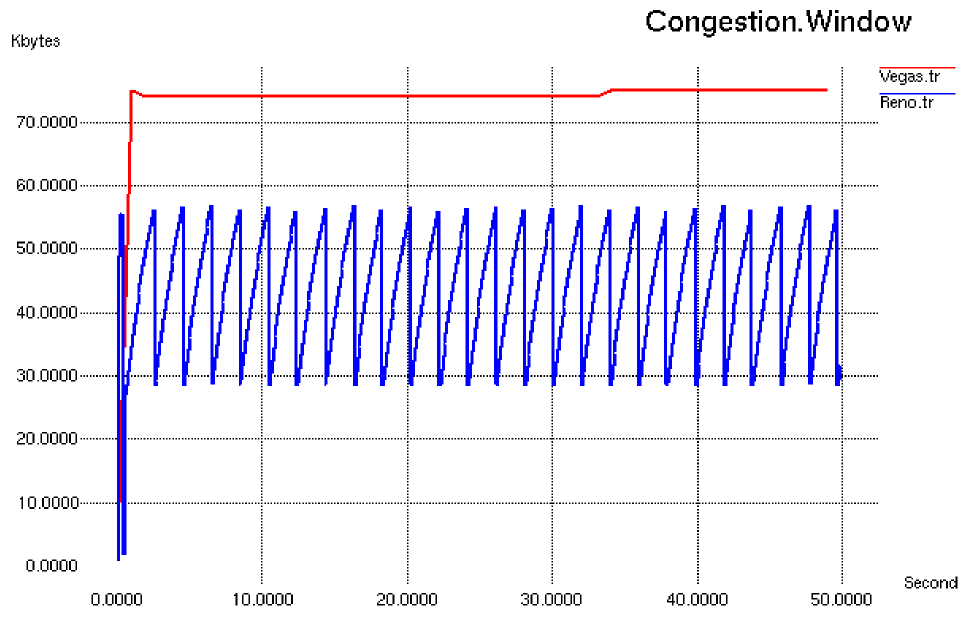

Figure 10. Comparison of *cwnd* for TCP Reno and Vegas (α = 40, β = 80).

on window size uniquely, then if we decide to choose a parameters values we must observe the selection of each value separately and individually and in isolation from the other parameter value.

Conclusion

We have presented in this paper the simulation results obtained for evaluating of TCP Vegas over LTE network model with different parameters. Several studies establish that TCP Vegas does achieve higher efficiency than Reno, causes much fewer packet retransmissions, and is not biased against the connections with longer RTT's. In our research we proofed that the default parameters of congestion window algorithm could not produce a competitor performance with other TCP variants, especially TCP Reno, but when we used other values of alpha and beta parameters we got a high performance notified to the degree of vulnerability.

However, while this paper leaves some questions unanswered, TCP Vegas clearly showed significant improvements with respect to TCP Reno, as highlighted by their effective use of graphs and performance measurements.

FUTURE WORK

Future work of this research will be on the compilation of developing or modifying the congestion window algorithm of TCP Vegas, and compare the results of this amendment with standard results.

ACKNOWLEDGMENT

This study is sponsored by Universiti Kebangsaan Malaysia (UKM) through the university research grant UKM-OUP-ICT-36-182/2011.

REFERENCES

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.

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.

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

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

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.

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.

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.

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

Zhang H, Bian Z (2002). Evaluation of Different TCP Congestion Control Algorithms Using NS-2.CMPT 885-3: Special Topics: High-Per, formance Networks, ENSC 835-3.