Communications and Network, 2013, 5, 148-156
http://dx.doi.org/10.4236/cn.2013.52016 Published Online May 2013 (http://www.scirp.org/journal/cn)

Comparative Analysis of TCP-Protocol Operation Algorithms in Self-Similar Traffic

Abed Saif Alghawli
College of Science and Humanities, Salman Bin Abdulaziz University, Aflaj, KSA
Email: alghauly@yahoo.com

Received March 7, 2013; revised April 7, 2013; accepted May 7, 2013

ABSTRACT

This paper presents simulation modelling of network under the conditions of self-similar traffic and bottleneck occurrence. The comparative analysis of different TCPs (NewReno, Reno, Tahoe and etc.) has been conducted along with the testing of various algorithms of these protocols activity. The use of TCP Vegas has been proved to be the most effective.

Keywords: Self-Similar Traffic; NewReno; Tahoe; Parallel TCP and Piggybacking

1. Introduction

TCP is the dominant protocol of the Internet. It delivers data in the form of byte streams, establishes the connection and is used in applications that require guaranteed delivery of messages. TCP applies batch totals to verify their integrity and releases application processes from timeouts and retransmissions to ensure reliability.

Numerous investigations of the Internet processes showed that the statistical characteristics of traffic possess the ability of time-scale invariance (self-similarity). Such an effect is caused by the specific character of file distributions along servers, their sizes, as well as by a typical behavior of users. It was found that data streams, which initially do not exhibit self-similarity properties, after being processed at the host server and at active network elements, start showing pronounced signs of self-similarity. This fact may cause fast buffer overloads even with low use factor. If no action is taken to limit the incoming traffic, then the queues on the most loaded lines will grow indefinitely and eventually exceed the size of the buffers at the corresponding nodes. This leads to the fact that the packages reentering to the nodes with full buffers will be reset and are to be retransmitted, and that in turn results in wasting network resources [1-4].

Traffic regulation in TCP assumes the existence of two independent processes: delivery control operated by the recipient using the Window parameter and congestion control operated by the sender by using the CWND—congestion window and the slow-start procedure (SSTHRETH slow start threshold). The first process monitors filling in the recipient's input buffer and the second one registers the channel congestion, the losses related and reduces the traffic level. The congestion window CWND and the SSTHRETH slow start procedure provides reconciling the full loading of the virtual connection and the current channel opportunities and minimizing the packet losses in case of overloading.

Some amendments and additions are constantly recorded in TCP in order to solve the problems appearing in the course of the protocol or to improve its performances for systems with focused specialization.

The purpose of this paper is to conduct a comparative analysis of TCP protocol types and algorithms of their work and to identify strengths and weaknesses of each algorithm in different operation scenarios in the network.

2. TCP Protocols

Let us consider the main types of TCP protocols: Tahoe, Reno, NewReno, SACK and Vegas [4-8].

TCP-Tahoe algorithm is the oldest and the most widely used one. The implementation of this algorithm has added many new algorithms and improvements to its earlier implementations. The new algorithms include: Slow Start, Congestion Avoidance, and Fast Retransmission. The improvements involve the modification of time reporting of the packets passing over the communication channel to the destination and back to set the timeout retransmission. The meaning of the congestion avoidance algorithm is to keep the value of CWND within possible maximum values. In fact this optimization is carried out with the help

of packet losses. If there are no packet losses, then the value of CWND reaches Window per default, as having been specified in the configuration of TCP driver.

For the connections with large windows the TCP Timestamps algorithm is applied. The timestamps allow accurate measurement of the round trip time RTT for the further correction of a value of the retransmission timer.

The paper is particularly focused on the algorithm of Rapid retransmission because it is modified in subsequent versions of TCP. When operated at the algorithm of Rapid retransmission after receiving a small number of duplicate confirmations for one TCP (ACK) packet, the data source concludes that the packet was lost and retransmits the packet and all packets sent after it with zero wait state for the retransmission timer, which leads to the reduction of the through-put capacity and to the increase of the already high channel loading level.

Getting the duplicate ACK is not a reliable sign of a packet loss. Duplicate ACKs arise even when changing the route exchange. For this reason the signal of loss is considered as the getting of three ACK packets in succession.

When the buffer is overfilled, an arbitrary number of segments will be lost. In such a case several scenarios may take place. The basic version—slow start launches within classical TCP-Tahoe algorithm when a segment loss and resulting timeout (RTO) at the sender's side since the sender will not receive the acknowledgment ACK for the lost segment. Slow start involves the installation of the congestion window equal to 1and the slow start threshold equal to half the value of CWND at which the timeout took place. The CWND reduces to one because the sender does not have any information about the status of the network. Then, after each i-th confirmation $CWND_{i+1} = CWND_i + 1$. This formula works until CWND is equal to ssthresh. Then the increase of CWND becomes linear. The point of this algorithm is to keep the value of CWND within maximum possible values. In fact this optimization is carried out with the help of packet losses. If there are no packet losses, the value of CWND reaches the value of the window by default as having been defined in the configuration of the TCP driver.

TCP Reno keeps the extensions included in Tahoe but it changes the operation of Fast retransmit by adding Fast Recovery. This new algorithm prevents the channel from being in an empty state after Fast retransmission and is not switched to Slow start phase to fill the channel after a single packet loss. Fast Recovery assumes that each received duplicate ACK is one package escaped from the channel. Thus, during Fast Recovery a TCP sender is able to calculate the amount of data sent.

The TCP sender enters into Fast Recovery after receiving the initial threshold of duplicate ACK. This threshold is usually set to be equal to three. Once the threshold of the duplicate ACK is received, the sender retransmits one packet and reduces its congestion window by half. Instead of Slow start, as in Tahoe TCP, Reno sender uses an additional parish of duplicate ACK to synchronize the subsequent outgoing packets.

In TCP-Reno normally the window size varies cyclically. The window size increases until the segment loss. TCP-Reno involves two phases of window re-sizing: a phase of slow start and congestion avoidance.

Developers named fast recovery algorithm—NewReno, as it differs greatly from the basic Reno algorithm. The proposed algorithm has certain advantages in comparison with the canonical Reno in different scenarios. However, there is one scenario when canonical Reno exceeds NewReno—when reordering the flow of packets.

NewReno algorithm uses the variable Recover (restore), whose initial value is equal to the original packet sequence number, and the algorithms of fast retransmission and fast recovery. When the option of the selective acknowledgment SACK is available, the sender knows which packages must be resent again at the phase of fast recovery.

The main implementation stages of the algorithm are:

1) Three duplicate ACKs: start of Fast Retransmit or without fast retransmit (the entry to the phase of fast retransmit and fast recovery is not applicable).

2) Entering fast retransmit phase: the congestion window is enlarged voluntarily in several segments (three) that have left the network and are buffered by the recipient.

3) Fast recovery: the congestion window is enlarged voluntarily in order to account the segment that has left the network.

4) Fast recovery, continuation: the segment is transmitted if it is allowed with the new CWND and window values announced by the recipient.

5) When ACK arrives with acknowledging the receipt of new data, then this ACK can be the confirmation related to the retransmission at the stage 2 or later retransmission.

6) Retransmission timeouts (RTO): after RTO the highest sequence number of the transmitted segment is recorded into the variable recover and the quit from the fast recovery stage is performed if it is possible.

7) Partial acknowledgement. The re-send of more than one packet and the resetting of the timer of the retransmission after the retransmission can be the optional response for a partial acknowledgment. In case of multiple packet losses the re-send of two packets when receiving the partial acknowledgement provides faster system recovery. Such an approach requires less time than N RTT in case of N packages loss. However, at the absence of the SACK option, the slow start provides sufficiently rapid system recovery without sending extra packets.

8) Specifies that TCP sender responds to the partial ACKs with the reduction of the congestion window to an amount corresponding to the volume of valid data. Thus, only one packet sent earlier is transmitted in response to each partial confirmation, but new packages can be sent as well, depending on the amount of the delivery confirmed partially.

9) Exclusion of multiple fast retransmissions.

TCP Vegas protocol implementation fits well in networks where it is necessary to determine the available bandwidth and adjust dynamically the optimal parameters. Vegas algorithm evaluates the buffering that occurs in the network and controls the rate of the flow correspondingly. The algorithm is able to calculate and reduce the flow rate before the packet loss occurs. It controls the size of the window by monitoring provided by the RTT sender (transmission time of the packets over the communication channel to the destination and back) for packets having been sent earlier. If there is an increase of RTT, the system recognizes that the network is close to overload and reduces the width of the window. If RTT is reduced, the sender will determine that the network has overcome the congestion and will increase the size of the window. Consequently, the window size in an ideal situation will tend to the desired value. This TCP modification requires high resolution of the sender's timer.

TCP SACK algorithm uses "Options" field of the TCP frame header for additional information about the packages received by a recipient. If there was a loss, then each segment of triple ACK, sent by the receiving station, contains information about the frame caused the sending of this segment. Thus, the sender after receiving the frame has the data not only about the lost frame but also about the frames that reached the recipient successfully. As a result the unnecessary re-send of the segments successfully buffered at the recipient side is avoided.

The same as in the case with Reno, TCP SACK goes into Fast recovery regime when receiving 3 duplicate ACKs. During Fast recovery the sender supports Pipe variable representing the number of packets in the network. This variable increases whenever a new segment has been sent and decreases if further confirmation has been obtained. The transfer of a new packet in the network is accepted if Pipe value is less than the congestion window.

The sender also maintains the data structure that stores the confirmations from the SACK option being under the acknowledgment. If the sender gets the permitted transfer, then he sends the next packet from the list of packages considered to be lost. If there are no such packets, then a new packet is transmitted. For connections with large windows the timestamp algorithm is applied. The timestamps help to conduct accurate time measurement of RTT access for further correction of values of the retransmission timer.

In case when the option of the selective acknowledgement SACK is available, the sender is aware which packages must be re-sent to the phase of Fast Recovery. In the absence of SACK option there is no sufficient information on the packages which are to be re-sent. As received three duplicate acknowledgments (DUPACK), the sender considers the package to be lost and sends it again. After that the sender may receive additional duplicate confirmation because the recipient provides the acknowledgement of packages that are in transmit when the sender switched to a mode of Fast Retransmit. In the case of loss of several packets from one window the sender receives new data when the confirmation of the packets resent is obtained. If one package was lost and there was no change in the order of packages, then the package confirmation will mean the successful delivery of all previous packages before switching to Fast Retransmit. However, if several packages were lost then the confirmation of the re-send packages acknowledge the delivery of some but not all packets sent before switching to Fast Retransmit mode.

3. Results of Simulation Modelling

To conduct the simulation in the network simulator Opnet, a simulative network analog consisting of several senders, recipients and the router between them was constructed **Figure 1**. The bottleneck of the network analog was the router and the output channel. In the course of numerical study the TCP operational algorithms the buffer sizes of the router and the recipient and the bandwidth at the router output have been altered. The traffic in the network under the examination presents a self-similar random process with user-specific parameters. One of the parameters is Hurst exponent, which characterizes the long-range dependence of the process and lies in the range [0.5, 1].

The characteristics for the comparative analysis of the network were: the number of lost data, the buffer capac-

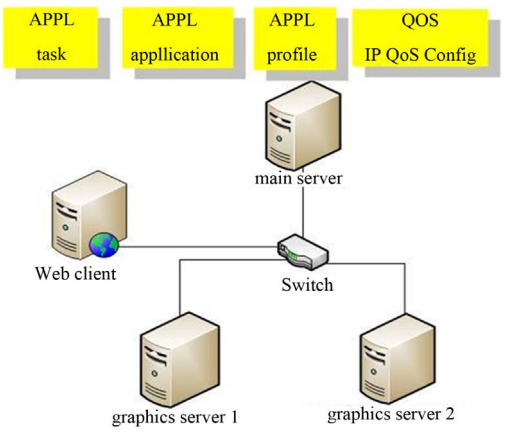

Figure 1. Simulative network.

ity of the router, the channel utilization and the network performance.

Study of CWND changes. The objects involved in the connections are found to be synchronized to some extent. This is due to the fact that at any conflict, associated with the increase in the width of the window when the buffer is full, all incoming cells belonging to packages are rejected.

On the assumption of instant readiness of the sender to transfer and that the time spread of cells does not exceed the time of packets forwarding in the input channel, all the connections will send the cells during the transmission time of packets involved into the collision. Consequently, all the connections lose packets and reduce the width of the window by half within RTT.

The simulation modeling testified that for TCP-Vegas protocol for one missing segment the CWND reduction is absent (line 1 in **Figure 2**). When there is one missing segment for TCP-Tahoe protocol (line 2) and TCP Reno (line 3), then the CWND reduction occurs well before and thus the channel resources are not used efficiently.

Figure 3 shows the number of sent and received segments and the corresponding change in the window size. The presented dependency is typical for all TCP protocols. From the graph it is seen that the number of sent segments increases linearly, in accordance with the size gain of the window. However, the amount of missed data at the window size reduction is well in excess of tolerable losses determined by Qos (Quality of Service).

In the case of NewReno algorithm the number of lost data and channel utilization decreases in comparison with Reno algorithm.

This is due to the fast retransmission and fast recovery options. However, at the option of selective acknowledgment (SACK) if operating at Reno the number of lost data is only slightly more than for NewReno. From the graph shown in **Figure 4** it is obvious that TCP-SACK protocol resizes CWND less. The loading of the buffer will be more uniform and the use of channel resources will also be superior to using Reno algorithm.

In the case of NewReno algorithm the number of lost data and channel utilization decreases in comparison with Reno algorithm. This is due to the fast retransmission and fast recovery options. However, at the option of selective acknowledgment (SACK) if operating at Reno the number of lost data is only slightly more than for Ne-

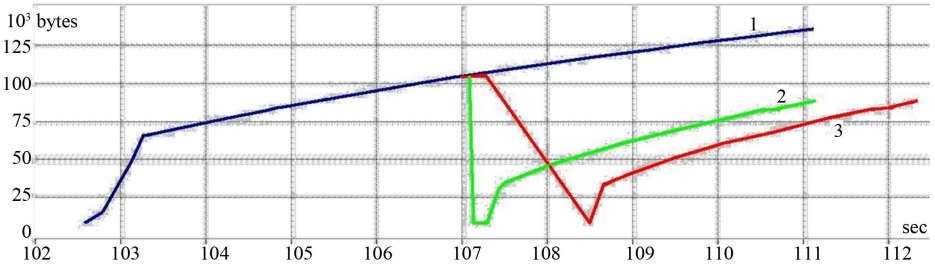

Figure 2. CWND changing: line 1—Tahoe when one segment loss; line 2—Tahoe when one segment loss; line 3—TCP Reno when one segment loss.

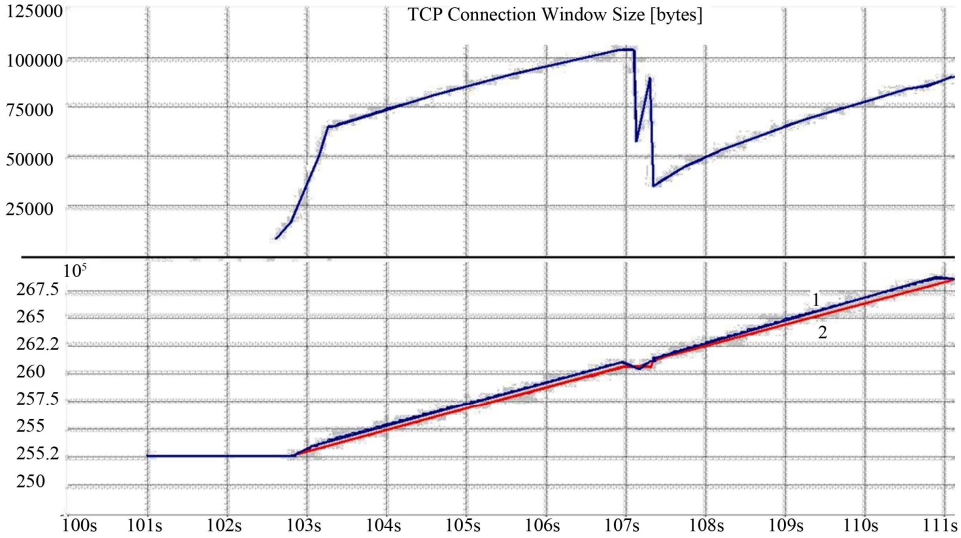

Figure 3. CWND changing (above) and the number of transmitted (line 1) and received segments (line 2) with time (below).

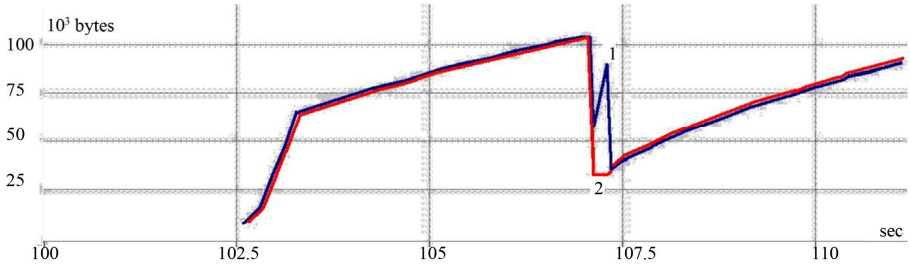

Figure 4. CWND changing for TCP Reno (line 1) and TCP SACK (line 2) when one segment loss.

wReno. From the graph shown in **Figure 4** it is obvious that TCP-SACK protocol resizes CWND less. The loading of the buffer will be more uniform and the use of channel resources will also be superior to using Reno algorithm.

In TCP-Tahoe the use of TCP group acknowledgments motivates the reduction in the window width to one after a loss to avoid the burst of packets caused by their retransmission. This in turn leads to an exponential window size gain at the slow start required for networks with a large product of the band by the delay. On the other hand, this exponential increase causes serious fluctuations in traffic, which, if the buffer size is less than one third of the product of the band by the delay, cause the buffer overflow and the second slow start decreasing the carrying capacity. TCP-Reno is trying to prevent this fact by reducing the window size twice when the loss is detected. Though under ideal conditions this does provide improved bandwidth, but in its present form TCP-Reno is too vulnerable to the interface effects and multiple packet losses to become a substitute for TCP-Tahoe. The main problem with TCP-Reno is the fact that there may be multiple constraints for the window associated with one episode of overload, and that multiple losses can result in timeout (which in practice leads to a significant decrease in carrying capacity when low-resolution timer is being used).

TCP-Vegas protocol tries to implement a number of improvements such as more sophisticated processing and evaluation of RTT. But for RTT fluctuations in Internet there are a lot of other reasons besides buffer flow. In order to improve the current version of TCP significantly it is necessary to avoid drastic cuts in the window size both in TCP-Tahoe and in TCP-Reno, except when there is a continuous overload (which causes massive packet losses). It is proposed to use TCP-Tahoe (in conjunction with network layer management to optimize the performance characteristics) in the case of isolated losses since this option is by far stabler than TCP-Reno.

Study of TCP in the case of overload. Simulation modelling of the network when using Reno algorithm with Fast recovery option showed that this algorithm is optimal only in the case of a single packet losses, *i.e.* when Reno sender transmits no more than one packet during the passage of one package through a communication channel to the destination and back. When there is a loss of a single packet, the algorithm Reno is substantially better as compared to Tahoe TCP but Reno degrades the network performance significantly if there was a loss of several packets within a single window Window. When operated at Tahoe which does not use the option of fast recovery the number of lost data is approximately the same as in NewReno algorithm but the traffic load is far less.

The results of the TCP-Reno study show that every connection usually loses about two packets in each episode of overloading. The losses occur when the buffer is full and one connection increases the window size per unit. When cells of this new package arrive into the buffer, they usually cause loss of cells belonging to two packets (the end of the package that came from another connection and the beginning of the following one). Therefore, on the average the loss of three packages is expected for one episode of overload.

The study of queueing and the use of network resources. The problem of optimal use of network resources for algorithms of fast recovery and fast retransmission in TCP-Reno algorithm associated with multiple fast retransmissions are relatively smaller than compared to the same problems that arise in the case of TCP-Tahoe algorithm, which does not use fast recovery. However, if no additional mechanisms associated with the use of Recover variable are applied then the unnecessary retransmission can occur when using TCP-Reno.

The paper represents comparative analysis of queueing in the buffer for different algorithms of TCP as the number of input data flows increases (**Table 1**). This is of particular importance, for example, for networks with multiple VPN channels and multi-port routers.

The RTT time rise is unpreventable with respect to the queue length growth. The packet losses start when achieving the lower threshold of the queue. Whenever the upper limit is exceeded, the incoming packages with lower priority are dropped out (see **Figure 5**). If only the packet dropping did not start until the buffer congestion, a linear increase of RTT would be expected. The amount

Table 1. The value of the mean length of a queue (mb) at different number of data flows.

	1	2	3	4	5	6	7
Tahoe	9900	11,700	11,900	15,000	15,300	15,600	16,000
Reno	9800	11,500	11,500	14,700	15,800	16,000	16,100
NewReno	9100	9500	9700	10,600	12,000	13,800	15,200
Sack	5300	5600	6900	7200	8600	8800	9100
Vegas	5200	5600	7000	7000	7700	8500	8900

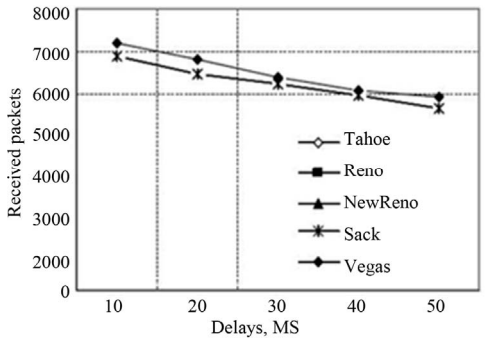

Figure 5. The number of received packets depending on packet delays.

of packet losses would increase almost spasmodically when achieving the limit value of the queue.

One of the most important indicators of the network quality is the channel utilization index. **Table 2** shows the values of this parameter while the evolution of the window size with the course of time.

In TCP-Vegas the ACK confirmation is combined with data packet (called piggybacking) instead of an independent transmission as in other algorithms. This saves fifty percent of time as contrasted with the normal performance of TCP ACK acknowledgement and does not waste extra time. Consequently, TCP-Vegas can transmit more data (**Table 3**).

The study of TCP in WAN. In order to guarantee the performance in highly loaded WAN (wide-area network), each TCP connection needs to have a reserved buffer and free transmission range along the whole network path. Usually the resource allocation is carried out in the interconnection step and makes the routers and switches form independent queues for each connection. Since the resource administration on the principle "the best possible" can be very expensive, the more acceptable alternative may be the resource reservation for every traffic class. The transfer rate varying with time available for every connection is defined on the network layer and is administered by the sender, so that different TCP connections are to be isolated from each other even if they use the same buffers together.

Table 2. Channel utilization index while the evolution of the window size with the course of time (ms).

	100	200	300	400	500	600	700
Tahoe	1	0.8	0.72	0.72	0.65	0.6	0.56
Reno	1	0.82	0.75	0.78	0.62	0.54	0.4
NewReno	1	0.87	0.79	0.85	0.74	0.68	0.6
Sack	1	0.93	0.88	0.99	0.82	0.87	0.99
Vegas	1	0.94	0.88	1	0.85	0.9	0.98

Table 3. The number of received packets depending on packet delays.

Algorithm TCP	Packet delays, ms				
	10	20	30	40	50
Tahoe	6869	6479	6215	5973	5673
Reno	6869	6479	6215	5973	5673
NewReno	6869	6479	6215	5973	5673
Sack	6869	6479	6215	5973	5673
Vegas	7200	6812	6400	6091	5934

Since the bias against connections with high RTT latency is associated with the mechanism of a window adaptation, it, theoretically, can be overcome by modifying the mechanism of bandwidth monitoring during the congestion avoidance phase, for example, by increasing the size of the window so that the rate of bandwidth growth for all connections to be one and the same. However, it is impossible to choose universal time scale for the window adjustment to be functional for networks with different bandwidths and topology. For example, sounding an additional band with rate of 1 Mb/s may be too fast for the network with a channel of 1 Mb/s but too slow for a GBIT network. In such a way to set this scheme to work it would be extremely significant to provide some exchanges between the network and the transport layer. Secondly, such a scheme would still be subjected to strong influence of certain TCP shortcomings such as the degradation of the performance in the presence of accidental losses and excessive delays associated with the attempts to use an additional band under the conditions of complete utilization of the channel. In summary, this modification cannot be considered as the representation of the best approach to the problem of optimality.

Figure 6 shows the number of sent useful data for whose computation it is necessary to subtract the number of re-sent data from the total number of sent data.

4. Future Evolution of TCP Protocol Future Evolution of TCP Protocol

Over the last years several new modifications of TCP

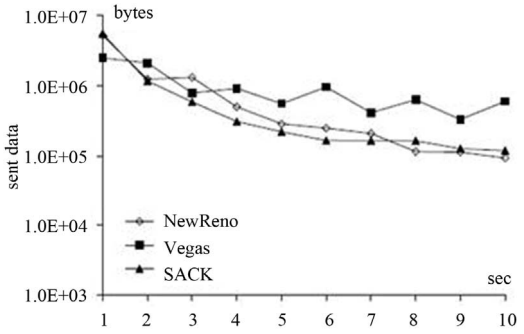

Figure 6. The number of sent useful data as a function of time.

protocol have been proposed: Binary Increase Control TCP (BIC-TCP), CUBIC TCP, Westwood TCP (TCPW), Parallel TCP Reno (P-TCP), Scalable TCP (S-TCP), Fast TCP, HighSpeed TCP (HS-TCP), HighSpeed TCP Low Priority (HSTCP-LP), Hamilton TCP (H-TCP), Yet Another Highspeed TCP (YeAH-TCP), Africa TCP, Compound TCP, etc. These protocols seek to resolve difficulties arising when working with modern fast (1.10 Gb/s) and long (RTT > 200 msek) channels. Almost all of them are based on certain older versions of TCP and differ by various means of congestion avoidance (exactly by different methods of determining the existence of packet losses that namely mean the origination of congestion). Different versions use different formulas to calculate CWND [9-14].

The description and analysis of the key features of the protocol modifications mentioned above are to follow.

At present TCP Reno with multiple parallel streams **P-TCP (Parallel TCP)** is the most commonly used for achieving high productivity. However, it can be super-aggressive and "unfair" to other versions of TCP protocol; the optimal number of parallel streams can vary significantly depending on the changes (for example, packet routes) or the use of networking opportunities.

To be effective in conditions of high-performance, the prime modern advanced protocols when using a single TCP-stream should provide characteristics similar to P-TCP (parallel Reno TCP) and, in addition, they need to have a better "justice" (in relation to other TCP versions) than P-TCP.

S-TCP (Scalable TCP) changes traditional algorithm of congestion control for TCP: exponential increase is used instead of additive increase and multiplicative decrease factor b is set to be equal to 0.125 to diminish the loss of productivity after the overload.

Fast TCP protocol is the only protocol based on Vegas TCP instead of TCP Reno. It uses queue delay and packet loss for detecting the overload [11,12]. This fact reduces large-scale packet losses due to the step-by-step algorithm in the transmitter resulting in rapid convergence to optimal parameters.

HighSpeed TCP **(HS-TCP)** is a modification of the congestion control mechanism in TCP; it improves TCP performance in high-speed networks with high latency. This modification behaves like TCP Reno for small CWND values but the more "aggressive" reaction function is used above selected CWND value. When CWND value is big (more than 38 packets, which is equal to the loss coefficient 1 out of 1000) this TCP version uses a table to indicate the extent of increasing the congestion window when ACK is received. In such a case less network bandwidth is used than 1/2 of CWND when packet loss.

The objective of **HSTCP-LP**, which is based on TCP-LP, is to use only the excess network bandwidth; however the other unused part of it can be applied by other TCP-flows. By giving a higher priority to all TCP-flows not using HSTCP-LP protocol, this version uses a simple two-class priority mechanism without any support from the network. HSTCP-LP has been implemented as the combination of HS-TCP and TCP-LP.

The disadvantage of HSTCP and STCP modification is their inappropriate band distribution for several flows with different RTT. In these circumstances the synchronization of losses for competing flows creates notable problems.

The slow response of TCP in high-speed networks of long length (fast long-distance networks) leads to the fact that there remains large unused carrying capacity. **BIC TCP** and **CUBIC TCP**—these are congestion control protocols designed to eliminate such a problem. BIC TCP is implemented and used in Linux kernel version 2.6.8 and higher. On default the model of protocol implementation was replaced by CUBIC TCP in version 2.6.19.

When a packet loss, **BIC TCP** reduces its window to a multiplicative factor. At first the reduction of the window size is set to maximum and after the reduction the window size is set to minimum. Then BIC TCP performs a binary search by using these two parameters, going to the "middle" between the maximum (Wmax) and minimum (Wmin). If there is no packet loss when updated window size, then this window size becomes the new minimum. If the packet loss occurs, then the window size becomes the new maximum. This process proceeds until the window increment is not less than a certain small constant Smin, and at this point, the window size is set to be equal to the current maximum. In the search for a new value of maximum, the window size increases at first slow to discover new maximum close by and, after some time of slow growth, if a new high is not detected (*i.e.* there are packet losses), then it is assumed that a new maximum is still further. Thus, the algorithm fails over a more rapid increase by passing to additive increase where the window size expands with a large permanent incrementation.

Satisfactory performance characteristics of BIC TCP are determined by slow increase around Wmax and linear magnification when additive increase and searching the maximum (**Figure 7(a)**).

CUBIC TCP is a protocol implementation of TCP with congestion control algorithm optimized for high performance networks with high latencies. CUBIC version is less aggressive and more systematic than BIC TCP, in which the window width value is a cubic function of time after the last event of overload, where the point of inflection is attached to the window but not to the event as it was in previous cases.

The CUBIC model uses a cubic function of window growth, which is very similar to the corresponding function BIC-TCP. In CUBIC, when implementing the window growth function, the time passed since the last event of overload is used. While most implementation models of standard TCP use convex functions of growth after a packet loss, the CUBIC applies both convex and concave sections of the cubic growth function of the window. **Figure 7** shows the evolution of the window in BIC (a) and CUBIC (b).

5. Conclusions

In summary, in the course of the experiments conducted and analysis of the results it was revealed that, when using TCP Vegas, the network quality of service is much better and the number of packet losses is much smaller than when operating at other TCP protocols. TCP Vegas protocol:

- is more stable when packet losses and enable to detect and retransmit a lost packet much faster than Tahoe;
- does not have to wait for 3 duplicate ACKs and consequently it is able to accomplish the retransmission of a lost packet faster than Reno and NewReno;

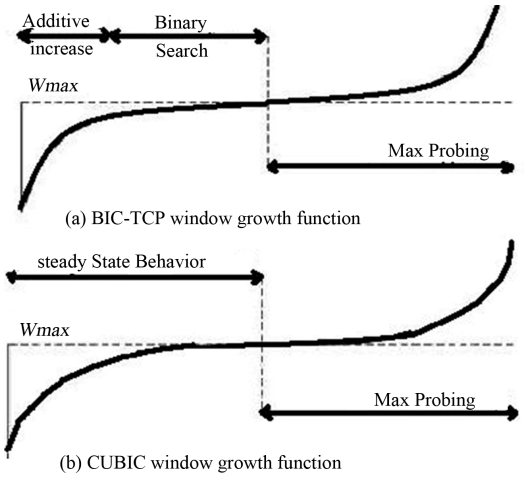

Figure 7. Window growth function for BIC-TCP and CUBIC.

- by modifying algorithms of congestion avoidance and slow start it realizes fewer retransmissions, which result in more efficient network resource utilizations in comparison with NewReno and Tahoe;
- it is enable to modify bandwidth measurements instead of packet loss according to the estimates of starting overload, which provides the best use of bandwidth and less congestions than Tahoe and SACK;
- aligns its rate of sending packets to the recipient in optimum bandwidth, causing stability as contrasted with SACK.

When considering the promising TCP modelsim two alternatives can be distinguished: BIC-TCP and CUBIC TCP. BIC-TCP model provides satisfactory scalability for high-speed networks, the equivalency for competing flows and stability with low oscillation of the window size. However, the growth function of BIC-TCP window may be too aggressive for TCP, in particular for small RTT values or for low-speed networks. Moreover, several phases of window control add unnecessary complexity to the protocol implementation and analysis of its characteristics. CUBIC TCP model lacks many of BIC TCP shortcomings but it also has some disadvantages and incompatibility with classical TCP protocols. That is why the study and improvement of TCP protocols is still an actual task.

REFERENCES

[1] W. Leland, M. Taqqu, W. Willinger and D. Wilson, "On the Self-Similarnature of Ethernet Traffic//IEEE/ACM," *Transactions of Networking*, Vol. 2, No. 1, 1994, pp. 1-15. doi:10.1109/90.282603

[2] W. Stollings, "High-Speed Networks and Internets. Performance and Quality of Service," W. Stollings, New Jersey, 2002.

[3] B. Sonkoly, B. Simon, T. A. Trinh and S. Molnar, "A Research Framework for Analyzing High Speed Transport Protocols Based on Control-theory," *Network Protocols and Algorithms*, Vol. 1, No. 2, 2009, pp. 1-26.

[4] SimonLeinen, "High-Speed TCP Variants," 2011. http://kb.pert.geant.net/twiki/bin/view/PERTKB/TcpHighSpeedVariants

[5] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, 1993, pp. 397-413.

[6] K. Fall and S. Floyd, "Simulation-Based Comparison of Tahoe, Reno, and Sack Tcp," *Computer Communication Review*, Vol. 26, No. 3, 2002, pp. 5-21. doi:10.1145/235160.235162

[7] W. Xia and W. Zhang, "End-to-End Solution of TCP Protocol in High Speed Network," 3*rd International Conference of Computer Research and Development* (*ICCRD*), 11-13 March 2011, pp. 284-288.

[8] B. Qureshi, M. Othman, S. Sabraminiam and N. A. Wati, "QTCP: An Optimized and Improved Congestion Control

Algorithm of High-Speed TCP Networks," Sprinter-Verlag, Berlin, Heidelberg, 2011, pp. 56-67.

[9] S. Ha, L. Le, I. Rhee and L. Xu, "Impact of Background Traffic on Performance of High-Speed TCP Variant Protocols," *Computer Networks*, Vol. 51, No. 7, 2007, pp. 1748-1762. doi:10.1016/j.comnet.2006.11.005

[10] H. Cai, D. Eun, S. Ha, I. Rhee and L. Xu, "Stochastic Ordering for Internet Congestion Control and Its Applications," *IEEE INFOCOM*, Anchorage, 6-12 May 2007.

[11] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha and I. Rhee, "Towards a Common TCP Evaluation Suite," PFLDnet, Manchester, 2008.

[12] I. Rhee and L. S. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variants," PFLDnet, Lyon, 2005.

[13] S. Molnár, B. Sonkoly and T. A. Trinh, "A Comprehensive TCP Fairness Analysis in High Speed Networks," *Computer Communications*, Vol. 32, No. 13-14, 2009, pp. 1460-1484.

[14] M. Welzl, M. Scharf and B. Briscoe, "RFC6077," *Open Research Issues in Internet Congestion Control*, February 2011.

