6-11-2019

Top Level Mesh

Matt Jasaitis

Tristen Islam

Recommended Citation

Jasaitis, Matt and Islam, Tristen, "Top Level Mesh" (2019). *Computer Engineering Senior Theses*. 154.
https://scholarcommons.scu.edu/cseng_senior/154

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 12, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Matt Jasaitis
Tristen Islam

ENTITLED

Top Level Mesh

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

B. Dezfouli
Thesis Advisor

Department Chair

Top Level Mesh

by

Matt Jasaitis
Tristen Islam

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 11, 2019

Top Level Mesh

Matt Jasaitis
Tristen Islam

Department of Computer Engineering
Santa Clara University
June 11, 2019

ABSTRACT

This paper will cover our report on the Top Level Mesh. We have built a web-based system for mesh network management. This system allows network utilities to be used from a web-based interface to monitor and manage the transfer of data. The system runs primarily on Raspberry Pis using Raspbian Linux. Users can access the system through web browsers to both configure the system and interact with the data on the network. We discuss our motivation for the project, design decisions made, technologies used and more throughout this report. We conclude with some lessons learned and future work to be done.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In todays world, access to information is more important than ever for timely decision-making. For example, networked medical devices need to send large amounts of data to let doctors know their patients' status, firefighters need to get satellite images of brush fires, and foreign aid workers need to get information on food supplies. All of these tasks are relatively simple when there is an an established wired or wireless infrastructure that can connect people to the applications they need. However, in many places in the world, a wireless network is not available. For example, there may be medical camps in remote areas that have to transfer data between themselves to the Internet. In addition, a hospital may have a large number of sensors to monitor a patient's condition that require guaranteed access. All of these scenarios require a robust infrastructure to manage the sending and receiving of data. We are trying to create a high performance wireless network that supports the delivery of high-priority traffic and is easy to deploy in any situation.

Currently, existing mesh software packages do not provide easy to deploy solutions that meet all the needs of ad hoc users such as emergency responders. For example, GoTenna provides a piece of hardware that connects to phones and enables data to be relayed between other phones with the same device. While this shows potential, it cannot be used to build large mesh networks (1). Another contender is Althea, a company which offers paid access to the Internet. However, it does not solve many of the problems that we are attempting to address with our bandwidth-aware routing. Althea has a payment system that is focused on setting up ISPs run by the community. It creates nodes in the mesh network and makes participation profitable for them (2). In short, both of these technologies, while similar, do not fully solve the issue of managing a complex network in the absence of infrastructure. Discussing every mesh product would take too long, but suffice it to say that solutions involving closed hardware, such as Google and Orbi, have their limitations and cannot accurately attune themselves to different environments. The software that comes the closest to what we are trying to accomplish is OpenWISP. However, this company lacks features that exploit the fact that mesh nodes are smart and have both edge and core functionality. OpenWISP offers some functionality to do edge tasks -

like quickly set up VPNs - but it does not offer the traffic management features that we want (3). One monitoring package that comes close to what we are attempting to do is Gotham - a visualization tool for batman-adv. This tool uses Hazelcast to help users visualize network topology. It also allows nodes to connect to the mesh automatically. While these features are useful, Gotham is built for one specific protocol (batman-adv.) and does not exploit all of the edge features of mesh architecture (4).

1.2 Solution

We have built a web-based system for the setup and configuration of wireless networks. Through a variety of techniques, our system provides a robust method of merging multiple wireless nodes into one network. The network is also able to manage up-and-down traffic to ensure priority data gets through. Additionally, the system provides near real-time data and graphics on the state of the network. This system differentiates itself from existing solutions by providing an integrated view of the network. We have developed software to live on the mesh and more efficiently move packets. From the web-based interface, a user is able to configure various properties of the mesh, as well as set various nodes as access points. The mesh network runs primarily on inexpensive hardware that is versatile. The nodes require minimal setup to be connected to the network and the management software. The initial connection to the main server where the web application is hosted could be through Ethernet or a wireless protocol that is pre-configured at start-up. The software is able to assign gateway nodes as well as configure pieces of the mesh that are specific for each protocol. Most of the protocols have routing algorithms built in, but this software is able to prioritize traffic.

We have implemented simple machine learning into the system for smarter traffic optimization. The web interface allows users to see which nodes are using a disproportionate amount of bandwidth and react accordingly.

In conclusion, our solution allows for the faster deployment and improved performance of wireless mesh networks. With this software, users have an easier time managing the complex protocols needed to set up a network and can do so from a unified web-based interface. The software also allows users to more readily diagnose and resolve problems in their network using performance metrics on the web interface.

Chapter 2

System Design

Functional requirements, nonfunctional requirements, and constraints are specified for the platform. Each requirement is categorized into ’Critical’, ’Recommended’, and ’Suggested’ to specify how essential each component is to the platform.

2.1 Functional

2.1.1 Critical

- Web-based manager must allow users to configure how traffic is distributed
- Web-based manager must facilitate file transfer of important files and mission critical data
- System must display metrics to users

2.1.2 Recommended

- Implementation of machine learning to improve traffic distribution
- Keep traffic distribution equitable
- Device security

2.1.3 Suggested

- Implementation of traffic storing and forwarding
- Swapping of base routing protocols
- Base security features

2.2 Non-Functional

2.2.1 Critical

- Web interface must look clear and intuitive

2.2.2 Recommended

- Minimal bugs

2.2.3 Suggested

- Configurations will be pushed out quickly

2.3 Constraints

- Must run on web browsers
- Must work from within the mesh
- Mesh must run on Raspberry Pis with Alfa wireless cards
- Web servers must be accessible through WiFi

2.4 Use Cases

Use cases for the two stakeholder groups - namely, network users and network administrators - are detailed below. The network administrator has all of the functionality for managing traffic, while the clients have the ability to access the rest of the network as well as push form data up.

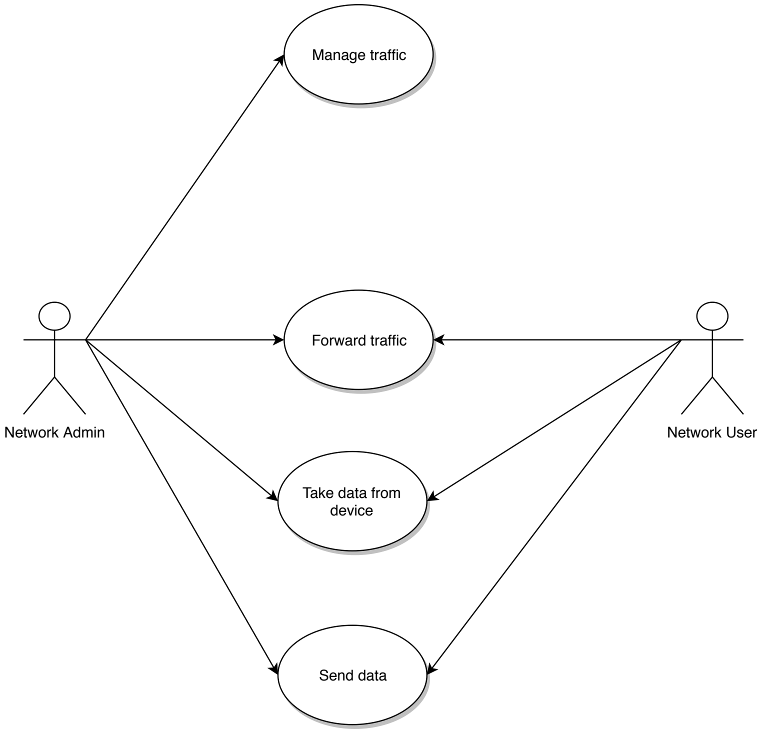


Figure 2.1: Use Cases for Internal & External Users

2.4.1 Forward Traffic

All mesh nodes can transfer traffic between each other. This will be a default action performed without user intervention. Traffic is forwarded through the 802.11s protocol and is handled by the Alfa WiFi cards on the Raspberry Pis.

Pre-conditions: The network is set up and nodes are connected to each other on the same ESSID.

Post-conditions: Traffic will be transferred between the nodes.

2.4.2 Manage Traffic

Node administrators are able to modify how the devices behave in a networked setting through the web interface. This involves choosing which IP addresses to prioritize and which ones to throttle. Administrators also have information on the current state of various devices. The throttling action is managed by the server that runs on the nodes. It is performed with the traffic control Linux utility.

Pre-conditions: The network is set up and nodes are connected to each other on the same ESSID.

Post-conditions: Nodes can prioritize different ports for the transmission and receiving of traffic. IP addresses can be blocked to improve performance if necessary. In short, basic networking commands are specified by the administrator and deployed to the network.

2.4.3 Take Data from Device

All users have access to the files on their current node and other nodes that they can contact. They can view these files to gain insight on how their network is behaving. The collection of data is managed through API endpoints on the nodes and is handled through the express utility. Most of the files that users can access with this action pertain to files containing various statistics or data on the network.

Pre-conditions: The network device is active and has a working WiFi connection.

Post-conditions: Users can view files on their device.

2.4.4 Send Network Data

Nodes have a ”chron” task setup to send data to each other intermittently. This gives every node an up-to-date record of what the network is doing. The chron tasks are primarily run through the HTTP protocol.

Pre-conditions: The network device is active and has a working WiFi connection.

Post-conditions: The form data is stored in a file and pushed to the admin system when applicable.

2.5 Activity Diagram

The network manager is able to send files to and get information from clients.

2.5.1 Mesh Manager

Figure 4.1 highlights the network manager's ability to manage traffic and get information from devices. The web manager can be run on almost any device and has the ability to view many important characteristics of connected nodes. The manager itself is run through a simple web user interface that allows users to easily navigate the various pages. Network managers have three main functions:

1. They can view information on connected mesh nodes.
2. They can edit traffic configuration rules.
3. They can push traffic configuration rules to the rest of the nodes.

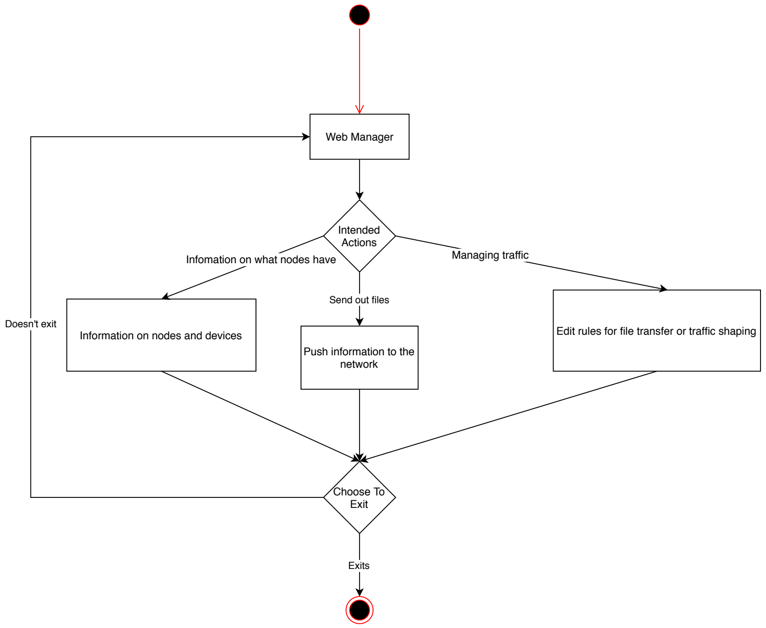

Web Manager
Intended Actions
Infomation on what nodes have
Send out files
Managing traffic
Doesn't exit
Information on nodes and devices
Push information to the network
Edit rules for file transfer or traffic shaping
Choose To Exit
Exits

Figure 2.2: Manager Activity Diagram

2.5.2 Mesh User

Users are able to access locally stored resources and make network requests. Users on the mesh can also access the wider Internet through gateway nodes if they are available. That being said, the system may be configured in ways that do not permit Internet access, so it is uncertain whether they will succeed. Users will also be able to take data from the nodes themselves through their API endpoints.

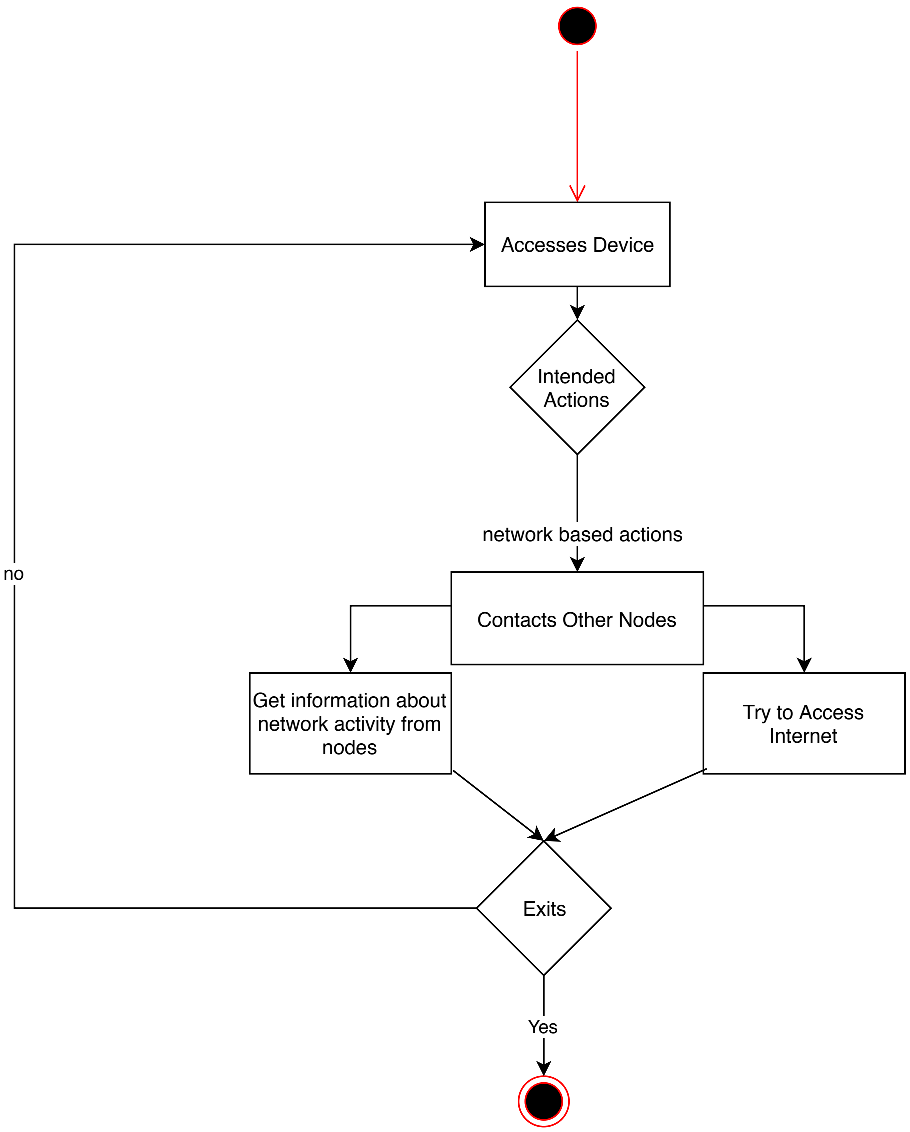


Figure 2.3: User Activity Diagram

2.6 High Level View Of System

Below, you can see the various views that we have in our frontend web application. These views include the main network status page, the network data page, the traffic control page and more.

2.6.1 Network Status

From this page, administrators can view network status. There is a visual display of the status of the nodes, as well as the traffic moving between them. The nodes are labeled with their IP addresses and the edges are dynamically configured based on data from the network. Elements on this page can be clicked on to get more detailed information on each node. The cloud at the top of this view represents the Internet.

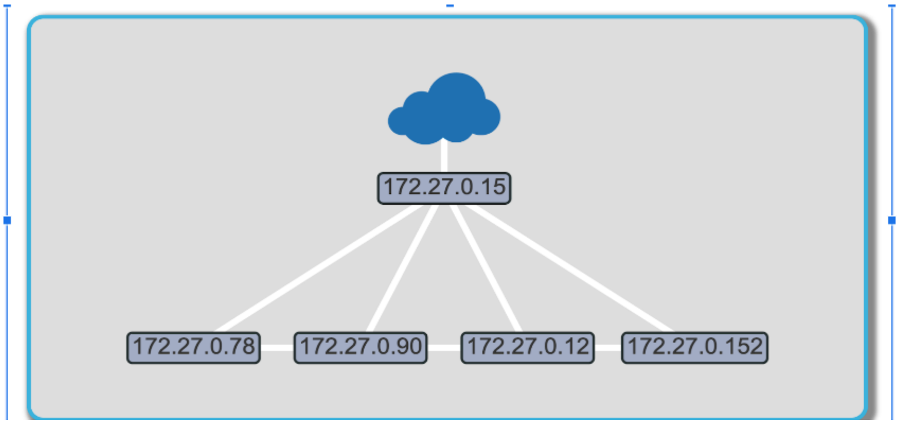


Figure 2.4: Network Status Mock-up Diagram

2.6.2 Network Data

The network status web page (based on the data collected from our software) provides a connectivity map of the network. From this page, users can see the traffic that each node sent and get an idea how the bandwidth is being used. This web page can give a large amount of relevant information, such as how much traffic each node has sent, as well as what protocols the nodes are using. Since protocols can give an indication of what sort of services are being utilized, this is valuable information.

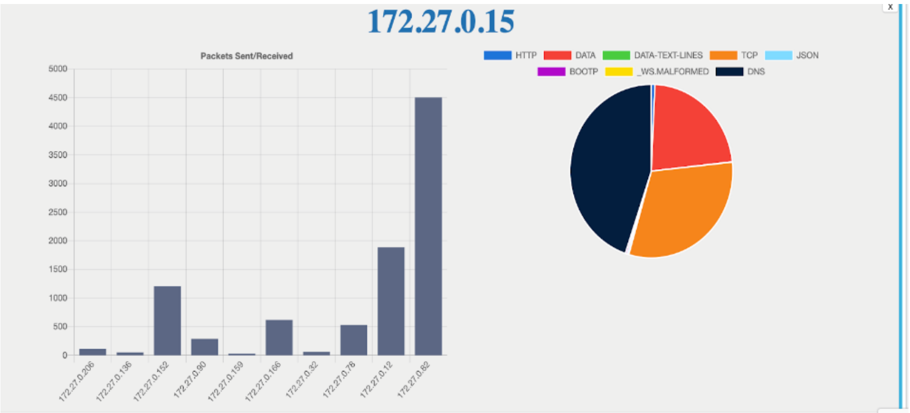
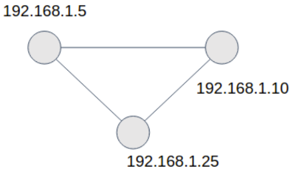


Figure 2.5: Network Status Mockup Diagram

2.6.3 ARP Table Graph

This is a graphic that appears when the individual nodes are queried directly from the API as opposed to viewed via the frontend. It shows how nodes are connected and gives users a cohesive view network topology from the perspective of that node. Many errors in networks (especially mesh networks) relate to ARP tables. If a node cannot find another node, its ARP table will likely reflect that. In an ideal scenario, every node can at least see the gateway node.



Name	IP Address	MAC Address
Mesh 1 (MPP)	192.168.1.1	00:24:73:1E:D7:BD
Mesh 2 (MAP)	192.168.1.16	00:24:73:1E:EB:3F
Mesh 3 (MAP)	192.168.1.5	00:24:73:1E:D6:6D
Mesh 4 (MAP)	192.168.1.10	00:24:73:1E:D7:4D
Mobile 1 (Station)	192.168.1.30	74:F0:6D:32:1B:1F
Mobile 2 (Station)	192.168.1.25	74:F0:6D:31:23:A6

from 2019-06-02 14-59-34.png

Figure 2.6: Network Status Mockup Diagram

2.6.4 Traffic Control

From this page, the administrators can prioritize IP addresses. Functionally, this system will call utilities like traffic control on the backend of the system. Once the request gets to one node, it is propagated to other nodes in the network. The goal of our system is that users could use this page to determine which nodes are misbehaving and then use the priorization function to throttle or help those nodes.

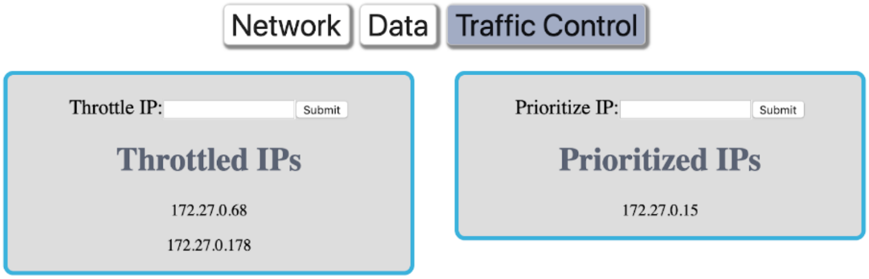
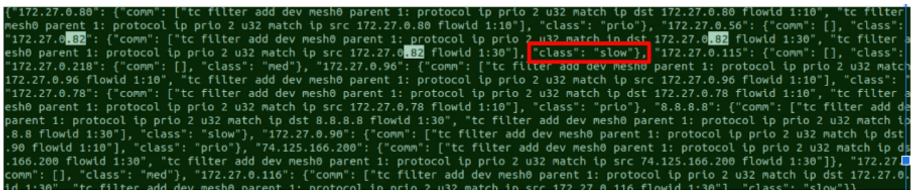


Figure 2.7: Traffic control

2.6.5 Auto Shaping

Our system can automatically shape traffic based on certain heuristics to keep the network running smoothly. This is done through data that we obtained experimentally, as well as with some basic machine learning.



```
{"172.27.0.80": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip dst 172.27.0.80 flowid 1:10", "tc filter
mesh0 parent 1: protocol ip prio 2 u32 match ip src 172.27.0.80 flowid 1:10"], "class": "prio"}, "172.27.0.56": {"comm": [], "class":
"172.27.0.82": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip dst 172.27.0.82 flowid 1:30", "tc filter a
esh0 parent 1: protocol ip prio 2 u32 match ip src 172.27.0.82 flowid 1:30"], "class": "slow"}, "172.27.0.115": {"comm": [], "class":
"172.27.0.218": {"comm": [], "class": "med"}, "172.27.0.96": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match
172.27.0.96 flowid 1:10", "tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip src 172.27.0.96 flowid 1:10"], "class": "
"172.27.0.78": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip dst 172.27.0.78 flowid 1:10", "tc filter a
esh0 parent 1: protocol ip prio 2 u32 match ip src 172.27.0.78 flowid 1:10"], "class": "prio"}, "8.8.8.8": {"comm": ["tc filter add de
parent 1: protocol ip prio 2 u32 match ip dst 8.8.8.8 flowid 1:30", "tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip
.8.8 flowid 1:30"], "class": "slow"}, "172.27.0.90": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip dst
.90 flowid 1:10"], "class": "prio"}, "74.125.166.200": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip ds
.166.200 flowid 1:30", "tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip src 74.125.166.200 flowid 1:30"]}, "172.27.
comm": [], "class": "med"}, "172.27.0.116": {"comm": ["tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip dst 172.27.0.
id 1:30", "tc filter add dev mesh0 parent 1: protocol ip prio 2 u32 match ip src 172.27.0.116 flowid 1:30"], "class": "slow"}}
```

Figure 2.8: Traffic control

2.7 Technologies

To create this platform, we chose the following technologies for their effectiveness, practicality and ease of integration.

- 802.11s

  This is our main routing system, though we utilize other wireless standards in the future. Research has shown that 802.11s is a reliable protocol and it serves all of our needs. Another advantage of this protocol is that it is supported by many Linux distributions (for our purposes, it works with Raspbian) and gives mesh nodes access to the full protocol stack with relative ease. (5)

- Raspberry Pis

  Raspberry Pis are our main network backbone. They can do many tasks by installing new software and are relatively easy to configure. Their ubiquitous nature has allowed them to become a staple of many fields of computing in recent years. These devices are very adaptable and there is a large amount of online information that helps with problem-solving.

- Alfa WiFi Cards

  Alfa WiFi cards can handle a wide range of protocols and are a very reliable choice for networking. They can run 802.11s, as well as batman-adv which makes switching protocols possible if needed.

- Raspbian

  This is the operating system on the Raspberry Pis. It is Debian-based and thus easy to work with and to configure. The wide array of support that this distribution has received makes it an appealing choice, as most issues can be easily solved through a simple web search.

- Ansible

  It works through SSH and it is used to configure the nodes. This is more of an internal tool rather than a tool for deployment, but it is vital for testing and configuring the nodes quickly.

- Node.js

  This is the backend JavaScript run-time environment for our server. Node.js's ability to combine client and server technologies helps us set up apps and features more quickly. It also gives us access to a more extensive library of software in the event that we need to add supplemental functionality.

- Angular

  This is the frontend framework for our user interfaces. It works well with Node.js and is fairly easy to use. Angular provides a wide variety of JavaScript packages that allow our application to be easily extended.

- Traffic Control Linux utility (TC)

  We use the TC utility for shaping traffic on the nodes. This utility is flexible and provides many different ways to shape traffic. It is particularly useful because it can split traffic from different IP addresses and ports into groups called qdiscs and then manage those qdiscs individually.

- Python

  This is the language that we use for data analysis on the network. Its wide variety of functionality is very helpful. Python also has the pip (package manager) utility that makes the installation of packages very simple. Python also has many utilities for data science and traffic capturing that makes setting up the system much easier. Notable utilities are PyShark for capturing traffic, NetInterfaces for dynamically configuring interfaces, Python ARP tables for getting a system's ARP table and scikit-learn.

- scikit-learn

  This is used for machine learning on traffic data. Most of the learning is done on data extracted from PCAP files. This utility has the ability to quickly make decision trees on data and can be used to determine what sort of entities the users want to throttle in a network.

2.8 Component State Diagrams

2.8.1 Setup of Individual Node

This is the state flow for an individual node. The node boots up and then receives its configuration from the main node. The main feature in this is that it will either be a gateway or not be a gateway. If a gateway node's external Internet goes down, then it becomes a non-gateway. To change its configuration, it must reboot.

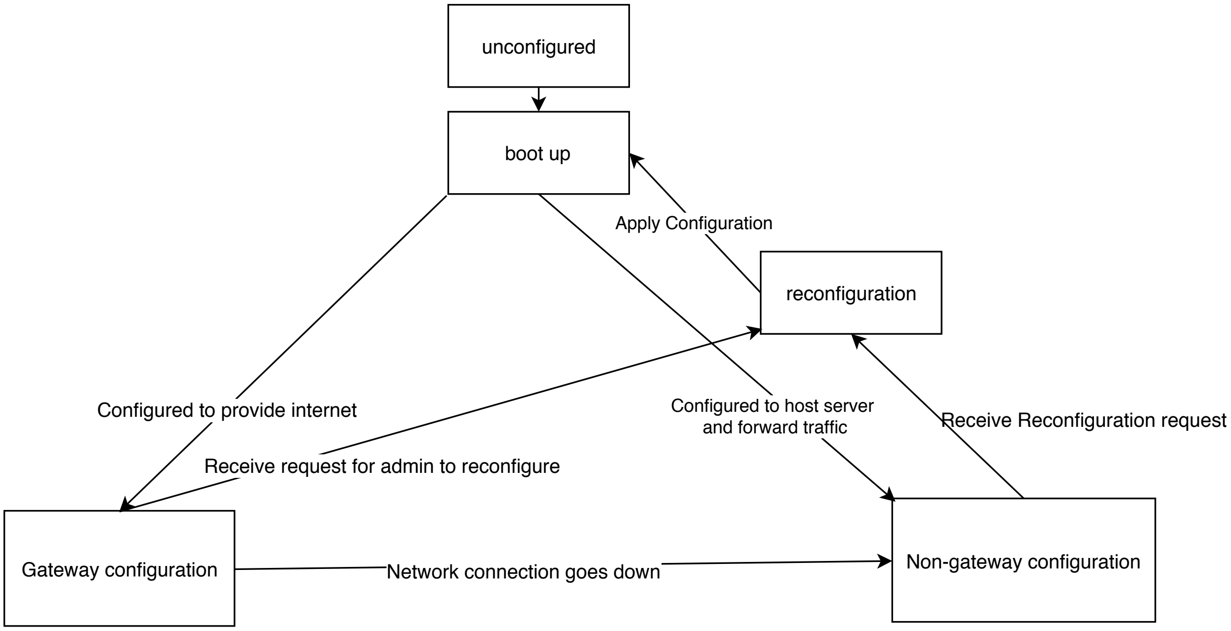

unconfigured
boot up
Apply Configuration
reconfiguration
Configured to provide internet
Configured to host server
and forward traffic
Receive Reconfiguration request
Receive request for admin to reconfigure
Gateway configuration
Network connection goes down
Non-gateway configuration

Figure 2.9: State diagram of node

2.8.2 Admin State Flow

The first step of the admin console is that it boots up. From there, it tries to ping other nodes. The console then obtains information on other nodes and sets up the console for configuring them. In the event of a network outage, it tries to ping the nodes again.

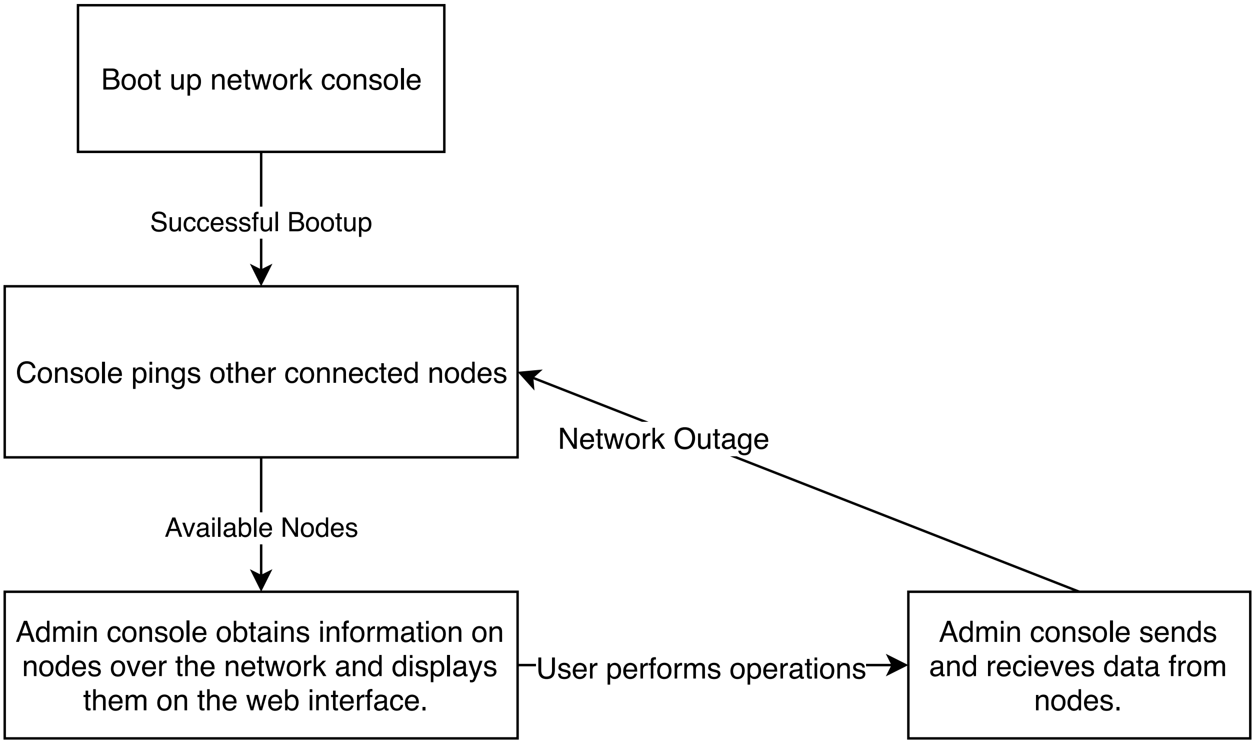

Boot up network console
Successful Bootup
Console pings other connected nodes
Network Outage
Available Nodes
Admin console obtains information on nodes over the network and displays them on the web interface.
User performs operations
Admin console sends and recieves data from nodes.

Figure 2.10: State diagram of admin console

Chapter 3

Implementation

3.1 Architectural Diagram

At a high level, our system is about collecting and managing data. We need to manage users' access to the Internet. We also need to manage the data that individual nodes collect and aggregate it. To that end, our architecture is somewhat data-centric. Below is a diagram showing our solution's architecture. As stated earlier, we have three kinds of nodes: gateway nodes, regular nodes and access points. Nodes are able to send data between themselves via our Node.js. Nodes store their own data locally on files that can be accessed and transferred later through APIs. Users on their devices are able to send and receive data on their network when connected to an access point node. Additionally, there is the admin controller (the laptop in this diagram), which provides access to the web interface. The admin controller receives data about other nodes and also has the ability to control and configure these nodes remotely. It is worth noting that, while the core nodes require mesh capabilities, clients connected to the system and the admin controller do not.

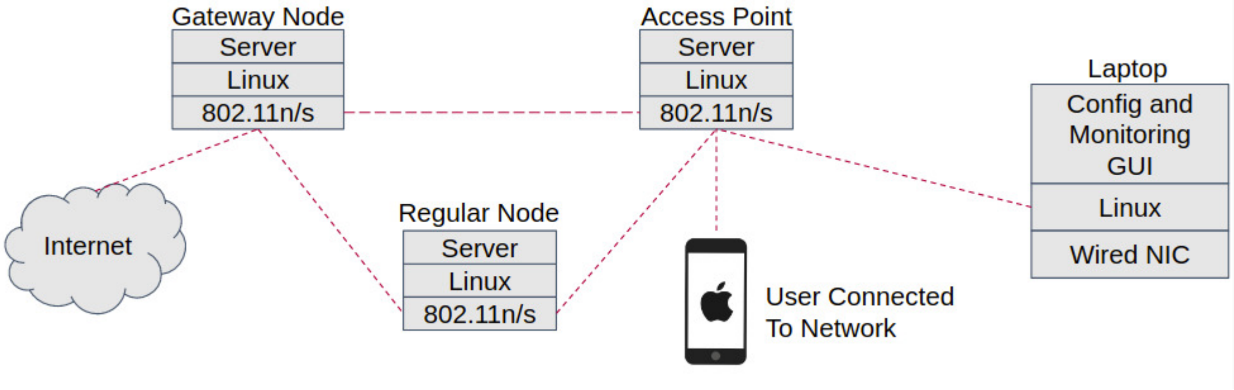


Figure 3.1: Architecture of Platform

3.2 Design Rationale

3.2.1 Technologies

We used 802.11s because it is a reliable protocol for getting things up and running. Studies show that 802.11s is more reliable than other protocols such as batman-adv. Thus its usage makes sense for this project. Additionally, it takes care of many of the simple routing operations and allows us to work at higher layers of the network stack.

The Raspberry Pi is a versatile piece of hardware. Our group also made use of OpenWrt systems for some tasks, but our main backbone is constructed with Raspberry Pis. This is due to Raspberry Pis having a wide array of software that they can handle almost any task. With a simple ”apt-get” command, they can quickly and reliably install needed software. This allows us to readily configure all layers of the network stack, as well as make use of tools like traffic control.

Alfa wireless cards are our main method of mesh access (though this is subject to change). Alfa wireless cards are reliable and can handle a wide variety of protocols.

Node.js allows JavaScript to be run on the backend and works well with Angular on the frontend. The diverse library of Node.js modules makes setting up our web server more convenient. The flexibility of Node.js also compliments the flexibility of our hardware and network architecture. By using Node.js, our system is future-proofed, and it is more adaptable to change based on a client’s needs.

Python has so many different libraries - everything from data analysis to traffic control - that we use it for several tasks in our project. Python is used to parse network data and to quickly set up machine learning models for the system. Scikit-learn is a more lightweight tool that can set up simple machine learning models. It allows us to effectively parse traffic and generate meaningful data without worrying about whether we implemented the algorithms correctly. This allows us to gather insights about network traffic.

Another advantage of Python is the ease with which it can interact with a variety of data formats and structures. JSON is a data format that is natively built into Python, so we did not need additional packages for it. Since JSON is ubiquitous to both Python and Node.js, it made sense to choose it.

3.2.2 Layout

We have a number of pages - including a network management page, a current state of network page and various data entry pages. The design is simple and lightweight, and it provides an intuitive graphical layout of the network’s status. We chose to emphasize simplicity over complexity in order to ensure a network admin can see valuable information without wasting time.

3.3 Description of System Implemented

3.3.1 Nodes

Nodes have a large number of features and are the backbone of our network. Every node has a small Node.js server running on it that handles API requests, as well as managing the data transfer between the nodes, running Python scripts and reading/writing to files.

- Traffic

  Nodes communicate traffic logs through API endpoints setup by Express.js, a package for Node.js. From there, they can parse received data, add to it with their own traffic that they have forwarded, or execute traffic shaping scripts.

- Shaping of Traffic

  Nodes shape traffic through the Linux program ”TC”. This utility has a wide array of functions that allow for the shaping and management of traffic.

- Data Exchange

  Nodes exchange data intermittently so that each node can have an up-to-date record of the traffic being sent through the network.

- Data

  Nodes store data in files on the server. The data itself is in JSON format. We did not see a need for the size and overhead of a central database.

3.3.2 Frontend

Unlike the nodes, the frontend for our system can be retrieved on any device that is connected to the network. The frontend queries the gateway node for data which contains metrics on the network as a whole. It can also send traffic shaping commands in the form of IP addresses that should be throttled or prioritized. The frontend is built with Angular which is a JavaScript framework built on top of Node.js.

- Graphics

  The frontend can take traffic data from the gateway and display it as a graphs that show network admins how their network is being used.

- Node Graph

  The frontend shows how nodes are connected in a graph representation. This allows for admins to view their networks in a very intuitive way.

- Traffic shaping

  The frontend can shape traffic and control what sort of packets are prioritized or slowed.

3.4 Node Connectivity

Nodes communicate via the HTTP protocol. They exchange data such as the traffic that they have forwarded. Each node has a server implementing an API that other nodes can query that allows traffic data to be exchanged.

Chron tasks are set up to run various Python scripts. These scripts do a number of things such as:

- Getting traffic from other nodes
- Querying the ARP tables of other nodes
- Generating a local graph of traffic on their system that can be queried in the event that there is no frontend that can be accessed
- Generating a graph of their ARP tables
- Querying the traffic control directions of other nodes

3.5 Networking Protocols

- Hostapd: Hostapd is the software that we use to turn the Raspberry Pis into WiFi access points.
- DNSMASQ: DNSMASQ is software that we use to run a DHCP server. Typically we have one DHCP server running on a gateway.
- Bridge-Utilities: We use bridge utilities to bridge the WiFi network into the mesh network. This allows for every node to be on one IP address range.

3.6 Traffic data

Traffic data is stored in a JSON object and we chose this approach to save storage space. By parsing the data and storing the parsed result - as opposed to storing the entire capture in a PCAP file - we save on storage by only storing the relevant pieces. The keys of this dictionary are IP addresses of mesh nodes. The keys then map to a nested dictionary that contains more information such as:

- Destination

  The dictionary stores the destinations that a node sent traffic to, as well as the number of packets that it sent to that destination. This is another useful statistic to tell network administrators how their bandwidth is being used.

- Time

  The time in which a particular packet was sent is also stored in the destination field of the dictionary. This is primarily to give users an idea of when traffic was sent, but in our current implementation it is not heavily used.

- Route

  This is a field in the data that essentially shows which node routed a certain node's traffic from source to destination. This helps network operators differentiate between core and edge nodes. This statistic is also used in the route count statistic.

- Protocol Dictionary

  The protocol dictionary is a dictionary of the protocols that a particular node sent and how many times it was used. This information gives insight into who is behind the node. This can also indicate whether encrypted protocols are being used. SSL and HTTPS indicate that a user is safely encrypting traffic.

- Length

  This is the average length of packets and is useful for determining what sort of website a user is visiting. A larger average length implies large data transfers.

- Count

  This is the count of packets that a particular node has sent.

- Class

  This is the type of node that the traffic recording script determines it is looking at.

3.7 ARP Data

ARP data is stored in a key-value dictionary format where the keys are the nodes and the value list is the IP addresses that the nodes can see. This is a much simpler structure than the traffic data but that is because less information is relevant. All that is needed for the graph is the interconnection of nodes so that is what is stored. This data can be parsed into a directed graph and displayed via the API or the frontend application.

3.8 Traffic Control Result

This data is a similar format to the ARP graphs with a key-value format. However, in this case, the values are the result of the forwarding operation. The main purpose of this data is to allow network operators to verify that nodes are making the correct assumptions about traffic shaping.

3.9 Algorithms and Techniques

3.9.1 Decision Trees

We use decision trees with the Python utility scikit-learn to do machine learning on packet data. The decision trees look at various factors of the packets such as what nodes they are connected to and how much traffic they have sent. From there, they can make decisions to determine how to throttle or prioritize nodes. The system can also take input from the admin, via the frontend application, to throttle and control traffic.

3.9.2 Statistics and Routing

One of the most important statistics that we use for shaping traffic is the route count. The route count is the number of nodes that a particular node routes traffic for. This statistic, more than any of the others, allows us to determine which nodes are necessary to the network and then throttle traffic accordingly. This is relevant for decisions such as making sure central nodes are able to send traffic and communicate.

Another statistic that we track is the average length of packets. This allows us to see the kind of traffic that a node is sending. This, along with packet count, helps us identify the biggest users of bandwidth. For example, this statistic helps us identify IPs that may be streaming videos or downloading large files so that we can them to ensure important traffic can get through the network.

3.9.3 Classification

Our system can classify nodes based on their behavior and what sort of traffic they send. The main classifications are as follows.

- Node

  A basic node that is forwarding traffic.

- Bridge

  An access point node that bridges regular nodes and user devices into the mesh.

- Gateway

  A node that connects to the larger internet.

What is tricky about node classification is that our system only classifies nodes by the traffic that it sees and its ARP table. This is because we wanted to make a system that does networking from as high of a level as possible.

To that end, we can use the route count statistic stated in the previous section to classify nodes. The route count allows the system to determine which nodes are forwarding traffic. As a general rule, nodes that are forwarding traffic through our mesh are important to the network and should not be throttled extensively. Additionally, nodes that are forwarding traffic are likely either gateways or bridges. The main focus of throttling is to ensure we only throttle exterior nodes that are using large amounts of data and not throttling our interior nodes.

Another statistic that is looked at for classification is a node's ARP table. Nodes that can ”see” more and interact with more nodes are central to the network. From this data, we can determine whether a node is a gateway, bridge or a user device. Thus, the number of ARP nodes is another statistic for node classification.

3.9.4 Heuristics

In addition to the machine learning, we use some heuristics to shape traffic. The simplest of these heuristics is checking whether a node's packet count is past a certain threshold, but there are other important statistics. Among these is the protocol count of a node - or the number of different protocols that it has. This shows the diversity of traffic that a node is sending.

3.10 Frontend Information

3.10.1 Sent Traffic

The frontend displays collected data to users. When selecting a node from our home page, seen in previous sections, the user can see how much traffic is sent by a node, where the traffic is going and which protocols are used. We put this data easily accessible because this is the most important and useful data to an admin. The network traffic is displayed in a bar chart seen in figure 15.1. For the sake of an example, if the network admin saw the large disparity of data being sent by the IP ending in .82, it would make sense to throttle that IP via our traffic control menu. The network protocols used are seen in a pi chart. The pi chart is interactive and by clicking a label in the legend, you can hide that protocol. This way, one heavily used protocol does not hide the breakdown of less used protocols. These graphs were created with the Chart.js library.

The breakdown of the protocols can be useful in a number of ways. For instance, a node that is predominately sending HTTP is likely making a large number of web requests. If a user is using SSH, they are likely configuring one of the nodes. It is information such as this that gives insight into network activity in a very unique way.

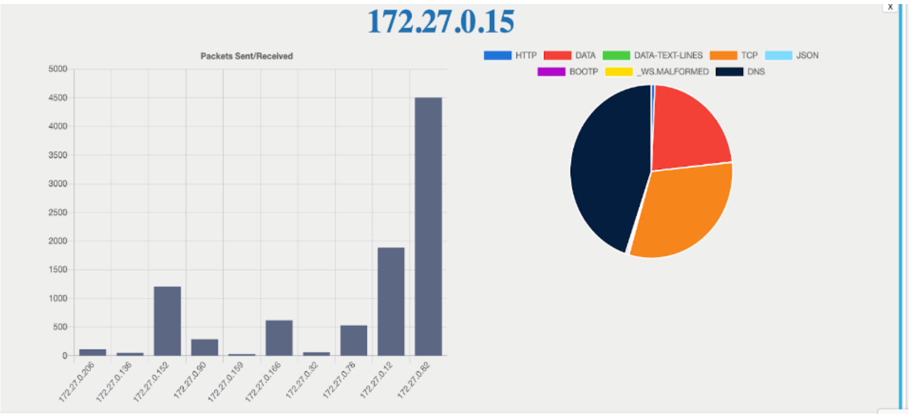


Figure 3.2: Network Status Mockup Diagram

3.10.2 Frontend Networking

As previously stated, the frontend works as a stand alone system and does not need mesh capabilities. In order to have the frontend available, all we have to do is create a build of the Angular project and then serve the project over an HTTP request to the node that is hosting it. The frontend is then able to gather the necessary data via HTTP requests to our API endpoints exposed on the gateway node. It takes the data files received from the API and parses them into data structures such as Arrays and Hashmaps. These data structures are the building blocks of our graphs, such as the one in Figure 15.1.

3.11 Node Functionality

While there is a main frontend that users are encouraged to utilize, another feature of our system is the ability to contact nodes directly. While this feature is used primarily for nodes to communicate between themselves, there are other use cases and actions that users can take with the nodes.

3.11.1 Graphical Images

The nodes themselves can display graphical images of data to users. This behavior is done by Python scripts on the backend, and it only displays images to keep the system lightweight. While not as powerful as the frontend, the images can give users valuable insights into their network. The types of images are listed below.

- ARP table graphs

  ARP table graphs show the connectivity of nodes. They are rendered as a simple PNG image to keep the system light weight. As previously stated, these graphs show users a node's view of the network. From there, the user can more quickly diagnose errors.

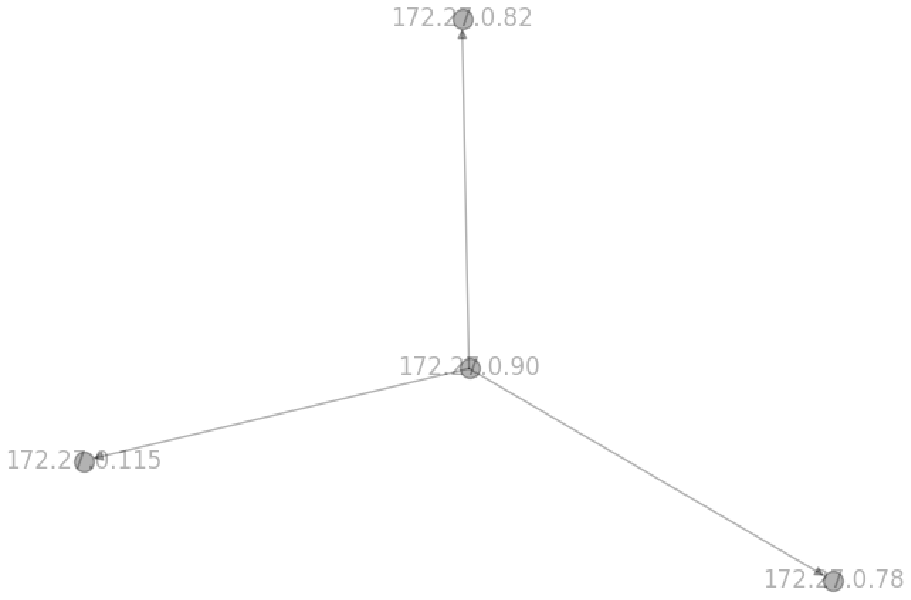


Figure 3.3: Network Status Mockup Diagram

- Traffic Data

  Nodes can generate and display a graph of how much traffic they are sending - similar to what is viewed on the frontend in a picture form. The graph is rendered as an image to save space. These graphs can quickly give a user insight into how their network is behaving.

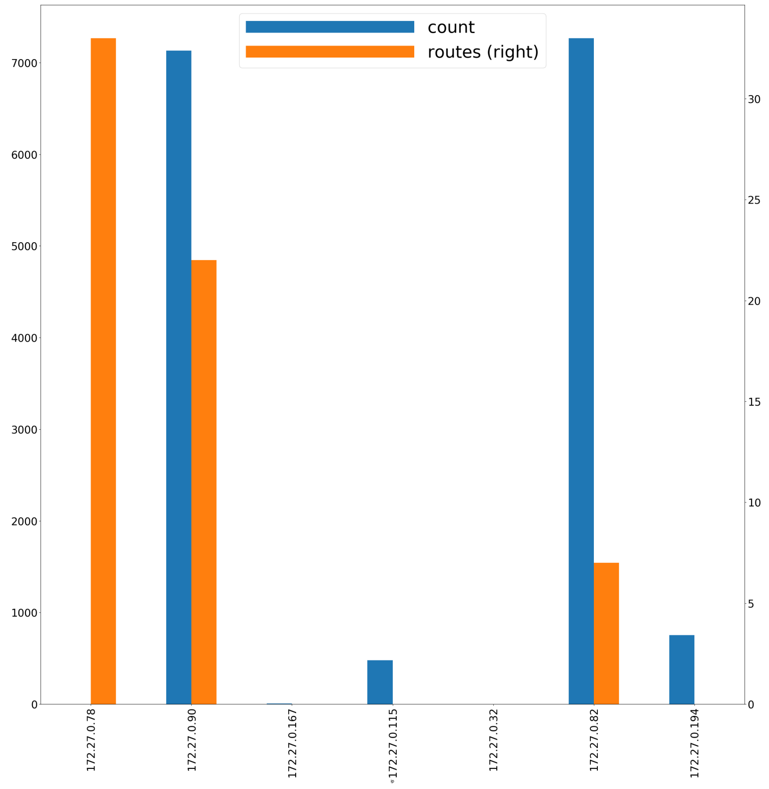


Figure 3.4: Network Status Mockup Diagram

3.11.2 Data

- Tc Info

  The frontend can display the results of a traffic shaping operation and show users how it classified nodes. This helps users know if the network has made any assumptions about shaping that they should attempt to rectify.

- Inputting Traffic Control Info

  There is an endpoint on the nodes for receiving traffic control information from the admin so that it can apply the changes.

- Listing Other Routes

  The node can list the available routes that it has. This helps give users a clear idea of a node's capabilities.

- Getting Traffic Data

Network users can retrieve the raw traffic data in the JSON data format.

- Getting the ARP Data

  Network users can get a node’s ARP data (that is used to make the ARP graph) in a raw JSON format.

3.12 Example Work Flow of System

The following is a time-line for the system running and a description of its processes.

1. The server on all of the nodes is started.

   The Node.js instance that runs all of the nodes is started. It is the Node.js backend that manages the chron tasks of the server.

2. The nodes receive traffic and parse it.

   A chron task is run that collects traffic. This task is simply a Python script that collects traffic and parses it into a JSON format that we determined to be effective. This also helps to remove extraneous information from the traffic and save space.

3. The nodes check their ARP table and convert it into the JSON format.

   This is done through Python scripts. A particular node is seen as a node on a graph and its ARP table is its connections.

4. The nodes shape traffic.

   The script to shape traffic is run and nodes prioritize or throttle traffic according to a variety of heuristics, as previously discussed.

5. The nodes exchange traffic with each-other and generate graphics

   These graphics in particular are primarily images.

6. The nodes receive traffic and generate graphics again, but this time with the information from other nodes.

   The information from other nodes allows nodes to make more informed decisions about their network.

3.13 Test Plan

3.13.1 Verification

Verification of the product is tested by checking that data is transferred correctly. Tests have been done on files and transferred data to ensure that they have remained intact. Data is sent through the network to test that connectivity is working. Most of this testing essentially involved verifying that our low level protocols integrated well with our high level protocols and that our networking protocol worked correctly.

- Traffic sent by client devices will be sent correctly.
- Files will be transferred without error.
- Nodes will be configures properly and will alert the network if they are not.

3.13.2 Validation

The design has been tested against all of the requirements and constraints. Unit tests from the requirements have been performed to ensure the correct functionality. The product has been compared to the activity diagrams to make sure that it behaved as expected.

- The nodes on the network are mesh nodes with the ability to forward traffic themselves.
- The system has the ability to notify users if there is an error.
- Work flow is consistent with activity diagrams.

Chapter 4

Concluding Remarks

4.1 Challenges Encountered

Throughout this project, we faced many challenges integrating various features into our system.

4.1.1 Technology

- **Routing**:

  One of the biggest challenges (that ultimately proved insurmountable) was to get dynamic routing (and by extension - load balancing) across all of the nodes. Part of the reason for this problem was that the protocol worked at a very low level so controlling it was difficult.

- **Maintaining a Cohesive Record of Traffic**:

  Because traffic could be sent through multiple nodes, maintaining a consistent record of where all of the traffic was going proved to be a challenge.

4.2 Technology

- **Dealing with external workload increase**:

  There were many times where the workload from other classes increased to the point where we were not able to work on the project for periods of time. This forced us to compromise on many features.

4.3 Suggested Changes

Overall, our mesh network was built to be flexible. This had a number of advantages - mainly an easier end user setup. However, this approach also had problems.

4.3.1 Node Functionality

- Individual Node Frontend:

  In our current setup, nodes have an API endpoint that can be accessed in the event that the frontend is not available to get data. A potentially effective change would be to have more frontend functionality for each node in order to make the system depend on a single component. This could consist of dynamic graphs like what was included in the main frontend.

- Recognizing the Frontend:

4.3.2 Frontend Functionality

- Shaping traffic:

  The frontend should posses expanded ability to shape traffic in diverse ways. Currently, it primarily works with IP addresses, but there are many changes that we could make to further extend the functionality the frontend.

4.3.3 Core System Changes

- BATMAN-ADV:

  While 802.11s served our needs well, batman-adv could potentially serve our needs better. This is because it has built-in API functionality and is put together in a way that makes the system easier to modify.

- Lower Level Traffic Control:

  While the TC utility is both flexible, easy to install and effective, it does not look at protocols at a low enough level. For this reason, other methods of shaping traffic could potentially be more useful. A utility that could interact more closely with the protocol itself could make our system much more extendable.

4.4 Lessons Learned

4.4.1 Implementation

- The finer details of the implementation of this project should have been determined in advance.
- Features like traffic control, while functional, were implemented in a way that could have interacted better with the rest of the technology.

At the start, we looked at certain aspects of this project at a very high level. Because of this, we attempted to accomplish them in the simplest way possible. While this worked, it later constrained us in other ways. For instance, because of the way we implemented features like shaping, other features like load balancing became difficult to implement without extensive editing of other components. Were we to do this again, we would handle things differently. We would try to bridge low level components directly to high level components.

4.4.2 Team Coordination

- A more tight meeting schedule would have likely increased our productivity.

Initially, we planned our project anticipating an increased workload from schoolwork, but we could not have anticipated the amount of issues that would derail us. It would have helped to try and make more specific test cases during fall quarter. This would have moved the project along and we would have had a clearer path forward. Earlier on, we should have focused on aspects like data storage at a more specific level and built test cases around that.

4.5 Ethics and Societal Impact

4.5.1 The Politics of Ownership

Currently, wireless networks are controlled by large corporations that have the power to determine what content can and cannot be accessed. In contrast, mesh networks hold the promise of wireless networks that are owned and operated by individuals.

The fundamental issue with ownership of communication infrastructure today is that it is controlled. Control has been proven to be an issue when determining what traffic should be sent through a network. This debate has come up in numerous occasions such as the recent debate over net neutrality. The issue of ownership and control of networks is multifaceted and has a large number of stakeholders.

Mesh technologies enable an alternative community-based ownership model, which in turn enables a different control model - one that is driven by the needs of the community, not investors. Most of the mesh technologies discussed provide different ways to manage resources and decide what traffic should be prioritized. However, the most common criticism of group-based decision making is that good intentions may still lead to bad outcomes. For example, a user might prioritize their traffic and unintentionally impact another user. People will often prioritize according to their own self-interest, and this could lead to misuse of resources. Thus, a secondary goal of these technologies is to ensure the fair sharing a network resources. The main opportunity of mesh networking is to create a network that is communally owned - that is, owned by individuals and not large corporations.

4.5.2 Privacy

One concern that could be brought up about our system is that users lack privacy since members of the mesh network have visibility into everyone's traffic. There are many nodes - each of which could be operated by a different person. While privacy is a concern, the system does allow encryption. More importantly commercial ISPs look at traffic today and consumers have little or no insight to how their data is used. As the recent issues with Facebook have shown, sometimes a big company will sell targeted data without anyone's knowledge or permission.

4.5.3 Throttling and Fairness

Traffic management is an important component of commercial wireless networks. One criticism of commercial ISPs is that they can throttle any traffic they wish. This can be done either to prioritize services (like video streaming) that they typically profit from. That said, a key component of our mesh project is that it throttles traffic. However, the main difference in our use of traffic management vs commercial networks is that we are focused on the equitable distribution of network resources and not profit. Our traffic shaping features could be tuned to prevent network misuse and keep these networks from getting congested by greedy users.

Mesh networking is based on ad hoc connections and thus traffic management is critical to smooth operation. The big issue then becomes how to throttle traffic and manage networks in an equitable way. The solution that we found was to monitor the number of packets sent, but there are other solutions and other fields of packets that can be looked at. However, sometimes it is difficult to inspect packets because they might be encrypted. This brings up the issue of control. Do you trust how users are using the bandwidth of this network node? The answer to this question depends on a large number of factors that vary on a case-by-case basis. Thus, being aware of which users are generating the most amount of traffic is critical. In our project, we developed a real time monitoring application to allow us to see the results of our algorithms.

4.5.4 Environmental Sustainability

Due to the low power requirement to run Raspberry Pis, our system has the potential to make networks more sustainable. For example using solar panels to power the nodes would allow its deployment in areas that have little or no infrastructure such as remote sites or in emergency situations. Conceivably, our mesh network could help people in remote areas by powering nodes with solar panels and creating wireless relays to the public Internet.

4.5.5 Economic Sustainability

This project also has the ability to make wireless access to the Internet much more economically viable and thus sustainable in remote or low income areas. That is because the low cost of the Raspberry Pis could make infrastructure affordable in scenarios where it is currently not profitable. For example, rural America has suffered from poor cellular coverage as large corporations do not see enough profit to justify deployment. This happens because there is not enough density of users to justify a wireless network. In contrast, community-based mesh networks have no profit objectives.

4.5.6 Societal Impact

Making information more accessible to all citizens in a society empowers people to make better decisions. Mesh networks allow people in unconnected areas to communicate and organize themselves. By allowing users to host their own web servers on their mesh network, communities could create their own low cost wireless services without the need to work with a large corporation.

4.5.7 Usability

Ordinarily, traffic management is considered to be a complex concept. Thus, one of the important aspects of our project was to create a simple web interface that anyone can understand and configure. One of the most important insights we gained during this project was that knowledge is critical to effective control. Whether a network is controlled by a corporation or a community, if you don't know what's going on, then you don't know what to change. A second insight was that all of the technologies must be integrated to enable large scale deployment. At least some insight into the issues that networks face is necessary in order to use them effectively and diagnose errors. Finding users who both need the technology and have the know-how to use it can be a challenge, but as more groups acquire technology skills, this should be doable.

Another consideration is that some societies prefer a simple life and do not want technologies such as this. There are people that do not want to connect to the Internet and don't feel like they need computers. In these settings, more education is needed in order to truly exploit the potential of mesh networking technology.

4.5.8 Lifelong Learning

Creating a traffic management solution for a mesh network required us to integrate a host of technologies. Previously, we studied these technologies individually, so getting them to work together was a big effort. But despite all of our work, there is still much more to do if our ambition is to support large scale deployment. To create something unique is hard work, and we learned the importance of continuous effort.

The technologies used in this project change constantly. Every year, new protocols are being developed or modified. Thus, one must be willing to keep up with the latest protocols and network techniques in order to be aware of what users want. Our software implements many techniques and uses many tools from a variety of sources. In this way, it has prepared us for the modern software universe in which a large number of technologies work together.

4.5.9 Compassion

One of the biggest issues in our society is the lack of communication. Time and time again, we see groups of people in our society that are isolated or marginalized. At its core, computer networking is about connecting groups and devices. Our technology has the ability to connect groups of people and improve their ability to coordinate. Since our technology uses a full protocol stack, it is compatible with most web software - meaning users can host almost anything. In this way, we have the ability to make the world much more compassionate by allowing groups to communicate and share their issues. This also extends to wireless sensors. When people have access to information, they become aware of problems in the world and can work to change them. Overall, our technology can make the world much more connected and compassionate, while preventing misuse of networks.

4.6 Conclusion

In this project we developed a web-based manager to monitor the mesh network and shape traffic within it. The backbone of this network was based on Raspberry Pi computers and Alfa WiFi cards. Each of the Raspberry Pis runs a Node.js mini server that schedules tasks such as collecting network data and shaping traffic.

The biggest lesson from this project was the importance of making complex technology easy to understand and control by users. Traffic management systems incorporate many layers of technology across the protocol stack. But unless the activity inside those layers is exposed in a way that is easy to understand, it is of no value. More importantly, users need a way to interact with the system. All of the technologies mentioned have their own logic models and capabilities. We chose to display the network performance via a web interface to simplify monitoring and management tasks.

The concept of layers is integral to mesh networks because their protocols operate on different layers of the TCPIP stack. Software and operating systems implement tasks in distinct ways that interact with other components and their stack of layers. This, in turn, causes the entire system to change its behavior. In addition, mesh protocols also have their own distinctive techniques to interact with compute devices. Opting to look at things at a high level of the protocol stack has given us access to powerful utilities and technologies. However, it also has its problems - such as forcing us to interact with hardware level components that, at times, have difficult to use APIs. Ultimately, this project showed that mesh networking can be approached in a multitude of ways, and that each of these approaches has its own benefits and challenges.

This however relates to one of the major disadvantages of our approach: we lacked ability to view the bottom of the stack - the physical interfaces. Ideally we would have wanted to control the WiFi signals however the web-interface made that difficult. While there were utilities in Linux that supposedly had the ability to monitor antenna performance, they never did quite what we wanted. This was an issue that technologies like batman-adv could potentially have solved.

Looking to the future, utilizing machine learning to shape traffic would seem like an ideal approach to improve the performance of our system. Machine learning would make the system more flexible, while utilizing the same core components. This could be accomplished by implementing deep learning based systems or perhaps just coding more sophisticated algorithms directly. However, other modifications to the software are more challenging. One of these modifications is switching protocols from 802.11s to batman-adv. Most of the server should still work after this switch since both protocols have a full protocol stack, but there are other considerations. For one, badman-adv has many powerful APIs, but the server would need to be configured to run them. Moreover, features like traffic shaping would potentially need to be reconsidered.

As a final takeaway, mesh networking - more than any other type of networking - is use case dependent. Each potential client will likely have needs that are specific to them. Thus, it is necessary to have a flexible system that is easily extendable. Our project has both of these features, and it served its use cases well. We did achieve most of the project objectives, such as the ability to monitor and manage traffic in real time, and we did this in way that can be easily extended. However, real time mesh traffic management would require more development before wide scale deployment is possible and I hope that this paper shed light on that.

Appendix A

User Manual

This is a guide for running the system.

A.1 Setting Up Nodes

Nodes need to be configured to start up their interface and connect to a single ESSID upon boot up. They also need to be configured to obtain an IP address through DHCP and forward traffic.

1. For each node, make sure that it has an 802.11s capable device.
2. Run these commands to start 802.11s
3. Run: sudo iw dev wlan1 interface add mesh0 type mp mesh_id MYMESHID
4. Run: sudo iw dev mesh0 set channel 4
5. Run: sudo ifconfig wlan1 down
6. Run: sudo ifconfig mesh0 up
7. Run: sudo ip addr add 10.1.100.10/24 dev mesh0
8. Install bridge utilities
9. Run: sudo brctl addbr br0
10. Run: sudo brctl addif br0 mesh0
11. If the node is an gateway run sudo brctl addif br0 wlan0
12. If the node is a gateway check that it can forward traffic to the internet through network address translation and that it is running a dhcp server. This will automatically point traffic toward it.
13. Clone the code from github
14. Install the requisite python utilities and node.js utilitites
15. Standard configurations of hostapd and dns masq can be used with the system. The ideal setup is to have DNSMASQ running on the gateway server and to have hostapd clients bridged in through bridge utilities. This setup allows for all nodes to be on one ip address range.
16. To start a node run sudo npm run start

The main items that are important for the network are that access point nodes are running Hostapd and that the bridge utilities are running on those computers to bridge them into the nodes connected by the mesh. The nodes also have to be configured so that they can connect to the public Internet via one of the nodes on the mesh. If these considerations are addressed, the server should work.

The process for installing Python dependencies will also vary greatly. The best way to get all of the dependencies is to repeatedly try and run all of the Python scripts and install the utilities that it says are missing. Search Google for instructions on how to install these utilities, as the process for installing them varies greatly.

A.2 API

The API is configured to run on port 3030 of the nodes. By going to port 3030, a list of valid APIs will be provided. From there, to get the desired information all that is required is to go to the URL in a browser. Example: curl 172.27.0.15:3030/getFile could be used to get a nodes traffic data.

A.3 Frontend

The computer running the frontend does not need to be 802.11s capable. It does, however, need to to run Node.js with the requisite dependencies and be able to contact one of the mesh nodes (preferably a node running a DHCP server).

Set the IP address for the frontend in all of the configuration files that specify it. From there, the frontend will know what node to query for its information.

The frontend can be easily navigated from the bar at the top. To install the frontend, clone the GitHub repository and run sudo npm install. From there, the steps to install the dependencies will vary greatly.

Appendix B

code

B.1 routes.js

```
const express = require(’express’);
const path = require(’path’);
const fileController = require(’../Controllers/fileController.js’);

const fs = require(’fs’);
var os = require(’os’);
var ifaces = os.networkInterfaces();

var ip = ’172.27.0.82’;
var chosenAddress=’172.27.0.82’;

module.exports = function(app, express) {
  console.log("running router script");
  let router = express.Router();
    // server ready to accept connections here
    console.log("runing network interfaces list for ROUTES");
    //const pythonProcess = spawn(’python’, [’protocolplot.py’]);
  var networkInterfaces = os.networkInterfaces( );


  console.log( networkInterfaces );
  //for(let ni of networkInterfaces){
  for (var key in networkInterfaces) {
      //if (networkInterfaces.hasOwnProperty(key)) { // this will check if key is owned by data object and not by any of it’s ancestors
        console.log(key+’: ’+networkInterfaces[key]); // this will show each key with it’s value
        for(k in networkInterfaces[key]){
          console.log(k+’: ’+networkInterfaces[key][k]);
          /*for(l in networkInterfaces[key][k]){
            console.log(l+": "+networkInterfaces[key][k][l])
          }*/
          if("address" in networkInterfaces[key][k] && k==0){

            console.log("address is "+networkInterfaces[key][k]["address"])
            var newaddr= networkInterfaces[key][k]["address"]
            if (newaddr.includes("172.27.0")){
              chosenAddress=newaddr;
              ip=chosenAddress
            }
          }
        /*for(let el of networkInterfaces[key][k]){
          console.log(el)
        }*/

      }
    //}
}
app.on(’listening’, function () {
  // server ready to accept connections here
  console.log("runing network interfaces list for ROUTES");
  //const pythonProcess = spawn(’python’, [’protocolplot.py’]);
  var networkInterfaces = os.networkInterfaces( );


  console.log( networkInterfaces );
  //for(let ni of networkInterfaces){
    for (var key in networkInterfaces) {
      //if (networkInterfaces.hasOwnProperty(key)) { // this will check if key is owned by data object and not by any of it’s ancestors
        console.log(key+’: ’+networkInterfaces[key]); // this will show each key with it’s value
        for(k in networkInterfaces[key]){
          console.log(k+’: ’+networkInterfaces[key][k]);
          /*for(l in networkInterfaces[key][k]){
            console.log(l+": "+networkInterfaces[key][k][l])
          }*/
          if("address" in networkInterfaces[key][k] && k==0){

            console.log("address is "+networkInterfaces[key][k]["address"])
            var newaddr= networkInterfaces[key][k]["address"]
            if (newaddr.includes("172.27.0")){
              chosenAddress=newaddr;
            }
          }
          /*for(let el of networkInterfaces[key][k]){
            console.log(el)

          }*/

        }
      //}
    }

});

//setup a function that is listening for request and send back the files that are requested
//https://stackoverflow.com/questions/25463423/res-sendfile-absolute-path
app.get("/getFile", (req, res) => {
  var networkInterfaces = os.networkInterfaces( );


  console.log( networkInterfaces );
  //for(let ni of networkInterfaces){
    for (var key in networkInterfaces) {
      //if (networkInterfaces.hasOwnProperty(key)) { // this will check if key is owned by data object and not by any of it’s ancestors
        console.log(key+’: ’+networkInterfaces[key]); // this will show each key with it’s value
        for(k in networkInterfaces[key]){
          console.log(k+’: ’+networkInterfaces[key][k]);
          /*for(l in networkInterfaces[key][k]){
            console.log(l+": "+networkInterfaces[key][k][l])
          }*/
          if("address" in networkInterfaces[key][k]){

            console.log("address is "+networkInterfaces[key][k]["address"])
            var newaddr= networkInterfaces[key][k]["address"]
            if (newaddr.includes("172.27.0") && k==0){
              ip=newaddr;
            }
          }
          /*for(let el of networkInterfaces[key][k]){
            console.log(el)
          }*/

        }
      //}
    }
    console.log("chosen address is "+chosenAddress)
  console.log("recieved request for files "+ip);
  res.sendFile(path.join(__dirname, '../../', 'iplog' + ip + '.json'));
})
app.get("/editTrafficControl", (req, res) => {
  console.log(req.body);
  var stream = fs.createWriteStream("my_file.txt");
  stream.once('open', function(fd) {
    stream.write(req.body);
    stream.end();
  });
  res.status(200).send({
    data: "done"
  })
});
app.get("/", (req, res) => {
  console.log("running router script");

  var response="";
  app._router.stack.forEach(function(r){
    if (r.route && r.route.path){
    console.log(r.route.path)
    //res.send(r.route.path)
    response+=r.route.path+"\r\n"
    }
  });
  res.status(200).send(response);


  /*res.status(200).send({
    data: "hello world"
  });*/
})
app.get("/getTcDirections", (req, res) => {
  res.sendFile(path.join(__dirname, '../../', 'tcdirs'+'.json'));
})
app.get("/getIplist", (req, res) => {
  res.sendFile(path.join(__dirname, '../../', 'iplist'+'.json'));
})
app.get("/getImage1", (req, res) => {
  res.sendFile(path.join(__dirname, '../../', 'output.png'));
})
```

```
  app.get("/getGraph", (req, res) => {
    res.sendFile(path.join(__dirname, '../../', 'griplog'+ip+'.png'));
  })
  app.get("/getProtocolGraph", (req, res) => {
    res.sendFile(path.join(__dirname, '../../', 'protgriplog'+ip+'.png'));
  })
  app.get("/getArp", (req, res) => {
    res.sendFile(path.join(__dirname, '../../', 'arpfile'+ip+'.json'));
  })
  app.get("/getArpGraph", (req, res) => {
    console.log('graph'+ip+'.png')
    res.sendFile(path.join(__dirname, '../../', 'graph'+ip+'.png'));
  })
  app.get("/getTcInfo", (req, res) => {
    console.log('graph'+ip+'.png')
    res.sendFile(path.join(__dirname, '../../', 'tcdirs.json'));
  })
  app.get("/getShapeFile", (req, res) => {
    console.log('graph'+ip+'.png')
    res.sendFile(path.join(__dirname, '../../', 'shapefile'+ip+'.json'));
  })
  var bodyParser = require('body-parser')
  //app.use(express.bodyParser());
  app.use(bodyParser.json());
  app.post('/recieveTcInfo',function(req,res){
    //var user_name=req.body.user;
    //var password=req.body.password;
    //console.log("User name = "+user_name+", password is "+password);
    //res.end("yes");
    console.log(req.body.toString())
    console.log(JSON.stringify(req.body))
    console.log(res.body)
    fs.writeFile('./tcdirs.json',JSON.stringify(req.body), function(err) {
    if (err) {
      console.log("E!");

        res.end("failure");
    } else {
      //fs.writeFile('./iplog172.27.0.90.json', body, function(err) {
      //console.log("file saved "+ipname.toString());
      //console.log(ipname)
        res.end("success");
    }
   })
   });
}
```

B.2 filecontroller.js

```
const cron = require('node-cron');
const path = require('path');
const sh = require('shelljs');
const {spawn} = require('child_process');
const fs = require('fs');
const request = require('request');
var gotResponse=1;
const mainServerList = ['172.27.0.90'];
var ipList = ['172.27.0.78','172.27.0.90','172.27.0.82','172.27.0.12'];
var os = require(’os’);
var ifaces = os.networkInterfaces();
var chosenAddress=’172.27.0.82’;


//each server on the nodes will intermittently send http requests out to every other node and ask for them to send their files back

//setup a function that is listening for request and send back the files that are requested
//https://stackoverflow.com/questions/25463423/res-sendfile-absolute-path

module.exports.runScripts = function() {
  cron.schedule(’* * * * *’, () => {
    //call a function to run python scripts in a local directory every 3 minutes;
    console.log("runing traffic manage");
    const pythonProcess = spawn(’python3.5’, [’nodecap9.py’]);
    console.log("spawned process");
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returpned from python script
      console.log("printing data");
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./newlog.json’, ’utf8’);
      //console.log(sentPackets);
    });

    pythonProcess.stderr.on(’data’, (data) => {
          console.log(data.toString());
    });
  });
}
module.exports.runScriptsTraffic = function() {
  cron.schedule(’* * * * *’, () => {
    //call a function to run python scripts in a local directory every 3 minutes;
    console.log("runing traffic control");
    const pythonProcess = spawn(’python3.5’, [’pytc5.py’]);
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returned from python script
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./internthing1.json’, ’utf8’);
      //console.log(sentPackets);
    });

    pythonProcess.stderr.on(’data’, (data) => {
          console.log(data.toString());
    });
  });
}
module.exports.runScriptsArp = function() {
  cron.schedule(’* * * * *’, () => {
    //call a function to run python scripts in a local directory every 3 minutes;
    console.log("runing arp script control");
    const pythonProcess = spawn(’python3.5’, [’getarp5.py’]);
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returned from python script
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./internthing1.json’, ’utf8’);
      //console.log(sentPackets);
    });

  pythonProcess.stderr.on('data', (data) => {
     console.log(data.toString());
  });
 });
}
module.exports.createGraph = function() {
 cron.schedule('* * * * *', () => {
  //call a function to run python scripts in a local directory every 3 minutes;
  console.log("runing create graph");
  const pythonProcess = spawn('python3.5', ['trafficplot7.py']);
  pythonProcess.stdout.on('data', (data) => {
   // Do something with the data returned from python script
   console.log(data.toString());
   //let sentPackets = fs.readFileSync('./internthing1.json', 'utf8');
   //console.log(sentPackets);
  });

  pythonProcess.stderr.on('data', (data) => {
     console.log(data.toString());
  });
 });
}
module.exports.createProtocolGraph = function() {
 cron.schedule('* * * * *', () => {
  //call a function to run python scripts in a local directory every 3 minutes;
  console.log("runing create graph");
  const pythonProcess = spawn('python3.5', ['protocolplot.py']);
  pythonProcess.stdout.on('data', (data) => {
   // Do something with the data returned from python script
   console.log(data.toString());
   //let sentPackets = fs.readFileSync('./internthing1.json', 'utf8');
   //console.log(sentPackets);
  });

  pythonProcess.stderr.on('data', (data) => {
     console.log(data.toString());
  });
 });
}
module.exports.createArpGraph = function() {

 cron.schedule('* * * * *', () => {
  //call a function to run python scripts in a local directory every 3 minutes;
  console.log("runing create arp graph");
  const pythonProcess = spawn('python3.5', ['generatefullarp.py']);
  pythonProcess.stdout.on('data', (data) => {
   // Do something with the data returned from python script
   console.log(data.toString());
   //let sentPackets = fs.readFileSync('./internthing1.json', 'utf8');
   //console.log(sentPackets);
  });

  pythonProcess.stderr.on('data', (data) => {
     console.log(data.toString());
  });
 });
 cron.schedule('* * * * *', () => {
  //call a function to run python scripts in a local directory every 3 minutes;
  console.log("runing create arp graph");
```

```
    const pythonProcess = spawn(’python3.5’, [’comparearpgraphs3.py’]);
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returned from python script
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./internthing1.json’, ’utf8’);
      //console.log(sentPackets);
    });

    pythonProcess.stderr.on(’data’, (data) => {
          console.log(data.toString());
    });
  });
}
module.exports.autoShape = function() {
  cron.schedule(’* * * * *’, () => {
    //call a function to run python scripts in a local directory every 3 minutes;
    console.log("runing auto shape");
    const pythonProcess = spawn(’python3.5’, [’autoshape9.py’]);
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returned from python script
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./internthing1.json’, ’utf8’);
      //console.log(sentPackets);
    });

    pythonProcess.stderr.on(’data’, (data) => {
          console.log(data.toString());
    });
  });
}
module.exports.genArp = function() {
  cron.schedule(’* * * * *’, () => {
    //call a function to run python scripts in a local directory every 3 minutes;
    console.log("runing auto shape");
    const pythonProcess = spawn(’python3.5’, [’generatefullarp.py’]);
    pythonProcess.stdout.on(’data’, (data) => {
      // Do something with the data returned from python script
      console.log(data.toString());
      //let sentPackets = fs.readFileSync(’./internthing1.json’, ’utf8’);
      //console.log(sentPackets);
    });

    pythonProcess.stderr.on(’data’, (data) => {
          console.log(data.toString());
    });
  });
}

module.exports.getFiles = function() {
  cron.schedule(’* * * * *’, () => {
    console.log("running get file");
    //call functsion to make http requests to the other nodes in the network
    //for each ip in ips, make a request
    var filePath=path.join(__dirname,"../../iplist.txt")

    ipList=fs.readFileSync(filePath)
    console.log(ipList.toString().split(’,’))
    ipList=ipList.toString().split(’,’)
    console.log(ipList)
    for(var ip1 in ipList){
      console.log(chosenAddress)
      if(ipList[ip1] == chosenAddress || ipList[ip1].length<=0)continue;
      console.log(’requesting ip ’+ipList[ip1]);
      (function(ipname){
        console.log(’requesting ip name’+ipname);
        request(’http://’+ipList[ip1]+’:3030/getFile’, function (error, response, body) {
          console.log(’attempting to write to file’+ipname);
          console.log(’error is ’+error);
          console.log(’response is ’+response);
          console.log(’body is from ’+ipname.toString()+’ ’+body);
          if(error == null && response != null){
          fs.writeFile(’./iplog’+ipname+’.json’, body, function(err) {
            if (err) {
              console.log("E!");
            } else {
              //fs.writeFile(’./iplog172.27.0.90.json’, body, function(err) {
              console.log("file saved "+ipname.toString());
              console.log(ipname)
            }
          })
          }else{
            console.log("nothing recieced");
          }
        });
      })(ipList[ip1]);
    }
  })

}
module.exports.getFilesArp = function() {
  cron.schedule(’* * * * *’, () => {
    console.log("running get file arp ");
    var filePath=path.join(__dirname,"../../iplist.txt")

    ipList=fs.readFileSync(filePath)
    console.log(ipList.toString().split(’,’))
    ipList=ipList.toString().split(’,’)
    console.log(ipList)
    for(var ip1 in ipList){
      console.log(chosenAddress)
      if(ipList[ip1] == chosenAddress || ipList[ip1].length<=0)continue;
      console.log(’requesting ip ’+ipList[ip1]);
      (function(ipname){
        console.log(’requesting ip name’+ipname);
        request(’http://’+ipList[ip1]+’:3030/getArp’, function (error, response, body) {
          console.log(’attempting to write to file’+ipname);
          console.log(’error is ’+error);
          console.log(’response is ’+response);
          console.log(’body is from ’+ipname.toString()+’ ’+body);
          if(error == null && response != null){
          fs.writeFile(’./arpfile’+ipname+’.json’, body, function(err) {
            if (err) {
              console.log("E!");
            } else {
              //fs.writeFile(’./iplog172.27.0.90.json’, body, function(err) {
              console.log("file saved "+ipname.toString());
              console.log(ipname)
            }
```

```
     })
     }else{
      console.log("nothing recieced");
     }
    });
   })(ipList[ip1]);
  }

  /*
  //call functsion to make http requests to the other nodes in the network
  //for each ip in ips, make a request
  var filePath=path.join(__dirname,"../../iplist.txt")

  ipList=fs.readFileSync(filePath)
  console.log(ipList.toString().split(','))
  ipList=ipList.toString().split(',')
  console.log(ipList)
  for(var ip1 in ipList){
   console.log(chosenAddress)
   if(ipList[ip1] == chosenAddress || ipList[ip1].length<=0)continue;
   console.log('requesting ip '+ipList[ip1]);
   request('http://'+ipList[ip1]+':3030/getArp', function (error, response, body) {
    console.log('attempting to write to file');
    console.log('error is '+error);
    console.log('arp response is '+response);
    console.log('body is from '+ipList[ip1].toString()+' '+body);
    if(error == null && response != null){
    fs.writeFile('./arpfile'+ipList[ip1]+'.json', body, function(err) {
     if (err) {
      console.log("E!");
     } else {
      //fs.writeFile('./iplog172.27.0.90.json', body, function(err) {
      console.log("file saved "+ipList[ip1].toString());
      console.log(ipList[ip1])
     }
    })
    }else{
     console.log("nothing recieced");
    }
   });
  }*/
 })

}
module.exports.getTcDirections = function() {
 cron.schedule('* * * * *', () => {

  console.log("running get tc directions");
  //call function to make http requests to the other nodes in the network
  //for each ip in ips, make a request

  for(var mIp in mainServerList){
   console.log('http://'+mainServerList[mIp]+':3030/getTcDirections');
   request('http://'+mainServerList[mIp]+':3030/getTcDirections'/*).on('response'*/, function (error, response, body) {
    console.log('attempting to write to file');
    console.log('attempting to write to file');
    console.log('error is '+error);
    console.log('response is '+response);
```

```
        console.log('body is '+body);
        if(error==null){
          fs.writeFile('./tcdirs.json', body, function(err) {
            if (err) {
              gotResponse=0
              console.log("E!");
            } else {
              //fs.writeFile('./test1.json', body, function(err) {

              console.log("file saved in orig list");
            }
          })
        }else{
          gotResponse=0
          console.log("error with request")
        }
      });
    }
    if(gotResponse==0){
      console.log("defaulting to old list")
      for(var mIp in ipList){
        console.log('http://'+ipList[mIp]+':3030/getTcDirections');
        request('http://'+ipList[mIp]+':3030/getTcDirections', function (error, response, body) {
          console.log('attempting to write to file');
          console.log('error is '+error);
          console.log('response is '+response);
          console.log('body is '+body);
          if(error==null){
            fs.writeFile('./test1.json', body, function(err) {
              if (err) {
                console.log("E!");
              } else {
                //fs.writeFile('./test1.json', body, function(err) {
                  gotResponse=1
                console.log("file saved");
              }
            })
          }else{
            console.log("error with request")
          }
        });
      }


      gotResponse=1;
    }
  })
}
module.exports.getNetworkInterfaces = function() {
  console.log("runing network interfaces list");
  //const pythonProcess = spawn('python', ['protocolplot.py']);
  var networkInterfaces = os.networkInterfaces( );


  console.log( networkInterfaces );
  //for(let ni of networkInterfaces){
    for (var key in networkInterfaces) {
      //if (networkInterfaces.hasOwnProperty(key)) { // this will check if key is owned by data object and not by any of it's ancestors
```

```
   console.log(key+': '+networkInterfaces[key]); // this will show each key with it's value
   for(k in networkInterfaces[key]){
    console.log(k+': '+networkInterfaces[key][k]);
    /*for(l in networkInterfaces[key][k]){
     console.log(l+": "+networkInterfaces[key][k][l])
    }*/
    if("address" in networkInterfaces[key][k]){

     console.log("address is "+networkInterfaces[key][k]["address"])
     var newaddr= networkInterfaces[key][k]["address"]
     if (newaddr.includes("172.27.0") && k==0){
      chosenAddress=newaddr;
      console.log("is the mac "+networkInterfaces[key][k]["mac"])
      console.log("is the internal "+networkInterfaces[key][k]["internal"])
      console.log("is the cidr "+networkInterfaces[key][k]["cidr"])
      console.log("is the family "+networkInterfaces[key][k]["family"])
      console.log("on address "+key+" is k"+k);
     }
    }
    /*for(let el of networkInterfaces[key][k]){
     console.log(el)
    }*/

   }
  //}
 }
 console.log("chosen address is "+chosenAddress)
cron.schedule('* * * * *', () => {
 //call a function to run python scripts in a local directory every 3 minutes;
 console.log("runing network interfaces list");
 //const pythonProcess = spawn('python', ['protocolplot.py']);
 var networkInterfaces = os.networkInterfaces( );


 console.log( networkInterfaces );
 //for(let ni of networkInterfaces){
  for (var key in networkInterfaces) {
   //if (networkInterfaces.hasOwnProperty(key)) { // this will check if key is owned by data object and not by any of it's ancestors
    console.log(key+': '+networkInterfaces[key]); // this will show each key with it's value
    for(k in networkInterfaces[key]){
     console.log(k+': '+networkInterfaces[key][k]);
     /*for(l in networkInterfaces[key][k]){
      console.log(l+": "+networkInterfaces[key][k][l])
     }*/
     if("address" in networkInterfaces[key][k]){

      console.log("address is "+networkInterfaces[key][k]["address"])
      var newaddr= networkInterfaces[key][k]["address"]
      if (newaddr.includes("172.27.0")){
       chosenAddress=newaddr;
      }
     }
     /*for(let el of networkInterfaces[key][k]){
      console.log(el)
     }*/

    }
   //}
```

```
     }
     console.log("chosen address is "+chosenAddress)
    //}

  });
}
```

B.3 nodecap9.py

```
import traceback
import json
import pyshark
from collections import Counter
import os
import datetime
#cap = pyshark.LiveCapture(interface='wlan1')
#cap.sniff(packet_count=50)
from sklearn import tree
import netifaces as ni
from joblib import dump, load
import math
bridgeflag=0

def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print( str(interface))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
                     if 'br' in interface: bridgeflag=1
              except:
                     print( "eror reading interface")
              print( "chosen interface is "+str(chosenInterface))
       return chosenInterface
interface=chooseInterface()
ni.ifaddresses(interface)
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
clf = tree.DecisionTreeClassifier()
try:
    clf = load('tree'+ip+'.joblib')
except:
    #continue
    print("error reading data")

print(ip) # should print "192.168.100.37"
packetlog=dict();
userlog=dict();
storedlog=dict();
templog=dict();


import glob
for filename in glob.glob('iplog*'):
    print(filename)
    #if ip in filename: continue;
```

```
#if ip in filename: continue;


with open(filename,’r’)as f:
 try:
   templog= json.load(f)
 except:
    continue
 #print((str(templog)))
 #storedlog={x: storedlog.get(x,0)+templog.get(x,0) for x in set(storedlog).union(templog)}
 #if templog[]
 for key,val in list(templog.items()):


          if key in storedlog:
                 print("key is in stored log "+str(key))
                 if ’172.27.0’ not in key: continue
                 if not isinstance(storedlog[key],dict): storedlog[key]=dict()
                 if ’remoteadded’ not in storedlog[key]: stored[key][’remoteadded’]=list()
                 if isinstance(storedlog[key],dict) and ’remoteadded’ in storedlog[key]:
                     if isinstance(storedlog[key][’remoteadded’],list):
                         #if filename[(filename.find(’g’)+1):(filename.find(’.j’))] not in storedlog[key][’remoteadded’]: storedlog[key][’remoteadded’].append(filename[(filename.find(’g’)+1):(filename.find(’.j’))])
                         if ’originip’ in templog and templog[’originip’] not in storedlog[key][’remoteadded’]: storedlog[key][’remoteadded’].append(templog[’originip’])

                 if ’remoteadded’in val:
                     for ij in val[’remoteadded’]:
                         if ij not in storedlog[key][’remoteadded’]:storedlog[key][’remoteadded’].append(ij)





                 if ’route’ not in storedlog[key]:
                     storedlog[key][’route’]=[]
                 if ’route’ in storedlog[key] and ’route’ in val:

                     #storedlog[key][’route’]=val[’route’]+storedlog[key][’route’]
                     for il in val[’route’]:
                         if il not in storedlog[key][’route’]: storedlog[key][’route’].append(il)
                     #if filename[(filename.find(’g’)+1):(filename.find(’.j’))] not in storedlog[key][’route’]: storedlog[key][’route’].append(filename[(filename.find(’g’)+1):(filename.find(’.j’))]) #remove later
                 if ’route’ in storedlog[key] and ’route’ not in val:
                     print("adding file name")

                     #if filename[(filename.find(’g’)+1):(filename.find(’.j’))] not in storedlog[key][’route’]: storedlog[key][’route’].append(filename[(filename.find(’g’)+1):(filename.find(’.j’))])

                 #print("debug route "+str(storedlog[key][’route’]))
                 #storedlog[key][’remoteadded’]=filename[5:17]
                 #try:
                 #print "doing addition "+str(storedlog[key])+" and "+str(val)
```

```
                        '''
                        storedlog[key]['count']= storedlog[key]['count']+val['count']
                        for key1,val1 in list(val.items()):
                            if not isinstance(val1,dict) or 'count' not in val1:
                                storedlog[key][key1]=val1
                                continue
                            print("key 1 is "+str(key1)+" val 1 is"+str(val1))
                            if key1=='count': continue
                            if key1 in storedlog[key] and 'count' in storedlog[key][key1]:
                                storedlog[key][key1]=val1
                                if storedlog[key][key1]['count']!=val1['count']:
                                    storedlog[key][key1]['count']= storedlog[key][key1]['count']+val1['count']
                                else:
                                    storedlog[key][key1]['count']= storedlog[key][key1]['count']
                            elif key1 in storedlog[key]:
                                storedlog[key][key1]=val1
                        '''



                            #print "result is "+str(storedlog[key])
                        #except:
                            #print "error doing addition"
                            #exit()


                else:
                    print("adding new key to stored log")
                    #val['remoteadded']=filename[5:17]
                    if isinstance(val,dict) and 'remoteadded' in val:
                        if isinstance(val['remoteadded'],list):
                            val['remoteadded'].append(templog['originip'])
                        else:

                            val['remoteadded']=[templog['originip']]
                    elif isinstance(val,dict):
                        if 'originip' in templog: val['remoteadded']=[templog['originip']]
                        else:val['remoteadded']=[]# fix later

                    print("adding value "+str(val))
                    storedlog[key]=val
    #os.system('mv '+filename+' '+'oldlog'+filename[4:])

    print("printing new log")
    print((json.dumps(storedlog)))

userlog=dict(storedlog)
#if ip in storedlog: userlog=storedlog[ip]
storedlogbackup=dict(storedlog)
#with open("datathing1.json",'r')as f:
# userlog= json.load(f)

externpacketlog=dict()
#with open("externthing1.json",'r')as f:
# externpacketlog= json.load(f)
#print json.dumps(userlog)
internpacketlog=dict()
#with open("internthing1.json",'r')as f:
```

```
# internpacketlog= json.load(f)
#userlog=dict();
print((json.dumps(storedlog)));
#print json.dumps(userlog);
#print json.dumps(internpacketlog)
#print json.dumps(externpacketlog)
print("sniffing packets")
errorcount =0;
cap = pyshark.LiveCapture(interface=interface)
print("sniffed packets")
cap.sniff(packet_count=500)
for packet in cap:
    ipstr='ip'
    if ipstr not in packet: ipstr='ipv6'
    #print packet
    #print packet.highest_layer
    try:
      if ('tcp' in packet or 'udp' in packet) and ('172.27.0' in str(packet[ipstr].src)):
        print("key is "+str(packet[ipstr].src))
        #print(packet)
        #print packet.highest_layer
        #print packet.tcp.dstport
        print (packet[ipstr].src)

        try:
          if packet[ipstr].src in userlog:
                    print("adding to dict again ")
                    #packetlog[packet.tcp.dstport] = 1+packetlog[packet.tcp.dstport];
                    print(("added to user log "+str(packet[ipstr].src)+" "+str(userlog[packet[ipstr].src])))
                    userlog[packet[ipstr].src]['count'] = userlog[packet[ipstr].src]['count']+1;
                    #userlog[packet[ipstr].src]['length'] = int(packet.length)
                    userlog[packet[ipstr].src]['totalsize'] = int(packet.length)+int(userlog[packet[ipstr].src]['totalsize'])
                    userlog[packet[ipstr].src]['length'] = int(userlog[packet[ipstr].src]['totalsize']/userlog[packet[ipstr].src]['count'])
                    #userlog[packet[ipstr].src]['time'].append(str((packet.sniff_timestamp)))
                    if 'route' not in userlog[packet[ipstr].src] and packet[ipstr].src != ip and packet[ipstr].dst != ip:
                        print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                        print("adding route")
                        userlog[packet[ipstr].src]['route']=[ip]
                    elif packet[ipstr].src != ip and packet[ipstr].dst != ip and ip not in userlog[packet[ipstr].src]['route']:
                        print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                        print("adding route")
                        userlog[packet[ipstr].src]['route'].append(ip)
                    if packet.highest_layer not in userlog[packet[ipstr].src]['protocol']:
                        print("adding new protocol")
                        userlog[packet[ipstr].src]['protocol'][packet.highest_layer]=1
                    else:
                        userlog[packet[ipstr].src]['protocol'][packet.highest_layer]=userlog[packet[ipstr].src]['protocol'][packet.highest_layer]+1


                    if packet[ipstr].dst in userlog[packet[ipstr].src]:
                            #userlog[packet[ipstr].src]+1;
                            print(("incrementing user log before "+str(userlog[packet[ipstr].src][packet[ipstr].dst])))
                            userlog[packet[ipstr].src][packet[ipstr].dst][’count’]= userlog[packet[ipstr].src][packet[ipstr].dst][’count’]+1;
                            print(("incrementing user log after "+str(userlog[packet[ipstr].src][packet[ipstr].dst])))
                            if ’time’ in userlog[packet[ipstr].src][packet[ipstr].dst]: userlog[packet[ipstr].src][packet[ipstr].dst][’time’].append(str(packet.sniff_timestamp))
                            else: userlog[packet[ipstr].src][packet[ipstr].dst][’time’]= [str(packet.sniff_timestamp)];
                    else:
                        userlog[packet[ipstr].src][packet[ipstr].dst]= dict();
                        userlog[packet[ipstr].src][packet[ipstr].dst][’count’]= 1;
                        userlog[packet[ipstr].src][packet[ipstr].dst][’time’]= [str(packet.sniff_timestamp)];

                    #if packet[ipstr].src== "172.27.0.90":
                    #if packet[ipstr].src == "172.27.0.90" or packet[ipstr].src== "172.27.0.155" or packet[ipstr].src == "172.27.0.15" or packet[ipstr].src =="172.27.0.227":
                        #internpacketlog[packet[ipstr].src] = internpacketlog[packet[ipstr].src]+1;
                    #else:
                        #externpacketlog[packet[ipstr].dst] = externpacketlog[packet[ipstr].dst]+1;

          else:
                    #print "adding to dict"
                    #packetlog[packet.tcp.dstport]=1;

                    #userlog[packet[ipstr].src]=1;
                    userlog[packet[ipstr].src]=dict()
                    userlog[packet[ipstr].src][’count’]=1
                    userlog[packet[ipstr].src][’totalsize’]=packet.length
                    userlog[packet[ipstr].src][’protocol’]=dict()
                    userlog[packet[ipstr].src][’protocol’][packet.highest_layer]=1
                    userlog[packet[ipstr].src][packet[ipstr].dst]= dict();
                    userlog[packet[ipstr].src][packet[ipstr].dst][’count’]= 1;
                    if ’route’ not in userlog[packet[ipstr].src] and packet[ipstr].src != ip and packet[ipstr].dst != ip:
                        print("adding route")
                        print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                        userlog[packet[ipstr].src][’route’]=[ip]
                    elif packet[ipstr].src != ip and packet[ipstr].dst != ip and ip not in userlog[packet[ipstr].src][’route’]:
                        print("adding route")
                        print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                        userlog[packet[ipstr].src][’route’].append(ip)

                    ’’’
                    elif packet[ipstr].src != ip:
                        print("appending route")
                        userlog[packet[ipstr].src][’route’].append(ip)
                    ’’’
                    #else:
                        #userlog[packet[ipstr].src][’route’].append(ip)


```

```
                  userlog[packet[ipstr].src]['port']=dict()
                  #userlog[packet[ipstr].src]['time']=[str(packet.sniff_timestamp)]
                  if 'dstport' in packet:
                      userlog[packet[ipstr].src]['port'][packet[ipstr].dstport]=1


      except:
        errorcount=errorcount+1;
        print("error with fields")
        print((traceback.format_exc()))
        #print packet
    if ('tcp' in packet or 'udp' in packet) and ('172.27.0' in str(packet[ipstr].dst)):
      print("key is "+str(packet[ipstr].dst))
      #print(packet)
      #print packet.highest_layer
      #print packet.tcp.dstport
      print (packet[ipstr].dst)

      try:
        if packet[ipstr].dst in userlog:
                  print("adding to dict again ")
                  #packetlog[packet.tcp.dstport] = 1+packetlog[packet.tcp.dstport];
                  print(("added to user log "+str(packet[ipstr].dst)+" "+str(userlog[packet[
                      ipstr].dst])))
                  userlog[packet[ipstr].dst]['count'] = userlog[packet[ipstr].dst]['count']+1;
                  #userlog[packet[ipstr].dst]['length'] = int(packet.length)
                  userlog[packet[ipstr].dst]['totalsize'] = int(packet.length)+int(userlog[
                      packet[ipstr].dst]['totalsize'])
                  userlog[packet[ipstr].dst]['length'] = int(userlog[packet[ipstr].dst]['
                      totalsize']/userlog[packet[ipstr].dst]['count'])
                  if 'route' not in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and
                      packet[ipstr].src != ip:
                      print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[
                          ipstr].dst))
                      userlog[packet[ipstr].dst]['route']=[ip]
                  elif 'route' in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and ip
                      not in userlog[packet[ipstr].dst]['route'] and packet[ipstr].src != ip:
                      print("appending route ")
                      print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[
                          ipstr].dst))
                      userlog[packet[ipstr].dst]['route'].append(ip)
                  if packet.highest_layer not in userlog[packet[ipstr].dst]['protocol']:
                      print("adding new protocol")
                      userlog[packet[ipstr].dst]['protocol'][packet.highest_layer]=1
                  else:
                      userlog[packet[ipstr].dst]['protocol'][packet.highest_layer]=userlog[packet
                          [ipstr].dst]['protocol'][packet.highest_layer]+1


                  if packet[ipstr].src in userlog[packet[ipstr].dst]:
                         #userlog[packet[ipstr].dst]+1;
                         print(("incrementing user log before "+str(userlog[packet[ipstr].dst][
                             packet[ipstr].src])))
                         userlog[packet[ipstr].dst][packet[ipstr].src]['count']= userlog[packet[
                             ipstr].dst][packet[ipstr].src]['count']+1;
                         print(("incrementing user log after "+str(userlog[packet[ipstr].dst][
                             packet[ipstr].src])))
                         if 'time' in userlog[packet[ipstr].dst][packet[ipstr].src]['time']:
                             userlog[packet[ipstr].dst][packet[ipstr].src]['time'].append(str(
                                    packet.sniff_timestamp));
                                else: userlog[packet[ipstr].dst][packet[ipstr].src]['time']= [str(packet.sniff_timestamp)];
                        else:
                            userlog[packet[ipstr].dst][packet[ipstr].src]= dict();
                            userlog[packet[ipstr].dst][packet[ipstr].src]['count']= 1;
                            userlog[packet[ipstr].dst][packet[ipstr].src]['time']= [str(packet.sniff_timestamp)];

                        #if packet[ipstr].src== "172.27.0.90":
                        #if packet[ipstr].src == "172.27.0.90" or packet[ipstr].src== "172.27.0.155" or packet[ipstr].src == "172.27.0.15" or packet[ipstr].src =="172.27.0.227":
                            #internpacketlog[packet[ipstr].src] = internpacketlog[packet[ipstr].src]+1;
                        #else:
                            #externpacketlog[packet[ipstr].dst] = externpacketlog[packet[ipstr].dst]+1;

            else:
                        #print "adding to dict"
                        #packetlog[packet.tcp.dstport]=1;

                        #userlog[packet[ipstr].src]=1;
                        userlog[packet[ipstr].dst]=dict()
                        userlog[packet[ipstr].dst]['count']=1
                        userlog[packet[ipstr].dst]['totalsize']=packet.length
                        userlog[packet[ipstr].dst]['protocol']=dict()
                        userlog[packet[ipstr].dst]['protocol'][packet.highest_layer]=1
                        userlog[packet[ipstr].dst][packet[ipstr].src]= dict();
                        userlog[packet[ipstr].dst][packet[ipstr].src]['count']= 1;
                        userlog[packet[ipstr].src]['port']=dict()
                        if 'route' not in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and packet[ipstr].src != ip:
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].dst]['route']=[ip]
                        elif packet[ipstr].dst != ip and packet[ipstr].src != ip and ip not in userlog[packet[ipstr].dst]['route']:
                            print("appending route " +str(ip)+" ")
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].dst]['route'].append(ip)

                        #userlog[packet[ipstr].src]['time']=[str(packet.sniff_timestamp)]
                        if 'srcport' in packet:
                            userlog[packet[ipstr].src]['port'][packet[ipstr].srcport]=1


         except:
            errorcount=errorcount+1;
            print("error with fields")
            print((traceback.format_exc()))
            #print packet

    except:
        print("error ")
        print((traceback.format_exc()))
        #print packet

#print json.dumps(packetlog)
```

```
#import netifaces as ni
#ni.ifaddresses(’wlan1’)
#ip = ni.ifaddresses(’wlan1’)[ni.AF_INET][0][’addr’]
#print ip # should print "192.168.100.37
for key,val in list(userlog.items()):
    if not isinstance(val,dict)or ’route’ not in val or not isinstance(val[’route’],list): continue
    for el in val[’route’]:
        print("checking route "+el)
        routecount=0
        for key1,val1 in userlog.items():
            print(val1)
            if isinstance(val1,dict) and ’route’ in val1 and isinstance(val1[’route’],list) and el in val1[’route’] and key1 != el:
                if len(val1[’protocol’])>2 or val1[’count’]>20:
                    routecount=routecount+1+len(val1[’protocol’])
        if el not in userlog:
            userlog[el]=dict()
        print("adding route count to "+str(el)+" which is "+str(routecount) )
        if ’routecount’ not in userlog[el] or userlog[el][’routecount’]<routecount: userlog[el][’routecount’]=routecount

arplog=dict()
try:
    with open("fullarp"+ip+".json",’r’)as f:
        arplog= json.load(f)
except:
    print("error reading file")
classifierinit=0
try:
    clf = load(’tree’+ip+’.joblib’)
    classifierinit=1
except:
    print("error with classifier")
for key,val in list(userlog.items()):
    if isinstance(val,dict) and ’route’ in val:
        for el in list(val[’route’]):
            print("checking "+str(el))
            treepred=’node’
            if el in userlog:
                if (el in userlog and ’class’ in userlog[el] and userlog[el][’class’] != ’node’) or classifierinit ==0 : continue
                try:
                    ’’’
                    pdata=[]
                    val=userlog[el]
                    if ’count’ in val and ’length’ in val: pdata.append(val[’count’]*int(math.log(val[’length’])))
                    else: pdata.append(0)
                    if ’count’ in val: pdata.append(val[’count’])
                    else: pdata.append(0)
                    if ’length’ in val: pdata.append(val[’length’])
                    else: pdata.append(0)
                    #pdata.append(0)
                    dests=0
                    #if ’class’ in val: pdata[’class’]=int(val[’class’])
                    #else: pdata.append(int(’node’))
                    if ’routecount’ in val: pdata.append(val[’routecount’])
                    else: pdata.append(0)
                    for key1,val1 in val.items():
                        print(key1)
                        if '172.27.0' in key1:
                            dests=dests+1
                    pdata.append(dests)
                    '''
                    pdata=[]
                    if 'length' in val:
                        pdata.append((val['count']*int(math.log(val['length']))))
                        pdata.append(val['count'])
                        pdata.append(val['length'])
                        #pdata.append(0)
                        dests=0
                        #if 'class' in val: pdata['class']=int(val['class'])
                        #else: pdata.append(int('node'))
                        if 'routecount' in val: pdata.append(val['routecount'])
                        else: pdata.append(0)
                        for key1,val1 in val.items():
                            #print(key1)
                            if '172.27.0' in key1:
                                dests=dests+1
                        pdata.append(dests)
                        try:
                            arpdict=dict()
                            with open("arpfile"+ip+".json") as f:
                                arpdict=json.load(f)
                            if key in arpdict:
                                pdata.append(len(arpdict[key]))
                            else:
                                pdata.append(0)
                        except:
                                pdata.append(0)

                    treepred=clf.predict([pdata])[0]
                    print("tree pred is"+treepred+" for "+str(el))
                    userlog[el]['class']=treepred
                    #break
                    continue
                except:
                    if 'routecount'in userlog[el] and userlog[el]['routecount']>6: userlog[el]['class']='gateway'
                    else: userlog[el]['class']='node'

            if el not in userlog and '172.27.0.' in el:

                userlog[el]=dict()
                userlog[el]['class']='bridge'
            elif '172.27.0.' in el:
                if 'routecount'in userlog[el] and userlog[el]['routecount']>6: userlog[el]['class']='gateway'
                else: userlog[el]['class']='node'

for key,val in userlog.items():
    if isinstance(val,dict) and( 'class' not in val or val['class']=='node'):
                if 'routecount' in userlog[key] and userlog[key]['routecount']>=3: userlog[key]['class']='gateway'
                else: userlog[key]['class']='node'
                for key1,val1 in val.items():
                    if '172.27.0' in key1 and key1 in userlog:
                        if 'routecount' in userlog[key1] and userlog[key1]['routecount']>=3: userlog[
                         key1]['class']='gateway'
                     else: userlog[key1]['class']='node'
                 elif '172.27.0' in key1 and key1 not in userlog:userlog[key1]['class']='node'


savedLog= dict();
#storedlog=userlog;

print((json.dumps(storedlogbackup)))
print((json.dumps(storedlog)))
#print json.dumps(internpacketlog)
#print json.dumps(externpacketlog)
import random
dictid=random.randint(1,21)*5
userlog['id']=dictid
userlog['originip']=ip
userlog['bridge']=bridgeflag

packetfile='iplog'+ip+'.json';
with open(packetfile,'w') as outfile:
    json.dump(userlog,outfile)
with open('datathing1.json','w') as outfile:
    json.dump(userlog,outfile)


import os
print("new val is")
os.system('cat '+packetfile)

print(("\n\n\nold val is "+str(storedlog)))
exit()
#os.system('cat internthing1.json')
#os.system('cat externthing1.json')
```

B.4 autoshape9.py

```
import pandas as pd
import traceback
import json
import math
import os

from sklearn import tree
from joblib import dump, load
import socket
def valid_ip(address):
   try:
       socket.inet_aton(address)
       return True
   except:
       return False
ip='172.27.0.82'
def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print(( str(interface)))
              try:
```

```
                        ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                        if '172.27.0.' in ip:
                                chosenInterface=interface
                except:
                        print( "eror reading interface")
                print(( "chosen interface is "+str(chosenInterface)))
        return chosenInterface
import netifaces as ni
interface=chooseInterface()
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
data=dict()
with open('iplog'+ip+'.json','r')as f:
  data= json.load(f)
prevlog=dict()
try:
        with open("shapefile"+ip+".json",'r')as f:
            prevlog= json.load(f)
except:
    print("error reading json dictionary")
datalist=[]
classlist=[]
resultlist=[]
sublist=[]
manualforward=dict()
with open('tcdirs'+'.json','r')as f:
  manualforward= json.load(f)
priolist=manualforward['prioritize']
slowlist=manualforward['slow']
for key,val in data.items():
    if not isinstance(val,dict) or 'count' not in val: continue

    pdata=dict()
    print(str(key)+" "+str(val['count']))
    pdata['ip']=key
    if 'length' in val:
        pdata['countlen']=val['count']*int(math.log(val['length']))
        pdata['count']=val['count']
        pdata['length']=val['length']
        pdata['dests']=0
        if 'class' in val: pdata['class']=val['class']
        else: pdata['class']='node'
        if 'routecount' in val: pdata['routecount']=val['routecount']
        else: pdata['routecount']=0
        for key1,val1 in val.items():
            if not valid_ip(key1): continue
            subdests=dict()
            subdests['ip']=key1
            subdests['count']=val1['count']
            sublist.append(subdests)

            print(key1)
            if '172.27.0' in key1:
                pdata['dests']=pdata['dests']+1

    else:
        pdata['count']=val['count']
        pdata['length']=0
        pdata['class']=val['class']
        if 'routecount' in val: pdata['routecount']=val['routecount']
        else: pdata['routecount']=0
    try:
        arpdict=dict()
        with open("arpfile"+ip+".json") as f:
            arpdict=json.load(f)
        if key in arpdict:
            pdata['arp']=len(arpdict[key])
        else:
            pdata['arp']=0
    except:
            #pdata.append(0)
            pdata['arp']=0
    #datalist.append(pdata)


    if not isinstance(val,dict) or 'count' not in val or 'length' not in val: continue
    classdata=[]
    print(str(key)+" "+str(val['count']))
    #pdata['ip']=key
    classdata.append(val['count']*int(math.log(val['length'])))
    classdata.append(val['count'])
    classdata.append(val['length'])
    #classdata.append(0)
    dests=0
    #if 'class' in val: classdata['class']=int(val['class'])
    #else: classdata.append(int('node'))
    if 'routecount' in val: classdata.append(val['routecount'])
    else: classdata.append(0)
    for key1,val1 in val.items():
        print(key1)
        if '172.27.0' in key1:
            dests=dests+1
    classdata.append(dests)
        #print(clf.predict([classdata]))
    #if key=='172.27.0.82': resultlist.append('node')
    #elif key=='172.27.0.15': resultlist.append('gate')
    #elif key=='172.27.0.90': resultlist.append('gate')
    #elif key=='172.27.0.78': resultlist.append('bridge')

    #elif key=='172.27.0.12': resultlist.append('bridge')
    #else: resultlist.append('node')
    try:
        arpdict=dict()
        with open("arpfile"+ip+".json") as f:
            arpdict=json.load(f)
        if key in arpdict:
            classdata.append(len(arpdict[key]))
        else:
            classdata.append(0)
    except:
            classdata.append(0)


            #datalist.append(classdata)
    pdata['tree']=classdata

    datalist.append(pdata)
    #classlist.append(classdata)
    if key in priolist:
        classlist.append(classdata)
        resultlist.append('prioritize')
    elif key in slowlist:

        classlist.append(classdata)
        resultlist.append('slow')
    else:
        #print("doing nothing")
        classlist.append(classdata)
        resultlist.append('middle')
        #if pdata['count']>500:
         # resultlist.append('slow')
        #elif pdata['count']<100:
        # resultlist.append('prioritize')

        #else: resultlist.append('middle')
print(str(datalist))
print(str(sublist))
manualforward=dict()
with open('tcdirs'+'.json','r')as f:
  manualforward= json.load(f)
shapelist=dict()
print(str(data))
os.system("sudo tc qdisc del dev "+interface+" root")
os.system("tc qdisc add dev "+interface+" root handle 1: htb default 30")
os.system("tc class add dev "+interface+" parent 1: classid 1:1 htb rate 6mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:10 htb rate 5mbit ceil 10mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:20 htb rate 4mbit ceil 6mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:30 htb rate 40kbit ceil 40kbit burst 10k")
for el in manualforward['prioritize']:
    shapelist[el]=dict()
    shapelist[el]['comm']=[]
    faststring1="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el+" flowid 1:10"
    faststring2="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el+" flowid 1:10"
    print(faststring1)
    os.system(faststring1)
    print(faststring2)
    os.system(faststring2)
    shapelist[el]['class']='prio'
    shapelist[el]['comm'].append(faststring1)
    shapelist[el]['comm'].append(faststring2)

for el in manualforward['slow']:
    shapelist[el]=dict()
    shapelist[el]['comm']=[]
    slowstring1="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el+" flowid 1:30"
    slowstring2="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el+" flowid 1:30"
    print(slowstring1)
    os.system(slowstring1)
    print(slowstring2)
    os.system(slowstring2)
    shapelist[el]['class']='slow'
    shapelist[el][’comm’].append(slowstring1)
    shapelist[el][’comm’].append(slowstring2)
clf = tree.DecisionTreeClassifier()
print("data list is "+str(datalist))
print("result list is "+str(resultlist))
clf = clf.fit(classlist, resultlist)
dump(clf, ’shapetree’+ip+’.joblib’)
clf = load(’shapetree’+ip+’.joblib’)
print(str(shapelist))
for el in datalist:
    print(el)
    if el[’ip’] in shapelist: continue
    try:
        print(str(el[’tree’]))
        print("prediction for "+str(el[’ip’])+" "+str(clf.predict([el[’tree’]])[0]))
        prediction=clf.predict([el[’tree’]])[0]
        if prediction == ’prioritize’:
            shapelist[el[’ip’]]=dict()
            shapelist[el[’ip’]][’comm’]=[]
            faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:10"
            print(faststring)
            shapelist[el[’ip’]][’class’]=’prio’
            shapelist[el[’ip’]][’comm’].append(faststring)

            os.system(faststring)
            faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:10"
            print(faststring)
            os.system(faststring)
            shapelist[el[’ip’]][’comm’].append(faststring)
        elif prediction==’slow’:
            shapelist[el[’ip’]]=dict()
            shapelist[el[’ip’]][’comm’]=[]
            shapelist[el[’ip’]][’class’]=’slow’
            slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:30"
            print(slowstring)
            os.system(slowstring)
            shapelist[el[’ip’]][’comm’].append(slowstring)
            slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:30"
            print(slowstring)
            os.system(slowstring)
            shapelist[el[’ip’]][’comm’].append(slowstring)
        else:
            shapelist[el[’ip’]]=dict()
            shapelist[el[’ip’]][’comm’]=[]
            ’’’
            shapelist[el[’ip’]]=dict()
            shapelist[el[’ip’]][’comm’]=[]
            midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:20"
            print(midstring)
            os.system(midstring)
            shapelist[el[’ip’]].append(midstring)
            midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:20"
            print(midstring)
            os.system(midstring)
            shapelist[el[’ip’]].append(midstring)
            '''

            if (el[’count’]> el[’length’] or el[’dests’]>4 or el[’count’]> 300) and (’class’ not in el or el[’class’] == ’node’) and (el[’ip’] not in prevlog or el[’ip’] in prevlog and prevlog[el[’ip’]][’class’] != ’slow’):

                slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:30"
                print(slowstring)
                os.system(slowstring)
                shapelist[el[’ip’]][’class’]=’slow’
                shapelist[el[’ip’]][’comm’].append(slowstring)
                slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:30"
                print(slowstring)
                os.system(slowstring)
                shapelist[el[’ip’]][’comm’].append(slowstring)
            elif el[’class’]==’gateway’ or el[’class’]==’bridge’:
                shapelist[el[’ip’]][’class’]=’prio’

                faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:10"
                print(faststring)
                os.system(faststring)
                shapelist[el[’ip’]][’comm’].append(faststring)
                faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:10"
                print(faststring)
                os.system(faststring)
            else:

                midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:20"
                print(midstring)
                os.system(midstring)
                midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:20"
                print(midstring)
                os.system(midstring)
                shapelist[el[’ip’]][’class’]=’med’



        continue
    except:
        print((traceback.format_exc()))
        print("faulire with tree")
        exit()

    if (el[’count’]> el[’length’] or el[’dests’]>4 or el[’count’]> 300) and (’class’ not in el or el[’class’] == ’node’):

        slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:30"
        print(slowstring)
        os.system(slowstring)
        slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+
            el[’ip’]+" flowid 1:30"
        print(slowstring)
        os.system(slowstring)
    elif el[’class’]==’gateway’ or el[’class’]==’bridge’:

        faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:10"
        print(faststring)
        os.system(faststring)
        faststring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:10"
        print(faststring)
        os.system(faststring)
        os.system(faststring)
    else:

        midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:20"
        print(midstring)
        os.system(midstring)
        os.system(midstring)
        midstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:20"
        print(midstring)
        os.system(midstring)
        os.system(midstring)


for el in sublist:
    if el[’count’]>300 and el[’ip’] not in shapelist:
        shapelist[el[’ip’]]=dict()
        shapelist[el[’ip’]][’comm’]=[]
        print("leaf prdction")
        slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+el[’ip’]+" flowid 1:30"
        print(slowstring)
        os.system(slowstring)
        shapelist[el[’ip’]][’comm’].append(slowstring)

        slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip src "+el[’ip’]+" flowid 1:30"
        print(slowstring)
        os.system(slowstring)
        shapelist[el[’ip’]][’comm’].append(slowstring)
with open("shapefile"+ip+".json",’w’) as outfile:
    print(json.dump(shapelist,outfile))
```

autoshape9.py

```
import traceback
import json
import pyshark
from collections import Counter
import os
import datetime
#cap = pyshark.LiveCapture(interface=’wlan1’)
#cap.sniff(packet_count=50)
from sklearn import tree
import netifaces as ni
from joblib import dump, load
```

```
import math
bridgeflag=0

def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print( str(interface))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
                     if 'br' in interface: bridgeflag=1
              except:
                     print( "eror reading interface")
              print( "chosen interface is "+str(chosenInterface))
       return chosenInterface
interface=chooseInterface()
ni.ifaddresses(interface)
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
clf = tree.DecisionTreeClassifier()
try:
    clf = load('tree'+ip+'.joblib')
except:
    #continue
    print("error reading data")

print(ip) # should print "192.168.100.37"
packetlog=dict();
userlog=dict();
storedlog=dict();
templog=dict();


import glob
for filename in glob.glob('iplog*'):
    print(filename)
    #if ip in filename: continue;
    #if ip in filename: continue;


    with open(filename,'r')as f:
     try:
       templog= json.load(f)
     except:
        continue
     #print((str(templog)))
     #storedlog={x: storedlog.get(x,0)+templog.get(x,0) for x in set(storedlog).union(templog)}
     #if templog[]
     for key,val in list(templog.items()):


              if key in storedlog:
                     print("key is in stored log "+str(key))
                     if '172.27.0' not in key: continue
                     if not isinstance(storedlog[key],dict): storedlog[key]=dict()
                     if 'remoteadded' not in storedlog[key]: stored[key]['remoteadded']=list()
                     if isinstance(storedlog[key],dict) and 'remoteadded' in storedlog[key]:
                        if isinstance(storedlog[key]['remoteadded'],list):
```

```
        #if filename[(filename.find('g')+1):(filename.find('.j'))] not in storedlog[key]['remoteadded']: storedlog[key]['remoteadded'].append(filename[(filename.find('g')+1):(filename.find('.j'))])
        if 'originip' in templog and templog['originip'] not in storedlog[key]['remoteadded']: storedlog[key]['remoteadded'].append(templog['originip'])

if 'remoteadded'in val:
    for ij in val['remoteadded']:
        if ij not in storedlog[key]['remoteadded']:storedlog[key]['remoteadded'].append(ij)





if 'route' not in storedlog[key]:
    storedlog[key]['route']=[]
if 'route' in storedlog[key] and 'route' in val:

    #storedlog[key]['route']=val['route']+storedlog[key]['route']
    for il in val['route']:
        if il not in storedlog[key]['route']: storedlog[key]['route'].append(il)
    #if filename[(filename.find('g')+1):(filename.find('.j'))] not in storedlog[key]['route']: storedlog[key]['route'].append(filename[(filename.find('g')+1):(filename.find('.j'))]) #remove later
if 'route' in storedlog[key] and 'route' not in val:
    print("adding file name")

    #if filename[(filename.find('g')+1):(filename.find('.j'))] not in storedlog[key]['route']: storedlog[key]['route'].append(filename[(filename.find('g')+1):(filename.find('.j'))])

#print("debug route "+str(storedlog[key]['route']))
#storedlog[key]['remoteadded']=filename[5:17]
#try:
#print "doing addition "+str(storedlog[key])+" and "+str(val)
'''
storedlog[key]['count']= storedlog[key]['count']+val['count']
for key1,val1 in list(val.items()):
    if not isinstance(val1,dict) or 'count' not in val1:
        storedlog[key][key1]=val1
        continue
    print("key 1 is "+str(key1)+" val 1 is"+str(val1))
    if key1=='count': continue
    if key1 in storedlog[key] and 'count' in storedlog[key][key1]:
        storedlog[key][key1]=val1
        if storedlog[key][key1]['count']!=val1['count']:
            storedlog[key][key1]['count']= storedlog[key][key1]['count']+val1['count']
        else:
            storedlog[key][key1]['count']= storedlog[key][key1]['count']
    elif key1 in storedlog[key]:
        storedlog[key][key1]=val1
'''



```

```
                            #print "result is "+str(storedlog[key])
                        #except:
                            #print "error doing addition"
                            #exit()


                else:
                    print("adding new key to stored log")
                    #val['remoteadded']=filename[5:17]
                    if isinstance(val,dict) and 'remoteadded' in val:
                        if isinstance(val['remoteadded'],list):
                            val['remoteadded'].append(templog['originip'])
                        else:

                            val['remoteadded']=[templog['originip']]
                    elif isinstance(val,dict):
                        if 'originip' in templog: val['remoteadded']=[templog['originip']]
                        else:val['remoteadded']=[]# fix later

                    print("adding value "+str(val))
                    storedlog[key]=val
    #os.system('mv '+filename+' '+'oldlog'+filename[4:])

    print("printing new log")
    print((json.dumps(storedlog)))

userlog=dict(storedlog)
#if ip in storedlog: userlog=storedlog[ip]
storedlogbackup=dict(storedlog)
#with open("datathing1.json",'r')as f:
# userlog= json.load(f)

externpacketlog=dict()
#with open("externthing1.json",'r')as f:
# externpacketlog= json.load(f)
#print json.dumps(userlog)
internpacketlog=dict()
#with open("internthing1.json",'r')as f:
# internpacketlog= json.load(f)
#userlog=dict();
print((json.dumps(storedlog)));
#print json.dumps(userlog);
#print json.dumps(internpacketlog)
#print json.dumps(externpacketlog)
print("sniffing packets")
errorcount =0;
cap = pyshark.LiveCapture(interface=interface)
print("sniffed packets")
cap.sniff(packet_count=500)
for packet in cap:
    ipstr='ip'
    if ipstr not in packet: ipstr='ipv6'
    #print packet
    #print packet.highest_layer
    try:
       if ('tcp' in packet or 'udp' in packet) and ('172.27.0' in str(packet[ipstr].src)):
         print("key is "+str(packet[ipstr].src))
         #print(packet)
         #print packet.highest_layer
         #print packet.tcp.dstport
         print (packet[ipstr].src)

         try:
            if packet[ipstr].src in userlog:
                        print("adding to dict again ")
                        #packetlog[packet.tcp.dstport] = 1+packetlog[packet.tcp.dstport];
                        print(("added to user log "+str(packet[ipstr].src)+" "+str(userlog[packet[ipstr].src])))
                        userlog[packet[ipstr].src]['count'] = userlog[packet[ipstr].src]['count']+1;
                        #userlog[packet[ipstr].src]['length'] = int(packet.length)
                        userlog[packet[ipstr].src]['totalsize'] = int(packet.length)+int(userlog[packet[ipstr].src]['totalsize'])
                        userlog[packet[ipstr].src]['length'] = int(userlog[packet[ipstr].src]['totalsize']/userlog[packet[ipstr].src]['count'])
                        #userlog[packet[ipstr].src]['time'].append(str((packet.sniff_timestamp)))
                        if 'route' not in userlog[packet[ipstr].src] and packet[ipstr].src != ip and packet[ipstr].dst != ip:
                           print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            print("adding route")
                            userlog[packet[ipstr].src]['route']=[ip]
                        elif packet[ipstr].src != ip and packet[ipstr].dst != ip and ip not in userlog[packet[ipstr].src]['route']:
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            print("adding route")
                            userlog[packet[ipstr].src]['route'].append(ip)
                        if packet.highest_layer not in userlog[packet[ipstr].src]['protocol']:
                            print("adding new protocol")
                            userlog[packet[ipstr].src]['protocol'][packet.highest_layer]=1
                        else:
                            userlog[packet[ipstr].src]['protocol'][packet.highest_layer]=userlog[packet[ipstr].src]['protocol'][packet.highest_layer]+1


                        if packet[ipstr].dst in userlog[packet[ipstr].src]:
                               #userlog[packet[ipstr].src]+1;
                               print(("incrementing user log before "+str(userlog[packet[ipstr].src][packet[ipstr].dst])))
                               userlog[packet[ipstr].src][packet[ipstr].dst]['count']= userlog[packet[ipstr].src][packet[ipstr].dst]['count']+1;
                               print(("incrementing user log after "+str(userlog[packet[ipstr].src][packet[ipstr].dst])))
                               if 'time' in userlog[packet[ipstr].src][packet[ipstr].dst]: userlog[packet[ipstr].src][packet[ipstr].dst]['time'].append(str(packet.sniff_timestamp))
                               else: userlog[packet[ipstr].src][packet[ipstr].dst]['time']= [str(packet.sniff_timestamp)];
                        else:
                            userlog[packet[ipstr].src][packet[ipstr].dst]= dict();
                            userlog[packet[ipstr].src][packet[ipstr].dst]['count']= 1;
                            userlog[packet[ipstr].src][packet[ipstr].dst]['time']= [str(packet.sniff_timestamp)];

                        #if packet[ipstr].src== "172.27.0.90":
                        #if packet[ipstr].src == "172.27.0.90" or packet[ipstr].src== "172.27.0.155" or packet[ipstr].src == "172.27.0.15" or packet[ipstr].src =="172.27.0.227":
                            #internpacketlog[packet[ipstr].src] = internpacketlog[packet[ipstr].src]+1;
                        #else:
                            #externpacketlog[packet[ipstr].dst] = externpacketlog[packet[ipstr].dst]+1;

            else:
                        #print "adding to dict"
                        #packetlog[packet.tcp.dstport]=1;

                        #userlog[packet[ipstr].src]=1;
                        userlog[packet[ipstr].src]=dict()
                        userlog[packet[ipstr].src]['count']=1
                        userlog[packet[ipstr].src]['totalsize']=packet.length
                        userlog[packet[ipstr].src]['protocol']=dict()
                        userlog[packet[ipstr].src]['protocol'][packet.highest_layer]=1
                        userlog[packet[ipstr].src][packet[ipstr].dst]= dict();
                        userlog[packet[ipstr].src][packet[ipstr].dst]['count']= 1;
                        if 'route' not in userlog[packet[ipstr].src] and packet[ipstr].src != ip and packet[ipstr].dst != ip:
                            print("adding route")
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].src]['route']=[ip]
                        elif packet[ipstr].src != ip and packet[ipstr].dst != ip and ip not in userlog[packet[ipstr].src]['route']:
                            print("adding route")
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].src]['route'].append(ip)

                        '''
                        elif packet[ipstr].src != ip:
                            print("appending route")
                            userlog[packet[ipstr].src]['route'].append(ip)
                        '''
                        #else:
                            #userlog[packet[ipstr].src]['route'].append(ip)


                        userlog[packet[ipstr].src]['port']=dict()
                        #userlog[packet[ipstr].src]['time']=[str(packet.sniff_timestamp)]
                        if 'dstport' in packet:
                            userlog[packet[ipstr].src]['port'][packet[ipstr].dstport]=1


         except:
            errorcount=errorcount+1;
            print("error with fields")
            print((traceback.format_exc()))
            #print packet
        if ('tcp' in packet or 'udp' in packet) and ('172.27.0' in str(packet[ipstr].dst)):
         print("key is "+str(packet[ipstr].dst))
         #print(packet)
         #print packet.highest_layer
         #print packet.tcp.dstport
         print (packet[ipstr].dst)

         try:
            if packet[ipstr].dst in userlog:
                        print("adding to dict again ")
```

```
                #packetlog[packet.tcp.dstport] = 1+packetlog[packet.tcp.dstport];
                print(("added to user log "+str(packet[ipstr].dst)+" "+str(userlog[packet[ipstr].dst])))
                userlog[packet[ipstr].dst][’count’] = userlog[packet[ipstr].dst][’count’]+1;
                #userlog[packet[ipstr].dst][’length’] = int(packet.length)
                userlog[packet[ipstr].dst][’totalsize’] = int(packet.length)+int(userlog[packet[ipstr].dst][’totalsize’])
                userlog[packet[ipstr].dst][’length’] = int(userlog[packet[ipstr].dst][’totalsize’]/userlog[packet[ipstr].dst][’count’])
                if ’route’ not in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and packet[ipstr].src != ip:
                   print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                   userlog[packet[ipstr].dst][’route’]=[ip]
                elif ’route’ in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and ip not in userlog[packet[ipstr].dst][’route’] and packet[ipstr].src != ip:
                   print("appending route ")
                   print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                   userlog[packet[ipstr].dst][’route’].append(ip)
                if packet.highest_layer not in userlog[packet[ipstr].dst][’protocol’]:
                   print("adding new protocol")
                   userlog[packet[ipstr].dst][’protocol’][packet.highest_layer]=1
                else:
                   userlog[packet[ipstr].dst][’protocol’][packet.highest_layer]=userlog[packet[ipstr].dst][’protocol’][packet.highest_layer]+1


                if packet[ipstr].src in userlog[packet[ipstr].dst]:
                      #userlog[packet[ipstr].dst]+1;
                      print(("incrementing user log before "+str(userlog[packet[ipstr].dst][packet[ipstr].src])))
                      userlog[packet[ipstr].dst][packet[ipstr].src][’count’]= userlog[packet[ipstr].dst][packet[ipstr].src][’count’]+1;
                      print(("incrementing user log after "+str(userlog[packet[ipstr].dst][packet[ipstr].src])))
                      if ’time’ in userlog[packet[ipstr].dst][packet[ipstr].src][’time’]: userlog[packet[ipstr].dst][packet[ipstr].src][’time’].append(str(packet.sniff_timestamp));
                      else: userlog[packet[ipstr].dst][packet[ipstr].src][’time’]= [str(packet.sniff_timestamp)];
                else:
                   userlog[packet[ipstr].dst][packet[ipstr].src]= dict();
                   userlog[packet[ipstr].dst][packet[ipstr].src][’count’]= 1;
                   userlog[packet[ipstr].dst][packet[ipstr].src][’time’]= [str(packet.sniff_timestamp)];

                #if packet[ipstr].src== "172.27.0.90":
                #if packet[ipstr].src == "172.27.0.90" or packet[ipstr].src== "172.27.0.155" or packet[ipstr].src == "172.27.0.15" or packet[ipstr].src =="172.27.0.227":
                   #internpacketlog[packet[ipstr].src] = internpacketlog[packet[ipstr].src]+1;
                #else:
                   #externpacketlog[packet[ipstr].dst] = externpacketlog[packet[ipstr].dst]+1;

        else:
                #print "adding to dict"
                #packetlog[packet.tcp.dstport]=1;

```

```
                        #userlog[packet[ipstr].src]=1;
                        userlog[packet[ipstr].dst]=dict()
                        userlog[packet[ipstr].dst]['count']=1
                        userlog[packet[ipstr].dst]['totalsize']=packet.length
                        userlog[packet[ipstr].dst]['protocol']=dict()
                        userlog[packet[ipstr].dst]['protocol'][packet.highest_layer]=1
                        userlog[packet[ipstr].dst][packet[ipstr].src]= dict();
                        userlog[packet[ipstr].dst][packet[ipstr].src]['count']= 1;
                        userlog[packet[ipstr].src]['port']=dict()
                        if 'route' not in userlog[packet[ipstr].dst] and packet[ipstr].dst != ip and packet[ipstr].src != ip:
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].dst]['route']=[ip]
                        elif packet[ipstr].dst != ip and packet[ipstr].src != ip and ip not in userlog[packet[ipstr].dst]['route']:
                            print("appending route " +str(ip)+" ")
                            print("adding route "+str(ip)+" "+str(packet[ipstr].src)+" "+str( packet[ipstr].dst))
                            userlog[packet[ipstr].dst]['route'].append(ip)

                        #userlog[packet[ipstr].src]['time']=[str(packet.sniff_timestamp)]
                        if 'srcport' in packet:
                            userlog[packet[ipstr].src]['port'][packet[ipstr].srcport]=1


         except:
            errorcount=errorcount+1;
            print("error with fields")
            print((traceback.format_exc()))
            #print packet

    except:
        print("error ")
        print((traceback.format_exc()))
        #print packet

#print json.dumps(packetlog)
#import netifaces as ni
#ni.ifaddresses('wlan1')
#ip = ni.ifaddresses('wlan1')[ni.AF_INET][0]['addr']
#print ip # should print "192.168.100.37
for key,val in list(userlog.items()):
    if not isinstance(val,dict)or 'route' not in val or not isinstance(val['route'],list): continue
    for el in val['route']:
        print("checking route "+el)
        routecount=0
        for key1,val1 in userlog.items():
            print(val1)
            if isinstance(val1,dict) and 'route' in val1 and isinstance(val1['route'],list) and el in val1['route'] and key1 != el:
                if len(val1['protocol'])>2 or val1['count']>20:
                    routecount=routecount+1+len(val1['protocol'])
        if el not in userlog:
            userlog[el]=dict()
        print("adding route count to "+str(el)+" which is "+str(routecount) )
        if 'routecount' not in userlog[el] or userlog[el]['routecount']<routecount: userlog[el]['routecount']=routecount

```

```
arplog=dict()
try:
   with open("fullarp"+ip+".json",'r')as f:
      arplog= json.load(f)
except:
   print("error reading file")
classifierinit=0
try:
   clf = load('tree'+ip+'.joblib')
   classifierinit=1
except:
   print("error with classifier")
for key,val in list(userlog.items()):
   if isinstance(val,dict) and 'route' in val:
      for el in list(val['route']):
         print("checking "+str(el))
         treepred='node'
         if el in userlog:
            if (el in userlog and 'class' in userlog[el] and userlog[el]['class'] != 'node') or classifierinit ==0 : continue
            try:
               '''
               pdata=[]
               val=userlog[el]
               if 'count' in val and 'length' in val: pdata.append(val['count']*int(math.log(val['length'])))
               else: pdata.append(0)
               if 'count' in val: pdata.append(val['count'])
               else: pdata.append(0)
               if 'length' in val: pdata.append(val['length'])
               else: pdata.append(0)
               #pdata.append(0)
               dests=0
               #if 'class' in val: pdata['class']=int(val['class'])
               #else: pdata.append(int('node'))
               if 'routecount' in val: pdata.append(val['routecount'])
               else: pdata.append(0)
               for key1,val1 in val.items():
                  print(key1)
                  if '172.27.0' in key1:
                     dests=dests+1
               pdata.append(dests)
               '''
               pdata=[]
               if 'length' in val:
                  pdata.append((val['count']*int(math.log(val['length']))))
                  pdata.append(val['count'])
                  pdata.append(val['length'])
                  #pdata.append(0)
                  dests=0
                  #if 'class' in val: pdata['class']=int(val['class'])
                  #else: pdata.append(int('node'))
                  if 'routecount' in val: pdata.append(val['routecount'])
                  else: pdata.append(0)
                  for key1,val1 in val.items():
                     #print(key1)
                     if '172.27.0' in key1:
                        dests=dests+1
                  pdata.append(dests)
```

```
                        try:
                            arpdict=dict()
                            with open("arpfile"+ip+".json") as f:
                                arpdict=json.load(f)
                            if key in arpdict:
                                pdata.append(len(arpdict[key]))
                            else:
                                pdata.append(0)
                        except:
                                pdata.append(0)

                    treepred=clf.predict([pdata])[0]
                    print("tree pred is"+treepred+" for "+str(el))
                    userlog[el][’class’]=treepred
                    #break
                    continue
                except:
                    if ’routecount’in userlog[el] and userlog[el][’routecount’]>6: userlog[el][’class’]=’gateway’
                    else: userlog[el][’class’]=’node’

            if el not in userlog and ’172.27.0.’ in el:

                userlog[el]=dict()
                userlog[el][’class’]=’bridge’
            elif ’172.27.0.’ in el:
                if ’routecount’in userlog[el] and userlog[el][’routecount’]>6: userlog[el][’class’]=’gateway’
                else: userlog[el][’class’]=’node’

for key,val in userlog.items():
    if isinstance(val,dict) and( ’class’ not in val or val[’class’]==’node’):
                if ’routecount’ in userlog[key] and userlog[key][’routecount’]>=3: userlog[key][’class’]=’gateway’
                else: userlog[key][’class’]=’node’
                for key1,val1 in val.items():
                    if ’172.27.0’ in key1 and key1 in userlog:
                        if ’routecount’ in userlog[key1] and userlog[key1][’routecount’]>=3: userlog[key1][’class’]=’gateway’
                        else: userlog[key1][’class’]=’node’
                    elif ’172.27.0’ in key1 and key1 not in userlog:userlog[key1][’class’]=’node’


savedLog= dict();
#storedlog=userlog;

print((json.dumps(storedlogbackup)))
print((json.dumps(storedlog)))
#print json.dumps(internpacketlog)
#print json.dumps(externpacketlog)
import random
dictid=random.randint(1,21)*5
userlog[’id’]=dictid
userlog[’originip’]=ip
userlog[’bridge’]=bridgeflag

packetfile=’iplog’+ip+’.json’;
with open(packetfile,’w’) as outfile:
    json.dump(userlog,outfile)
```

```
with open('datathing1.json','w') as outfile:
    json.dump(userlog,outfile)


import os
print("new val is")
os.system('cat '+packetfile)

print(("\n\n\nold val is "+str(storedlog)))
exit()
#os.system('cat internthing1.json')
#os.system('cat externthing1.json')
```

B.5 genarp5.py

```

import python_arptable
from python_arptable import ARPTABLE
import netifaces as ni
import json
import os
import socket

def valid_ip(address):
   try:
       socket.inet_aton(address)
       return True
   except:
       return False
def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print(( str(interface)))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
              except:
                     print( "eror reading interface")
              print(( "chosen interface is "+str(chosenInterface)))
       return chosenInterface
interface=chooseInterface()
gateway=''
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
for key,val in (ni.gateways()).items():
       if isinstance(val,list):
              for el in val:
                     if interface in el:
                            print("gateway is "+el[0])
                            gateway=el[0]
       else:
              if interface in val:
                     print("gateway is"+val[0])
                     gateway=val[0]
print("gateway is "+gateway)
if gateway == '':
    templog=dict()
    with open('iplog'+ip+'.json','r') as infile:
        templog=json.load(infile)
    print(templog)
    if ip in templog:
        if 'class' in templog[ip]:
            if templog[ip]['class']=='gateway' or templog[ip]['class']=='gate':
                gateway=ip
if gateway == '':
    gateway=ip
print(ARPTABLE)
arpdict=ARPTABLE
print(arpdict)
iplist=[gateway]

print(iplist)
ipdict=dict()
ipdict[ip]=iplist
print(ipdict)

print(ARPTABLE)
arpdict=ARPTABLE
print(arpdict)
iplist=[gateway]

#iplist.append(ip)
for di in arpdict:
    print(di)
    if '172.27.0' in di['IP address'] and di['IP address'] not in iplist and di['HW address'] != '00:00:00:00:00:00' and di['Device']==interface:
        response = os.system("ping -c 1 " + di['IP address'])
        print(response)
        if response ==0:
                print("hostup "+str(el))
                iplist.append(di['IP address'])
        else: print("error")

ipdict[ip]=iplist
print(iplist)
import glob
for filename in glob.glob('arpfile*'):
        print("filename")






        with open(filename,'r')as f:
                try:
                        templog= json.load(f)
                        print((str(templog)))
                        for key,val in list(templog.items()):
                                if valid_ip(key): ipdict[key]=val

                                for i in val:
                                        response = os.system("ping -c 1 " +i)
                                        print(response)
                                        if response ==0:
                                            if i not in iplist: iplist.append(i)
```

```


                      print("hostup "+i)

        except:
            continue

with open("arpfile"+ip+".json",'w') as outfile:
  print(json.dump(ipdict,outfile))
#file=open("iplist.txt","w")
#for ip in iplist:
# file.write(ip+",")
dictlist=dict()
dictlist['ips']=iplist
print(dictlist)
with open("iplist.json",'w') as outfile:
    print(json.dump(dictlist,outfile))

file=open("iplist.txt","w")
for ip in iplist:
    file.write(ip+",")
```

B.6 comparearpgraphs3.py

```
import traceback
import json
import pyshark
from collections import Counter
import os
import matplotlib
matplotlib.use('Agg')
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import json
import scipy
#cap = pyshark.LiveCapture(interface='wlan1')
#cap.sniff(packet_count=50)
def valid_ip(address):
  try:
    socket.inet_aton(address)
    return True
  except:
    return False
import netifaces as ni
def chooseInterface():
    import netifaces as ni
    chosenInterface='wlp3s0'
    for interface in ni.interfaces():
        print( str(interface))
        try:
            ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
            if '172.27.0.' in ip:
                chosenInterface=interface
        except:
            print( "eror reading interface")
        print( "chosen interface is "+str(chosenInterface))
        return chosenInterface
interface=chooseInterface()
ni.ifaddresses(interface)
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
print(ni.gateways())
gateway=''
for key,val in (ni.gateways()).items():
        if isinstance(val,list):
                for el in val:
                        if interface in el:
                                print("gateway is "+el[0])
                                gateway=el[0]
        else:
                if interface in val:
                        print("gateway is"+val[0])
                        gateway=val[0]
print("gateway is "+gateway)
#exit()
print(ip) # should print "192.168.100.37"
packetlog=dict();
userlog=dict();
storedlog=dict();
templog=dict();
fromlist=[]
tolist=[]



try:
        with open("arpfile"+ip+".json",'r')as f:
                templog= json.load(f)
                print((str(templog)))
                for key,val in list(templog.items()):
                        print("is key "+str(key))
                        if valid_ip(key) or '172.27.0' in key:
                                storedlog[key]=val
                                for i in val:
                                        if valid_ip(key) or '172.27.0' in key:
                                                fromlist.append(key)
                                                tolist.append(i)
except:
        exit()
        #continue
'''
with open("fullarp"+ip+".json",'w') as outfile:
    print(json.dump(storedlog,outfile))
'''
print(fromlist)
print(tolist)
df = pd.DataFrame({ 'from':fromlist, 'to':tolist})
df

# Build your graph. Note that we use the DiGraph function to create the graph!
G=nx.from_pandas_edgelist(df, 'from', 'to', create_using=nx.DiGraph() )

# Make the graph
pos = nx.spring_layout(G,scale=100,k=1,iterations=20)
#layout = nx.kamada_kawai_layout(G, dist=20)
'''
```

```
df = pd.DataFrame(index=G.nodes(), columns=G.nodes())
for row, data in nx.shortest_path_length(G):
    for col, dist in data.items():
        df.loc[row,col] = dist*2
df = df.fillna(df.max().max())
layout = nx.kamada_kawai_layout(G, dist=df.to_dict())
'''
nx.draw(G, layout=pos,with_labels=True,node_size=100, alpha=0.3, arrows=True, edge_color='black', node_color='black')
plt.axis("off")
#nx.draw(G, with_labels=True, node_size=1500, alpha=0.3, arrows=True)
#pos = nx.spring_layout(G,k=0.6,scale=4)
#plt.show()
#plt.draw()
plt.savefig("graph"+ip+".png",dpi=100)
#plt.show()
#plt.show()
plt.close()
```

B.7 pytc5.py

```
def chooseInterface():
       import netifaces as ni
       chosenInterface='mesh0'
       for interface in ni.interfaces():
              print(( str(interface)))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
              except:
                     print( "eror reading interface")
              print(( "chosen interface is "+str(chosenInterface)))
       return chosenInterface
interface=chooseInterface()
print("starting script")
import json
prioritizedict=dict()
with open('tcdirs.json','r')as f:
    prioritizedict= json.load(f)
print(json.dumps(prioritizedict))

prioritizelist=prioritizedict["prioritize"]
slowlist=prioritizedict["slow"]
print(prioritizelist)

for i in prioritizelist:
    print(i)

import os
import sys

os.system("sudo tc qdisc del dev "+interface+" root")
os.system("tc qdisc add dev "+interface+" root handle 1: htb default 30")
os.system("tc class add dev "+interface+" parent 1: classid 1:1 htb rate 6mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:10 htb rate 5mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k")
os.system("tc class add dev "+interface+" parent 1:1 classid 1:30 htb rate 10kbit ceil 10kbit burst 10k")
os.system("tc qdisc add dev "+interface+" parent 1:10 handle 10: sfq perturb 10")
os.system("tc qdisc add dev "+interface+" parent 1:20 handle 20: sfq perturb 10")
os.system("tc qdisc add dev "+interface+" parent 1:30 handle 30: sfq perturb 10")
os.system("tc filter add dev "+interface+" protocol ip parent 1: prio 1 u32 match ip dport 22 0xffff flowid 1:10")
os.system("tc filter add dev "+interface+" parent 1: protocol ip prio 1 u32 match ip dst 10.0.0.97 match ip dport 1234 0xffff flowid 1:30")

for i in prioritizelist:
   slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+i+" flowid 1:10"
   os.system(slowstring)
   print(slowstring)

for i in slowlist:
   slowstring="tc filter add dev "+interface+" parent 1: protocol ip prio 2 u32 match ip dst "+i+" flowid 1:30"
   os.system(slowstring)
   print(slowstring)
```

B.8 trafficplot7.py

```
#import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from io import StringIO
import json
import math
import netifaces as ni
import socket

def valid_ip(address):
   try:
       socket.inet_aton(address)
       return True
   except:
       return False
def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print(( str(interface)))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
              except:
                     print( "eror reading interface")
              print(( "chosen interface is "+str(chosenInterface)))
       return chosenInterface

interface=chooseInterface()
ni.ifaddresses(interface)
```

```
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
data=dict()
with open('iplog'+ip+'.json','r')as f:
  data= json.load(f)
print(str(data))
datalist=[]
for key,val in data.items():
    try:
        if not valid_ip(key): continue


        pdata=dict()
        #print(str(key)+" "+str(val['count']))
        pdata['ip']=key
        #if 'length' in val:
        if 'length'in val and 'count' in val: pdata['countlen']=val['count']*val['length']
        else: pdata['countlen']=0
        if 'count' in val:pdata['count']=val['count']
        else: pdata['count']=0
        if 'length'in val :pdata['length']=val['length']
        else: pdata['length']=0
        if 'routecount' in val: pdata['routes']=val['routecount']
        else: pdata['routes']=0
        datalist.append(pdata)
    #pdata['totalsize']=int((val['totalsize']))
      #else:
                #pdata['count']=val['count']
    except:
        #print(str(key)+" "+str(val['count']))
        continue;
    #print("appending to list "+str(pdata))
    #datalist.append(pdata)
'''
for ind,el in enumerate(datalist):
    print(str(el))
    routecount=0
    locip=el['ip']
    for key,val in data.items():
        print("is val "+str(val))
        if not isinstance(val,dict): continue
        if 'route' in val and locip in val['route']: routecount=routecount+1
    el['routes']=routecount
'''
print(str(datalist))
df = pd.DataFrame(datalist)
#ax=df.plot.add_subplot()
'''
ax1 = df1.plot()
ax2 = ax1.twinx()
ax2.spines['right'].set_position(('axes', 1.0))
df2.plot(ax=ax2)
ax3 = ax1.twinx()
ax3.spines['right'].set_position(('axes', 1.1))
df3.plot(ax=ax3)
#plt.show()
'''
'''
import pylab as pl
fig = pl.figure()
```

```
ax1 = pl.subplot(111,ylabel=’count’)
#ax2 = gcf().add_axes(ax1.get_position(), sharex=ax1, frameon=False, ylabel=’axes2’)
ax2 =ax1.twinx()
ax2.set_ylabel(’length’)
ax1.bar(df.index,df.count, width =0.4, color =’g’, align = ’center’)
ax2.bar(df.index,df.length, width = 0.4, color=’r’, align = ’edge’)
ax1.legend([’count’], loc = ’upper left’)
ax2.legend([’length’], loc = ’upper right’)
fig.show()
’’’
#fig = plt.figure() # Create matplotlib figure

#ax = fig.add_subplot(111) # Create matplotlib axes
#ax2 = ax.twinx() # Create another axes that shares the same x-axis as ax.

#width = 0.4

#df[’count’].plot(kind=’bar’, color=’red’,x=’ip’, ax=ax, width=width, position=1)
#df[’length’].plot(kind=’bar’, color=’blue’,x=’ip’ ,ax=ax2, width=width, position=0)

#ax.set_ylabel(’count’)
#ax2.set_ylabel(’length’)
plot1=df.plot(kind=’bar’,x=’ip’, y=[’count’,’routes’],figsize=(30,30),fontsize=25,secondary_y=’routes’)
#df.plot(kind=’bar’,secondary_y=True,x=’ip’,y=’length’)
fig=plot1.get_figure()
fig.savefig("griplog"+ip+".png")

#plt.show()
```

B.9 generatefullarp.py

```
import traceback
import json
import pyshark
from collections import Counter
import os
import python_arptable
from python_arptable import ARPTABLE

#cap = pyshark.LiveCapture(interface=’wlan1’)
#cap.sniff(packet_count=50)
import netifaces as ni
def chooseInterface():
       import netifaces as ni
       chosenInterface=’wlp3s0’
       for interface in ni.interfaces():
              print( str(interface))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0][’addr’]
                     if ’172.27.0.’ in ip:
                            chosenInterface=interface
              except:
                     print( "eror reading interface")
              print( "chosen interface is "+str(chosenInterface))
       return chosenInterface
interface=chooseInterface()
ni.ifaddresses(interface)
```

```
ip = ni.ifaddresses(interface)[ni.AF_INET][0][’addr’]
print(ni.gateways())
gateway=’’
for key,val in (ni.gateways()).items():
        if isinstance(val,list):
                for el in val:
                        if interface in el:
                                print("gateway is "+el[0])
                                gateway=el[0]
        else:
                if interface in val:
                        print("gateway is"+val[0])
                        gateway=val[0]
print("gateway is "+gateway)
#exit()
print(ip) # should print "192.168.100.37"
packetlog=dict();
userlog=dict();
storedlog=dict();
templog=dict();
fromlist=[]
tolist=[]
iplist=[gateway]
import glob
for filename in glob.glob(’arpfile*’):
    print(filename)
    #if ip in filename: continue;

    try:
        with open(filename,’r’)as f:
                templog= json.load(f)
                print((str(templog)))
                for key,val in list(templog.items()):
                        print(str(key)+" "+str(val))
                        if key != ip: storedlog[key]=val
                        elif key==ip and filename[filename.find(’e’)+1:filename.find(’.j’)]==ip:
                                storedlog[key]=val

                        ’’’
                        for i in val:
                                response = os.system("ping -c 1 " +i)
                                print(response)
                                if response ==0:
                                        print("hostup "+i)
                                        if i not in storedlog[ip]: storedlog[ip].append(i)
                                        if i not in iplist: iplist.append(i)
                                #iplist.append(di[’IP address’])
                                fromlist.append(key)
                                tolist.append(i)
                        ’’’
    except:
            print("error")
            continue

with open("fullarp"+ip+".json",’w’) as outfile:
    print(json.dump(storedlog,outfile))
arplog=dict()
with open("fullarp"+ip+".json",’r’)as f:
    arplog= json.load(f)
```

```
print(arplog)
import os
'''
for key,val in list(arplog.items()):
   response = os.system("ping -c 1 " + key)
   print(response)
   if response ==0: print("hostup")
   else: print("error")
   for el in list(val):
      response = os.system("ping -c 1 " + el)
      print(response)
      if response ==0: print("hostup "+str(el))
      else: print("error")
print(ARPTABLE)
arpdict=ARPTABLE
print(arpdict)
iplist=[gateway]
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
#iplist.append(ip)
for di in arpdict:
   print(di)
   if '172.27.0' in di['IP address'] and di['IP address'] not in iplist and di['HW address'] != '00:00:00:00:00:00':
      iplist.append(di['IP address'])
'''
print(iplist)
```

B.10 protocolplot.py

```
import pandas as pd
import json
import math
import json
import math
import netifaces as ni
import matplotlib
matplotlib.use('Agg')
def chooseInterface():
       import netifaces as ni
       chosenInterface='wlp3s0'
       for interface in ni.interfaces():
              print(( str(interface)))
              try:
                     ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
                     if '172.27.0.' in ip:
                            chosenInterface=interface
              except:
                     print( "eror reading interface")
              print(( "chosen interface is "+str(chosenInterface)))
       return chosenInterface

interface=chooseInterface()
ni.ifaddresses(interface)
ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']

data=dict()
with open('iplog'+ip+'.json','r')as f:
  data= json.load(f)
```

```
print(str(data))
datalist=[]
for key,val in data.items():
  try:
      pdata=dict()
      print(str(key)+" "+str(val['count']))
      for key1,val1 in val['protocol'].items():
        pdata['ipprot']=key+key1
        pdata['count']=val1
        datalist.append(pdata)
  except:
    continue


print(str(datalist))
df = pd.DataFrame(datalist)
print (df)
plot1=df.plot(kind='bar',x='ipprot', y='count',figsize=(100,20))
fig=plot1.get_figure()
fig.savefig("protgriplog"+ip+".png")
'''
print (df)
plot1=df.plot()
fig=plot1.get_figure()
fig.savefig("output.png")
'''
```

B.11 Frontend Angular Build

```
(window["webpackJsonp"] = window["webpackJsonp"] || []).push([["main"],{

/***/ "./node_modules/moment/locale sync recursive ^\\.\\/.*$":
/*!**************************************************!*\
  !*** ./node_modules/moment/locale sync ^\.\/.*$ ***!
  \**************************************************/
/*! no static exports found */
/***/ (function(module, exports, __webpack_require__) {

var map = {
  "./af": "./node_modules/moment/locale/af.js",
  "./af.js": "./node_modules/moment/locale/af.js",
  "./ar": "./node_modules/moment/locale/ar.js",
  "./ar-dz": "./node_modules/moment/locale/ar-dz.js",
  "./ar-dz.js": "./node_modules/moment/locale/ar-dz.js",
  "./ar-kw": "./node_modules/moment/locale/ar-kw.js",
  "./ar-kw.js": "./node_modules/moment/locale/ar-kw.js",
  "./ar-ly": "./node_modules/moment/locale/ar-ly.js",
  "./ar-ly.js": "./node_modules/moment/locale/ar-ly.js",
  "./ar-ma": "./node_modules/moment/locale/ar-ma.js",
  "./ar-ma.js": "./node_modules/moment/locale/ar-ma.js",
  "./ar-sa": "./node_modules/moment/locale/ar-sa.js",
  "./ar-sa.js": "./node_modules/moment/locale/ar-sa.js",
  "./ar-tn": "./node_modules/moment/locale/ar-tn.js",
  "./ar-tn.js": "./node_modules/moment/locale/ar-tn.js",
  "./ar.js": "./node_modules/moment/locale/ar.js",
  "./az": "./node_modules/moment/locale/az.js",
  "./az.js": "./node_modules/moment/locale/az.js",
  "./be": "./node_modules/moment/locale/be.js",
  "./be.js": "./node_modules/moment/locale/be.js",
  "./bg": "./node_modules/moment/locale/bg.js",
  "./bg.js": "./node_modules/moment/locale/bg.js",
  "./bm": "./node_modules/moment/locale/bm.js",
  "./bm.js": "./node_modules/moment/locale/bm.js",
  "./bn": "./node_modules/moment/locale/bn.js",
  "./bn.js": "./node_modules/moment/locale/bn.js",
  "./bo": "./node_modules/moment/locale/bo.js",
  "./bo.js": "./node_modules/moment/locale/bo.js",
  "./br": "./node_modules/moment/locale/br.js",
  "./br.js": "./node_modules/moment/locale/br.js",
  "./bs": "./node_modules/moment/locale/bs.js",
  "./bs.js": "./node_modules/moment/locale/bs.js",
  "./ca": "./node_modules/moment/locale/ca.js",
  "./ca.js": "./node_modules/moment/locale/ca.js",
  "./cs": "./node_modules/moment/locale/cs.js",
  "./cs.js": "./node_modules/moment/locale/cs.js",
  "./cv": "./node_modules/moment/locale/cv.js",
  "./cv.js": "./node_modules/moment/locale/cv.js",
  "./cy": "./node_modules/moment/locale/cy.js",
  "./cy.js": "./node_modules/moment/locale/cy.js",
  "./da": "./node_modules/moment/locale/da.js",
  "./da.js": "./node_modules/moment/locale/da.js",
  "./de": "./node_modules/moment/locale/de.js",
  "./de-at": "./node_modules/moment/locale/de-at.js",
  "./de-at.js": "./node_modules/moment/locale/de-at.js",
  "./de-ch": "./node_modules/moment/locale/de-ch.js",
  "./de-ch.js": "./node_modules/moment/locale/de-ch.js",
  "./de.js": "./node_modules/moment/locale/de.js",
  "./dv": "./node_modules/moment/locale/dv.js",
  "./dv.js": "./node_modules/moment/locale/dv.js",
  "./el": "./node_modules/moment/locale/el.js",
  "./el.js": "./node_modules/moment/locale/el.js",
  "./en-SG": "./node_modules/moment/locale/en-SG.js",
  "./en-SG.js": "./node_modules/moment/locale/en-SG.js",
  "./en-au": "./node_modules/moment/locale/en-au.js",
  "./en-au.js": "./node_modules/moment/locale/en-au.js",
  "./en-ca": "./node_modules/moment/locale/en-ca.js",
  "./en-ca.js": "./node_modules/moment/locale/en-ca.js",
  "./en-gb": "./node_modules/moment/locale/en-gb.js",
  "./en-gb.js": "./node_modules/moment/locale/en-gb.js",
  "./en-ie": "./node_modules/moment/locale/en-ie.js",
  "./en-ie.js": "./node_modules/moment/locale/en-ie.js",
  "./en-il": "./node_modules/moment/locale/en-il.js",
  "./en-il.js": "./node_modules/moment/locale/en-il.js",
  "./en-nz": "./node_modules/moment/locale/en-nz.js",
  "./en-nz.js": "./node_modules/moment/locale/en-nz.js",
  "./eo": "./node_modules/moment/locale/eo.js",
  "./eo.js": "./node_modules/moment/locale/eo.js",
  "./es": "./node_modules/moment/locale/es.js",
  "./es-do": "./node_modules/moment/locale/es-do.js",
  "./es-do.js": "./node_modules/moment/locale/es-do.js",
  "./es-us": "./node_modules/moment/locale/es-us.js",
  "./es-us.js": "./node_modules/moment/locale/es-us.js",
  "./es.js": "./node_modules/moment/locale/es.js",
  "./et": "./node_modules/moment/locale/et.js",
  "./et.js": "./node_modules/moment/locale/et.js",
  "./eu": "./node_modules/moment/locale/eu.js",
  "./eu.js": "./node_modules/moment/locale/eu.js",
  "./fa": "./node_modules/moment/locale/fa.js",
  "./fa.js": "./node_modules/moment/locale/fa.js",
  "./fi": "./node_modules/moment/locale/fi.js",
  "./fi.js": "./node_modules/moment/locale/fi.js",
  "./fo": "./node_modules/moment/locale/fo.js",
  "./fo.js": "./node_modules/moment/locale/fo.js",
  "./fr": "./node_modules/moment/locale/fr.js",
  "./fr-ca": "./node_modules/moment/locale/fr-ca.js",
  "./fr-ca.js": "./node_modules/moment/locale/fr-ca.js",
  "./fr-ch": "./node_modules/moment/locale/fr-ch.js",
  "./fr-ch.js": "./node_modules/moment/locale/fr-ch.js",
  "./fr.js": "./node_modules/moment/locale/fr.js",
  "./fy": "./node_modules/moment/locale/fy.js",
  "./fy.js": "./node_modules/moment/locale/fy.js",
  "./ga": "./node_modules/moment/locale/ga.js",
  "./ga.js": "./node_modules/moment/locale/ga.js",
  "./gd": "./node_modules/moment/locale/gd.js",
  "./gd.js": "./node_modules/moment/locale/gd.js",
  "./gl": "./node_modules/moment/locale/gl.js",
  "./gl.js": "./node_modules/moment/locale/gl.js",
  "./gom-latn": "./node_modules/moment/locale/gom-latn.js",
  "./gom-latn.js": "./node_modules/moment/locale/gom-latn.js",
  "./gu": "./node_modules/moment/locale/gu.js",
  "./gu.js": "./node_modules/moment/locale/gu.js",
  "./he": "./node_modules/moment/locale/he.js",
  "./he.js": "./node_modules/moment/locale/he.js",
  "./hi": "./node_modules/moment/locale/hi.js",
  "./hi.js": "./node_modules/moment/locale/hi.js",
  "./hr": "./node_modules/moment/locale/hr.js",
  "./hr.js": "./node_modules/moment/locale/hr.js",
  "./hu": "./node_modules/moment/locale/hu.js",
  "./hu.js": "./node_modules/moment/locale/hu.js",
  "./hy-am": "./node_modules/moment/locale/hy-am.js",
  "./hy-am.js": "./node_modules/moment/locale/hy-am.js",
  "./id": "./node_modules/moment/locale/id.js",
  "./id.js": "./node_modules/moment/locale/id.js",
  "./is": "./node_modules/moment/locale/is.js",
  "./is.js": "./node_modules/moment/locale/is.js",
  "./it": "./node_modules/moment/locale/it.js",
  "./it-ch": "./node_modules/moment/locale/it-ch.js",
  "./it-ch.js": "./node_modules/moment/locale/it-ch.js",
  "./it.js": "./node_modules/moment/locale/it.js",
  "./ja": "./node_modules/moment/locale/ja.js",
  "./ja.js": "./node_modules/moment/locale/ja.js",
  "./jv": "./node_modules/moment/locale/jv.js",
  "./jv.js": "./node_modules/moment/locale/jv.js",
  "./ka": "./node_modules/moment/locale/ka.js",
  "./ka.js": "./node_modules/moment/locale/ka.js",
  "./kk": "./node_modules/moment/locale/kk.js",
  "./kk.js": "./node_modules/moment/locale/kk.js",
  "./km": "./node_modules/moment/locale/km.js",
  "./km.js": "./node_modules/moment/locale/km.js",
  "./kn": "./node_modules/moment/locale/kn.js",
  "./kn.js": "./node_modules/moment/locale/kn.js",
  "./ko": "./node_modules/moment/locale/ko.js",
  "./ko.js": "./node_modules/moment/locale/ko.js",
  "./ku": "./node_modules/moment/locale/ku.js",
  "./ku.js": "./node_modules/moment/locale/ku.js",
  "./ky": "./node_modules/moment/locale/ky.js",
  "./ky.js": "./node_modules/moment/locale/ky.js",
  "./lb": "./node_modules/moment/locale/lb.js",
  "./lb.js": "./node_modules/moment/locale/lb.js",
  "./lo": "./node_modules/moment/locale/lo.js",
  "./lo.js": "./node_modules/moment/locale/lo.js",
  "./lt": "./node_modules/moment/locale/lt.js",
  "./lt.js": "./node_modules/moment/locale/lt.js",
  "./lv": "./node_modules/moment/locale/lv.js",
  "./lv.js": "./node_modules/moment/locale/lv.js",
  "./me": "./node_modules/moment/locale/me.js",
  "./me.js": "./node_modules/moment/locale/me.js",
  "./mi": "./node_modules/moment/locale/mi.js",
  "./mi.js": "./node_modules/moment/locale/mi.js",
  "./mk": "./node_modules/moment/locale/mk.js",
  "./mk.js": "./node_modules/moment/locale/mk.js",
  "./ml": "./node_modules/moment/locale/ml.js",
  "./ml.js": "./node_modules/moment/locale/ml.js",
  "./mn": "./node_modules/moment/locale/mn.js",
  "./mn.js": "./node_modules/moment/locale/mn.js",
  "./mr": "./node_modules/moment/locale/mr.js",
  "./mr.js": "./node_modules/moment/locale/mr.js",
  "./ms": "./node_modules/moment/locale/ms.js",
  "./ms-my": "./node_modules/moment/locale/ms-my.js",
  "./ms-my.js": "./node_modules/moment/locale/ms-my.js",
  "./ms.js": "./node_modules/moment/locale/ms.js",
  "./mt": "./node_modules/moment/locale/mt.js",
  "./mt.js": "./node_modules/moment/locale/mt.js",
  "./my": "./node_modules/moment/locale/my.js",
  "./my.js": "./node_modules/moment/locale/my.js",
  "./nb": "./node_modules/moment/locale/nb.js",
  "./nb.js": "./node_modules/moment/locale/nb.js",
  "./ne": "./node_modules/moment/locale/ne.js",
  "./ne.js": "./node_modules/moment/locale/ne.js",
  "./nl": "./node_modules/moment/locale/nl.js",
  "./nl-be": "./node_modules/moment/locale/nl-be.js",
  "./nl-be.js": "./node_modules/moment/locale/nl-be.js",
  "./nl.js": "./node_modules/moment/locale/nl.js",
  "./nn": "./node_modules/moment/locale/nn.js",
  "./nn.js": "./node_modules/moment/locale/nn.js",
  "./pa-in": "./node_modules/moment/locale/pa-in.js",
  "./pa-in.js": "./node_modules/moment/locale/pa-in.js",
  "./pl": "./node_modules/moment/locale/pl.js",
  "./pl.js": "./node_modules/moment/locale/pl.js",
  "./pt": "./node_modules/moment/locale/pt.js",
  "./pt-br": "./node_modules/moment/locale/pt-br.js",
  "./pt-br.js": "./node_modules/moment/locale/pt-br.js",
  "./pt.js": "./node_modules/moment/locale/pt.js",
  "./ro": "./node_modules/moment/locale/ro.js",
  "./ro.js": "./node_modules/moment/locale/ro.js",
  "./ru": "./node_modules/moment/locale/ru.js",
  "./ru.js": "./node_modules/moment/locale/ru.js",
  "./sd": "./node_modules/moment/locale/sd.js",
  "./sd.js": "./node_modules/moment/locale/sd.js",
  "./se": "./node_modules/moment/locale/se.js",
  "./se.js": "./node_modules/moment/locale/se.js",
  "./si": "./node_modules/moment/locale/si.js",
  "./si.js": "./node_modules/moment/locale/si.js",
  "./sk": "./node_modules/moment/locale/sk.js",
  "./sk.js": "./node_modules/moment/locale/sk.js",
  "./sl": "./node_modules/moment/locale/sl.js",
  "./sl.js": "./node_modules/moment/locale/sl.js",
  "./sq": "./node_modules/moment/locale/sq.js",
  "./sq.js": "./node_modules/moment/locale/sq.js",
  "./sr": "./node_modules/moment/locale/sr.js",
  "./sr-cyrl": "./node_modules/moment/locale/sr-cyrl.js",
  "./sr-cyrl.js": "./node_modules/moment/locale/sr-cyrl.js",
  "./sr.js": "./node_modules/moment/locale/sr.js",
  "./ss": "./node_modules/moment/locale/ss.js",
  "./ss.js": "./node_modules/moment/locale/ss.js",
  "./sv": "./node_modules/moment/locale/sv.js",
  "./sv.js": "./node_modules/moment/locale/sv.js",
  "./sw": "./node_modules/moment/locale/sw.js",
  "./sw.js": "./node_modules/moment/locale/sw.js",
  "./ta": "./node_modules/moment/locale/ta.js",
  "./ta.js": "./node_modules/moment/locale/ta.js",
  "./te": "./node_modules/moment/locale/te.js",
  "./te.js": "./node_modules/moment/locale/te.js",
  "./tet": "./node_modules/moment/locale/tet.js",
  "./tet.js": "./node_modules/moment/locale/tet.js",
  "./tg": "./node_modules/moment/locale/tg.js",
  "./tg.js": "./node_modules/moment/locale/tg.js",
  "./th": "./node_modules/moment/locale/th.js",
  "./th.js": "./node_modules/moment/locale/th.js",
  "./tl-ph": "./node_modules/moment/locale/tl-ph.js",
  "./tl-ph.js": "./node_modules/moment/locale/tl-ph.js",
  "./tlh": "./node_modules/moment/locale/tlh.js",
  "./tlh.js": "./node_modules/moment/locale/tlh.js",
  "./tr": "./node_modules/moment/locale/tr.js",
  "./tr.js": "./node_modules/moment/locale/tr.js",
  "./tzl": "./node_modules/moment/locale/tzl.js",
  "./tzl.js": "./node_modules/moment/locale/tzl.js",
  "./tzm": "./node_modules/moment/locale/tzm.js",
  "./tzm-latn": "./node_modules/moment/locale/tzm-latn.js",
  "./tzm-latn.js": "./node_modules/moment/locale/tzm-latn.js",
  "./tzm.js": "./node_modules/moment/locale/tzm.js",
  "./ug-cn": "./node_modules/moment/locale/ug-cn.js",
  "./ug-cn.js": "./node_modules/moment/locale/ug-cn.js",
  "./uk": "./node_modules/moment/locale/uk.js",
  "./uk.js": "./node_modules/moment/locale/uk.js",
  "./ur": "./node_modules/moment/locale/ur.js",
  "./ur.js": "./node_modules/moment/locale/ur.js",
  "./uz": "./node_modules/moment/locale/uz.js",
  "./uz-latn": "./node_modules/moment/locale/uz-latn.js",
  "./uz-latn.js": "./node_modules/moment/locale/uz-latn.js",
  "./uz.js": "./node_modules/moment/locale/uz.js",
  "./vi": "./node_modules/moment/locale/vi.js",
  "./vi.js": "./node_modules/moment/locale/vi.js",
  "./x-pseudo": "./node_modules/moment/locale/x-pseudo.js",
  "./x-pseudo.js": "./node_modules/moment/locale/x-pseudo.js",
  "./yo": "./node_modules/moment/locale/yo.js",
  "./yo.js": "./node_modules/moment/locale/yo.js",
  "./zh-cn": "./node_modules/moment/locale/zh-cn.js",
  "./zh-cn.js": "./node_modules/moment/locale/zh-cn.js",
  "./zh-hk": "./node_modules/moment/locale/zh-hk.js",
  "./zh-hk.js": "./node_modules/moment/locale/zh-hk.js",
  "./zh-tw": "./node_modules/moment/locale/zh-tw.js",
  "./zh-tw.js": "./node_modules/moment/locale/zh-tw.js"
};
```

```


function webpackContext(req) {
  var id = webpackContextResolve(req);
  return __webpack_require__(id);
}
function webpackContextResolve(req) {
  var id = map[req];
  if(!(id + 1)) { // check for number or string
    var e = new Error("Cannot find module ’" + req + "’");
    e.code = ’MODULE_NOT_FOUND’;
    throw e;
  }
  return id;
}
webpackContext.keys = function webpackContextKeys() {
  return Object.keys(map);
};
webpackContext.resolve = webpackContextResolve;
module.exports = webpackContext;
webpackContext.id = "./node_modules/moment/locale sync recursive ^\\.\\/.*$";

/***/ }),

/***/ "./src/$$_lazy_route_resource lazy recursive":
/*!**********************************************************!*\
  !*** ./src/$$_lazy_route_resource lazy namespace object ***!
  \**********************************************************/
/*! no static exports found */
/***/ (function(module, exports) {

function webpackEmptyAsyncContext(req) {
  // Here Promise.resolve().then() is used instead of new Promise() to prevent
  // uncaught exception popping up in devtools
  return Promise.resolve().then(function() {
    var e = new Error("Cannot find module ’" + req + "’");
    e.code = ’MODULE_NOT_FOUND’;
    throw e;
  });
}
webpackEmptyAsyncContext.keys = function() { return []; };
webpackEmptyAsyncContext.resolve = webpackEmptyAsyncContext;
module.exports = webpackEmptyAsyncContext;
webpackEmptyAsyncContext.id = "./src/$$_lazy_route_resource lazy recursive";

/***/ }),

/***/ "./src/app/app-routing.module.ts":
/*!***************************************!*\
  !*** ./src/app/app-routing.module.ts ***!
  \***************************************/
/*! exports provided: AppRoutingModule */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "AppRoutingModule",
    function() { return AppRoutingModule; });
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*!
    @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
/* harmony import */ var _angular_router__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! @angular/router */ "./node_modules/@angular/router/fesm5/router.js");
/* harmony import */ var _home_home_component__WEBPACK_IMPORTED_MODULE_2__ = __webpack_require__(/*! ./home/home.component */ "./src/app/home/home.component.ts");
var __decorate = (undefined && undefined.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length, r = c < 3 ? target : desc === null ? desc = Object.getOwnPropertyDescriptor(target, key) : desc, d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") r = Reflect.decorate(decorators, target, key, desc);
    else for (var i = decorators.length - 1; i >= 0; i--) if (d = decorators[i]) r = (c < 3 ? d(r) : c > 3 ? d(target, key, r) : d(target, key)) || r;
    return c > 3 && r && Object.defineProperty(target, key, r), r;
};



var routes = [
    { path: '', component: _home_home_component__WEBPACK_IMPORTED_MODULE_2__["HomeComponent"] },
];
var AppRoutingModule = /** @class */ (function () {
    function AppRoutingModule() {
    }
    AppRoutingModule = __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["NgModule"])({
            imports: [
                _angular_router__WEBPACK_IMPORTED_MODULE_1__["RouterModule"].forRoot(routes)
            ],
            exports: [
                _angular_router__WEBPACK_IMPORTED_MODULE_1__["RouterModule"]
            ]
        })
    ], AppRoutingModule);
    return AppRoutingModule;
}());



/***/ }),

/***/ "./src/app/app.component.css":
/*!***********************************!*\
  !*** ./src/app/app.component.css ***!
  \***********************************/
/*! no static exports found */
/***/ (function(module, exports) {

module.exports = ""

/***/ }),

/***/ "./src/app/app.component.html":
/*!************************************!*\
  !*** ./src/app/app.component.html ***!
  \************************************/
/*! no static exports found */
/***/ (function(module, exports) {

module.exports = "<router-outlet></router-outlet>\n\n"
```

```

/***/ }),

/***/ "./src/app/app.component.ts":
/*!***********************************!*\
  !*** ./src/app/app.component.ts ***!
  \***********************************/
/*! exports provided: AppComponent */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "AppComponent", function () { return AppComponent; });
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
var __decorate = (undefined && undefined.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length, r = c < 3 ? target : desc === null ? desc = Object.getOwnPropertyDescriptor(target, key) : desc, d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") r = Reflect.decorate(decorators, target, key, desc);
    else for (var i = decorators.length - 1; i >= 0; i--) if (d = decorators[i]) r = (c < 3 ? d(r) : c > 3 ? d(target, key, r) : d(target, key)) || r;
    return c > 3 && r && Object.defineProperty(target, key, r), r;
};

var AppComponent = /** @class */ (function () {
    function AppComponent() {
        this.title = 'meshFrontend';
    }
    AppComponent = __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["Component"])({
            selector: 'app-root',
            template: __webpack_require__(/*! ./app.component.html */ "./src/app/app.component.html"),
            styles: [__webpack_require__(/*! ./app.component.css */ "./src/app/app.component.css")]
        })
    ], AppComponent);
    return AppComponent;
}());



/***/ }),

/***/ "./src/app/app.module.ts":
/*!*******************************!*\
  !*** ./src/app/app.module.ts ***!
  \*******************************/
/*! exports provided: AppModule */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "AppModule", function() { return AppModule; });
/* harmony import */ var _angular_platform_browser__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! @angular/platform-browser */ "./node_modules/@angular/platform-browser/fesm5/platform-browser.js");
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
/* harmony import */ var _angular_forms__WEBPACK_IMPORTED_MODULE_2__ = __webpack_require__(/*! @angular/forms */ "./node_modules/@angular/forms/fesm5/forms.js");
/* harmony import */ var _app_component__WEBPACK_IMPORTED_MODULE_3__ = __webpack_require__(/*! ./app.component */ "./src/app/app.component.ts");
/* harmony import */ var _home_home_component__WEBPACK_IMPORTED_MODULE_4__ = __webpack_require__(/*! ./home/home.component */ "./src/app/home/home.component.ts");
/* harmony import */ var _app_routing_module__WEBPACK_IMPORTED_MODULE_5__ = __webpack_require__(/*! ./app-routing.module */ "./src/app/app-routing.module.ts");
/* harmony import */ var _angular_common_http__WEBPACK_IMPORTED_MODULE_6__ = __webpack_require__(/*! @angular/common/http */ "./node_modules/@angular/common/fesm5/http.js");
/* harmony import */ var _swimlane_ngx_graph__WEBPACK_IMPORTED_MODULE_7__ = __webpack_require__(/*! @swimlane/ngx-graph */ "./node_modules/@swimlane/ngx-graph/release/index.js");
/* harmony import */ var _swimlane_ngx_graph__WEBPACK_IMPORTED_MODULE_7___default = /*#__PURE__*/__webpack_require__.n(_swimlane_ngx_graph__WEBPACK_IMPORTED_MODULE_7__);
/* harmony import */ var _angular_platform_browser_animations__WEBPACK_IMPORTED_MODULE_8__ = __webpack_require__(/*! @angular/platform-browser/animations */ "./node_modules/@angular/platform-browser/fesm5/animations.js");
var __decorate = (undefined && undefined.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length, r = c < 3 ? target : desc === null ? desc = Object.getOwnPropertyDescriptor(target, key) : desc, d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") r = Reflect.decorate(decorators, target, key, desc);
    else for (var i = decorators.length - 1; i >= 0; i--) if (d = decorators[i]) r = (c < 3 ? d(r) : c > 3 ? d(target, key, r) : d(target, key)) || r;
    return c > 3 && r && Object.defineProperty(target, key, r), r;
};









var AppModule = /** @class */ (function () {
    function AppModule() {
    }
    AppModule = __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_1__["NgModule"])({
            declarations: [
                _app_component__WEBPACK_IMPORTED_MODULE_3__["AppComponent"],
                _home_home_component__WEBPACK_IMPORTED_MODULE_4__["HomeComponent"]
            ],
            imports: [
                _angular_platform_browser__WEBPACK_IMPORTED_MODULE_0__["BrowserModule"],
                _app_routing_module__WEBPACK_IMPORTED_MODULE_5__["AppRoutingModule"],
                _angular_common_http__WEBPACK_IMPORTED_MODULE_6__["HttpClientModule"],
                _swimlane_ngx_graph__WEBPACK_IMPORTED_MODULE_7__["NgxGraphModule"],
                _angular_platform_browser_animations__WEBPACK_IMPORTED_MODULE_8__["BrowserAnimationsModule"],
                _angular_forms__WEBPACK_IMPORTED_MODULE_2__["FormsModule"]
            ],
            schemas: [_angular_core__WEBPACK_IMPORTED_MODULE_1__["CUSTOM_ELEMENTS_SCHEMA"]],
            providers: [],
            bootstrap: [_app_component__WEBPACK_IMPORTED_MODULE_3__["AppComponent"]]
        })
```

```
	], AppModule);
	return AppModule;
}());



/***/ }),

/***/ "./src/app/home/home.component.css":
/*!*****************************************!*\
  !*** ./src/app/home/home.component.css ***!
  \*****************************************/
/*! no static exports found */
/***/ (function(module, exports) {

module.exports = ".graph-container {\n\theight: 700px; \n\twidth: 70%;\n\tmargin: 0px auto;\n}\n\n.node:hover {\n\tcursor:pointer;\n}\n\n.selectedTab {\n\tbackground-color: #A3ADC4;\n}\n\n.navButtons {\n\tmargin:0px auto;\n\ttext-align: center;\n\tmargin-top: 25px;\n\tmargin-bottom: 25px;\n}\n\n.navButton {\n\tmargin-left: 5px;\n\tmargin-right: 5px;\n\tfont-size: 30px;\n\tborder-radius: 5px;\n\tbox-shadow: 2px 2px 2px 2px #888888;\n}\n\n.navButton:hover {\n\tcursor: pointer;\n}\n\n.navButton:focus {\n\toutline: none;\n}\n\n#mynetwork {\n border: 5px solid #05B2DC;\n border-radius: 20px;\n background: #dddddd;\n display: inline-block;\n height: 500px;\n width: 70%;\n box-shadow: 5px 5px 5px 5px #888888;\n}\n\n.exit {\n\tfloat: right;\n}\n\n.network {\n\ttext-align: center;\n}\n\n.bottomBut {\n\ttext-align: center;\n\tdisplay: block;\n\tmargin:0px auto;\n\tfont-size: 24px;\n}\n\n.view-data-modal {\n\tposition: absolute;\n\ttop: 50%;\n\tleft: 50%; \n\t-webkit-transform: translate(-50%, -50%); \n\t transform: translate(-50%, -50%);\n\tbackground-color: #efefef;\n\ttext-align: center;\n\twidth: 90%;\n\theight: 80%;\n\tborder: 5px solid #05B2DC;\n\tborder-radius: 5px;\n}\n\n.view-data-title {\n\tfont-size: 45px;\n\tcolor: #0270B1;\n\tmargin-top: 4px;\n\tmargin-bottom: 4px;\n}\n\n.dataTab {\n\tbackground-color: #efefef;\n}\n\n.barChart {\n\theight: 80%;\n\twidth: 45%;\n\tdisplay: inline-block;\n}\n\n.piChart {\n\theight: 80%;\n\twidth: 45%;\n\tdisplay: inline-block;\n}\n\n.chartContainer {\n\twidth: 50%;\n\tdisplay: inline-block;\n\tvertical-align: top;\n}\n\n.piChartContainer {\n\twidth: 50%;\n\tdisplay: inline-block;\n\tvertical-align: top;\n}\n\n.tcTab {\n\ttext-align: center;\n}\n\n.inputContainer {\n\tdisplay: block;\n\twidth: 100%;\n\tmargin: 0px auto;\n\ttext-align: center;\n\tmargin-top: 20px;\n\tmargin-bottom: 20px;\n\tfont-size: 20px;\n}\n\n.lists {\n\twidth: 70%;\n\tmargin: 0px auto;\n}\n\n.ipList {\n\tdisplay: inline-block;\n\tvertical-align: top;\n\tmargin-left: 20px;\n\tmargin-right: 20px;\n\tborder: 4px solid #05B2DC;\n\twidth: 40%;\n\tbackground-color: #dddddd;\n\tborder-radius: 10px;\n}\n\n.listTitle {\n\tcolor: #5B6272;\n}"

/***/ }),

/***/ "./src/app/home/home.component.html":
/*!******************************************!*\
  !*** ./src/app/home/home.component.html ***!
  \******************************************/
/*! no static exports found */
/***/ (function(module, exports) {

module.exports = "<!-- <button (click)=\"test()\">Test</button> -->\n\n<div class=\"navButtons\">\n\t<button [ngClass]=\"{'selectedTab': currentTab == 'network'}\" class=\"navButton\" (click)=\"currentTab='network'; closeModal()\">Network</button>\n\t<button [ngClass]=\"{'selectedTab': currentTab == 'data'}\" class=\"navButton\" (click)=\"currentTab='data'\">Data</button>\n\t<button [ngClass]=\"{'selectedTab': currentTab == 'tc'}\" class=\"navButton\" (click)=\"currentTab='tc'\">Traffic Control</button>\n</div>\n\n<div [hidden]=\"currentTab != 'network'\" class=\"network\">\n\t<div #graphContainer id=\"mynetwork\" class=\"graph-container\"></div>\n</div>\n\n<div [hidden]=\"!viewingIp || currentTab != 'network'\" class=\"view-data-modal\"> \n\t<div class=\"exit\">\n\t\t<button (click)=\"closeModal()\">X</button>\n\t</div>\n\t<h2 class=\"view-data-title\">\n\t\t{{viewingIp}}\n\t</h2>\n\t<div class=\"barChart\">\n\t\t<canvas id
```

```
    =\"trafficChart\">{{packetChart}}</canvas>\n\t</div>\n\t<div class=\"piChart\">\n\t\t<canvas id
    =\"protChart\">{{protChart}}</canvas>\n\t</div>\n\t<button class=\"bottomBut\" (click)=\"
    closeModal()\">Close</button>\n</div>\n\n<div [hidden]=\"currentTab != 'data'\" class=\"dataTab
    \">\n\t<div class=\"chartContainer\">\n\t\t<canvas id=\"traffic\">{{testChart}}</canvas>\n\t\t<
    canvas id=\"length\">{{lengthChart}}</canvas>\n\t</div>\n\t<div class=\"piChartContainer\">\n\t
    \t<canvas id=\"protocols\">{{totalProtChart}}</canvas>\n\t\t<canvas id=\"externals\">{{
    externalChart}}</canvas>\n\t</div>\n</div>\n\n<div class=\"tcTab\" [hidden]=\"currentTab != 'tc
    '\">\n\t<div class=\"lists\">\n\t\t<div class=\"ipList\">\n\t\t\t<div class=\"inputContainer
    \">\n\t\t\t\t<label>Throttle IP:</label>\n\t\t\t\t<input [(ngModel)]=\"throttleInput\" name=\"
    ti\">\n\t\t\t\t<button (click)=\"throttleIP()\">Submit</button>\n\t\t\t</div>\n\t\t\t<h1 class
    =\"listTitle\">Throttled IPs</h1>\n\t\t\t<p *ngFor=\"let ip of slowedIps\">{{ip}}</p>\n\t\t</
    div>\n\t\t<div class=\"ipList\">\n\t\t\t<div class=\"inputContainer\">\n\t\t\t\t<label>
    Prioritize IP:</label>\n\t\t\t\t<input [(ngModel)]=\"prioritizeInput\" name=\"pi\">\n\t\t\t\t<
    button (click)=\"prioritizeIP()\">Submit</button>\n\t\t\t</div>\n\t\t\t<h1 class=\"listTitle\">
    Prioritized IPs</h1>\n\t\t\t<p *ngFor=\"let ip of priorityIps\">{{ip}}</p>\n\t\t</div>\n\t</div
    >\n</div>"

/***/ }),

/***/ "./src/app/home/home.component.ts":
/*!****************************************!*\
  !*** ./src/app/home/home.component.ts ***!
  \****************************************/
/*! exports provided: HomeComponent */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "HomeComponent", function
    () { return HomeComponent; });
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*!
    @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
/* harmony import */ var _mesh_service__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! ../
    mesh.service */ "./src/app/mesh.service.ts");
/* harmony import */ var chart_js__WEBPACK_IMPORTED_MODULE_2__ = __webpack_require__(/*! chart.js
    */ "./node_modules/chart.js/dist/Chart.js");
/* harmony import */ var chart_js__WEBPACK_IMPORTED_MODULE_2___default = /*#__PURE__*/__webpack_
    require__.n(chart_js__WEBPACK_IMPORTED_MODULE_2__);
/* harmony import */ var vis__WEBPACK_IMPORTED_MODULE_3__ = __webpack_require__(/*! vis */ "./node_
    modules/vis/dist/vis.js");
/* harmony import */ var vis__WEBPACK_IMPORTED_MODULE_3___default = /*#__PURE__*/__webpack_require
    __.n(vis__WEBPACK_IMPORTED_MODULE_3__);
var __decorate = (undefined && undefined.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length, r = c < 3 ? target : desc === null ? desc = Object.
        getOwnPropertyDescriptor(target, key) : desc, d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") r = Reflect.decorate
        (decorators, target, key, desc);
    else for (var i = decorators.length - 1; i >= 0; i--) if (d = decorators[i]) r = (c < 3 ? d(r)
        : c > 3 ? d(target, key, r) : d(target, key)) || r;
    return c > 3 && r && Object.defineProperty(target, key, r), r;
};
var __metadata = (undefined && undefined.__metadata) || function (k, v) {
    if (typeof Reflect === "object" && typeof Reflect.metadata === "function") return Reflect.
        metadata(k, v);
};
```

```
/*
    Pi IPs:
    172.27.0.15 (gateway) x
    172.27.0.32 (access point) x
    172.27.0.12 (routing)
    172.27.0.90 (access point)
    172.27.0.78 (routing) x
*/
var HomeComponent = /** @class */ (function () {
    function HomeComponent(meshService) {
        this.meshService = meshService;
        this.trafficChart = [];
        this.lengthChart = [];
        this.totalProtChart = [];
        this.externalChart = [];
        this.ips = ["172.27.0.15", "172.27.0.152", "172.27.0.12", "172.27.0.90", "172.27.0.78"];
        this.runningIps = [];
        this.IPChart = [];
        this.packetChart = [];
        this.protChart = [];
        //traffic control
        this.slowedIps = ["172.27.0.68", "172.27.0.178"];
        this.priorityIps = ["172.27.0.15"];
        this.graphData = {};
    }
    HomeComponent.prototype.ngOnInit = function () {
        //first, get IPs and Link Files
        //Second get Packet Data
        this.currentTab = "network";
        // this.networkData = {"172.27.0.15": {"172.27.0.90": {"count": 287}, "totalsize": 64440, "
            count": 287, "protocol": {"TCP": 240, "DATA-TEXT-LINES": 2, "JSON": 19, "HTTP": 26}, "
            length": 224},
        // "172.27.0.78": {"totalsize": 630244, "port": {}, "224.0.0.251": {"count": 389}, "count":
            5095, "172.27.0.90": {"count": 4618}, "172.27.255.255": {"count": 3}, "35.222.85.5": {"
            count": 11}, "35.224.99.156": {"count": 74}, "protocol": {"TCP": 137, "NBNS": 3, "DATA-
            TEXT-LINES": 1, "HTTP": 24, "_WS.MALFORMED": 1, "JSON": 7, "DNS": 4922}, "length": 123},
        // "172.27.0.90": {"totalsize": 764260, "port": {}, "172.27.0.82": {"count": 4618},
            "172.27.0.115": {"count": 10}, "count": 5313, "172.27.0.78": {"count": 287},
            "172.27.0.135": {"count": 62}, "172.27.0.32": {"count": 336}, "protocol": {"TCP": 540, "
            DATA-TEXT-LINES": 3, "HTTP": 59, "_WS.MALFORMED": 1, "BOOTP": 116, "JSON": 51, "DNS":
            4543}, "length": 143},
        // "172.27.0.32": {"totalsize": 86898, "port": {}, "count": 336, "172.27.0.90": {"count":
            336}, "protocol": {"TCP": 232, "BOOTP": 54, "JSON": 25, "HTTP": 25}, "length": 258},
        // "172.27.0.12": {"172.27.0.90": {"count": 62}, "totalsize": 21204, "count": 62, "protocol
            ": {"BOOTP": 62}, "length": 342},
        // "172.27.0.115": {"173.194.12.91": {"count": 12}, "protocol": {"QUIC": 410, "TCP": 48, "
            SSL": 17, "HTTP": 16, "XML": 1, "DNS": 12}, "224.0.0.251": {"count": 2}, "172.217.6.78":
             {"count": 31}, "port": {}, "172.27.0.90": {"count": 10}, "239.255.255.250": {"count":
            15}, "172.217.6.68": {"count": 19}, "length": 803, "172.217.0.42": {"count": 35},
            "74.125.170.123": {"count": 11}, "74.125.170.122": {"count": 298}, "count": 504,
            "52.49.211.202": {"count": 13}, "216.58.194.174": {"count": 31}, "216.58.195.67": {"
            count": 5}, "172.217.5.106": {"count": 15}, "totalsize": 405081, "216.58.195.68": {"
            count": 7}}};
        this.networkLinks = { "172.27.0.15": ["172.27.0.90", "172.27.0.12", "172.27.0.152",
            "172.27.0.78"], "172.27.0.152": ["172.27.0.12"], "172.27.0.12": ["172.27.0.90"],
            "172.27.0.90": ["172.27.0.78"] };
        this.getData();
    };
    HomeComponent.prototype.getTrafficControl = function () {
        //General logic:
        //Initial:
        //Parse JSON Data and get array of slowed and prioritized IPs
        //Add IP:
        //Add new IP to current list, add to JSON Object, send to server to update
        //Remove IP:
        //Remove from current list, remove from JSON Object, send to server to update file
        //Make a call to API endpoint to get a list of the currently slowed IPs
        //Make a call to API endpoint to get a list of the currently prioritized IPs
    };
    HomeComponent.prototype.throttleIP = function () {
    };
    HomeComponent.prototype.prioritizeIP = function () {
    };
    HomeComponent.prototype.createNetGraph = function () {
        var nodes = new vis__WEBPACK_IMPORTED_MODULE_3__["DataSet"]();
        nodes.add({ id: "Internet", x: 0, y: -600, shape: 'image', image: '../../assets/images/internet.png', size: 50 });
        var x = 0;
        var y = -500;
        for (var i = 0; i < this.ips.length; i++) {
            nodes.add({ id: this.ips[i], label: this.ips[i], x: x, y: y });
            if (i == 0) {
                x = 300;
                y = -300;
            }
            else {
                x -= 200;
                y = -300;
            }
        }
        // create an array with edges
        var edges = new vis__WEBPACK_IMPORTED_MODULE_3__["DataSet"]();
        edges.add({ from: "Internet", to: "172.27.0.15" });
        for (var key in this.networkLinks) {
            for (var i = 0; i < this.networkLinks[key].length; i++) {
                edges.add({ from: key, to: this.networkLinks[key][i], color: 'red' });
            }
        }
        // create a network
        var container = this.graph.nativeElement;
        // provide the data in the vis format
        var data = {
            nodes: nodes,
            edges: edges
        };
        var width = 400;
        var height = 400;
        var options = {
            edges: {
                color: {
                    color: 'white',
                    highlight: 'blue'
                },
                smooth: true,
                length: 300,
                width: 8
            },
            interaction: {
                dragNodes: false,
                dragView: false,
                hover: true,
                zoomView: false
            },
            physics: {
                stabilization: true,
            },
            nodes: {
                fixed: {
                    x: true,
                    y: true
                },
                shape: 'box',
                font: {
                    size: 28
                },
                color: {
                    background: '#A3ADC4',
                    border: '#253031'
                },
                borderWidth: 3
            }
        };
        var network = new vis__WEBPACK_IMPORTED_MODULE_3__["Network"](container, data, options);
        //click event handler
        var self = this;
        network.on("click", function (params) {
            var trafficLabels = [];
            var trafficData = [];
            if (params.nodes[0] != "Internet") {
                self.viewingIp = params.nodes[0];
                var data = self.networkData[params.nodes[0]];
                for (var key in data) {
                    if (data[key].count && trafficLabels.length < 21) {
                        trafficLabels.push(key);
                        trafficData.push(data[key].count);
                    }
                    else if (trafficLabels.length > 20) {
                        break;
                    }
                }
                self.packetChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"]('trafficChart',
                      {
                    type: 'bar',
                    data: {
                        datasets: [{
                                data: trafficData,
                                backgroundColor: "#5C6784"
                            }],
                        labels: trafficLabels
                    },
                    options: {
                        legend: {
                            display: false
                        },
                        maintainAspectRatio: false,
                        title: {
                            display: true,
                            text: "Packets Sent/Received"
                        }
                    }
                });
                var protLabels = [];
                var protData = [];
                for (var key in data.protocol) {
                    protLabels.push(key);
                    protData.push(data.protocol[key]);
                }
                self.protChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"](’protChart’, {
                    type: ’pie’,
                    data: {
                        datasets: [{
                                data: protData,
                                backgroundColor: ["#0074D9", "#FF4136", "#2ECC40", "#FF851B", "#7FDBFF
                                     ", "#B10DC9", "#FFDC00", "#001f3f", "#39CCCC", "#01FF70", "#85144b
                                     ", "#F012BE", "#3D9970", "#111111", "#AAAAAA", "#CAE1FF", "#00E5EE
                                     ", "#00C78C", "#8FBC8F", "#FFD700", "#FFA500", "#FF6347"]
                            }],
                        labels: protLabels
                    }
                });
            }
        });
    };
    HomeComponent.prototype.getData = function () {
        var _this = this;
        this.meshService.getPacketData().subscribe(function (data) {
            var trafficCounts = [];
            var routeCounts = [];
            var lengths = [];
            var prots = [];
            var protCounts = [];
            var protMap = new Map();
            var externalIps = [];
            var externalIpsCount = [];
            console.log(data);
            _this.networkData = data;
            for (var key in _this.networkData) {
                for (var prot in _this.networkData[key]["protocol"]) {
                    if (protMap.has(prot))
                        protMap.set(prot, protMap.get(prot) + _this.networkData[key][’protocol’][prot
                          ]);
                    else
                        protMap.set(prot, _this.networkData[key][’protocol’][prot]);
                }
                if (!_this.ips.includes(key) && _this.networkData[key].count) {
                    externalIps.push(key);
                    externalIpsCount.push(_this.networkData[key].count);
                }
            }
            for (var i = 0; i < _this.ips.length; i++) {
                if (_this.ips[i] in _this.networkData) {
                    trafficCounts.push(_this.networkData[_this.ips[i]].count);
                    lengths.push(_this.networkData[_this.ips[i]].length);
                    routeCounts.push(_this.networkData[_this.ips[i]].routecount);
                }
                else {
```

```
        trafficCounts.push(0);
        lengths.push(0);
    }
}
console.log(routeCounts);
prots = Array.from(protMap.keys());
protCounts = Array.from(protMap.values());
var protColors = [];
_this.createNetGraph();
_this.externalChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"]('externals', {
    type: 'bar',
    data: {
        labels: externalIps,
        datasets: [
            {
                data: externalIpsCount,
                backgroundColor: "#1D263B"
            }
        ]
    },
    options: {
        legend: {
            display: false
        },
        scales: {
            xAxes: [{
                    display: true,
                    ticks: {
                        fontSize: 15
                    }
                }],
            yAxes: [{
                    display: true,
                    ticks: {
                        fontSize: 15
                    }
                }],
        },
        title: {
            display: true,
            text: "External Traffic",
            fontColor: "black",
            fontSize: 30
        },
        gridLines: {
            color: "black",
            lineWidth: 10
        }
    }
});
_this.trafficChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"]('traffic', {
    type: 'bar',
    data: {
        labels: _this.ips,
        datasets: [
            {
                label: "Packets",
                data: trafficCounts,
                backgroundColor: "#70A0AF"
```

```
                        }
                    ]
                },
                options: {
                    legend: {
                        display: true
                    },
                    scales: {
                        xAxes: [{
                                display: true,
                                ticks: {
                                    fontSize: 15
                                }
                            }],
                        yAxes: [{
                                display: true,
                                ticks: {
                                    fontSize: 15
                                }
                            }],
                    },
                    title: {
                        display: true,
                        text: "Node Traffic",
                        fontColor: "black",
                        fontSize: 30
                    },
                    gridLines: {
                        color: "black",
                        lineWidth: 10
                    }
                }
            });
            _this.lengthChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"]("length", {
                type: ’bar’,
                data: {
                    labels: _this.ips,
                    datasets: [
                        {
                            label: "Length",
                            data: lengths,
                            backgroundColor: "#4CB963"
                        },
                        {
                            label: "Route Count",
                            data: routeCounts,
                            backgroundColor: "#706993"
                        }
                    ]
                },
                options: {
                    legend: {
                        display: true
                    },
                    scales: {
                        xAxes: [{
                                display: true,
                                ticks: {
                                    fontSize: 15
                                }
                            }],
                        yAxes: [{
                                display: true,
                                ticks: {
                                    fontSize: 15
                                }
                            }],
                    },
                    title: {
                        display: true,
                        text: "Length of Packets",
                        fontColor: "black",
                        fontSize: 30
                    },
                    gridLines: {
                        color: "black",
                        lineWidth: 10
                    }
                }
            });
            _this.totalProtChart = new chart_js__WEBPACK_IMPORTED_MODULE_2__["Chart"]("protocols", {
                type: 'pie',
                data: {
                    labels: prots,
                    datasets: [
                        {
                            data: protCounts,
                            backgroundColor: ["#0074D9", "#FF4136", "#2ECC40", "#FF851B", "#7FDBFF", "#B10DC9", "#FFDC00", "#001f3f", "#39CCCC", "#01FF70", "#85144b", "#F012BE", "#3D9970", "#111111", "#AAAAAA"]
                        }
                    ]
                },
                options: {
                    title: {
                        display: true,
                        text: "Protocols Used",
                        fontColor: "black",
                        fontSize: 30
                    },
                }
            });
        });
    };
    HomeComponent.prototype.sendTrafficControlData = function () {
        var body = {
            throttle: this.throttleInput,
            priority: this.prioritizeinput
        };
        this.meshService.sendTrafficControlData(body).subscribe(function (data) {
            console.log(data);
        });
    };
    HomeComponent.prototype.closeModal = function () {
        this.viewingIp = null;
        if (this.packetChart)
            this.packetChart.destroy();
        if (this.protChart)
```

```
            this.protChart.destroy();
    };
    __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["ViewChild"])('graphContainer'),
        __metadata("design:type", _angular_core__WEBPACK_IMPORTED_MODULE_0__["ElementRef"])
    ], HomeComponent.prototype, "graph", void 0);
    HomeComponent = __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["Component"])({
            selector: 'app-home',
            template: __webpack_require__(/*! ./home.component.html */ "./src/app/home/home.component.html"),
            styles: [__webpack_require__(/*! ./home.component.css */ "./src/app/home/home.component.css")]
        }),
        __metadata("design:paramtypes", [_mesh_service__WEBPACK_IMPORTED_MODULE_1__["MeshService"]])
    ], HomeComponent);
    return HomeComponent;
}());



/***/ }),

/***/ "./src/app/mesh.service.ts":
/*!*******************************!*\
  !*** ./src/app/mesh.service.ts ***!
  \*********************************/
/*! exports provided: MeshService */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "MeshService", function() { return MeshService; });
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
/* harmony import */ var _angular_common_http__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! @angular/common/http */ "./node_modules/@angular/common/fesm5/http.js");
var __decorate = (undefined && undefined.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length, r = c < 3 ? target : desc === null ? desc = Object.getOwnPropertyDescriptor(target, key) : desc, d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") r = Reflect.decorate(decorators, target, key, desc);
    else for (var i = decorators.length - 1; i >= 0; i--) if (d = decorators[i]) r = (c < 3 ? d(r) : c > 3 ? d(target, key, r) : d(target, key)) || r;
    return c > 3 && r && Object.defineProperty(target, key, r), r;
};
var __metadata = (undefined && undefined.__metadata) || function (k, v) {
    if (typeof Reflect === "object" && typeof Reflect.metadata === "function") return Reflect.metadata(k, v);
};


var MeshService = /** @class */ (function () {
    function MeshService(http) {
        this.http = http;
    }
    MeshService.prototype.getPacketData = function () {
        return this.http.get('http://172.27.0.15:3030/getFile');
    };
    MeshService.prototype.getLinks = function () {
        return this.http.get(’http://172.27.0.15:3030/getArp’);
    };
    MeshService.prototype.getIPs = function () {
        return this.http.get(’http://172.27.0.15:3030/getIpList’);
    };
    MeshService.prototype.sendTrafficControlData = function (body) {
        return this.http.post(’http://172.27.0.15:3030/editTrafficControl’, body);
    };
    MeshService = __decorate([
        Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["Injectable"])({
            providedIn: ’root’
        }),
        __metadata("design:paramtypes", [_angular_common_http__WEBPACK_IMPORTED_MODULE_1__["HttpClient"]])
    ], MeshService);
    return MeshService;
}());



/***/ }),

/***/ "./src/environments/environment.ts":
/*!*****************************************!*\
  !*** ./src/environments/environment.ts ***!
  \*****************************************/
/*! exports provided: environment */
/***/ (function(module, __webpack_exports__, __webpack_require__) {

"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony export (binding) */ __webpack_require__.d(__webpack_exports__, "environment", function() { return environment; });
// This file can be replaced during build by using the ‘fileReplacements‘ array.
// ‘ng build --prod‘ replaces ‘environment.ts‘ with ‘environment.prod.ts‘.
// The list of file replacements can be found in ‘angular.json‘.
var environment = {
    production: false
};
/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as ‘zone.run‘, ‘zoneDelegate.invokeTask‘.
 *
 * This import should be commented out in production mode because it will have a negative impact
 * on performance if an error is thrown.
 */
// import ’zone.js/dist/zone-error’; // Included with Angular CLI.


/***/ }),

/***/ "./src/main.ts":
/*!*********************!*\
  !*** ./src/main.ts ***!
  \*********************/
/*! no exports provided */
/***/ (function(module, __webpack_exports__, __webpack_require__) {
```

```
"use strict";
__webpack_require__.r(__webpack_exports__);
/* harmony import */ var _angular_core__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! @angular/core */ "./node_modules/@angular/core/fesm5/core.js");
/* harmony import */ var _angular_platform_browser_dynamic__WEBPACK_IMPORTED_MODULE_1__ = __webpack_require__(/*! @angular/platform-browser-dynamic */ "./node_modules/@angular/platform-browser-dynamic/fesm5/platform-browser-dynamic.js");
/* harmony import */ var _app_app_module__WEBPACK_IMPORTED_MODULE_2__ = __webpack_require__(/*! ./app/app.module */ "./src/app/app.module.ts");
/* harmony import */ var _environments_environment__WEBPACK_IMPORTED_MODULE_3__ = __webpack_require__(/*! ./environments/environment */ "./src/environments/environment.ts");




if (_environments_environment__WEBPACK_IMPORTED_MODULE_3__["environment"].production) {
    Object(_angular_core__WEBPACK_IMPORTED_MODULE_0__["enableProdMode"])();
}
Object(_angular_platform_browser_dynamic__WEBPACK_IMPORTED_MODULE_1__["platformBrowserDynamic"])().bootstrapModule(_app_app_module__WEBPACK_IMPORTED_MODULE_2__["AppModule"])
    .catch(function (err) { return console.error(err); });


/***/ }),

/***/ 0:
/*!***************************!*\
  !*** multi ./src/main.ts ***!
  \***************************/
/*! no static exports found */
/***/ (function(module, exports, __webpack_require__) {

module.exports = __webpack_require__(/*! /Users/Matt/Desktop/Coding/JavaScript/MeshApp/MeshAppFrontend/meshFrontend/src/main.ts */"./src/main.ts");


/***/ })

},[[0,"runtime","vendor"]]]);
//# sourceMappingURL=main.js.map
```

Bibliography

[1] goTenna Website, "gotenna." https://www.gotenna.com/.

[2] J. Tremback, "Faster, cheaper, decentralized internet." https://althea.org/.

[3] O. Contributors, "Openwisp." http://openwisp.org/whatis.html.

[4] M. S. Singh and V. Talasila, "A practical evaluation for routing performance of batman-adv and hwmn in a wireless mesh network test-bed," in *2015 International Conference on Smart Sensors and Systems (IC-SSS)*, pp. 1–6, Dec 2015.

[5] J. Kim, M. Seo, and S. Lee, "G.o.t.h.a.m. - mesh network management and visualization for batman-adv," Jan 2017.

[6] "Advanced traffic control." https://wiki.archlinux.org/index.php/Advanced_traffic_control.

[7] p. justin, "In defense of the slim protocol," Jan 2018.

[8] A. Rao, E. Dimogerontakis, M. Selimi, A. Ali, L. Navarro, and A. Sathiaseelan, "Blockchain for economically sustainable wireless mesh networks," 11 2018.

[9] "Openwisp 2 documentation." http://openwisp.io/docs/.

[10] M. Jnemann, "802.11s wireless mesh network." https://mjuenema.github.io/80211s_wireless_mesh/.

[11] "open80211s." https://github.com/o11s/open80211s/wiki/HOWTO.