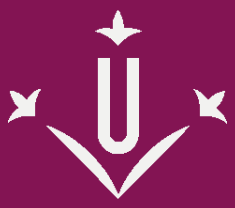

UNIVERSITAT DE LLEIDA

Escola Politècnica Superior

EXERCISE 2

ICT PROJECT: COMMUNICATION SERVICES AND SECURITY

Rafael Câmara Pereira
David Sarrat González
Daniel Vieira Cordeiro

TABLE OF CONTENTS

1. Introduction

In this assignment, it was made an analysis of a simulation of concurrency between three different implementations of TCP (Transmission Control Protocol) agents TCP Vegas, TCP Tahoe, and TCP Reno using the Network Simulator - ns-2, a discrete event simulator targeted at networking tool. By doing that, it was possible to observe the behavior of the algorithms on simulated bottleneck situations and analyze various metrics, like performance and amount of sent and lost packages, as well as characteristics of management of congestion control on the different TCP agents.

1.1. TCP Tahoe

TCP Tahoe is an implementation of congestion control that introduced the idea that duplicate ACKs likely mean a lost packet [1]. The ideas behind TCP Tahoe came from a 1988 paper by Jacobson and Karels, which introduced the following Congestion Window (cwnd) adjustment rules:

- If three duplicate ACKs are received
    - a fast retransmit is performed
    - the agent sets the slow start threshold to half of the current congestion window
    - reduces the congestion window to 1 MSS (cwnd = 1MSS)
    - and resets to slow start state [2].

1.2. TCP Reno

Created in 1990, TCP Reno has the same algorithms implemented as the TCP Tahoe, with the addition of the fast recovery algorithm, meaning that if three duplicate ACKs are received, a fast retransmit is performed and it does not enter the slow start phase [2]. Instead, it enters a phase called Fast Recovery in which:

- The congestion window is cut in half (cwnd = cwnd/2)
- The slow start threshold is set to the value of the new congestion window.

Both TCP Reno and TCP Tahoe work with queues to control the maximum number of segments in the attempt of having a better margin on bandwidth [1].

1.3. TCP Vegas

Created by two University of Arizona researchers, TCP Vegas controls its window size by observing RTTs (round-trip times) of packets that the sender host has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles the window size. If RTTs become small, on the other hand, the sender host of TCP Vegas determines that the network is relieved from the congestion, and increases the window size again. Hence, the window size in an ideal situation is expected to be converged to an appropriate value. Another feature of TCP Vegas in its congestion control algorithm is a slow slow-start mechanism. The rate of increasing its window size in the slow-start phase is one half of that in TCP Tahoe and TCP Reno. Namely, the window size is incremented every other time an ACK packet is received [4].

2. Simulation Environment

In order to create and run the simulation, a script in the TCL language was created ("labDeliver.tcl"), following the network architecture as described in figure 1. It was executed using the Network Simulator - ns-2, a tool that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks [4].

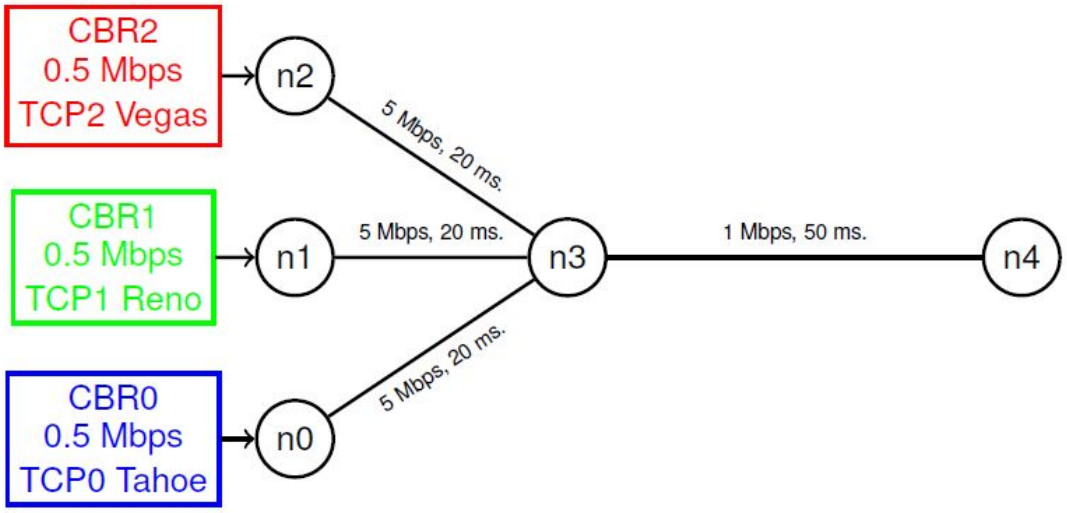

Figure 1: Network architecture. Source: [5]

The simulation consists of 5 nodes, being 3 transmitter agents (N0, N1, and N2), connected to a receiver (N3), which forwards the incoming traffic to a fifth node (N4). Nodes 0 to 2 are different TCP Agents, respectively Vegas, Tahoe, and Reno, all three are all connected to node 3 via a duplex communication line (ns duplex-link), with 5 Mbps bandwidth, 20 ms delay, and a drop tail queue configuration. While n3 has the queue size limit of 20 segments to n4, being connected to it via a duplex-link, with 1 Mbps bandwidth, 50 ms delay, and a drop tail queue configuration.

All of the TCP agents have a source generator that generates transmission of 0.5 Mbps over each agent, however, operating with different times of the activity:

- TCP Vegas is active for 2 seconds with a 2-seconds pause
- TCP Reno is active for 1 second with a 1-second pause
- TCP Tahoe is active for 0.5 seconds with a 0.5-seconds pause

All of them operate in a loop throughout the 20 seconds of simulation, being that, at time 0 (zero) they are all activated for the first time. This configuration generates is better explained in figure 2:

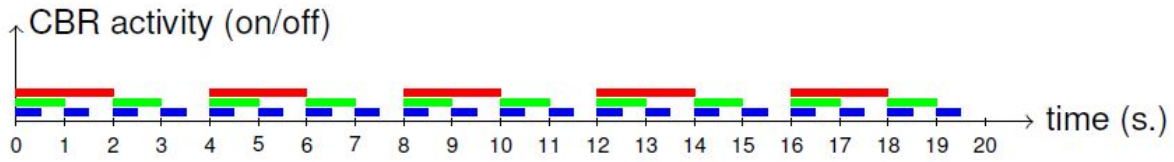

Figure 2: TCP agents active times. Source: [5]

The execution of simulation generates two log files, containing the results of the execution, named “ex2-trace.rtt” and “ex2-trace.tr” containing, respectively, metrics like Round Trip Time (RTT), Estimated RTT (SRTT), CWND and Maximum CWND (cwmax) through time by node, and information on packages sent and received.

3. Simulation Results

In order to evaluate the results, a Jupyter notebook (“Assignment2.ipynb”) was created, in which the log files are imported and data contained is extracted and processed, and by doing so, we are able to better analyze the results, as well as to create tables and graphs.

In the .tcl code, using the “trace-all” command, it was traced all the transportation of packages over the network simulation with the objective of getting the number of packages losses and packages transported from the TCP agents up to the sink TCP agent. While the “writeAgent()” function is used to record the actual values of a variety of metrics in a given moment of the execution.

3.1. Lost packages

In this section, we will display a table with the lost packets relative to each node during the simulation. In order to calculate this amount, we have selected the rows at the stacktrace with an event type of ‘d’, that signalizes the moment that the packet is dropped from the stack.

Since we have verified that all the packets were lost in the source node 3 stack, we have made use of the source address of the packet, and the results were:

Lost packages	
Node 0 (TCP Tahoe)	10
Node 1 (TCP Reno)	11
Node 2 (TCP Vegas)	1
TOTAL	**22**

Table 1: Lost packages

3.2. Transferred bytes

Below is the amount of transferred bytes relative to each node during the simulation. The transferred bytes were calculated using the event type of '-', which symbolizes the subtraction of the packet from the source node stack, and transmission to the destination node. In that case, the lost packets were not calculated in node 3, since they were actually transferred between the source address and the third node, never leaving this node to the destination address.

Transferred bytes	
Node 0 (TCP Tahoe)	765480
Node 1 (TCP Reno)	759240
Node 2 (TCP Vegas)	940000
Node 3 (Receiver)	2516560
Node 4 (Sink)	94400
TOTAL	**5075680**

Table 2: Transferred bytes

3.3. Congestion Windows

The Congestion Window (cwnd), is a variable defined by TCP equivalent to Advertised Window (used in flow control), but at the transmitter end [5]. The congestion window is one of the factors that determine the number of bytes that can be sent out at any time, being also a means of stopping a link between the sender and the receiver from becoming overloaded with too much traffic. It is calculated by estimating how much congestion there is on the link [2].

In this subsection, it is shown the congestion window behavior for each one of the source nodes. This figure below illustrates the functional differences between each TCP agent regarding the recovery of a congestion state.

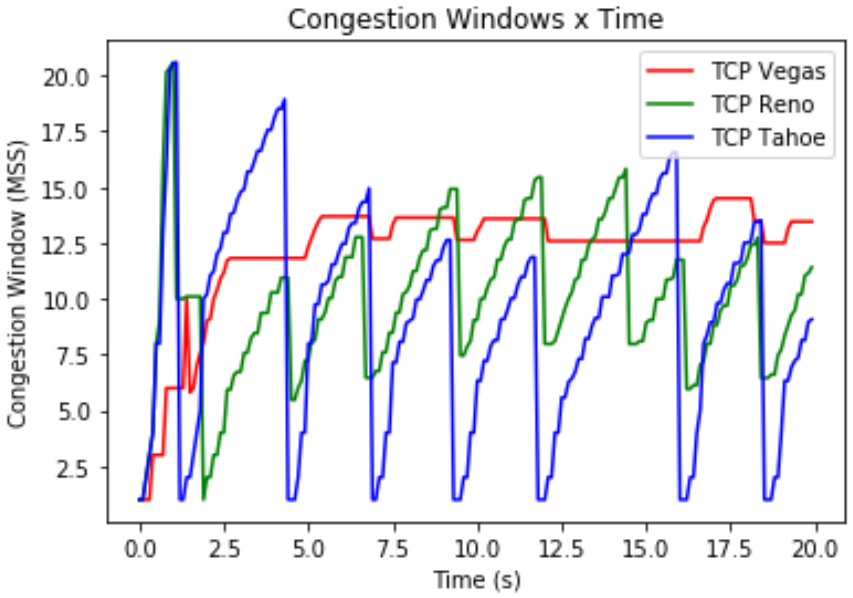

Figure 3: Congestion Window x Time Plot

4. Results Analysis

In this section we will be explaining more about the results obtained, comparing how each type of TCP affects the congestion window and the packet loss.

Analyzing the packet losses, we can see that with TCP Tahoe and TCP Reno there were much more lost packets, and seeing the congestion window behavior between them and TCP Vegas, we can notice that when the window becomes larger than 14, which is the maximum value of convergence for TCP Vegas, they start to lose packets. Since they are always retransmitting fast, the recovery is not maintained for long, so by the time, we can see that they can lose packets even with a smaller window, like between 7 seconds and 12 seconds for TCP Tahoe and between 15 seconds and 20 seconds for TCP Reno.

As for TCP Vegas, we can see the first and only lost packet around 2 seconds of the graph, that was occurred by some network congestion allied with a very short RTT (since the grow of the congestion window was very vertical) right before the loss, and then the recovery and convergence had not saturated the network, so the behavior was better.

Regarding the congestion windows themselves and the TCP implementations, TCP Tahoe and TCP Reno make use of slow start and fast retransmit, so we can see that they have a similar graphic, growing exponentially from the beginning of the transmission and every time that the recovery of the congestion is activated. the main difference between them is that when TCP Reno detects that a congestion takes place it starts a fast recovery phase, cutting the congestion window in a half, avoiding the retransmission from the initial congestion window value of 1.

On the other hand, TCP Vegas has the same features of TCP Tahoe but uses the round-trip-time to calculate the congestion window. The slow start rows at half of the rate of the other algorithms, but with the time is expected that the window gets a convergence in its size, in this case, between 12.5 and 14. Also, in the event of a lost packet, the recovery does not start from the initial window value of 1, it drops until starting to receive the acknowledgments.

5. Conclusions

In this practice, we were able, not only to have a better visualization of a network architecture involving multiple TCP agents but we also had the opportunity to see the variety of algorithms implemented by them in action. And by doing so, generating data on each execution, allowing us to observe the differences in performance.

As a result of the treatment of the data obtained in the simulations, we were able to generate the tables and graphs shown above, for a better understanding and interpretation of them. In this manner, we were able to compare the different TCP agents and their behavior when in a congestion situation.

6. References

[1] Peter L Dordal. An introduction to computer networks. Department of Computer Science Loyola University Chicago, 2018. Available at: http://intronetworks.cs.luc.edu/current/html/.

[2] TCP Congestion Control. Available at: https://en.wikipedia.org/wiki/TCP_congestion_control.

[3] Kenji Kurata. Fairness Comparisons Between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas. Available at: http://www.isoc.org/inet2000/cdproceedings/2d/2d_2.htm.

[4] The Network Simulator - ns-2. Available at: https://www.isi.edu/nsnam/ns/.

[5] Cèsar Fernàndez. Communication Services and Security TCP Congestion Lab. Departament d'Informàtica Universitat de Lleida, 2018.

[6] Nam's ISI. The network simulator - ns-2, 2011.

7. Collaborations

For the development of both codes (“Assignment2.ipynb” and “labDeliver.tcl”) our team has collaborated mainly with the following team in the class:

- Óscar López Luna and Guillem Orellana Trullols

In addition, small exchanges of information, ideas, and help have taken place with the following teams in the class:

- Gerard Donaire Alós and Roger Truchero Visa
- Emilia Naambo Shikeenga and Léo Raymond Jean Siegel