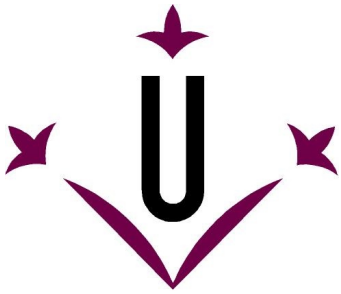

University of Lleida

Master's Degree in Informatics Engineering

Higher Polythecnic School

Exercise 2

ICT Project: Communication Services and Security

Cèsar Fernández Camón

Albert Pérez Datsira

Jeongyun Lee

April 4, 2021

Table of contents

1 Introduction

This assignment is focused to work with three different implementations of TCP[1] congestion avoidance agents, TCP Tahoe, TCP Reno and TCP Vegas; and the corresponding flow controls mechanisms.

All by understanding the network simulator ns-2 based on the programming pure script language well-called TCL[2], created by John Ousterhout in 1988.

With the aim of simulating a specific scenario and make an analysis observing the behaviour of the algorithms applied on the simulated bottleneck situation.

[1] TCP: Transmission Control Mechanism
[2] TCL: Tool Command Language

2 Simulation Environment

The Network Simulator - ns2, was used in order to carry out the simulation as its highly used among academic students and PhD Research scholars because of its benefits such us complex tecnhiques are easily simulated, experimental setup is controlled, complex and simple scenario can be easily tested and, of course, less cost.

Ns2 is a discrete event simulator targeted at networking reserach and development process. Provides better substancial support for simulation of TCP, routing and multicast protocols over wired and wireless (local and satellite) networks.

In addition, Ns2 is a mixture of OTCL[3] script and C++ language. So using OTcl library we can design and run simulations by TCL interprets.

3 Simulation Scenario

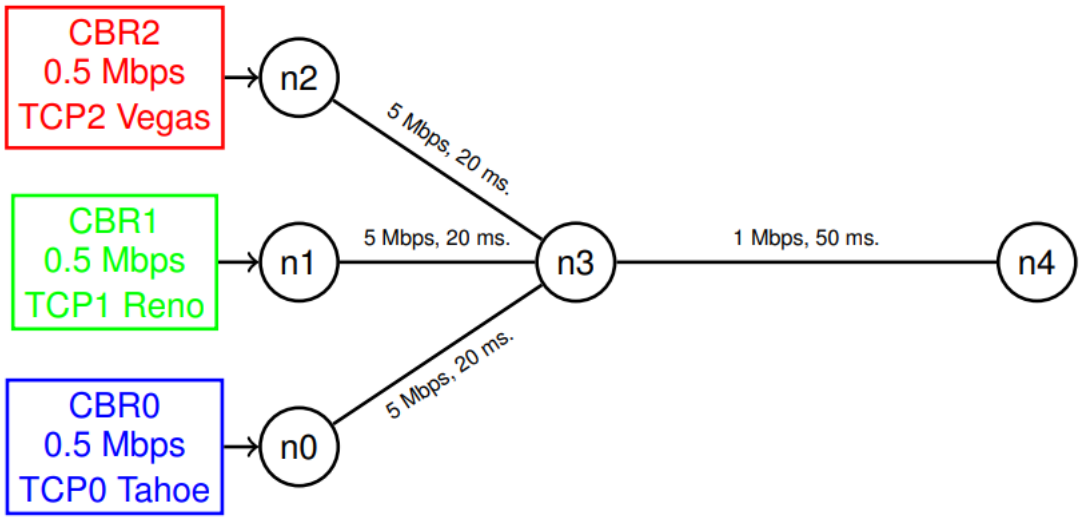

Figure 1: Simulation scenario schema

The simulation consists of 5 nodes, being the first 3 ones transmitter agents (n0, n1,n2), connected to a receveir or intermediate node (n3) operating such a router, which forwards the incoming traffic to the last node (n4).

Nodes from 0 to 2 are different kind of TCP Agents, respectively Tahoe, Reno and Vegas. All three are connected to the node 3 via a duplex communication link (ns duplex-link), meaning that is a point-to-point system allowing simultaneously communication in both directions, with 5 Mbps bandwith and 20 ms physical delay.

[3]OTCL: Object oriented extension of TCL created by David Wetherall at MIT (Massachusetts Institue of Technology)

With regard to node 3, it has the queue with a DropTail procedure configuration and a limit enqueued size of 20 segments acting as a bottle-neck for the simulation scenario. Also, this last one is connected via a duplex-link to the node 4, with 1 Mbps bandwith and 50 ms delay.

Furthermore, all TCP Agents have a source generator, called CBR, that generates transmission of 0.5 Mbps over each agent using the default configuration, 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 simulation long, being activated initially at zero time.

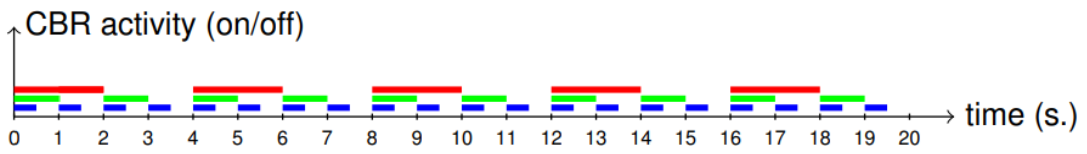

Figure 2: CBRs activity schema

Finally, the three main TCP Agents are detailed in the following subsections.

3.1 TCP Tahoe

TCP Tahoe is the first TCP variant which includes the first congestion control algorithm. This algorithm is developed by Jacobson and Karels in 1986.

Moreover, TPC Tahoe represents the second generation of TCP versions, which includes two new techniques, Congestion Avoidance and Fast Retransmission to TCP.

For congestion avoidance Tahoe uses 'Additive Increase/Multiplicative Decrease' algorithm but applying the Slow-Start algorithm improves. A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window as a threshold value. It then set cwnd[4] to one and starts slow start algorithm until it reaches the trheshold value. After that it increments linearly until it encounters a packet loss. Thus it increase it window slowly as it approaches the bandwidth capacity.

In addition, Tahoe also introduce the Fast Retransmit improvement mechanism of congestion control, based on the concept of resending the unacknowledge segment after three duplicate

[4]cwnd: Congestion Window

ACKs[5] are received, insted of waiting for the retransmission timer to expire. So, when this happens, cwnd size is reset to one packet and the process of Slow-Start is initiated. Thus, saving around half of the total TimeOut, which implies improving throughput by 20% approx.

3.2 TCP Reno

Created in 1990, TCP Reno implementation retained the enhancements incorporated into TCP Tahoe but modified the Fast Retransmit mechanism and include a new technique to improve even further the congestion control called Fast Recovery.

Fast Recovery prevents the communication channel from going empty after Fast Retransmit, thereby avoiding the need to Slow-Start to re-fill it after a single packet loss. Meaning its behaviour is not as aggressive as Tahoe in the reduction of the congestion window.

The idea is basically avoid the Slow-Start phase when duplicate ACKs arrive, being more sensitive to that event by reducing to half the congestion window and then increase linearly. Because if the TimeOut does not occur, time will be saved by recovering faster or on the contrary Slow-Start will be applied.

3.3 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.

TCP Vegas is an implementation which is a modification of TCP Reno. It builds on the fact that proactive measure to encounter congestion are much more efficient than reactive ones, proposing its own unique retransmission and congestion control strategies.

TCP Vegas detects congestion at an incipient stage based on increasing Round-Trip Time (RTT) values of the packets in the connection unlike other flavors such as Tahoe, Reno, etc., which detect congestion only after it has actually happened via packet loss.

If observed RTTs become larger, TCP Vegas recognizes that the network begins to be congested, and throttles the window size. On the other hand, if RTTs become smaller, 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.

[5] ACK: Acknowledgment

4 Simulation Results

In order to carry out the simulation and obtain the results, a TCL script was created and the log files generated during its execution were obtained.

At first instance, we introduced ourself to this programming language by means of the examples provided in class. We dedicated some time to perform the simulations shown to better understand the operation and mechanics of both the simulator and the script language.

From here, we were able to build our script (script.tcl), which you can consult for more information thanks to the comments added in-line, to represent the specific scenario required, and thus, obtain the two following log files considered.

The first one (sim-trace.tr) represents the scenario packets trace, which refers to all the activity related to the packets written using the own ns2 event format. Besides that, the other one (sim-trace.rtt) contain specific values such as RTT, cwnd, etc. about the all three TCP agents.

Moreover, it must be said, the results are focused on a few metrics. From the first log file we obtained the lost packets and the transferred bytes to focus the analysis on the network congestion for each agent. On the other hand, from the second log file we got at each time so important values such as the congestion window for each agent, to better understand how it evolved over time.

Finally, to analyse the metrics obtained, a Python3 script (process-data.py) was created to process the log files and extract the results and the corresponding plots. But also, may be found a Jupyter Notebook (process-data.ipynb) containing more detailed and structured information, in which the log files are imported and processed as the way the script does but benefiting from the interface provided by the Jupyter environment.

4.1 Lost packets & Transferred bytes

	Lost Packets	**Transferred Bytes**
TCP Tahoe	10	766520
TCP Reno	11	759240
TCP Vegas	1	940000
Total	22	2465760

We can see that the first two tcp agents, TCP Tahoe and TCP Reno obtain almost the same losses, but comparing those to TCP Vegas, this last one acquires a substancial advantage.

As discussed in previous sections the description of these protocols, we can emphasize that the results are as expected because of the algorithms used for each one by differentiating two groups. TCP Tahoe and Reno using a reactive policy while TCP Vegas use a preventive policy, in both cases to avoid the network congestion.

In addition, as this simulation was only 20 sec long, the difference would be much substantive if it got longer.

4.2 Congestion window over time

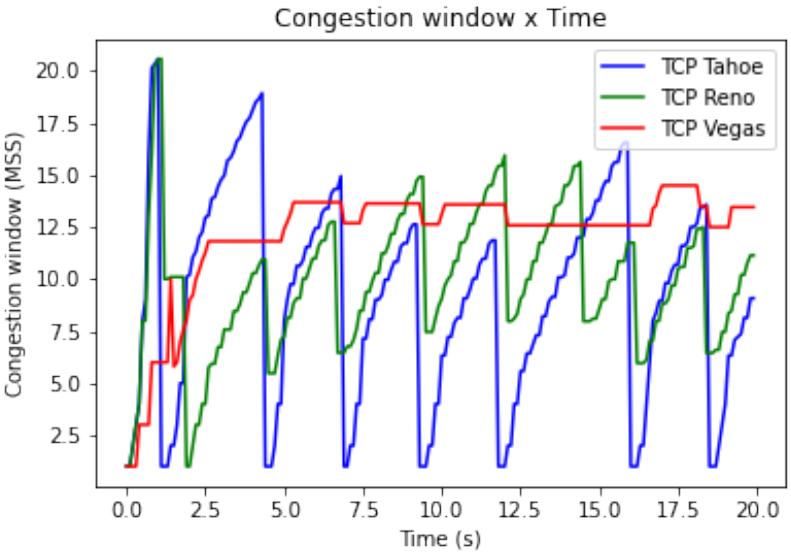

Figure 3: Congestion Window

In the previuos plot it could be observed the congestion window values over time for each TCP Agent implied.

First, we quickly saw a big difference of convergion between TCP Vegas and the other two agents. Exactly as we studied in class.

Meanwhile, TCP Vegas mantain a ratius of convergion better, the other two ones doesn't estimate in the same way the congestion window as they are more reactive/sensitive to networks events such as timeOuts.

But, for the other hand, among TCP Tahoe and Reno, as this second uses Fast Retransmit and Fast Recovery simultaneously, it converges better than its precedessor.

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 a congestion situation occurs.

References

[1] Exploration and evaluation of traditional TCP congestion control techniques

[2] Berkeley Project2 SACKRENEVEGAS

[3] Comparison of TCP congestion control mechanisms Tahoe, Newreno and Vegas IOS Journals paper

[4] TCP Performance - CUBIC, Vegas & Reno paper