Packet Pair

NETMET Lab Exercises 4

Introduction

In this session we will measure information about the quality of a network link or route between a client and a server. These measurements can be the *bandwidth*, the *jitter,* or even the *datagram loss*.

In order to gain this data, we will first use iPerf with the protocol TCP and UDP.

Then, we will see a method called Packet Pair in order to estimate the bandwidth.

Finally, we will use the Packet pair method in order to mimic the bandwidth estimation feature of iPerf by coding a simple client and server program.

Using iPerf to measure quality of network link

Tip : Remember you can use **man** command to have information about how to use a linux command.

Create pods from 2 different EdgeNet nodes. One will be the server whereas the other will be the client.

- Uni-directional Bandwidth estimation
  Execute iperf in server mode on one pod : **iperf -s**
  Execute iperf in client mode on the other : **iperf -c <ip_address_of_server_node>**

  You should see the maximum achievable bandwidth between the two nodes. What is it ?

- Bi-directional Bandwidth estimation
  Execute iperf in server mode on one pod : **iperf -s**
  Execute iperf in client mode on the other : **iperf -c <ip_address_of_server_node> -r**

  This time, iPerf is measuring the maximum bandwidth in both directions one direction after another.
  You can do the both directions simultaneously using this command client-side :
  iperf -c <ip_address_of_server_node> -d
  Is there a change between measuring the bandwidth iteratively or simultaneously ? Why ?

- Changing the TCP window size
  Execute iperf in server mode on one pod : **iperf -s -w 4000**
  Execute iperf in client mode on the other : **iperf -c <ip_address_of_server_node> -w 2000**

  Here we changed the TCP window size for both the client and the server. Explain why a window too small can give poor performance.

- Jitter and datagram loss
  Execute iperf in server mode on one pod : **iperf -s -u -i 1**
  Execute iperf in client mode on the other : **iperf -c <ip_address_of_server_node> -u**

  You should see information about jitter and datagram loss. Is the link between the two pods in a good shape ?

Tip : To keep a good link quality, the packet loss should not go over 1 %. A high packet loss rate will generate a lot of TCP segment retransmissions which will affect the bandwidth. The jitter value is particularly important on network links supporting voice over IP (VoIP) because a high jitter can break a call.

Understand the underlying network concepts of iPerf

Perform some iperf commands you just used in the previous along with capturing the network data via a sniffer tool (server side + client side) and to answer these questions.

Questions :

- Define *bandwidth*, *jitter,* and *datagram loss* precisely.
- For each of these metrics, give which protocol iPerf is using to have this information and explain why.
- For each of these metrics, explain how iPerf does to compute this information.

- Cite one protocol-independent reason that prevents iPerf to be precise ?
- Cite one constraint using iPerf with TCP ?
- Cite one constraint using iPerf with UDP ?

Mimic iPerf logic with Packet-Pair method

Now we also want to experiment with the packet pair method (see slides of the lecture).

For this we use a very simple client and server program written in Python. The **server** program listens to UDP packets and when it gets a pair of packets, it outputs the time difference between the two and it calculates the speed. The **client** simply sends two packets back-to-back.

Try to implement the packet-pair method by implementing both client and server.

Tip : In Python, you can use the built-in library socket to do the network interaction between the server and the client.