Setting up the QUIC testing environment

Carmine Bianco

May 2, 2017

1 Part 1: Setting up the Virtual Machines

In order to be able to test the implementation of the QUIC protocol used for this work, two Virtual Machines running on the same PC are needed. Thus, the following steps are needed:

1. (a) Install VirtualBox, by typing in

   ```
   sudo apt-get install virtualbox
   ```

   in the terminal.

   (b) Download the ISO file for Ubuntu 14.04 from http://releases.ubuntu.com/14.04/.

   (c) Set up a Virtual Machine. Choose *Ubuntu 64 bit* as the OS. Call the VM *Client*. Follow the on-screen instructions for choosing the path where the Ubuntu ISO has been stored, and so on.

2. (a) In the VM's context menu, go to *Devices* → *Insert Guest Additions CD Image*.

   (b) Install the Guest Additions on the VM by following the on-screen instructions, then reboot the VM.

   (c) Open the VM's terminal and type in

   ```
   sudo adduser <chosen username> vboxsf
   ```

   This will enable sharing folders with the host machine. Reboot the VM.

   (d) Go into *Devices* → *Shared Folder Settings*. Click on the "Plus" symbol. Choose the folder with all the necessary data and tick all boxes (*Read only, Auto-mount, Make permanent*).

   (e) Reboot the VM again. Now you should find your kernel data under */media/sf_<folder name>*

3. Open the terminal and type in

   ```
   sudo apt-get build-dep linux-image-$(uname -r)
   ```

4. Use the *cd* command to go to the folder where your kernel installation files are stored and type in the terminal

   ```
   sudo dpkg -i *.deb
   sudo reboot
   ```

5. During the reboot, hold the **Shift** key. This will open the GRUB menu. There, choose *Advanced options for Ubuntu*, find the custom kernel's name and press Enter.

6. Install the *Virtualbox Guest Additions* from scratch by repeating steps 2a) and 2b). Thus, you will be able to access the shared folder again.

7. (a) After rebooting the VM, open the terminal and type in

   ```
   cd /usr/include/netinet
   sudo apt-get install vim
   sudo vim in.h
   ```

   You'll find a list of protocols, each corresponding to a number.

   (b) Press a to go into insert mode, then append to the list the following lines:

   ```
   IPPROTO_QUIC = 18 /*QUIC protocol*/
   #define IPPROTO_QUIC IPPROTO_QUIC
   ```

   This will let the system gain knowledge of the existence of this protocol.

   (c) Press Esc, then type *:wq* and enter to exit and save.

The same procedure needs to be done for the server. It is either possible to follow the steps from scratch (this time around, naming the machine *Server* instead of *Client* or to clone the first VM.
To do this, open the Virtualbox Manager, right click on the VM and select *Clone*. Change the name of the clone to *Server*, tick the *Reinitialize the MAC address of all network cards* box and choose the "Full clone" option in the following dialog window.

2 Part 2: Setting up the internal network

1. (a) In the Virtualbox Manager window, click for each VM on the *Settings* button.

   (b) Select the *Network* window and change the *Attached to* field from NAT to **Internal Network**. Rename the network to *intnet*. Do the same for both machines and check in the *Advanced* part whether the two network interfaces' MAC addresses are different.

2. Open the host machine's terminal and type in

```
VBoxManage dhcpserver add --netname intnet -ip 192.168.10.1
--netmask 255.255.255.0 --lowerip -192.168.10.2 --upperip
192.168.10.100 --enable
```

This command will create a DHCP server for the *intnet* internal network, which is needed to allocate each VM a different IP address.

3. Boot up the Virtual Machines as done in Part 1, Step 5. The IP address allocated to each VM can be read either in the lower taskbar by hovering over the network adapter symbol or in the terminal by typing

   ```
   ifconfig
   ```

4. For both machines, copy the *Test code* subfolder from the shared folder into your home folder. **Only on the client machine**, also copy the *File.pdf* test file into the same folder.

5. On both machines, go using *cd* to the folder where the test code is stored and run the *./run_first.sh* shell script.

6. Open *quicServer.c* on the server VM, *quicClient.c* on the client VM with the command

   ```
   vim quic<Server/Client>.c
   ```

   Press a to go into Insert mode. In the lines

   ```
   serv_addr.sin_addr.s_addr= inet_addr("10.13.13.102");
   client_addr.sin_addr.s_addr= inet_addr("10.13.13.101");
   ```

   replace 10.13.13.102 and 10.13.13.101 with your server and client IP addresses respectively. Save the files by pressing Esc, typing *:wq* and entering.

7. On the server machine, compile *quicServer.c* using the command

   ```
   gcc -o quicServer quicServer.c
   ```

   On the client machine, compile *quicClient.c* using the command

   ```
   gcc -o quicClient quicClient.c
   ```

8. Run *./quicServer* on the server machine. **Only after the program outputs *Waiting to receive file...***, run *./quicClient* on the client machine. If everything has been done right, the file should be correctly sent by the client and receive by the server.
   Before repeating the test, delete the received file (rm New.pdf in the server terminal window), otherwise further transfers won't be completed.

3 Part 3: Setting up Wireshark for network testing

As stated in the thesis, the tool Wireshark is needed in order to get testing data from the network.

1. Make sure both VMs are connected to the Internet. To do this, after you have powered off the machines, go into the *Settings* → *Network* menu, go to *Adapter 2*, tick the *Enable network adapter* box and select *Attached to NAT*.

2. Boot up both machines, open the terminal windows and write[1] the following commands:

   ```
   sudo apt-get install wireshark
   sudo groupadd wireshark
   sudo usermod -a -G wireshark YOUR_USER_NAME
   sudo chgrp wireshark /usr/bin/dumpcap
   sudo chmod 750 /usr/bin/dumpcap
   sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/dumpcap
   sudo getcap /usr/bin/dumpcap
   ```

   These commands are meant to give the current user all the necessary privileges for the tool. Afterwards, log out and back into the VM.

3. Open Wireshark (at least on one machine). Choose the **eth0** capture interface and press the green *Start* button.

4. At the end of each connection test, press the red *Stop* button. You will now be able to analyze the statistics.

   (a) In the *Statistics→Summary* menu, data such as the average throughput in MBps are stored.

   (b) The throughput graph can be found in the *Statistics→IO Graph* window. The user is then able to filter some of the connections as well as use a rolling average filter on the graph.

   It is also possible to save the capture file by clicking on the corresponding icon.

5. The *netem* tool, which is included in Linux, makes possible for the users to emulate network delay and loss characteristics. In order to enable them, type in the terminal

   ```
   modprobe sch_netem
   ```

[1] These commands are taken from a Q&A thread in the Wireshark.org forum: https://ask.wireshark.org/questions/16343/install-wireshark-on-ubuntu

Delay and loss can be then adjusted using the commands

```
sudo tc qdisc add dev eth0 root netem delay Xms loss Y%
```

From the second time onwards, replace the word *add* with *change*. The current properties of the network can be checked by typing in the terminal

```
tc qdisc list
```

As far as this document goes, the testing has been facing two major issues. First of all, since the *send* system call in the *quicClient* function only asks the protocol for data transmission without waiting for further acknowledgments, whenever all data chunks have been sent, the socket is closed successfully on the client side and the server side just crashes while reading the next chunk (there is nothing to read anymore, since the file descriptor points to a now invalid socket!). This problem has been encountered in a large majority of the tests with a packet loss rate $\neq 0\%$.
While other solutions have been tried out (like using the *shutdown()* call instead of *close()* at the end of the client function, or setting the *SO_LINGER* flags accordingly, the only solution which currently provides for a working simulation is waiting, at the end of the client function, for a small "acknowledgment" packet the server sends back after having processed the whole data.
The issue of comparing the data which this testing environment gives out to those in the thesis (Why such a big discrepancy? How did Mr. Suman get his values for the average network throughput?) and that of actually testing the TCP protocol in this environment (only changing the socket creation function won't work) still need to be solved, though.