Network Technology and Programming Lab
Assignment 2

Stefanos Papadakis , Manolis Spanakis

Deadline: 24 March 23:59

March 12, 2019

General Information

The goal of this assignment is to become familiar with socket programming and the challenges of measuring the end-to-end performance of a network. The implementation language is up to your choice, but we strongly recommend to choose a low-level one (preferably C/C++ and POSIX sockets) that will allow you to get a deep understanding of network programming.

In addition, the assignment requires to extend the network topology of the previous assignment in order support IPv6 too.

1. Introduction

In this assignment you have to implement a network measuring tool, that supports both IPv6 and IPv4 hosts. The tool will measure the end-to-end throughput of the network, jitter and an estimation of the one-way latency.

The tool works in a client-server model. The server waits for connections and the clients connect to the remote server to begin the experiment. There are two communication channels between each client and the server.

- The signalling one, which is used for information exchange and signalling between the server and the client. This communication channel uses **TCP**
- The experiment channel. This channel transfers data over **UDP** that are part of the network measurement process.

Your are free to create additional channels depending the runtime parameters of your program.

2. Parameters

Your program should be able to receive the following command line parameters. Note that you have to support exactly the syntax below.

Server/Client parameters:

- **-a**: In server mode, this argument specifies the IP address of the network interface that the program should bind. If none given the program binds in all available interfaces. In client mode this argument provides the IP address of the server to connect to.
- **-p**: In server mode this argument indicates the listening port of the primary communication TCP channel of the server. In client mode, the argument specifies the server port to connect to.

  Server parameters

- **-s**: The program acts like server.

  Client parameters

- **-c**: The program acts like client.
- **-l**: UDP packet size in bytes
- **-b**: The bandwidth in bits per second of the data stream that the client should sent to the server. The program should take into consideration the overhead of the lower TCP/IP stack layers in order the measurement to be as accurate as possible.
- **-n**: Number of parallel data streams that the client should create. For each one of these streams a new thread is assigned in both server and client.
- **-t**: Experiment duration in seconds. The client should send for the specified amount of time and the stop the data stream. Before it exits, it informs properly the server in order the later to print the measurement results. If this argument is not specified, the data stream should be continuously sending data until a user termination signal occurs.
- **-d**: Measure the one way delay, instead of throughput, jitter and packet loss.

3. Network Measurements

At each run the tool should perform either the one way delay or the throughput, goodput, jitter and packet loss measurement, depending on the argument specified. The measurement process is performed **at the server** and stops when the client sends appropriate message through the communication TCP channel. Then the results are printed to the server and are sent back to the client for printing to the terminal.

- **Average throughput:** The server measures the data that receives from the client until a termination signal from the control channel is received. When this happens it computes the effective throughput. As the tool can measure only the payload on the UDP packets, proper adjustments should be made to take into account the headers overhead of the different layers. Try these adjustments to be as accurate as possible.
- **Average goodput:** Same as throughput, but measuring only the bytes received from the UDP packets.
- **Packet loss percentage:** The tool should be able to report the percentage of lost packets. To achieve this each packet should contain a header with an appropriate counter. The server uses this counter to extract the packet loss. Special attention should be given in edge cases such as the change for packets to be lost at the end of the transmission. To overcome this problem, the client can send among with the termination signal and the counter of the last packet sent.

- **Average jitter:** Jitter is the difference between the inter-arrival times of consecutive packets.
- **Standard deviation of jitter:** Except the average, the tool should be able to specify the standard deviation of the jitter during the experiment.
- **One way delay:** In this type of measurement the tool exchange data in a different way. The most common technique to measure the one way delay is the RTT/2. We discussed in the class why this measurement is not so accurate. Extra bonus points will be given to teams that will try to estimate more accurately the one way delay using their tool.

4. Implementation considerations

Measuring accurately the time is quite critical in such type of applications. The timing function should provide enough precision and should be not subjected to timing adjustments from external sources (NTP, PTP, summer/winter time changes, etc).

Remember that the goal of this tool, is to extract accurate results about the network. Filling the data buffers with random data, or allocate and de-allocate continuously memory can severely affect the precision of your measurements. Fill only the necessary minimal information and use pre-allocated buffers.

Another issue that can destroy your measurements is the blocking nature of some network functions. You should identify these corner cases and deal with them. You do not need to start measuring time before a blocking call, that you do not known for how much time will block for example.

5. Experiments

Restore the network topology of the Assignment 2. Make the proper adjustments, so the network can support IPv6 with the same hierarchy. You can use any valid IPv6 addressing scheme.

6. Single Server - Single Client

Conduct the following set of experiments, reporting all the available statistics that the tool provides:

- PC1 to PC2, for all the possible combinations of packets sizes of 16, 64, 512, 1024 bytes and 100 Kbits, 1 Mbit, 10 Mbit, 100 Mbit, 1 Gbit throughput.
- PC2 to PC3, for all the possible combinations of packets sizes of 16, 64, 512, 1024 bytes and 100 Kbits, 1 Mbit, 10 Mbit, 100 Mbit, 1 Gbit throughput.

Try to explain your findings. Does the number of the intermediate nodes affect the results? If yes, are there any combinations of packet length - throughput, where the effect is more notable? Discuss.

Repeat the above measurements, using IPv6. Are any differences in the observed values? Discuss.

7. Single Server - Multiple Clients

Conduct the following set of experiments, reporting all the available statistics that the tool provides:

- PC2 to PC1, for all the possible combinations of packets sizes of 16, 64, 512, 1024 bytes and 100 Kbits, 1 Mbit, 10 Mbit, 100 Mbit, 1 Gbit throughput.
- At the same time make a measurement from PC3 to PC1, with the same parameters of the PC2 to PC1 stream

Discuss how the results are affected or not using multiple streams from different machines into a single server.