1. **Setting up an EC2 test server**

Create an ec2 instance with key pair value

It generates a .pem file

For ssh, do puttygen -> load -> load pem file -> save private key

Load private key on putty

Putty - left handside - ssh - auth - load private key

This is how you ssh into an ec2 instance

Create a folder in ec2 - mkdir wehe

Clone the repo there - git clone \*link\*

Cd ../src

Copy the ssl folder content there

Once you have cloned the repo on your EC2 machine (lets say it is now in the directory ~/home/wehe\_desktop\_py3/), download the ssl directory that contains the encryption key [here](https://nam05.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdrive.google.com%2Fopen%3Fid%3D1SrQU_1tCjyK7fP7qRTJVUM6oPq3iDtb_&data=02%7C01%7Csonparote.p%40northeastern.edu%7Ce1a2dcf205a4423d8f7708d7fc2e6157%7Ca8eec281aaa34daeac9b9a398b9215e7%7C0%7C0%7C637255148455744159&sdata=zbHjaKD%2FfpwK%2BmWAWJpzskEYXSeQcdWrS4FYbBPrpok%3D&reserved=0), and the replay files for YouTube and Netflix [here](https://nam05.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdrive.google.com%2Fopen%3Fid%3D1GtmaYQqjJLFW68HwG-cvRXzVtvbNw5kB&data=02%7C01%7Csonparote.p%40northeastern.edu%7Ce1a2dcf205a4423d8f7708d7fc2e6157%7Ca8eec281aaa34daeac9b9a398b9215e7%7C0%7C0%7C637255148455754157&sdata=F93xXJJb2E30vkzZyAPodXmSVsxxbx4ma9bdnRZtVk0%3D&reserved=0).

Copy and unzip both of them on your EC2 machine. Put the ssl directory to ~/home/wehe\_desktop\_py3/src/, and the replays to a directory like ~/home/replay\_files/.

Change the content ~/home/wehe\_desktop\_py3/src/folders.txt to where the replay files are (e.g., ~/home/replay\_files/Youtube\_12122018 etc.).

Try both sudo python3 replay\_analyzerServer.py --ConfigFile=configs.cfg --original\_ports=True,

sudo python3 replay\_server.py --ConfigFile=configs.cfg --original\_ports=True, install required packages.

Once everything is correctly installed, you should be able to run ./restartServers.sh which creates two screen sessions (replay and analyzer) running in the background.

Now you can change the server address in the Wehe mobile app to point to your EC2 machine, and run YouTube or Netflix replays.

1. **To install packages**

Install pip sudo python3 -m pip install psutil

For psutil, install python3-dev

For gcc

For tornado, install a particular version

<https://www.cyberciti.biz/faq/install-epel-repo-on-an-rhel-8-x/>

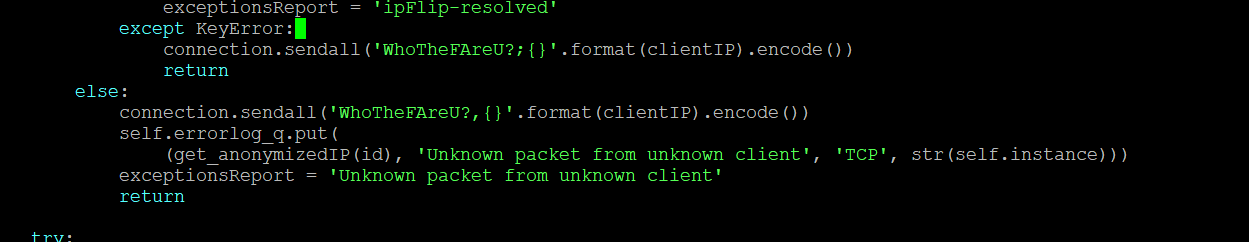
To install screen

1. **Errors encountered in the beginning**

Error connecting to side channel

\*\*\*CHECK ERROR LOGS: ('209.17.96.0', 'Unknown packet from unknown client', 'TCP', "('', 443)")\*\*\*

Replay server



Ps - Fa ; sudo kill pid

You can't use the public IP, but you can use the public hostname (ec2-IPADDRESS-.us-east-1.compute.amazonaws.com) because this will resolve to the internal IP address when called from inside EC2, and will resolve to the external IP from outside Amazon's network.

New client: wNIBQZ9FbT WebexRandom-04282020 1 DiffDetector 74 True

[15]\*\*\* Unknown replay name: WebexRandom-04282020 (wNIBQZ9FbT) \*\*\*

1. **Basic overview (Thanks to Derek)**

During the tests, the client and the server communicate. In the server code, I believe this happens in the handle() function in server\_replay.py. If you go to that function, there should be a function listing the steps to run each test. Basically, the user chooses several apps or ports to test. For each app or port, those steps get run twice (or possibly four times if the result is inconclusive).

A test contains two parts: the original replay, which contains actual traffic from an app/port, and the random replay, which contains random traffic for an app and is port 443 for port tests.

The client will randomly choose whether to run the original or random replay first.

To initiate a replay, the client will send some information about the replay that is about to happen (steps 0 and 1). Next, the client will ask the server for permission to run the replay (step 2). There are a variety of reasons why permission isn't granted, such as the server doesn't support the replay, the server is low on resources, or another client with the same IP is currently running a test.

After permission is granted, the client will send "noIperf" (step 3a). (mostly vestigial code) After that happens, the client sends to the server information about the client device (step 3b), such as location. Then, the server starts tcpdump (step 4).

In step 5a, the server sends the client a port mapping, which I believe helps the client know where to send packets, and in 5b, the server sends something called "sender count" (not sure what this is)

At this point, the client sends the replay packets to the server and throughput data is collected both on the server and client sides. All port tests and most apps use TCP, while a few apps use UDP.

After the client finishes sending packets, it tells the server that the packets have been sent (step 6), and the client sends to the server the throughput information collected on the client side (step 7). In step 8, the client sends "Result;No" (again not sure why). Step 9 closes the connection between the client and server.

After all this, one replay, either the original or the random replay, has run. The process starts all over again with the other replay.

When the other replay is finished, the client asks the server to analyze the two replays. Every test gets stored on the server, so the way a replay is identified is through randomID (which I believe is called realID in the handle() function), historyCount, and testID. randomID is a random 10 character string that identifies a client device. historyCount is the ID number of the test (yes, ID number of the test is historyCount, not testID), so for example, a historyCount of 25 means that test is the 25th app/port the client has run. The testID is the replay number. So, when testing apps, the original replay has ID 0, and the random replay has ID 1. When testing ports, the port being tested has ID 0, and port 443 has ID 1.

To analyze the replays, the client sends the randomID of the user, the historyCount of the current test, and a testID of 1. The server will that specific replay data and compare it to the data of testID 0, which is the other replay that was run. The server saves the analysis. The client then requests the results, and the server sends some data back. The client uses that data to show the user that either there is differentiation, no differentiation, or inconclusive. If there is differentiation or no differentiation, the client will move on to the next app/port to test. If the result is inconclusive, the client might automatically run the app/port again to attempt to come to a conclusive result. Whether an inconclusive result is rerun automatically can be changed by a button in the app. I'm not exactly sure where the code to analyze and send results is located on the server.

1. **Comparing paris traceroute and normal traceroute**

**Setting up paris traceroute -**

https://github.com/libparistraceroute/libparistraceroute/wiki/Installation

Paris traceroute vs normal traceroute

Paris -

1 52.15.0.101 45.565ms 45.567ms 45.576ms

2 100.65.26.0 65.117ms 65.104ms 65.111ms

3 100.66.12.148 32.649ms 32.655ms 32.662ms

4 100.66.15.64 15.421ms 15.433ms 15.441ms

5 100.66.7.65 17.732ms 17.739ms 17.778ms

6 100.66.4.123 634.605ms 634.612ms 634.632ms

7 100.65.9.193 0.426ms 0.429ms 0.436ms

8 15.230.39.197 1.472ms 1.475ms 1.457ms

9 15.230.39.210 0.899ms 0.901ms 0.901ms

10 52.93.239.122 1.244ms 1.277ms 1.280ms

11 100.92.53.128 17.839ms 17.842ms 17.845ms

12 100.92.48.28 11.888ms 11.890ms 11.891ms

13 100.92.48.45 11.026ms 11.029ms 11.030ms

14 100.92.49.44 15.673ms 15.680ms 15.683ms

15 100.92.49.29 11.303ms 11.307ms 11.307ms

16 52.93.133.104 11.161ms 11.166ms 11.165ms

17 100.91.163.4 11.842ms 11.840ms 11.847ms

18 100.91.163.19 16.761ms 16.766ms 16.767ms

19 100.91.160.18 11.308ms 11.294ms 11.301ms

20 100.91.160.9 11.193ms 11.196ms 11.197ms

21 100.91.177.19 11.022ms 11.025ms 11.025ms

22 100.100.8.17 11.225ms 11.225ms 11.259ms

23 100.100.65.200 11.271ms 11.275ms 11.277ms

24 100.100.65.195 50.664ms 50.669ms 50.670ms

25 100.100.2.32 11.208ms 11.211ms 11.210ms

26 99.82.181.25 10.895ms 10.937ms 10.932ms

27 \* \* \*

28 108.170.246.33 12.672ms 12.671ms 13.210ms

29 108.170.246.34 11.784ms 11.787ms 11.786ms

30 108.170.232.199 12.342ms 12.344ms 12.345ms

Normal -

1 52.15.0.97 31.212 ms 52.15.0.101 43.908 ms 52.15.0.97 31.156 ms

2 100.65.26.0 3.755 ms 100.65.25.48 8.211 ms 100.65.25.0 8.209 ms

3 100.66.12.64 8.032 ms 100.66.12.94 40.177 ms 100.66.12.78 3.810 ms

4 100.66.14.142 21.758 ms 100.66.15.138 16.387 ms 100.66.14.196 12.527 ms

5 100.66.7.97 10.557 ms \* 100.66.7.169 15.087 ms

6 100.66.4.21 16.995 ms \* 100.66.4.203 11.755 ms

7 100.65.8.129 0.484 ms 100.65.11.193 0.350 ms 100.65.9.97 0.452 ms

8 15.230.39.195 0.935 ms 15.230.39.221 0.902 ms 52.95.3.135 1.412 ms

9 15.230.39.210 1.354 ms 52.95.1.252 1.691 ms 15.230.39.70 0.848 ms

10 52.93.239.54 6.662 ms 52.93.239.78 0.562 ms 52.95.2.179 0.907 ms

11 100.92.53.156 10.877 ms 100.92.53.28 16.301 ms 100.92.53.0 12.066 ms

12 100.92.43.72 10.704 ms 100.92.43.98 11.603 ms 100.92.48.122 11.046 ms

13 100.92.48.89 14.558 ms 100.92.48.13 11.759 ms 100.92.48.39 11.708 ms

14 100.92.49.64 11.588 ms 100.92.44.44 11.032 ms 100.92.49.70 11.422 ms

15 100.92.44.127 11.385 ms 100.92.44.9 10.911 ms 100.92.49.125 10.806 ms

16 52.93.133.116 11.173 ms 52.93.132.60 14.273 ms 52.93.133.116 11.194 ms

17 100.91.163.80 11.414 ms 100.91.163.74 11.458 ms 100.91.163.72 11.468 ms

18 100.91.168.135 10.796 ms 100.91.168.63 10.831 ms 100.91.168.115 11.074 ms

19 100.91.159.88 11.258 ms 100.91.165.132 11.609 ms 100.91.164.40 16.991 ms

20 100.91.164.57 10.834 ms 100.91.159.83 11.487 ms 100.91.160.11 10.998 ms

21 100.91.177.137 16.589 ms 100.91.177.151 10.733 ms 100.91.177.175 10.596 ms

22 100.100.8.121 11.130 ms 100.100.6.119 11.144 ms 100.100.8.127 11.004 ms

23 100.100.90.72 10.962 ms 11.142 ms 100.100.88.200 10.897 ms

24 100.100.72.133 10.982 ms 100.100.80.3 11.305 ms 100.100.73.5 10.928 ms

25 100.100.4.4 11.282 ms 100.100.4.8 10.835 ms 100.100.4.10 11.324 ms

26 99.83.65.3 10.891 ms 99.82.181.25 10.991 ms 99.83.68.209 11.381 ms

27 108.170.240.112 10.930 ms \* 13.512 ms

28 142.250.232.78 10.909 ms \* 10.891 ms

29 209.85.252.46 17.487 ms \* 108.170.246.2 10.878 ms

30 108.170.232.199 11.837 ms 12.164 ms 209.85.254.95 16.859 ms

Load balancing? Multiple options?

**Why the different results ?** https://paris-traceroute.net/about/

main()

--- run() //sets up the environment

//creates objects

//before we run the tests

--- side\_channel.run() //last step of run

--- http\_server = gevent.server.StreamServer(...., handle()) //new client

-- traceroute

--- as soon as we run the tests, “new client...”

“Starting tcpdump….”

--- g.link(side\_channel.callback())

//create a copy of sidechannel

//for multiple clients

//g → current running thread

//g.link() runs the callback function after the thread finishes, the callback will have this instance as an argument

//after the test ends

// “side\_channel\_callback….”

**---- traceroute <- Initially traceroute was placed here**

// “stopping tcp dump”

**I ENDED UP ADDING THE TRACEROUTE BEFORE THE TESTS START. (CHECK**

**FOR “Running Traceroute!” IN LOGS.)**

**ADDING AT ABOVE MENTIONED POSITION CREATED A BUG THAT SET**

**SECONDARY SUCCESS TO FALSE. IT SHOULD ALWAYS BE TRUE.**

1. **Writing the subprocess**

(All the versions that failed)

subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

Command = [‘’’LD\_LIBRARY\_PATH="/usr/local/lib"‘’’, “paris-traceroute”,”-n”, dClient.id]

Traceroute = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

#Traceroute = subprocess.Popen(Command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, env={'LD\_LIBRARY\_PATH':'/usr/local/lib'})

#folder = resultsFolder + '/' + realID + '/tracerouteResults/'

#tracerouteFile = folder + 'tracerouteResults\_{}\_{}\_{}.json'.format(realID, historyCount, testID

#with open('tracerouteFile', a) as output :

#print(subprocess.check\_call(Command, shell=True, stdout=output,

env={'LD\_LIBRARY\_PATH':'/usr/local/lib'}))

Subprocess - <https://stackoverflow.com/questions/2502833/store-output-of-subprocess-popen-call-in-a-string>

Subprocess.pipe returns the output in a variable

Use subprocess.PIPE if you want to [get the output of the child process](https://stackoverflow.com/a/12606327/4279) (or pass input) as a string (variable) or just call subprocess.check\_output() that does it for you internally.

self.\_p = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

[communicate()](https://docs.python.org/3/library/subprocess.html#subprocess.Popen.communicate) returns a tuple (stdout\_data, stderr\_data). The data will be strings if streams were opened in text mode; otherwise, bytes.

log = open('some file.txt', 'a') # so that data written to it will be appended c = subprocess.Popen(['dir', '/p'], stdout=log, stderr=log, shell=True)

**Biggest takeaway here was to use pOpen instead of check\_output. Check\_output is a blocking call. We need it to be non blocking.**

1. **Saving the data**

Best format to store data?

Do the parameters for the command need to change?

Should the results be appended or time stamped?

1. **Discussion with Fan regarding analysis and the logistics - how is it done?**

Need to go through mail to dig this info. Can’t find it.

Achtung

home/fangfan/weheRsync/weheRsync.py

[https://console.cloud.google.com/storage/browser/archive-measurement-lab/wehe?project=measurement-lab](https://meet.google.com/linkredirect?authuser=0&dest=https%3A%2F%2Fconsole.cloud.google.com%2Fstorage%2Fbrowser%2Farchive-measurement-lab%2Fwehe%3Fproject%3Dmeasurement-lab)

Ec2 - amazon - wehersync -achtung

ec2-mlab-certain directory on achtung

Mlab future - mlab + achtung ---> <https://meet.google.com/linkredirect?authuser=0&dest=https%3A%2F%2Fconsole.cloud.google.com%2Fstorage%2Fbrowser%2Farchive-measurement-lab%2Fwehe%3Fproject%3Dmeasurement-lab>

1. **Miscellaneous - increasing size of ec2 - encountered low resources**

<https://medium.com/@m.yunan.helmy/increase-the-size-of-ebs-volume-in-your-ec2-instance-3859e4be6cb7#:~:text=Extend%20the%20partition%20by%20typing,show%2040GB%20of%20volume%20size>. :- increase size of volume

Xfs\_growfs instead of resizefs

1. **Miscellaneous - parsing**
2. **Docker basics**

Please just run this command. DO NOT SPEND TIME TRYING OTHER STUFF.

**sudo docker run -v /data:/data -v /home/ubuntu/wehe-py3/src:/wehe --env SUDO\_UID=$UID --net=host -it wehe 34.238.220.253**

**The test server is 34.238.220.253**

**All changes done to files will be transferred inside the docker using above command.**

Sudo docker images

sudo docker container ls -a

To remove image - sudo docker stop <id>

Sudo docker rm <id>

Sudo docker rmi <id>

1. **Run query for tridents**

<https://console.cloud.google.com/bigquery?project=measurement-lab&pli=1&j=bq:US:bquxjob_3677ce83_176730219de&page=queryresults>

WITH MAIN\_TABLE AS

(select TestTime, s.Source.IP as source, h.Source.IP,l.HopDstIP from `measurement-lab.aggregate.traceroute` as s, unnest(Hop) as h, unnest(h.Links) as l where Destination.IP="151.203.196.68" limit 50)

SELECT MAIN\_TABLE.HopDstIP, MAIN\_TABLE.TestTime, MAIN\_TABLE.source FROM MAIN\_TABLE ORDER BY MAIN\_TABLE.HopDstIP

\*\*may need changes\*\*. Sync up with Derek for that.

Any questions, please feel free to reach out to me at poorva.sonparote@gmail.com