Universitat de Lleida

màster en enginyeria informàtica

escola politècnica superior

curs 2020/2021

Communication Services and Security Exercise 2

Authors:
Lluís Mas
Radu Spaimoc

04/04/21

Universitat de Lleida

Contents

List of Figures

1 Objectives

This lab work had two main objectives. The first main objective was to understand the **Network Simulator (ns-2)**, which is a networking tool to create discrete event simulations. Secondly, after accomplishing the first objective, the goal was to implement a **Simulation Scenario** using a **TCL** script in ns-2. The requirements and the scenario structure are presented in the following sections.

By implementing this script the objective was to work with the **Transmission Control Protocol (TCP)** congestion and flow control mechanisms using different TCP agents as TCP Vegas, TCP Tahoe, and TCP Reno. After, analyzing this simulated scenario trace, was possible to compare the behavior of the agents and their characteristics when managing the congestion control.

2 Delivered Files

Our solution for this exercice includes the following files:

- **scenario_script:** Script to implement the presented scenario.
- **trace.rtt:** File with metrics obtained during the simulation.
- **trace.tt:** File containing the iformation of the transportation of the packages during the simulation.
- **Exercice 2 - Analysis.ipynb:** File containing the analysis of the previous files. Also, generates the Fig 4.

3 Simulation Tools

In this section are presented all the tools used in this simulation and all the environment characteristics needed to implement the presented scenario.

3.1 Environment

Before using the ns-2 program, the Fedora distribution was installed in VirtualBox a free and open-source hosted hypervisor.

3.2 Requirements

The network simulator ns-2 can be downladed from allinone web page, or running the following command on a terminal:

```
$ sudo apt install ns2
```

Also, the following packages are required:

- libX11-devel
- tcl
- tk
- gcc-c++
- libXt-devel

3.3 Analysis Tools

After implemnting the scenario, the obtained trace was analyzed using the **Jupyter notebook** under the Python language, tool that is integrated in Anaconda. The used Python packages to make a proper analysis were:

- numpy
- pandas
- matplotlib

4 Simulation Scenario

Figure 1 shows the structure of the implemented simulation scenario.

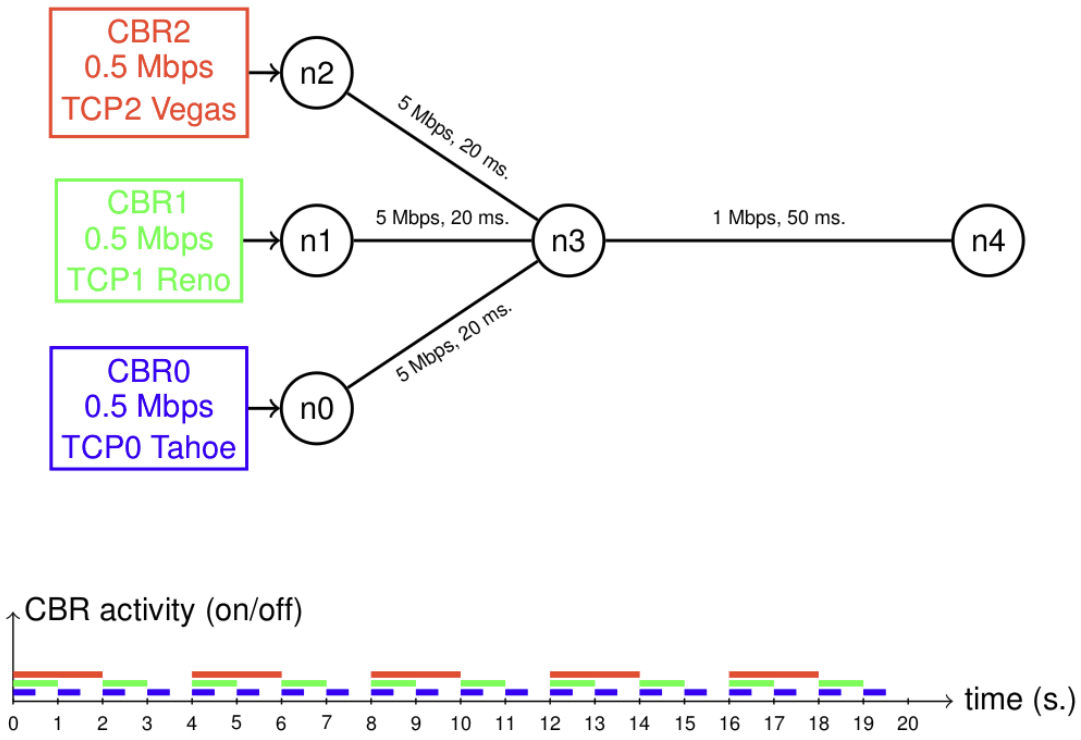

Figure 1: Simulation Scenario

As it's shown the scenario is formed by 5 nodes, being ***n0, n1, n2*** transmitter agents connected to the receiver ***n3*** which forwards the incoming traffic to the last node *n4*. Transmitter nodes are connected to ***n3*** throw a duplex communication line. As we can see all of them have ***5 Mbps*** bandwith and ***20 ms*** delay, n3 communication with n4 is also via a duplex link but with ***1 Mbps*** bandwith and ***50 ms*** delay.

The generated script simulates the proposed scenario durign 20 seconds, also implements the shown CBR activity operating with the different times shown for each node:

- **TCP0 Tahoe:** Active 2s and 2s paused.
- **TCP1 Reno:** Active 1s and 1s paused.
- **TCP2 Vegas:** Active 1/2s and 1/2 paused.

5 Scenario script

In order to create and run the previous simulation scenario a TCL script was created, *scenario_script.tcl* can be executed with the following command:

```
$ ns scenario_script.tcl
```

To implement the previous presented scenario, the script contains two sections, first a list of procedures and after the implementation of the scenario using the implemented and the default procedures. The script structure follows the example provided script *sim1.tcl* and is properly commented, the pseudocode below shows the main structure and a portion of the script:

```
...

# Prcedure to connect TCP node to sink
proc node_to_sink { agent } {...}

# Procedure to setup agents
proc agent_setup { index } {...}

# Procedure to record TCP times
proc record_tcp_times { } {...}

# Procedure to logg multiple values
proc write_agent { tcp n rfile now args } {...}

# Finishing procedure
proc finish {} {...}


# Nodes creation
set n(0) [$ns node]
...

# TCP agents (Default TCP agent: Tahoe)
set tcp_agents(0) [new Agent/TCP]
...

# Traffic CBR
set cbr_i(0) [new Application/Traffic/CBR]
...

# Duplex lines between nodes
```

```
$ns duplex-link $n(2) $n(3) 5Mb 20ms DropTail
...

agent_setup $i
...

node_to_sink $agent
...

# TCP Vegas ---> Active: 2s - Paused: 2s
for { set index 0 }  { $index < 20 }  { incr index 4 } { ....}

# TCP Reno ---> Active: 1s - Paused: 1s
for { set index 0 }  { $index < 20 }  { incr index 2 } {...}

# TCP Tahoe ---> Active: 0.5s - Paused: 0.5s
for { set index 0 }  { $index < 20 }  { incr index } {...}

# Execution commands
$ns at 0.0 "record_tcp_times"
# Duration control
$ns at 20.0 "finish"

$ns run
```

When setting the different TCP agents, NS documentation about TCP Vegas, explains that: *There are no methods or configuration parameters specific to this object. State variables are: v_alpha_, v_beta_, v_gamma_, v_gamma_*. We assigned as experimental values to **v_alpha_** and **v_beta_** the values 3 and 6 respectively.

6 Analysis Results

To analyze the obtained files and answer the proposed question, a Jupyter Notebook file was created ***Exercice 2 - Analysis.ipynb***. In it the data from the previous log files is extracted and processed to obtain the results in a fast and easy way at the same time that using the *matplotlib* library we can extract the Figure 4. The Figure 2 shows the network simulator trace format.

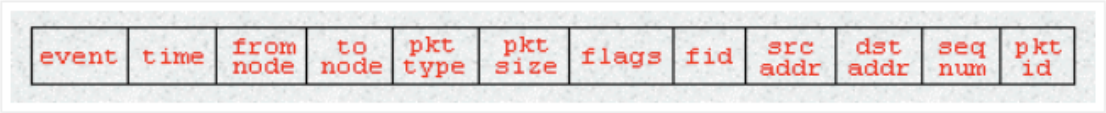

Figure 2: Event Types

The Figure 3 summarizes the different event types during a package transmission in the network simulator, in the following subsections it's explained how using these event types were obtained the different objectives.

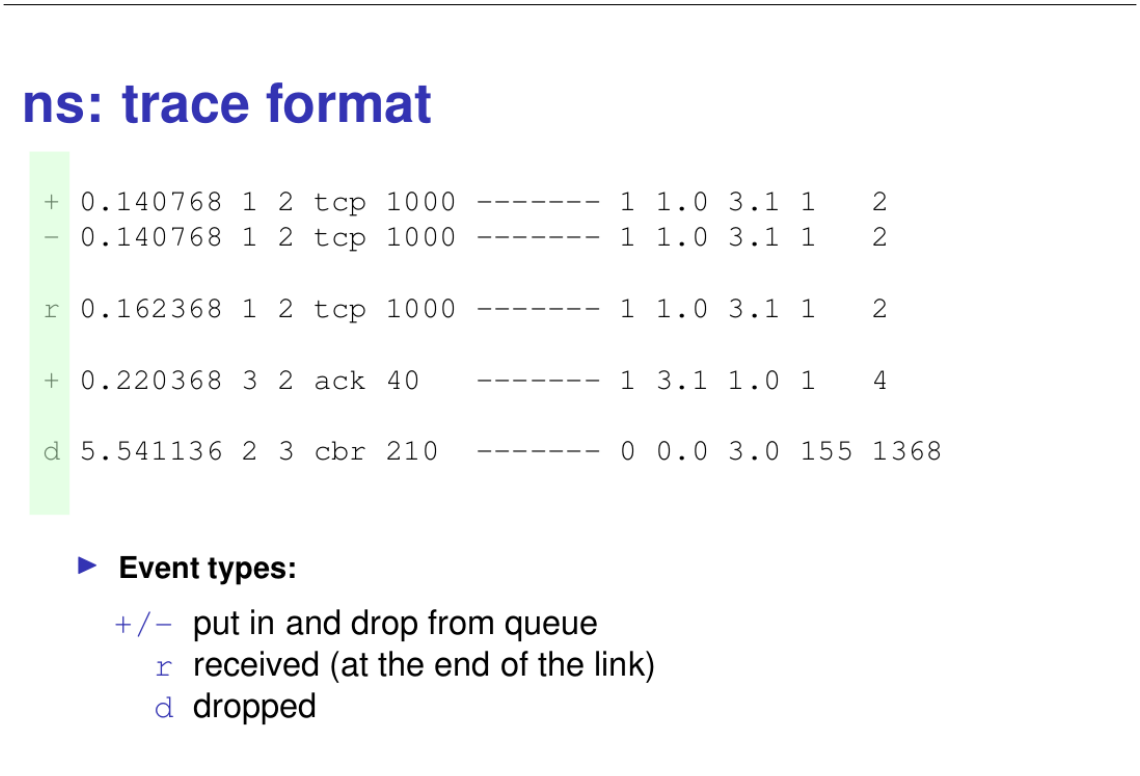

Figure 3: ns Trace Format - Event Types

6.1 Lost Packages

Table 6.1 shows the number of lost packages by each transmitter node, as we can see a total of **22** packets were lost. To obtain the number of packets lost first were selected the corresponding part of the stack trace with the source node at **n3**, then we selected the packets of the stack trace with an event type of **d**, as it's shown in the Figure 3 this event shows the moment when a packet is dropped of the stack.

Node	Lost Packages
n0	10
n1	11
n2	1
Total	**22**

Lost packages by node.

6.2 Transferred Bytes

Table 6.2 shows the number of transferred bytes for each node during the implemented scenario duration. As we can see a total of **5.075.680** bytes were transferred. To obtain the number of transferred bytes for each node was selected the corresponding part of the stack trace with an event type of **-**, which registers the transmission to the destination node and the removal of the packet from the source node queue. The lost packets in **n3** were not included.

Node	Transferred Bytes
n0	765.480
n1	759.240
n2	940.000
n3	2.516.560
n4	94.400
Total	**5.075.680**

Transferred bytes by node.

6.3 Congestion Windows

As we know the Congestion Windows (CWND) is one of the factors responsible of determining the number of bytes capable of be sended between the transmitter and the receiver. With it aims to be a measure of controlling a communication link between transmitter and receiver of no being overloaded.

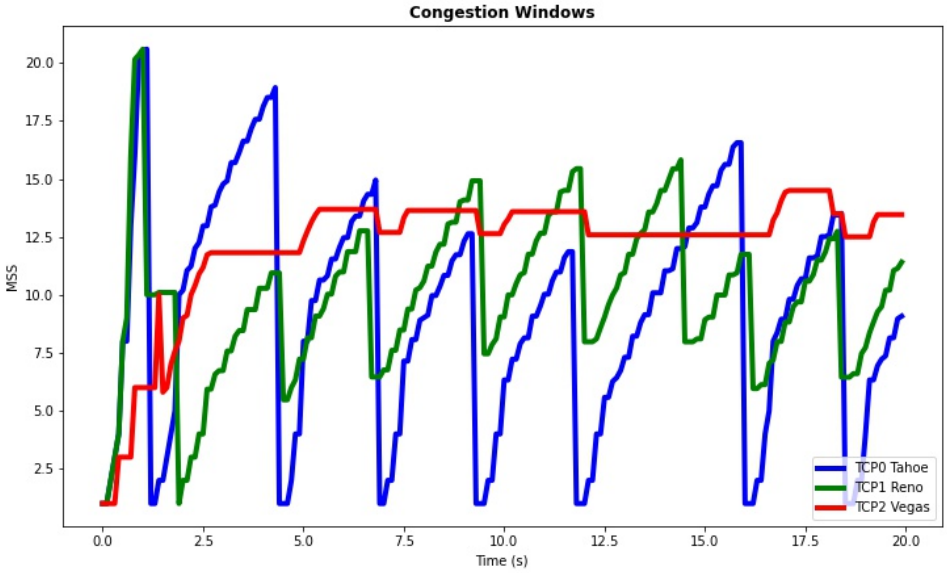

Figure 4: Congestion Windows

Figure 4 shows the **CWND** behaviour during the simulation for each transmitter node. Showing graphically the functional differences of the different TCP agents **Tahoe, Reno, Vegas** regarding the recovery of a congestion state.

7 Conclusions

In this section will be explained the different conclusions arrived after implementing the presented scenario throw the network simulator with it's specifications and analyzing the obtained trace.

From the Table 6.1 we can see the lost packages by the different TCP agents, where TCP Reno lost **50**% of the total packages followed by TCP Tahoe which lost near **45**% and TCP vegas lost only **5**% of the transmitted packages.

After analyzing the number of packages lost by the different TCP agents, Table 6.2 which shows the total of transferred bytes for each node directly agrees with Table 6.1 results. Node **n3** which receives the packages from the 3 transmitter nodes has the major amount of transferred bytes (2.516.560), followed by **n2** which is the TCP Vegas agent, and as we have seen it's the transmitter node with less packages lost (1).

Comparing the CWND behaviour for each algorithm in Figure 4, we can see that when the window is larger than the maximum value of convergence for TCP Vegas, which is somewehere between 14,5 and 15. Is where TCP Reno and TCP Tahoe start to lose packages. We assume that is because their fast retransmission which provoque that the recovery in not mantained for a long period, so by the time, the figure also shows us that they can lose packages with a smaller window to.

From TCP Vegas metrics presented in in Figure 4, we can see that the only lost package is somewhere before the first 2.5 seconds of the simulation. But, seeing the carth and the values during the simulation, we can conclude that the recovery and convergence had not satured the network at all.

TCP Tahoe and TCP Reno representation on the Figure 4 seems to have a similar behavior which tends to grow exponentially, from the first second of the simulation. We conclude that the similar behavior is because both use slow start and fast retransmit. Also in Figure 4 we can see the big difference between them. When TCP Reno detects congestion, starts what we know as "Fast Recovery" policy by reducing the congestion window in a half (with exception of the initial value of the window).

8 References

- [1] Fedora
- [2] VirtualBox
- [3] Transmission Control Protocol
- [4] TCP Vegas
- [5] TCP Reno
- [6] Difference between TCP Tahoe and Reno
- [7] TCP Implementations
- [8] Exercice Attachment
- [9] Downloading and installing ns-2
- [10] NSAM user information
- [11] NSAM Wiki
- [12] NS Documentation
- [13] N3
- [14] TCL language
- [15] Intronetworks
- [16] Congestion Window