Network Simulation with NS-3

Computer Networks II

October 26, 2016

1 Introduction

In this work, you will be using ns-3 to write simulation scripts and network packet analyzer (Wireshark/Tshark) to measure performance related to TCP and UDP, including making graphs.

2 NS-3

The ns-3 simulator is a discrete-event network simulator targeted primarily for research and educational use. The simulator is written entirely in C++ with optional Python bindings. Therefore, simulation scripts can be written in either C++ or Python. The NS-3 is primarily used on Linux systems, although support exists for FreeBSD and Cygwin (for Windows).

3 Python bindings

Python bindings allow the C++ code in NS-3 to be called from Python[1]. However, note that Python bindings for NS-3 are a work in progress and some limitations are known. For an example, callback based tracing is not yet properly supported for Python bindings. However, Python bindings provide an easy way of having hands on experience with NS-3.

4 Laboratory tasks

The laboratory consists of three tasks. The first two are warm up tasks. The two warm up exercise are step-by-step exercises designed for you to quickly get started running simulations in ns3 and to produce graphs. In the third exercise, you decide exactly what to simulate. It is an "open" exercise.

[1] Python bindings for NS-3 are being developed using a new tool called *PyBindGen*

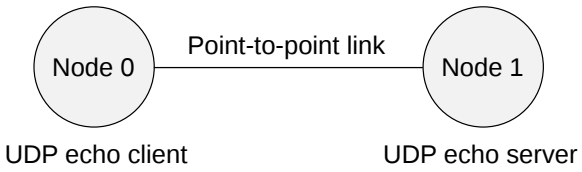

Figure 1: Network topology of task 1

4.1 Task 1

Completion of this task will give you basic understanding about NS-3 simulation. In this task, you will setup a simple client and server and trace data flows into *pcap* files and use the FlowMonitor.

Most of the work in this task has already been done in the example script `sim-udp.py`, which can be found in `Lab 2 Files` in Studentportalen. Start by downloading the script and simply run it (if using the virtual machine):

```
bash$ cd ~/ns-allinone-3.24.1/ns-3.24.1
bash$ ./waf shell
bash$ python sim-udp.py --latency=1
```

or, if using the Linux thin client machines in the lab rooms of ITC:

```
bash$ module load ns3
bash$ python sim-udp.py --latency=1
```

The output on the screen is the output from the analysis of the FlowMonitor. The simulator also creates two pcap files. Then continue with studying the script file and do these steps (if they are not already done in the scipt file):

1. Create a simple topology of two nodes (*Node0*, *Node1*) separated by a point-to-point link as shown in Figure 1.
2. Setup a UdpEchoClient on *Node0* and a UdpEchoServer on *Node1*. Let data rate of the client to be fixed.
3. Start the client application, and measure end to end throughput whilst varying the latency of the link.
4. Plot the end to end throughput against different latencies.
5. Now add another client application to *Node0* and a server instance to *Node1*. What do you need to configure to ensure that there is no conflict?

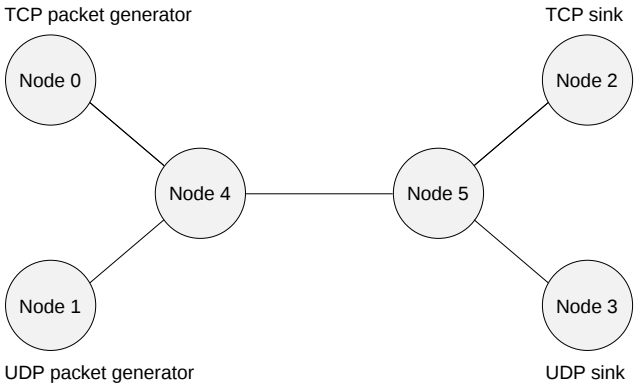

Figure 2: Network topology of task 2

4.2 Task 2

In this task, you will look at how TCP and UDP connections share bandwidth when both of them flow over a bottleneck link. For this task, download the script file `sim-tcp.py`, which can be found in `Lab 2 Files` in Studentportalen. Study the file to understand what happens. Then modify it to achieve the following (if not already implemented in the script file):

1. Create a simple dumbbell topology with two client nodes *Node0* and *Node1* on the left side of the dumbbell and two server nodes *Node2* and *Node3* on the right side of the dumbbell as shown in Figure 2. Let *Node4* and *Node5* form the bridge of the dumbbell. Use point-to-point links everywhere.
2. Install a TCP sink application instance on *Node2* such that a TCP packet generator on *Node0* connects to the sink on *Node2*.
3. Similarly, install an UDP sink application instance on *Node3* such that a UDP packet generator on *Node1* connects to the sink on *Node3*.
4. Start the TCP sink application and TCP packet generator at time 1 seconds.
5. Start the UDP application at after a while such that it clogs half the dumbbell bridge's link capacity.
6. Use pcap traces to record data flows at the sink nodes.
7. Extract instant throughput for both TCP and UDP flows at their sink nodes from pcap trace files.
8. Plot instant throughput for both TCP and UDP in the same graph against simulation time.

9. Now, enable logging for *TcpNewReno*, which is the default congestion control algorithm, in order to get updates to the congestion window size (*cwnd*).

10. Plot *cwnd* against time to see how *cwnd* is updated when the UDP data flow is started.

11. Repeat the above steps for different TCP congestion-avoidance algorithms such as *TcpReno* and *TcpTahoe* and compare the variations of instant throughput.

4.3 Task 3

Task 3 is the real task that is mandatory and you need to hand in. In this task, you are asked to simulate something. You may base your simulation on the warm up tasks by modifying parameter, topologies, protocols, applications, etc. Whatever you choose, you need to motivate your choices. Motivations could be about realistic scenarios that you do think arise and would require some extra insights. It could be about performance comparisons between protocols or parameters. Then formulate a question that you want to answer with the simulation.

Design the experiments that you want to run, i.e., what ns3 models and parameters you will use. Run several simulations and vary some of the models/parameters to investigate what happens. Preferably repeat the exact same experiment (with different random seeds) several times to calculate the average performance and confidence intervals.

Analyze the simulation results, produce some graphs, and draw conclusions. Present this at a demonstration session. Finally, write a report and hand in. You will also need to read another pair's report and give written constructive feedback. The details of the assignment is place in Studentportalen on the web page *Lab 2 - Simulation*.

In this task, you may work with the network topology of task 2, but you may, for instance, extend it as shown in Figure 3 or any other more complex network topology. In this task, you are free to select what to be changed and what to measure. For inspiration, here follows a list of things you could try:

- Change the data rate of the links
- Change propagation delay of the links
- Change the queue length or type (e.g., RED queue)
- Transport protocol (UDP or TCP)
- Different TCP versions (e.g., *Reno*, *NewReno*, *Tahoe*, *Westwood*, *Westwood+*)
- Introduce packet errors on some links

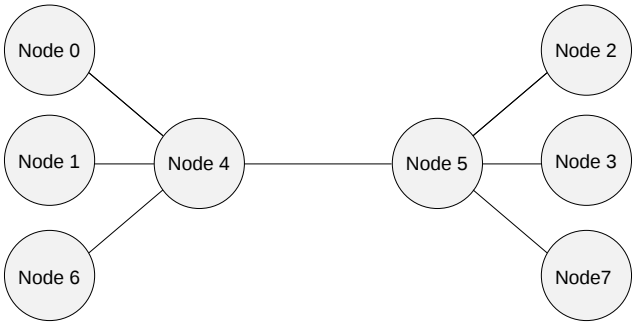

Figure 3: Network topology of task 3

- Use different types of applications (CBR, BulkTransfer, OnOff, ...)
- Measure through-put, packet loss, retransmissions, congestion window, fairness between flows, packet delays, and/or file download times, etc.

However, you have to be rational about your selection.

Appendix

How to extract instant throughput from pcap files?

In these tasks, instant throughput is number of bytes received during a particular time interval. In order to extract instant throughput from pcap files, wireshark or the `tshark` command can be used. Note that packet re-transmissions are not accounted in this instant throughput extraction.

The simplest way is to use wireshark. Just launch wireshark with your pcap file as follows (we assume your pcap file is called `sim-tcp-0-0.pcap`):

```
bash$ wireshark sim-tcp-0-0.pcap
```

From the menu *Statistics*, choose *IO Graphs*. Immediately, you will see a graph. You may want to zoom by playing around with the parameters *Tick interval* and *Pixels per tick*. Try `0.1 sec` and `2`. You can also add multiple graphs by selecting filters. Try, for instance, `tcp.analysis.lost_segment` or `tcp.analysis.retransmission`.

```
bash$ tshark -r <pcap file name> -t r -q -z io,stat,0.5
```

Here, 0.5 in `io,stat,1` is the time interval in seconds.

As the output of this command, lines similar to the followings are given.

```
================================
| IO Statistics                |
```

```
|                               |
| Interval size: 0.5 secs       |
| Col 1: Frames and bytes       |
|-------------------------------|
|              |1               |
| Interval     | Frames | Bytes |
|-------------------------------|
|  0.0 <>  0.5 |      3 |   170 |
|  0.5 <>  1.0 |      0 |     0 |
|  1.0 <>  1.5 |     22 | 19136 |
|  1.5 <>  2.0 |     38 | 20104 |
```

You can use the tool `plot_tput.py` to make a graph out of this data file. Just run the following to generate a plot based on a pcap file:

```
bash$ tshark -r sim-tcp-0-0.pcap -t r -q -z io,stat,0.5 > tputfile
bash$ python plot_tput.py tputfile
```