A Comparative Study of the TCP Control Algorithms

G. Kirov

Key Words: *Network simulation; TCP; congestion control.*

Abstract. *TThe publication presents a comparative study of the different congestion control mechanisms implemented by the Transmission Control Protocol (TCP): Slow-Start and Congestion Avoidance without Fast Retransmit, Tahoe, Reno, and New-Reno. The state of the art of the TCP control approaches is described. The advantages and drawbacks of the above-mentioned algorithms are investigated through the simulation investigations. The TCP performance analysis is based on two scenarios of the network simulation with different percentages of the packet loss.*

1. Introduction

Communications networks are becoming more and more complex for an analysis and congestion control. The idea to optimize the congestion problem is a challenging one. The importance of the congestion control algorithms is evidenced by the increasing attention that they have received in the recent years. Effective solution of this problem would lead to breakthroughs in areas such as wireless [3,11], satellite, and long-haul terrestrial networks.

The state of the art of the network congestion shows that it is a very difficult problem because there is no way to determine the network condition [9]. The congestion occurs when there is a lot of traffic in the networks [1,2,5]. Therefore, the congestion control can be defined as a multicriterial optimization task that has to estimate the following uncertain input parameters: number of users and applications that use the network, network capacity, congestion points, etc [6].

Only in the last few years the rapidly increasing bandwidths and delays have created a general need for increased attention to TCP flow-control mechanisms [13]. In order to optimize the network utilization the expert attention is focused on a managing of TCP flow-control window with different scaling. The main purpose of the paper is to analyse the TCP control mechanisms and to presents a comparative study of the different TCP control algorithms through a simulation analysis.

The paper is organized as follows: next section motivates the computer simulation use, section 3 describes the basic principles of the different TCP control algorithms, section 4 presents simulation research of the TCP congestion control approaches, the next section presents the comparative analysis of the simulation results, and the last section contains concluding remarks.

2. Network Simulation Approach

In the last decade the simulation became a powerful approach for analysis and design of complex network systems. According to simple definition the simulation is the imitation of the real process or system over the time. The behavior of a network system is studied by developing a simulation model. The one is a physical, mathematical, or logical representation of network processes and systems [3, 4]. A simulation model presents a set of assumptions about the network behaviour. Engineers use models to test different hypotheses by investigation of many „What-If" questions concerning the real system. Generally, there are two forms of network simulation: analytical modelling and computer simulation [5]. The first one uses mathematical analysis to predict the effect of changes to the existing systems or to investigate the performance of the new systems under different circumstances. The main disadvantage of this approach is that many real network systems are so sophisticated and they cannot be presented by a set of equations. In edition, the analytical modelling is inappropriate to simulate the dynamic nature of a network. Analytical approaches study the network performance based on over-simplified assumptions. This means the final results to be biased towards network performance under ideal conditions. The inaccuracy of analytical models focuses the expert attention on computer simulation to obtain correct results. The simulation is the better approach to investigate the network operation, Ethernet performance, and communication protocols [3,4]. In this instance, the investigation of the system behaviour over time always requires a computer simulation. With computer simulation approaches to performance evaluation, network systems can be modelled with almost any level of detail desired and the design space can be explored more finely than is possible with analytical-based approaches or measurements. Computer simulation can combine mathematical and empirical models easily, and incorporate measured characteristics of network devices and actual signals into analysis and design [2].

The network simulators are one of the most complex software tools that provide comprehensive development environment for simulation and performance analysis of communication networks. The main advantages of the simulators can be summarized in the following points [4]:

- Simulators enable the study of complex network systems.
- The effects of information and environmental changes on the model's behaviour can be analyzed.
- Simulators can be used to verify analytical models.
- Simulators are appropriate tools for test interoperability between network nodes.
- Network Simulators can be used with pedagogical purposes for training new operators, administrators, and users that can experiment in a simulated environment knowing that their mistakes couldnot cause any problems.

The main disadvantage of network simulator is that the building of simulation model requires special skills. The other problem is that a simulation modelling can be time-consuming.

The above considerations show that network simulators are appropriated tools to obtain adequate information on functionality and performance of communication networks and protocols.

3. TCP Control Algorithms

Over the past several years, TCP is the most used transport protocol all over the world. It is the basic transport protocol for Internet [10,12]. In this section are described some modifications of TCP that distinguish themselves by their congestion control algorithms [7].

3.1. Slow Start

The *TCP window* is the amount of unacknowledged data in flight between the sender and the receiver. When the TCP starts connection the sender injects the packets into network up to the maximum window size that is advertised by the receiver. This mechanism works propely if the sender and receiver are members of the same local network [8,12]. In other case, if there are slow links and routers between them, the congestion problem can arise. It can be explained with the fact that routers have queuing capability. If a router cannot transmit packets at a given destination, the packets in the queue wait for a further transmission. Taking into account that the router queue has a limited size the data transmission time may exceed and the packets will be discarded. In order to make an efficient use of network bandwidth, TCP has to control its own rate using the feedback from the network, i.e. to manage the rate of sending and receiving.

To resolve the above problem the slow start algorithm has been involved. The basic of this approach is the notion of a *congestion window* (cwnd). When the new connection is established the cwnd is initialized to one packet. Every time a packet with sequence number n arrives correctly at the receiver, the receiver *acknowledges* (ACK) the packet n by sending an ACK packet back to the sender. It contains the sequence number of next packet that it is waiting for. TCP uses an arrival of ACK as a trigger of new packet transmission, i.e. each time an ACK is received, the congestion window is increased by one packet [7]. The sender stops increasing the window size when one reaches the limit of the network capacity. The limit is defined as the minimum of the window that the sender can transmit and receiver can receive.

Fgure 1 shows that when TCP starts a connection between the sender and receiver the window size is set to one packet and the sender waits for its ACK. In the next step, after the ACK receiving, the congestion window is set from one to two, and two packets can be sent. In the case that each of those two packets is received correctly, the congestion window is increased to four [12].

3.2. Congestion Avoidance

Congestion avoidance is the algorithm that tries to solve the problem with lost packets. The congestion occurs when the

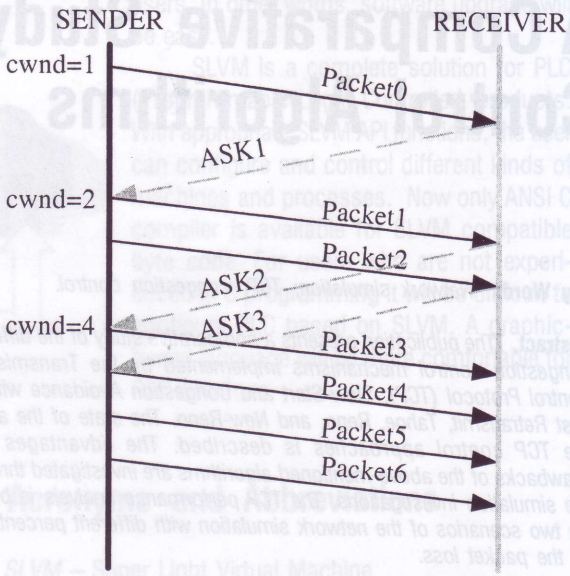


Figure 1. TCP window mechanism

rate at which packets arrive at a router is more than ones been sent [5]. In general, there are two indications of packet loss: a timeout occurring and the receipt of duplicate ACKs.

Congestion avoidance and slow start are different control algorithms that work together. The combined control mechanism introduces two parameters to adjust the amount of data being injected into the network: a congestion window (cwnd) and a *slow start threshold size* (ssthresh) [12].

The window size is defined by the following formula:

$$(1)\quad Window\ size = \min(advertized\ window, \mathrm{cwnd})$$

where *cwnd* is a window that sender can transmit, *advertised window* is flow control window, which is sent from receiver side.

When the new connection starts, TCP sets cwnd to one packet, ssthresh to arbitrary high value (65 535 bytes), and starts slow start mode. If the congestion is indicated by a timeout or the reception of duplicate ACKs, one-half of the current window size is saved in ssthresh and cwnd is set to one packet (i.e., slow start). TCP triggers Slow start at the beginning of a transfer, or after timeout and the window exponentially increases: send one packet, then two, then four, and so on every time an ACK is received. The TCP works in the slow start mode until the window size reaches ssthresh. After that TCP performs congestion avoidance. At this moment the cwnd is incremented by formula 2 each time an ACK is received, where packsize is the packet size and cwnd is maintained in bytes. In comparison with the slow start exponential growth, this is a linear growth of cwnd (*figure 2*).

$$(2)\quad \frac{(packSize * packSize)}{\mathrm{cwnd}}.$$

3.3. Fast Retransmit

The old TCP detects the network congestion and lost packets by using the *timeout* mechanism. When a packet is sent, TCP sets up its own timer to the *Retransmission Timeout Period* (RTO) for this packet. If receiver correctly receives packet,

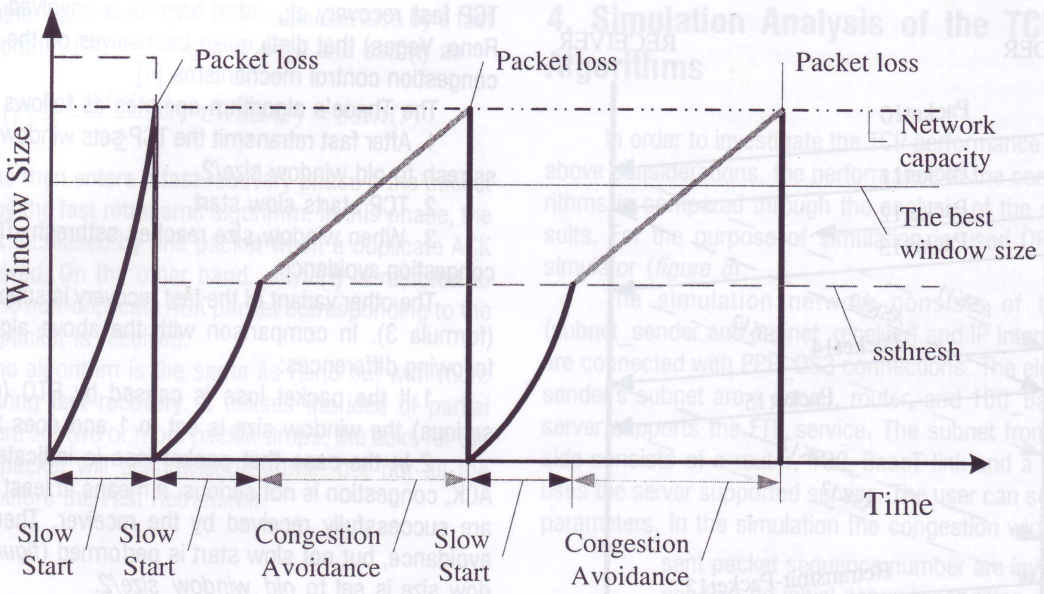


Figure 2. Slow start and congestion avoidance

TCP generates an immediate acknowledgment (ACK) corresponding to the data packet before the timer is expired. TCP assumes that the network is OK. After that TCP automatically informs the timer of the received ACK packet and continuously waits for the other ACK packets. In the case, that TCP doesnot receive required ACK within RTO period, sender will retransmit the packet whose timer is expired. Further, TCP starts slow start and sets cwnd to 1 and ssthresh to (old cwnd / 2) (*figure 3*).

Fast retransmit algorithm retransmits packets without waiting for retransmission timeout. It uses the ability of TCP to return the ACK if the packet is correctly transmit. If the received packet n is out of order the receiver acknowledges the packet n+1 by duplicate again ACK for the wrong packet n. The purpose of the duplicate ACK is to inform the sender that a packet was received is out of order. Therefore, fast retransmit uses duplicate as to trigger retransmission packets.

The duplicate ACK can by generated by *packet loss* or *packet reordering*. In the case of a reordering only one or two duplicate ACK will be generated before the reordered packet is received. Then the next ACK will be returned with the sequence number of another waited packet (*figure 4*). The TCP triggers the fast retransmit algorithm when TCP generates three or more duplicate ACK.

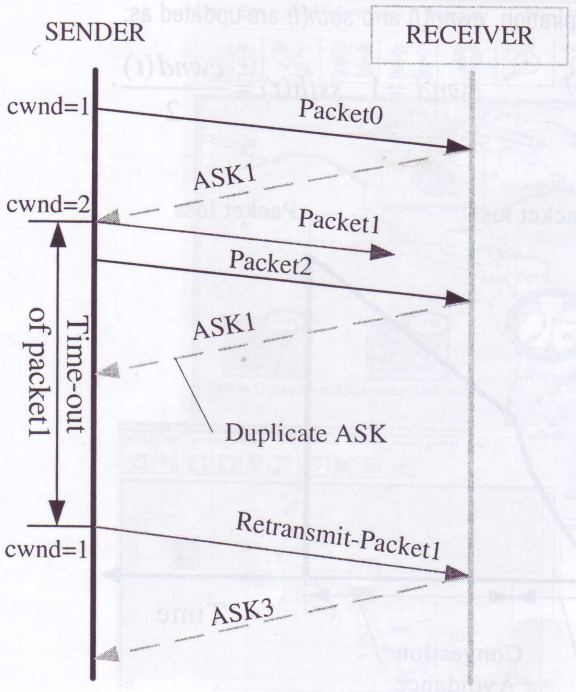


Figure 3. Timeout mechanism

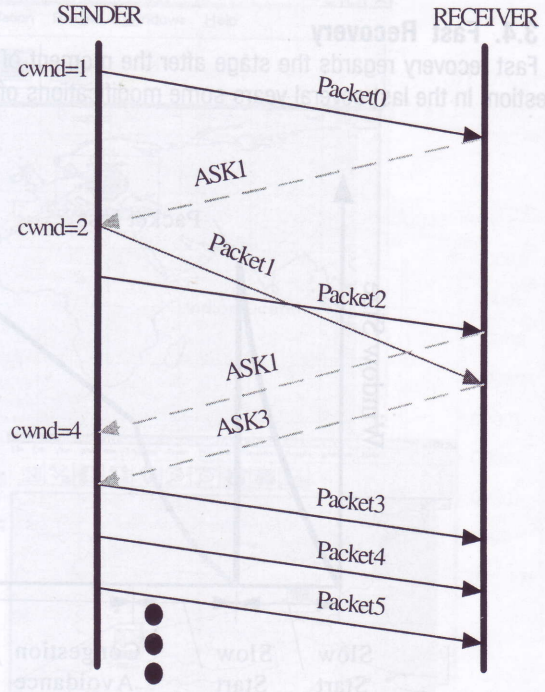


Figure 4. Packet reordering

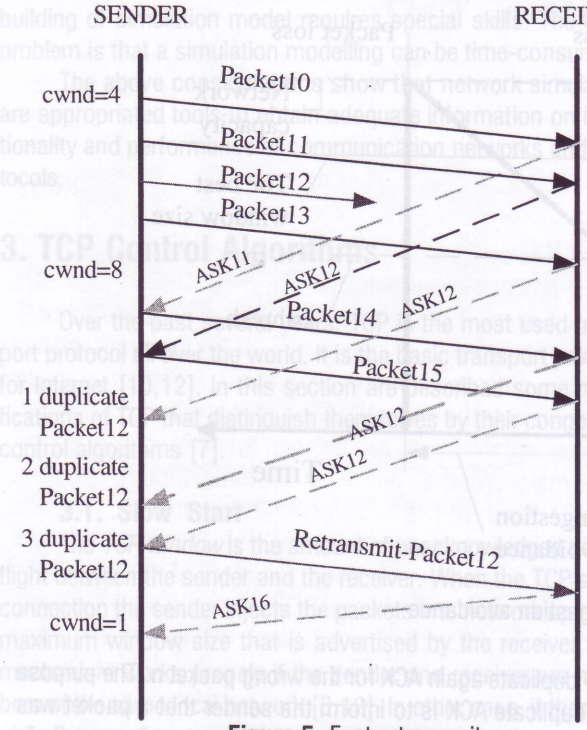


Figure 5. Fast retransmit

The *figure 5* shows how 3 duplicate ACKs lead to a fast retransmit. In this case only one packet (packet12) is dropped. After detecting a wrong one (packet12), the receiving side sends again the last ACK packet (ACK12). After receiving the third duplicate ACK12, the sender retransmits packet12. Then, according to the congestion control algorithm, TCP sets cwnd to 1, ssthresh to (old cwnd /2) = 6/2 = 3, and starts the slow start mode.

3.4. Fast Recovery

Fast recovery regards the stage after the moment of the congestion. In the last several years some modifications of the TCP fast recovery algorithm have been involved (e.g., Tahoe, Reno, Vegas) that distinguish themselves on the basis of their congestion control mechanisms [4].

The Thaoe's algorithm operates as follows (*figure 6*):

1. After fast retransmit the TCP sets window size to 0 and sstresh to old window size/2.

2. TCP starts slow start.

3. When window size reaches ssthresh, TCP triggers to congestion avoidance.

The other variant of the fast recovery is supposed by Reno (formula 3). In comparison with the above algorithm it has following differences:

1. If the packet loss is caused by RTO (congestion is serious) the window size is set to 1 and does Start slow.

2. In the case that packet loss is indicate by duplicate ACK, congestion is not serious. It means at least three packets are successfully received by the receiver. Then, congestion avoidance, but not slow start is performed (*figure 7*), i.e. window size is set to *old_window_size/2*.

TCP Reno has two phases in increasing its window size: slow start phase and congestion avoidance phase. When an ACK (acknowledgment) packet is received by TCP at the sender side at time $t + t_A$[sec], the current window size $cwnd(t + t_A)$ is updated from $cwnd(t)$ as follows:

$$(3)\quad cwnd(t+t_A)=\begin{cases} \text{Slow start:} & \\ cwnd(t)+1, & \text{if } cwnd(t)<ssth(t) \\ \text{Congestion avoidance:} & \\ cwnd(t)+\dfrac{1}{cwnd(t)}, & \text{if } cwnd(t)>ssth(t) \end{cases}$$

where $ssth(t)$ is a threshold value at which TCP changes its phase from the slow start phase to the congestion avoidance phase.

When packet loss is detected by retransmission timeout expiration, $cwnd(t)$ and $ssth(t)$ are updated as:

$$(4)\quad cwnd = 1,\ ssth(t) = \frac{cwnd(t)}{2}.$$

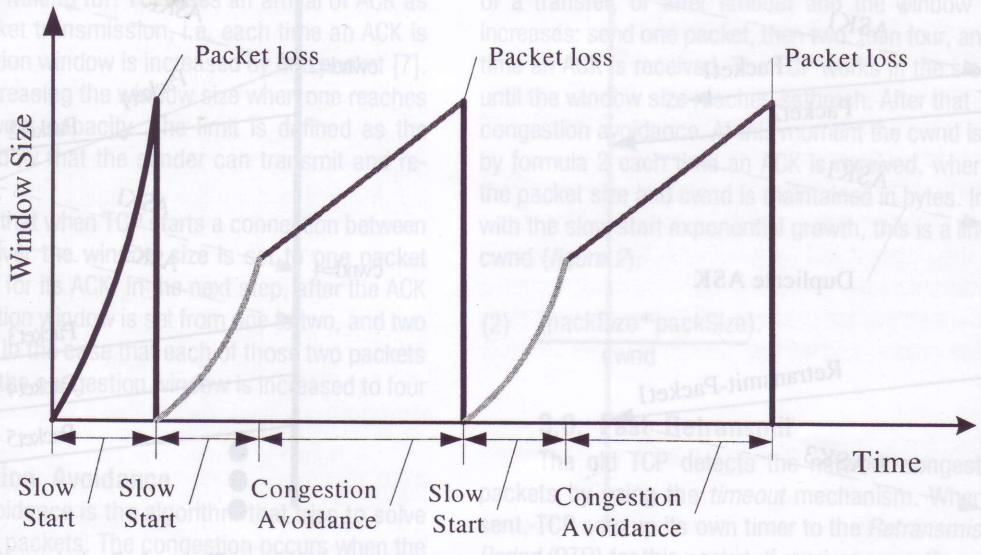


Figure 6. Tahoe's algorithm

On the other hand, when TCP detects packet loss by a fast retransmit algorithm [9], it changes *cwnd*(*t*) and *ssth*(*t*) as

$$ssth(t) = \frac{cwnd(t)}{2}, \quad cwnd(t) = ssth(t). \tag{5}$$

TCP Reno then enters a fast recovery phase if the packet loss is found by the fast retransmit algorithm. In this phase, the window size is increased by one packet when a duplicate ACK packet is received. On the other hand, *cwnd*(*t*) is restored to *ssth*(*t*) when the non-duplicate ACK packet corresponding to the retransmitted packet is received.

New Reno algorithm is the same as Reno but with more intelligence during fast recovery. It utilises the idea of partial acks: when there are two or more packet drops, the acks for the retransmitted packet will acknowledge some, but not all the packets sent before the Fast Retransmit.

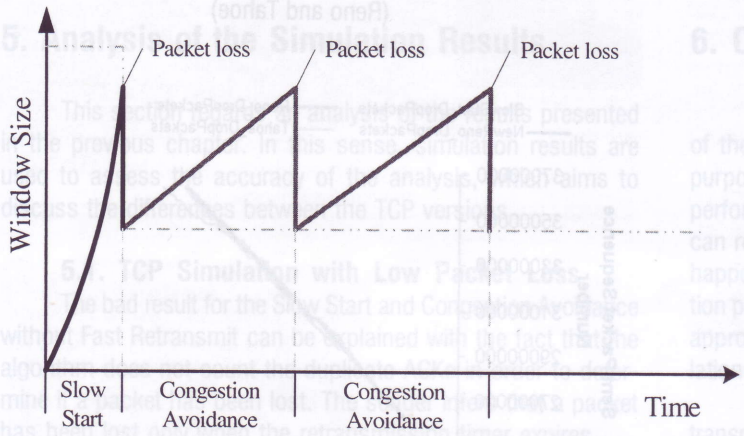


Figure 7. Reno's algorithm

4. Simulation Analysis of the TCP Control Algorithms

In order to investigate the TCP performance and verify the above considerations, the performance of the congestion algorithms is compared through the analysis of the simulation results. For the purpose of simulation is used OPNET network simulator (*figure 8*).

The simulation network consists of two subnets (subnet_sender and subnet_receiver) and IP Internet cloud that are connected with PPP_DS3 connections. The elements of the sender's subnet are a server, router, and 100_BaseT link. The server supports the FTP service. The subnet from the receiver side consists of a router, 100_BaseT link and a ftp client that uses the server supported service. The user can set a lot of TCP parameters. In the simulation the congestion window size and sent packet sequence number are investigated. Depending on initial network condition different simulation scenarios for the TCP control algorithms are performed:

- Slow-Start and Congestion Avoidance without Fast Retransmit.
- Tahoe: includes Fast Retransmit and Fast Recovery.
- Reno: adds modification to Fast Recovery.
- New-Reno: enhanced Reno TCP using a modified version of Fast Recovery.

In the first scenario the congestion window size and sent packet sequence number of the TCP control algorithms are studied when the packet loss is low (0.5%). *Figure 9*, *figure 10*, *figure 11* and *figure 12* show the simulation results.

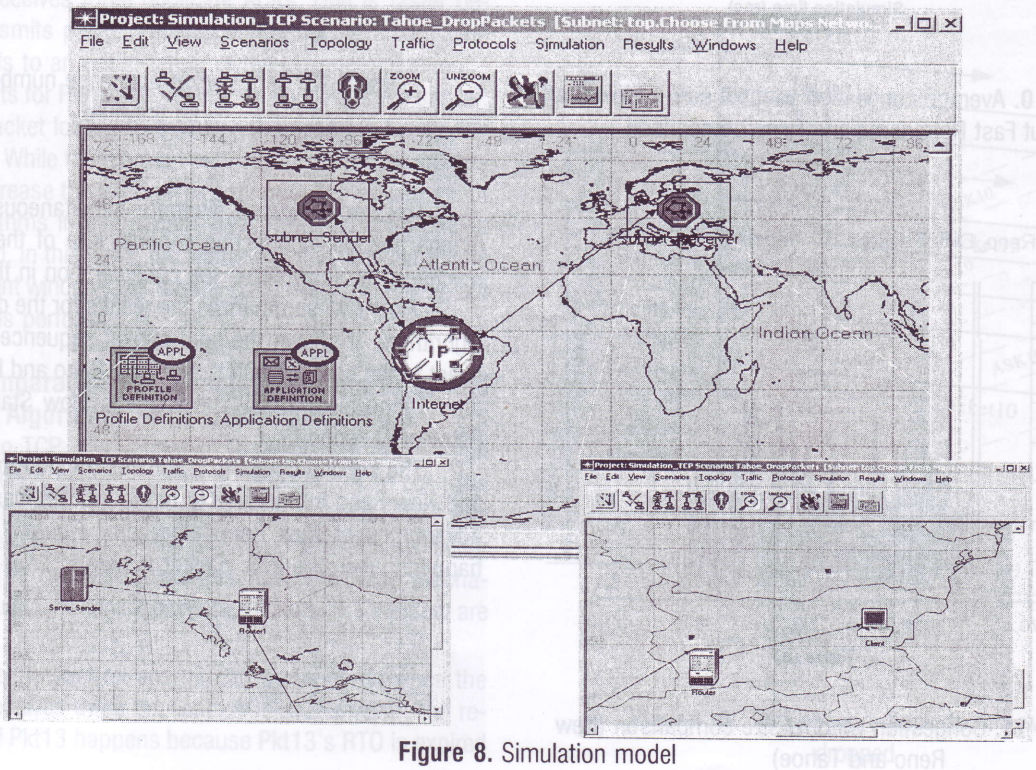


Figure 8. Simulation model

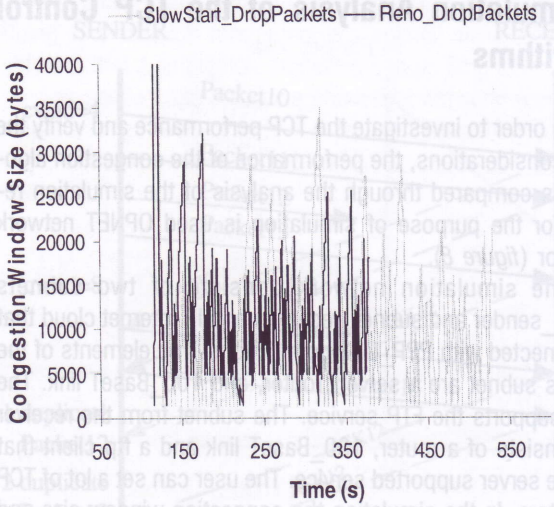


Figure 9. Slow Start without Fast Retransmit and Reno comparison

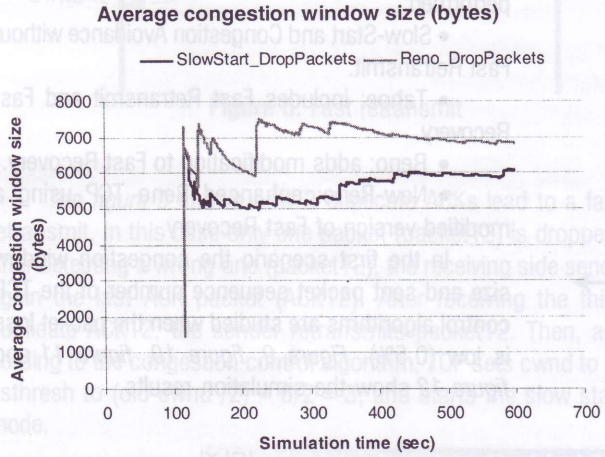


Figure 10. Average congestion window size – Slow Start without Fast Retransmit and Reno comparison

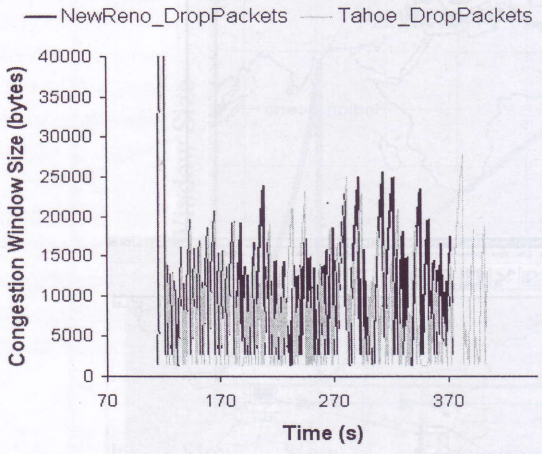


Figure 11. Congestion window size comparison (New Reno and Tahoe)

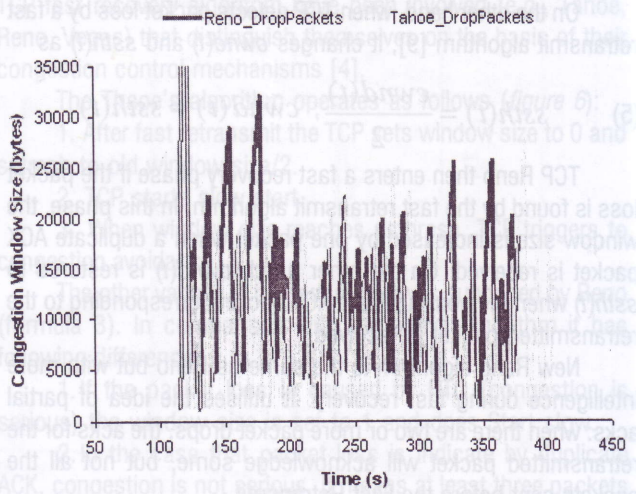


Figure 12. Congestion window size comparison (Reno and Tahoe)

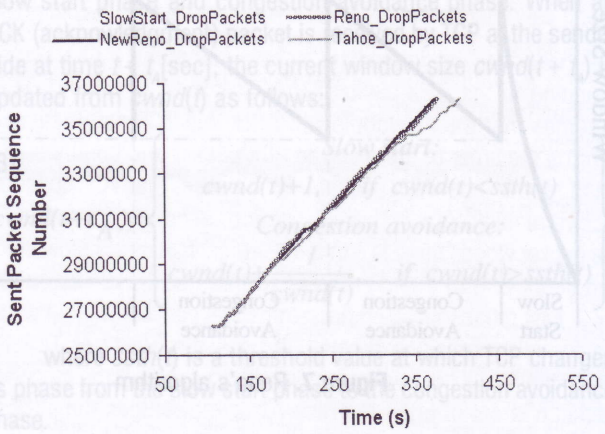


Figure 13. Sent packet sequence number in respect to the time

The above figures illustrate simultaneously the performance and the congestion window size of the different TCP mechanisms. It confirms the consideration in the chapter 2.

Figure 13 summarizes the results for the different control algorithms. It shows the sent packet sequence number in respect to the time. It depicts that New Reno and Reno are better than Tahoe algorithms. The worst is Slow Start without Fast Retransmit support.

The second scenario compares the performance of Reno and New Reno in the case of bigger packet loss. It shows the advantages of New Reno algorithm. *Figure 14* shows what happens when the loss packet is bigger.

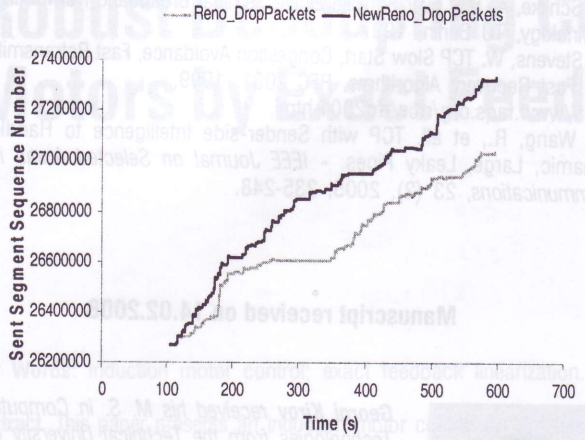


Figure 14. Comparison between New Reno and Reno

5. Analysis of the Simulation Results

This section regards an analysis of the results presented in the previous chapter. In this sense, simulation results are used to assess the accuracy of the analysis, which aims to discuss the differences between the TCP versions.

5.1. TCP Simulation with Low Packet Loss

The bad result for the Slow Start and Congestion Avoidance without Fast Retransmit can be explained with the fact that the algorithm does not count the duplicate ACKs in order to determine if a packet has been lost. The sender infers that a packet has been lost only when the retransmission timer expires.

The Tahoe mechanism shows better performance. It implements Slow Start, Congestion Avoidance and Fast Retransmit when sender receives three duplicate ACKs. One is faster because it retransmits packet without waiting for retransmission timeout. It leads to an optimization of the channel utilization.

The results for Reno and New Reno algorithms are similar. They induce packet losses to estimate the available bandwidth in the network. While there are no packet losses, the algorithms continue to increase the window size by one. The advantage of the both algorithms in comparison with Tahoe is when packet loss is detected. In this case, the window size is reduced to one half of the current window size and the congestion avoidance, but not slow start is performed.

5.2. Comparative TCP Simulation of the Reno and New Reno's Algorithms with High Packet Loss

New Reno TCP is a variant of Reno with a modification within Fast Recovery algorithm. This is done in order to solve the timeout problem when multiple packets are lost from the same window. *Figure 15* explains why the new Reno shows better performance. The diagram below describes the Reno mechanism in the situation where 2 or more packets in a window are dropped.

For Packet10, fast retransmit can be triggered since the sending side receives three duplicate ACK10. However, the retransmission of Pkt13 happens because Pkt13's RTO is expired – not fast retransmit. It is caused because there are no enough packets in transit for triggering the 2nd fast retransmit.

When 2 or more packets in a window are dropped the New Reno is more effective than Reno because of Fast Retransmit algorithm does not stop, and it is applied for packet 13. In this case, the TCP sender has the information to make intelligent decisions about which packets to retransmit and which packets not to retransmit during Fast Recovery.

In TCP Reno, the first partial ACK will trigger the sender out of the fast recovery phase. This will result in the requirement of timeouts when there are multiple losses in a window. In New Reno, a partial ACK indicates that another packet has been lost and the sender retransmits the first lost packet. Partial ACKs do not take New Reno out of Fast Recovery. It avoids requiring multiple fast retransmits from a single window because it retransmits one packet per RTT until all the lost packets are retransmitted.

6. Conclusions

The paper presents the basic techniques for TCP control of the optimum window size for a given connection. For the purpose of investigation a computer simulation approach for performance analysis of the algorithms is used. The simulation can reply of the questions that often go unanswered – what is happening network throughout taking into account the application performance, bandwidth utilization, network congestion and appropriate prioritization of user and application traffic. Simulations of the following TCP control mechanisms are performed:

- Slow-Start and Congestion Avoidance without Fast Retransmit.
- Tahoe: includes Fast Retransmit and Fast Recovery.

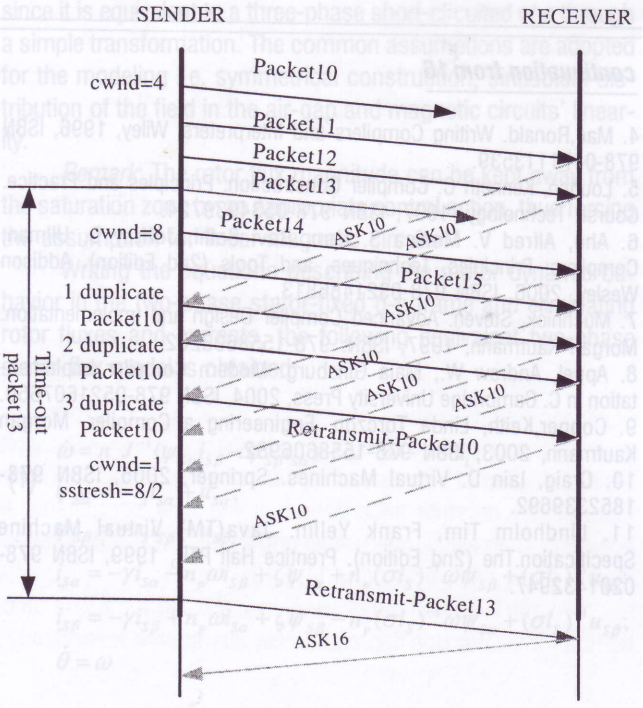


Figure 15. Two or more packets in a window are dropped

- Reno: adds modification to Fast Recovery.
- New-Reno: enhanced Reno TCP using a modified version of Fast Recovery.

The comparative study of the above-mentioned algorithms is done. The advantages of the Reno and New Reno algorithms are proved. The simulation results show that when the probability for packet loss is high the New Reno one is the most effective because it has very low probability of retransmission timeouts.

The simulation results will help experts make well-informed decisions on how to manage an Ethernet network and fine-tune the network parameters.

References

1. Douglas, C. Internetworking with TCP/IP (2). Prentice-Hall, Inc., 1991.
2. Douglas, C. Internetworking with TCP/IP (3). Prentice-Hall, Inc., 1991.
3. Ewerlid, A. Reliable Communication over Wireless Links, in Nordic Radio Symp. (NRS). Sweden, Apr. 2001.
4. Fall, K., S. Floyd.Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. - *Computer Communication Review*, 26 (3), July 1996, 5-21.
5. Firoiu, V., M. Borden. A Study of Active Queue Management for Congestion Control.Proc. IEEE INFOCOM, March 2000.
6. Jacobson, V. Congestion Avoidance and Control. - *Computer Communication Review*, 18 (4), August 1988, 314-329.
7. Jacobson, V. Congestion Avoidance and Control, in Proceedings of SIGCOMM '88 Workshop. ACM SIGCOMM, ACM Press, Stanford, CA, 1988, 314-329.
8. Mo, J. and J. Walrand. Fair End-to-end Window-based Congestion Control. - *IEEE/ACM Trans. Networking*, 8, 5 (Oct. 2000), 556-567.
9. Morris, R. Scalable TCP Congestion Control. IEEE INFOCOM 2000, Tel Aviv.
10. Padhye, J., S. Floyd. On Inferring TCP Behavior. - *Computer Communications Review ACM-SIGCOMM*, 31, August 2001.
11. Schilke, A. TCP over Satellite Links. Seminar Broadband Networking Technology, TU Berlin, 1997.
12. Stevens, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, 1999, http://www.faqs.org/rfcs/rfc2001.html.
13. Wang, R., et all. TCP with Sender-side Intelligence to Handle Dynamic, Large, Leaky Pipes. - *IEEE Journal on Selected Areas in Communications*, 23 (2), 2005, 235-248.

Manuscript received on 14.02.2008

Georgi Kirov *received his M. S. in Computer Technologies from the Technical University of Sofia, Bulgaria. He obtained his PhD degree in the field of the Intelligent Technologies and Computer Networks from the Institute of Computer and Communication Systems at the Bulgarian Academy of Sciences. Dr. Kirov is currently a research fellow at the Department of Knowledge Based Control Systems, Institute of Control and System Researches of the Bulgarian Academy of Sciences (BAS) with publications in the fields of intelligent technologies in computer simulation, system researches and communication networks.*

Contacts:
Institute of Control and System Research,
Bulgarian Academy of Sciences,
Acad. G. Bonchev str., bl. 2, P.O.Box 79,
1113 Sofia
e-mail: kirov@icsr.bas.bg

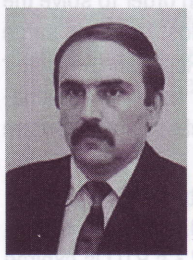
continuation from 16

4. Mac,Ronald. Writing Compilers and Interpreters. Wiley, 1996, ISBN 978-0471113539.
5. Louden, Kenneth C. Compiler Construction: Principles and Practice. Course Technology, 1997, ISBN 978-0534939724.
6. Aho, Alfred V. Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley, 2006, ISBN 978-0321486813.
7. Muchnick, Steven. Advanced Compiler Design and Implementation. Morgan Kaufmann, 1997, ISBN: 978-1558603202.
8. Appel, Andrew W., Maia Ginsburg. Modern Compiler Implementation in C. Cambridge University Press, 2004, ISBN 978-0521607650.
9. Cooper,Keith, Linda Torczon. Engineering a Compiler. Morgan Kaufmann, 2003, ISBN 978-1558606982.
10. Craig, Iain D. Virtual Machines. Springer, 2005, ISBN 978-1852339692.
11. Lindholm Tim, Frank Yellin. Java(TM) Virtual Machine Specification.The (2nd Edition). Prentice Hall PTR, 1999, ISBN 978-0201432947.

Manuscript received on 13.12.2007

Svetozar Petrov Rusinov *born in 1980. Obtained M. Sc. Degree in Computer Systems and Technologies in 2007 at Technical University of Gabrovo. He worked at Senior R&D Engineer in Johnson Controls Inc. His main research interests comprise the area of software architecture, software engineering, software reliability and embedded and real time software systems.*

Contacts:
e-mail: svetozar_rusinov@mail.bg



Raycho Todorov Ilarionov *(born 1957) is Assoc. Prof. in department Computer systems and technologies of Technical University of Gabrovo. His research interests include Computer Peripherals, Multimedia Systems, Digital Circuit Devices & Microprocessors Devices.*

Contacts:
e-mail: ilar@tugab.bg