УДК 681.3

RESEARCH OF COMPUTER NETWORK PROTOCOLS USING NS-2 SOFTWARE SIMULATIONS

Saif Talib Albasrawi, Yuri Ladyzhensky
Donetsk National Technical University, Donetsk, Ukraine
Department of Computer Engineering

Abstract

Saif Talib , Yuri Ladyzhensky . Research of Computer Network Protocols Using NS-2 Software Simulations *. The NS-2 simulator covers a very large number of applications, protocols, network types, network elements and traffic models. The goal of our papers to learn how to use NS simulator and to become acquainted with and understand the operations of some of the simulated objects using NS simulation.We focus on the analysis of the behavior of the simulated objects using NS simulation.*

Keywords*: Simulation, NS-2, Network, protocol*

Introduction. NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more... NS also implements multicasting and some of the MAC layer protocols for LAN simulations.

NS simulator is based on an object oriented simulator, written in C++, and a OTcl (an object oriented extension of Tcl) interpreter, used to execute user's command scripts[6-9]. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object.

Tracing Objects. NS simulation can produce both the visualization trace (for NAM) as well as an ASCII file trace corresponding to the events registered at the network. When we use tracing NS inserts four objects in the link: EnqT, DeqT, RecvT and DrpT, as indicated in figure 1.

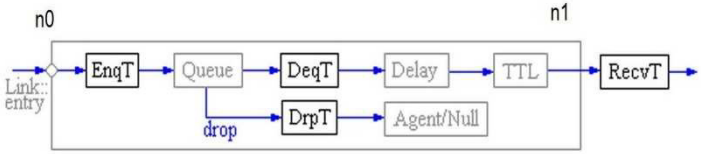

Figure 1: Tracing objects in a simplex link

EnqT registers information concerning a packet that arrives and is queued at the input queue of the link. If the packet overflows then information concerning the dropped packet are handled by DrpT. DeqT registers information at the instant the packet is dequed. Finally, RecvT gives us information about packets that have been received at the output of the link. NS allows us to get more information than through the above tracing. One way is by using queue monitoring.

Structure of trace files. When tracing into an output ASCII file, the trace is organized in 12 fields as follows in figure 2.

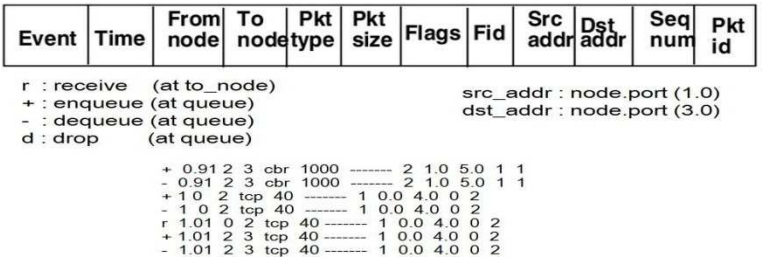

Figure 2: Fields appearing in a trace file

Packet. A NS packet is composed of a stack of headers, and an optional data space (see figure 3).A packet header format is initialized when a Simulator object is created, where a stack of all registered (or possibly useable) headers, such as the common header that is commonly used by any objects as needed, IP header, TCP header, RTP header (UDP uses RTP header) and trace header, is defined, and the offset of each header in the stack is recorded.

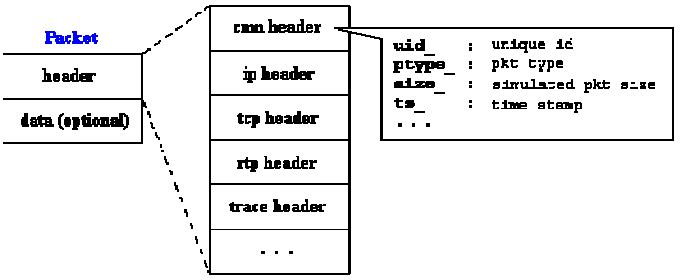

Figure 3. NS Packet Format

Description and simulation of TCP/IP. TCP (Transport Control Protocol) is the transport protocol that is responsible for the transmission of around 90% of the

Internet traffic.Understanding TCP is thus crucial for dimensioning the Internet. Although TCP is already largely deployed, it continues to envolve.

Objectives of TCP and window flow control.

TCP has several objectives:

- Adapt the transmission rate of packets to the available bandwidth,
- Avoid congestion at the network,
- Create a reliable connection by retransmitting lost packets.

In order to control the transmission rate, the number of packets that have not yet been received (or more precisely, for which the source has not obtained the information of good reception) is bounded by a parameter called a congestion window. We denote it by **W**(**cwnd** in the TCP code),this means that the source is obliged to wait and stop transmissions the number of packets that it had transmitted and that have not been"acknowledged" reaches **W**. In order to acknowledge packets and thus to be able to retransmit lost packets each transmitted packet has a sequence number.

Acknowledgements .The objectives of Acknowledgements (ACKs) are:

- Regulate the transmission rate of TCP, ensuring that packets can be transmitted only when other have left the network.
- Render the connection reliable by transmitting to the source information it needs so as to retransmit packets that have not reached the destination.

The ACK tells the source what is the sequence number of the packet it expects.A TCP packet is considered lost if three repeated ACKs for the same packets arrive at the source1, or when a packet is transmitted, there is a timer that starts counting. If its ACK does not arrive within a period $\mathbf{T_0}$, there is a "Time-Out" and the packet is considered to be lost. Retransmitting after three duplicated ACKS is called "fast retransmit". How to choose T_0? The source has an estimation of the average round trip time **RTT**, which is the time necessary for a packet to reach the destination plus the time for its ACK to reach the source. It also has an estimation of the variability of RTT. T0 is determined as follows:

$$\boldsymbol{T = \overline{RTT} + 4D}$$

Where $\boldsymbol{\overline{RTT}}$ is the current estimation of RTT, and **D** is the estimation of the variability of RTT. In order to estimate RTT, we measure the difierence **M** between the transmission time of a packet and the time its ACK returns. Then we compute

$$\boldsymbol{\overline{RTT} \leftarrow a * \overline{RTT} + (1 - a)M} \ ,$$

$$\boldsymbol{D \leftarrow aD + (1 - a)|\overline{RTT} - M|}$$

In order to decrease the number of ACKs in the system, TCP frequently uses the "delayed ACK" option where an ACK is transmitted for only every d packets that reach the destination. The standard value of d is 2. However, delaying an ACK till d > 1 packets are received could result in a deadlock in case that the window size

is one.Therefore, if the first packet (of an expected group of d packets) arrives at the destination, then after some time interval (typically 100ms) if d packets have not yet arrived, then an acknowledgement is generated without further waiting.

Dynamic Congestion Window. Since the beginnings of the eighties, during several years, TCP had a fixed congestion window. Networks at that time were unstable, there were many losses, large and severe congestion periods, during which the throughputs decreased substantially; there were many packet retransmissions and large delays. In order to solve this problem, it was proposed [1] to use a dynamic congestion window: its size can vary according to the network state. When the window is small, it can grow rapidly, and when it reaches large values it can only grow slowly. When congestion is detected, the window size decreases drastically. This dynamic mechanism allows resolving congestion rapidly and yet using efficiently the network's bandwidth.

Losses and a dynamic threshold W_{th}. Not only W is dynamic, W_{th} is too. It is fixed in TCP to half the value of W when there has been a packet loss. There are several variants of TCP. In the first variant, called "**Tahoe**", whenever a loss is detected then the window reduces to the value of 1 and a slow-start phase begins. This is a drastic decrease of the window size and thus of the transmission rate. In the other mostly used variants, called **Reno** or **New-Reno**, the window drops to 1 only if the loss is detected through a time-out. When a loss is detected through repeated ACKs then the congestion window drops by half. Slow start is not initiated and we remain in the"congestion avoidance" phase.

Initiating a Connection. To initiate a TCP connection, the source sends a " sync" packet to the destination. The destination then sends an ACK called" sync ACK". When receiving this ACK, TCP can start sending data.If either of these packets is lost then after a time-out expires (usually 3 or 6 seconds) then it is retransmitted. When a retransmitted packet is lost, the time-out duration doubles and the packet is sent again.

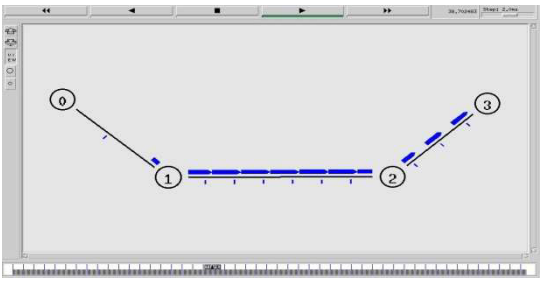

Figure 4:NAM graphic interface

Comparison of the throughput of TCP versions.We have done simulation of TCP versions (Tahoe ,Reno ,New-Reno and Vegas) respectively using TCL scripts for them to extract the throughput of each one of them, this network consist of 4 nodes(n0,n1,n2,n3) as shown in above figure 4,which is drawn by NAM software. The duplex link between n0 and n1 has 2Mbps of bandwidth and 10 millisecond of delay ,the simplex link between n1 and n2 forward and backward has 0.3Mbps of bandwidth and 100 millisecond of delay, The duplex between n2 and n3 has 0.5Mbps and 40 millisecond of delay. Each node uses a Drop Tail queue of which the maximum size is 20, the whole simulation duration is 100 second and "TCP" is set to start at 1.0 second and stop at 99.0 second. The 'TCP" agent is attached to n0 and a connection is established to TCP "Sink" agent attached to n3 , The maximum size of packet that a "TCP" agent can generate is 552 Byte, a TCP "Sink" agent generates and send ACK packets to the sender "TCP agent" and frees the received packets . We obtain an output file with the averaged received throughput of TCP (in bytes per second) as a function of time in node n3 "Sink" agent. where in our example each 1 second a new value of the throughput is obtained. In order to understand better the behavior of the system, we also plot the window size of each one of them by using Gnuplot software. Finally, we found that Vegas is more throughput from others as illustrated in table 1.

Type of TCP version	Number of Packets(Throughput)
Tahoe	5417
Reno	5941
New-Reno	5962
Vegas	6519

Table 1: The throughput of TCP versions

Tahoe. The problem with it is that it takes a complete timeout interval to detect a packet loss, and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn't send immediate ACK's, it sends cumulative acknowledgements, thus every time a packet is lost it waits for a Timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links as shown in figure 5. We see from figure 6 that from time 8.19 onwards a steady-state cyclic regime of TCP is attained,TCP is always in congestion avoidance, and its window size increase (almost linearly) until congestion occurs, Before time 8.19 we see a transient behavior in which TCP is in the slow-start phase, At time 3.7 there are losses at the slow-start phase, A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window(80) as a threshold value(40),and it then set congestion window(CWD) to 1,and then starts slow-start phase until it reaches the threshold value. and at time 7.5 it found packet loss ,so again it save the half of the current window(40) as a threshold value(20),and it then

set congestion window to 1, then start slow-start phase until it reaches the threshold value, After that it increments linearly until it encounters a packet loss and so on.

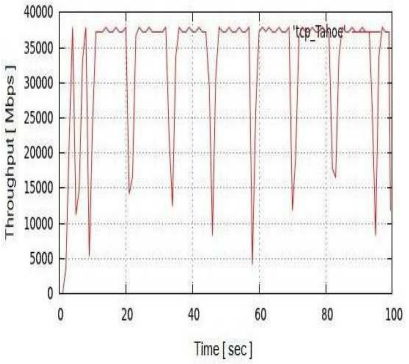

Figure 5:Throughput of TCP Tahoe

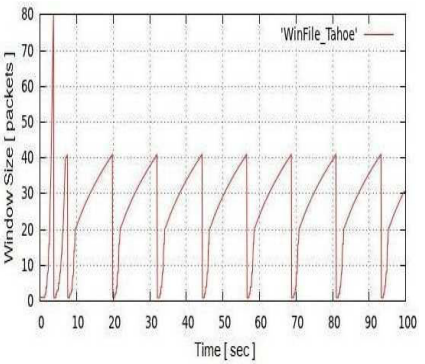

Figure 6:Window size of TCP Tahoe

Reno. It Performs very well over TCP when the packet losses are small as depicted in figure 7. But when we have multiple packet losses in one window then RENO doesn't perform too well and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. If there is multiple packet drops then the first info about the packet loss comes when we receive the duplicate ACK's. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one RTT.

We see from figure 8 that from time 7.3 onwards a steady-state cyclic regime of TCP is attained,TCP is always in congestion avoidance, and its window size increase (almost linearly),Before time 7.3 we see a transient behavior in which TCP is in the slow-start phase, At time 3.7 there are losses at the slow start phase, Reno save the half of current window(80)as a threshold value(40) and also set CWD to the same value and then slow-start phase until reach to the lost packet and so on.

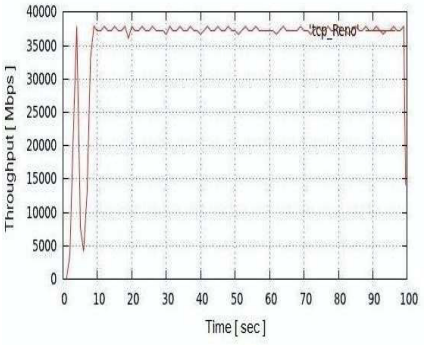

Figure 7:Throughput of TCP Reno

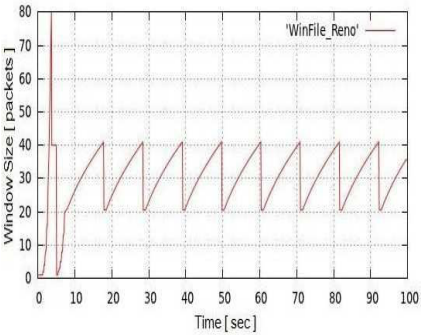

Figure 8:Window size of TCP Reno

New-RENO. It is a slight modification over TCP-RENO. It is able to detect multiple packet losses and thus is much more efficient that RENO in the event of multiple packet losses. Like Reno, New-Reno also enters into fast-retransmit when it receives multiple duplicate packets, however it differs from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast-recovery is acknowledged, but New-Reno suffers from the fact that its take one RTT to detect each packet loss as shown in figure 9. When the ACK for the first retransmitted segment is received only then can we deduce which other segment was lost .We see from figure 10 that New-Reno like Reno, but it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast-recovery is acknowledged.

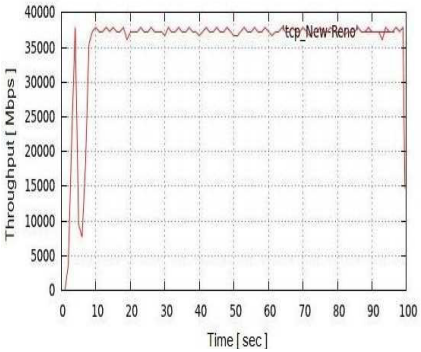

Figure 9:Throughput of TCP

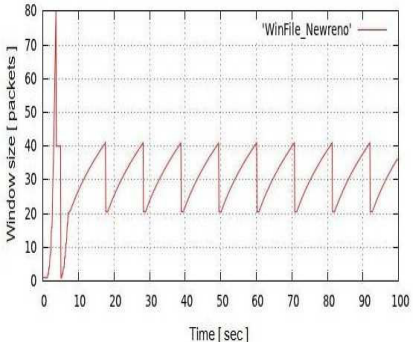

Figure 10:Window size of TCP New Reno

Vegas. It is better because it is much more robust in the face of lost packets. It can detect and retransmit lost packet much sooner than timeouts in ,and it doesn't have to always wait for 3 duplicate packets so it can retransmit sooner, and its congestion avoidance mechanisms to detect 'incipient' congestion are very efficient and utilize network resources much more efficiently as shown in figure 11.

As illustrated in figure 12, Vegas builds on the fact that proactive measure to encounter congestion are much more efficient than reactive ones.TCP Vegas is different from them in its behavior during congestion avoidance. It does not use the loss of segment to signal that there is congestion. It determines congestion by a decrease in sending rate as compared to the expected rate.

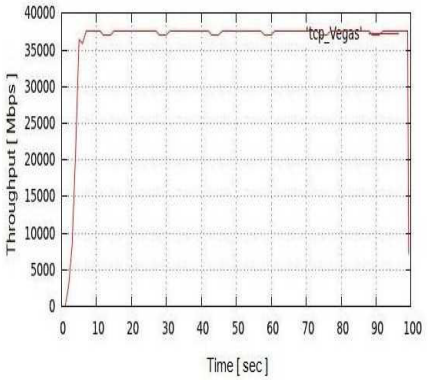

Figure 11:Throughput of TCP Vegas

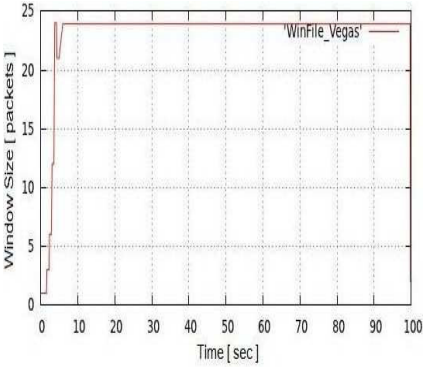

Figure 12:Window size of TCP Vegas

network components, etc., mainly by giving simple examples and explanations based on our experiences, this research proves that TCP(Vegas) has better throughput with Tahoe, Reno and New-Reno which is useful for the research of network protocols and integrating their advantages .

References

1. V.Jacobson. “Congestion Avoidance and Control”.SIGCOMM Symposium no Communication Architecture and protocols.
2. V.Jacobson “Modified TCp Congestion Control and Avoidance Algorithms ” .Technical.
3. S.Floyd, T.Henderson “The New-Reno Modification to TCP’s Fast Recovery Algorithm” RFC 2582.
4. L. S. Brakmo, L.L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas in Communication.
5. NS by Example , Jae Chung and Mark Claypool ,http://nile.wpi.edu/NS/
6. E.Altman,”A stateless approach for improving TCP performance using Diffserv” .
7. E.Altman and T.Jimenez,” Improving TCP over multihop networks using delayed ACK” .
8. W. Noureddine and F. Tobagi, “improving the Performance of Interactive TCP Applications using Service Differentiation” , Proceedings of IEEE Infocom, New-York ,USA ,June 2001
9. P. Pieda, J. Ethridge, M.Baines and F. Shallwani, “A network simulator differentiated services implementation “ , Open IP, Nortel Networks.