Daniel Butts

CMI – Rogers Final Report

**Breakdown of work**

September 2020 – October 2020:

A large portion of my time was spent getting familiar with the Geo-Studies team’s problems and goals. I looked into the resources that were provided by Mitch to get familiar with how the ciphering of NAS messages are handled and how his team was attempting to decipher them. In particular, I read the pertinent sections of the 3GPP TS 33.401, and 3GPP TS 24.301 technical specifications specific to the NAS count.

As requested by Mitch, I did some minor investigations into trying to find more efficient methods to estimate the NAS count, as the Rogers team was currently brute forcing it, but nothing came from my investigation.

November 2020

During the month of November, I focused more on getting myself familiar with 4G architecture and the different procedures that are carried out over the network. I then begun to develop flow charts that may be used to help determine the state of the devices during these different procedures. In particular, I created flow charts that dealt with the [LTE Attach Flow](https://github.com/dan032/CMI_Rogers/blob/master/docs/flowcharts/4G%20Flowchart-LTE%20Attach%20Flow%20Chart.svg), the [Tracking Area Update Procedure](https://github.com/dan032/CMI_Rogers/blob/master/docs/flowcharts/4G%20Flowchart-Tracking%20Area%20Update%20Procedure.svg), the [S1 Release Procedure](https://github.com/dan032/CMI_Rogers/blob/master/docs/flowcharts/4G%20Flowchart-S1%20-%20Release%20Procedure.svg), along with a chart for the [Control and User Plane Connection IDs](https://github.com/dan032/CMI_Rogers/blob/master/docs/flowcharts/4G%20Flowchart-Control%20and%20User%20Plane%20Connection%20IDs.svg)

December 2020

Seeing as Rogers was unable to provide us with network data by this point, I begun familiarizing myself with Python libraries such as PyShark and Scapy so that if at any point in the future we gained access to Rogers’ data or if we found our own, I would be in a better position to utilize it.

January 2021

I began exploring the different fields that existed for the different protocols used during 4G communication to determine which fields may be of interest. Specifically, I focused on the ID’s which may be required to map network traffic to a particular device (such as the IMSI, MME IDs, eNodeB IDs, TEIDs, etc). I began to develop a [prototype](https://github.com/dan032/CMI_Rogers/tree/master/first_parser) that would be able to parse PCAP files that contains 4G data into CSV files so they would be better suited for machine learning tasks.

February 2021

Continuing the work from the previous month, I focused on implementing the mappings of tunnel identifiers and other IDs to a specific device in the [prototype](https://github.com/dan032/CMI_Rogers/tree/master/first_parser) that I was developing. By the end of February, most of the IDs were properly mapped, though I was still encountering some issues mapping certain TEIDs as it was not particularly clear where they were being set.

March 2021

At the beginning of March, Khaled tasked me with investigating the paper and code related to this repository: <https://github.com/a9khan/trafficclassifier>. After recreating the network specified in the paper, I had to debug the script as there were several logical errors in the code that prevented it from saving data to CSV files. However, the more that I delved into the paper and code the more that I realized that the findings from this paper were not relevant for our work. Since it was solely focused on predicting OpenFlow traffic, and their network setup only allowed the controller to see higher level flow statistics on the network, the script could not easily be modified to support lower-level features such as accessing individual packets.

April 2021

Seeing as the previous scripts from the traffic classifier repository were not of use to our project, I began looking for new libraries that may be able to help as analyze network traffic (not specific to 4G). I ended up finding a library called [NFStream](https://www.nfstream.org/docs/api), which seems like a promising way to explore and analyze network flows while also providing greater detail about those flows than the scripts from the previous month (while allowing us to set up our network in such a manner that would not prevent us from directly accessing packet information). I also set up a Cron job so that gathered network traffic is constantly collected, and once an hour it is saved into a PCAP file and parsed into a CSV file by [NFStream](https://www.nfstream.org/docs/api). A benefit to using this library over other parsers that I have produced or used this year is its speed, it takes approximately one second to produce the CSV file for the data captured over an hour.

I also investigated the tool: D-ITG (Distributed Internet Traffic Generator) as a method for replicating network traffic. It was fairly straight forward to setup and allowed my virtual machines to produce different types of traffic (i.e. VoIP, select number of games, DNS, etc).

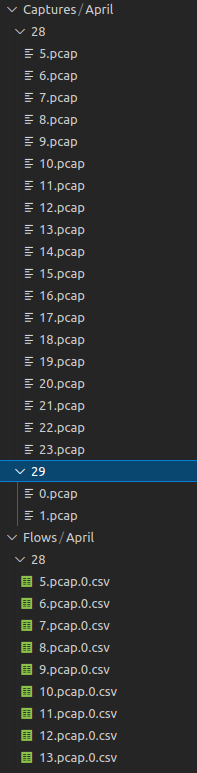


Figure File structure after running the script for a day on my host VM

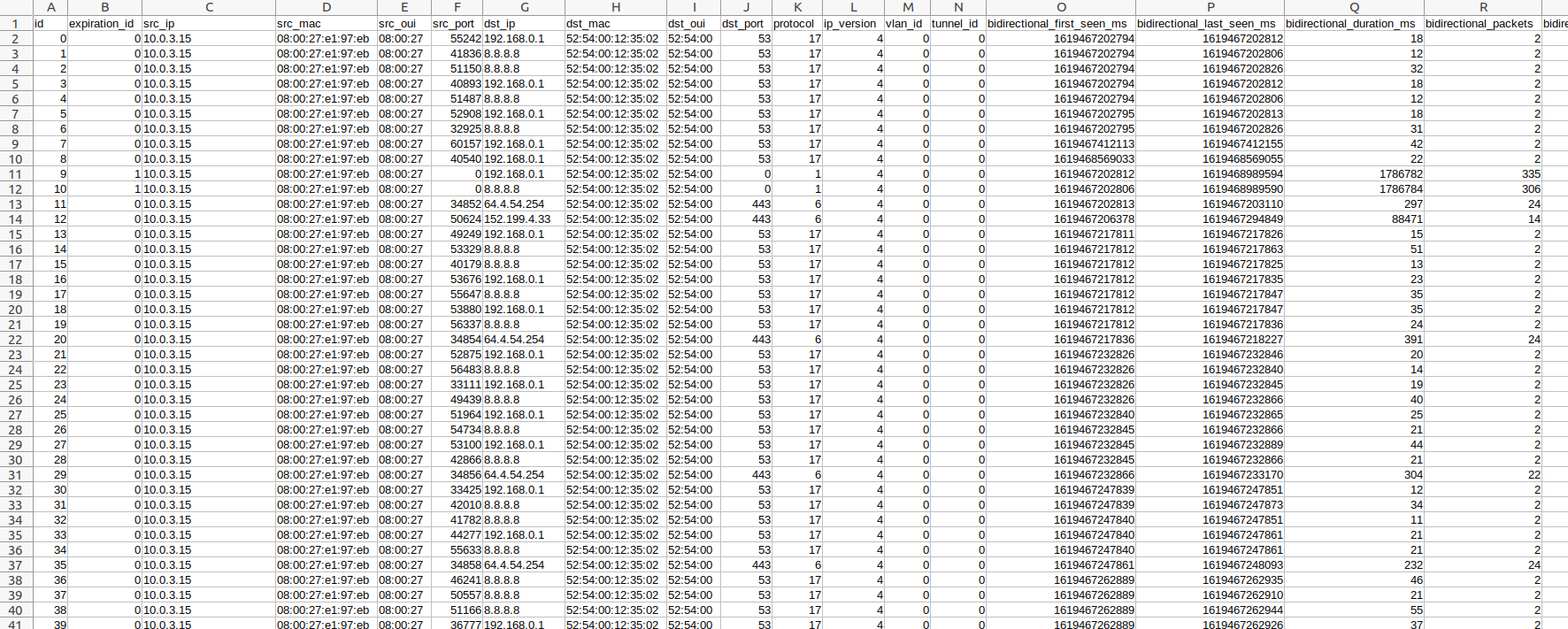


Figure Typical layout of a flow CSV.

Environment Setup

I am currently using 3 Virtual Machines, 2 hosts to produce and receive network traffic, and 1 VM acting as a switch.

Specifications for each VM (note these are excessive, you do not need as many resources allocated for each VM):

1. OS: Ubuntu 20.10
2. Each VM has 8GB of RAM
3. 1 CPU
4. 2 Network Adapters:
   1. NAT Adaptor
   2. Internal Network

If you are using Windows, ensure that you disable Hyper-V.

This is what my network is currently looking like, to avoid needing to set up the network every time you turn your VM’s on, ensure that you update your netplan accordingly (example shown in figure below). Initially I had used Open vSwitch to forward flows to a controller as specified by the traffic classifier paper, but it was determined to be unnecessary at this time.

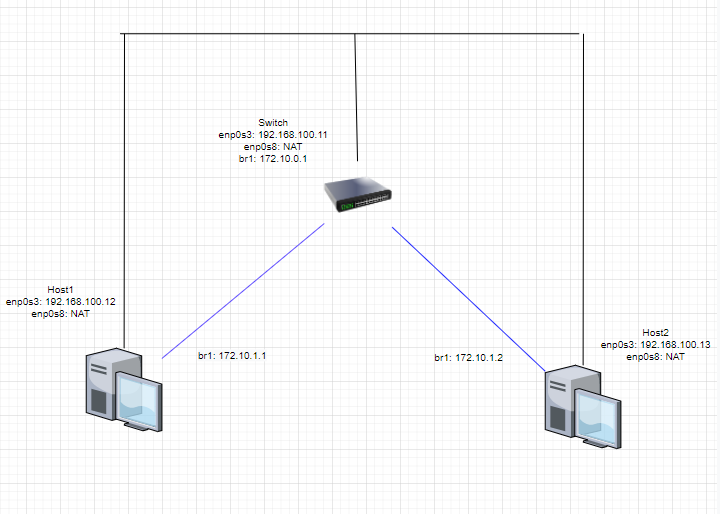


Figure Network Topology

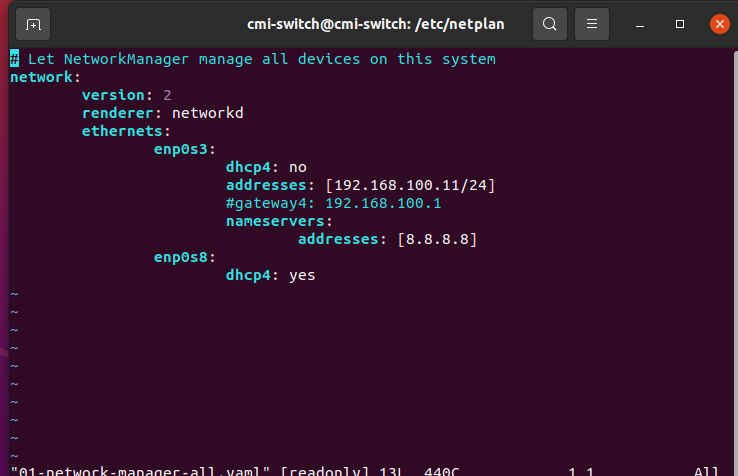


Figure 4 Example netplan for Switch

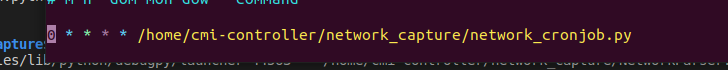
Open vSwitch Installation

1. Install Open vSwitch on all 3 machines
2. To set up the VXLANs perform the following:
   1. Run *ovs-vsctl add-br br1* on all 3 VMs
   2. Perform *ifconfig br1 [172.10.x.x]* on all 3 VMs
   3. On the switch run:
      1. ovs-vsctl add-port br1 vx1 -- set interface vx1 type=vxlan options:remote\_ip=192.168.100.12
      2. ovs-vsctl add-port br1 vx2 -- set interface vx1 type=vxlan options:remote\_ip=192.168.100.13
   4. On both hosts run:
      1. ovs-vsctl add-port br1 vx1 -- set interface vx1 type=vxlan options:remote\_ip=192.168.100.11
3. To confirm that the vxlan’s are setup, try pinging one host from another using the VXLAN IP (172.10.x.x)

After you confirm that the network is working properly, you need to install D-ITG from this link <http://traffic.comics.unina.it/software/ITG/>.

After installation, you will need to set up a receiver on one host, and a sender on another. To do so, move to your bin directory within the D-ITG directory, and run the command “./ITGRecv” on the receiver host. To send traffic from your sending host, you will need to do something similar, but the command will differ depending on the type of traffic that you want to send. For instance, to send DNS traffic, you could use the command “./ITGSend DNS -a 172.10.1.2 -t 30000” Refer to the documentation for more information.

Setting up the network gathering cronjob is straight forward. After ensuring that you have installed all necessary python libraries (in this case, you should only need to install NFStream and PyShark – which will require TShark) and downloaded the appropriate modules from my [GitHub](https://github.com/dan032/CMI_Rogers/tree/master/hourly_cronjob), type “sudo crontab -e” and then add this one line at the bottom of the file (location of python file will depend on where you saved it):



Note: You will need to change the current directory specified within the network\_cronjob.py file on line 11.