Machine Learning in Traffic Classification of SDN Based Networks

ECE 1508 Network Softwarization – Technologies and Enablers:

Final Project Report

Ahmed Khan (998325272)

April 15th, 2019

Abstract

Traffic Classification has become an important research area especially with the advancements of Machine Learning and Software Defined Networking. In this project, two machine learning models – Logistic Regression (supervised) and K-Means Clustering (unsupervised) were used to classify DNS, Telnet, Ping, and Voice traffic flows simulated by the Distributed Internet Traffic Generator (D-ITG) tool. Each host in the network was connected through an overlay network to an Open vSwitch (OVS). The OVS was connected to a Ryu controller which collected basic flow statistics between hosts. These statistics were then parsed by a Python traffic classification script which periodically outputted the learned traffic labels of each flow. Logistic Regression was found to work much better than K-Means Clustering. Further improvements in the project could be to add functionality to detect unique traffic flows between the same pair of source and destination.

Introduction

The classification of traffic flows in today's IP networks has become an important research area with the adoption of Machine Learning (ML) techniques and Software Defined Networking (SDN) principles. Traditional methodologies including identifying traffic based on port number and payload inspection are not effective due to the dynamic and encrypted nature of current traffic. This project will attempt to utilize Supervised and Unsupervised ML algorithms to classify flows by their required bandwidth, required QoS, and their application based on various flow level details as features.

As mentioned earlier, traffic classification using Machine Learning is a growing trend in the network analytics domain. One application of these techniques is in cybersecurity. Large datasets produced from enormous Internet traffic flows are difficult to process and analyze, even for talented experts in the field using sophisticated tools. In addition, ML classification and clustering of flows can help identify network hotspots and potential bottlenecks. When using bandwidth and QoS of flows as classifiers, we can use Traffic Engineering (TE) to adjust flow paths and add virtual resources to the network infrastructure. Finally, identification of applications or web-based protocols is important for forecasting future trends and ensuring the network can meet the demand. Network classification is therefore of great interest to ISPs, governments, and enterprise alike.

Implementation

A simple network topology was created using VirtualBox as shown in Figure 1. It consisted of 5 Virtual Machines: 1 Controller, 1 Layer 2 Switch, and 3 Hosts. The nodes in the network were modeled as VMs so that network traffic would experience some delay. Another option was to use Mininet but it wasn't chosen due to the network simulation taking place within a single VM. An overlay network was deployed

so that the traffic generated by the hosts went through the Switch VM instead of using the VirtualBox internal switching mechanism to communicate. The hosts in the network used OVS with 2 interfaces. The first interface was internal and the second connected towards the Switch VM OVS using VXLAN tunnelling. The Switch OVS had VXLAN interfaces towards the hosts and connected to the Controller VM directly (i.e. using underlay IP). The Ryu controller was deployed on the Controller VM.

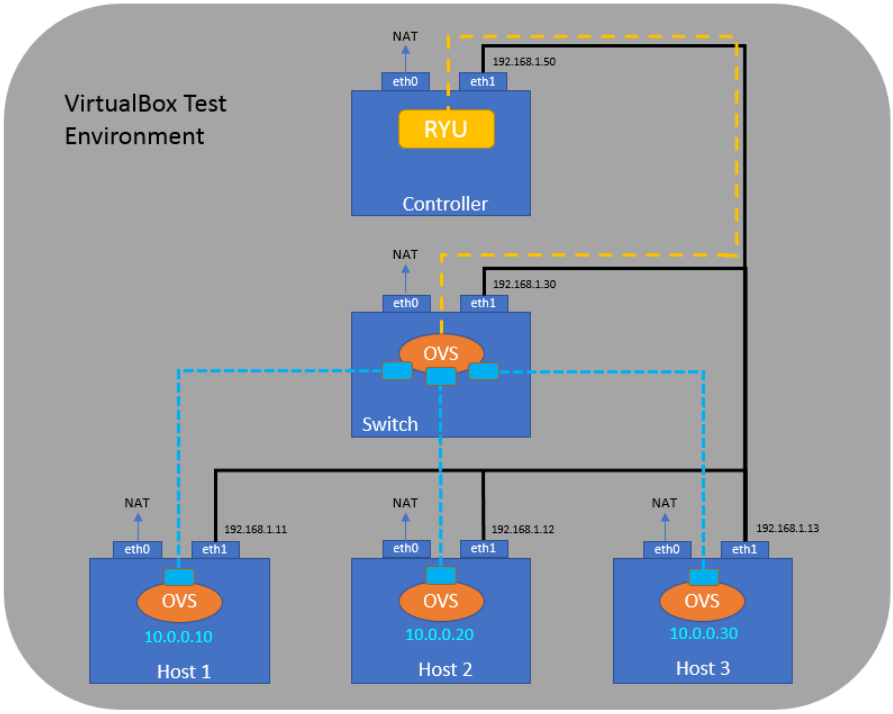

Figure 1: VirtualBox Network Simulation

After the network is successfully deployed and configured, the controller should be aware of packets flowing through the switch between any of the hosts. The sample Python script, simple_monitor_13.py, was modified to display the following flow information each second:

- time: the UTC clock value at the time of flow information
- datapath: the switch ID to identify the switch in Ryu

- in-port: the port receiving the incoming traffic
- eth_src: the source MAC address of the flow
- eth_dst: the destination MAC address of the flow
- out-port: the port sending the outgoing traffic
- total_packets: total packets of the flow so far
- total_bytes: total Bytes of the flow so far

This data was used as input data to the traffic_classifier.py script. The script used the input data to create a Flow object with the following attributes:

- time_start: the UTC clock value at the time the flow is first detected
- datapath: the switch ID to identify the switch in Ryu
- in-port: the port receiving the incoming traffic
- eth_src: the source MAC address of the flow
- eth_dst: the destination MAC address of the flow
- out-port: the port sending the outgoing traffic
- forward_packets: the total number of packets seen in the forward direction (src -> dst)
- forward_bytes: the total number of Bytes seen in the forward direction (src -> dst)
- forward_delta_packets: the number of packets seen since the last forward flow detection
- forward_delta_bytes: the number of Bytes seen since the last forward flow detection
- forward_inst_pps: the instantaneous packets per second in the forward direction (src->dst)
- forward_avg_pps: the average packets per second in the forward direction (src->dst)
- forward_inst_bps: the instantaneous Bytes per second in the forward direction (src->dst)
- forward_avg_bps: the average Bytes per second in the forward direction (src->dst)
- forward_status: the status (active/inactive) of the forward flow
- forward_last_time: the UTC clock value of the last time forward flow was detected

- reverse_packets: the total number of packets seen in the reverse direction (dst -> src)
- reverse _bytes: the total number of Bytes seen in the reverse direction (dst -> src)
- reverse _delta_packets: the number of packets seen since the last reverse flow detection
- reverse _delta_bytes: the number of Bytes seen since the last reverse flow detection
- reverse _inst_pps: the instantaneous packets per second in the reverse direction (dst->src)
- reverse _avg_pps: the average packets per second in the reverse direction (dst->src)
- reverse _inst_bps: the instantaneous Bytes per second in the reverse direction (dst->src)
- reverse _avg_bps: the average Bytes per second in the reverse direction (dst->src)
- reverse _status: the status (active/inactive) of the reverse flow
- reverse _last_time: the UTC clock value of the last time reverse flow was detected

The traffic_classifier.py script can do the following tasks:

1) Collect Training Data – collect training data for a specified traffic type. The traffic must be flowing between two hosts before the script is run.
2) Classify using Supervised Machine Learning – classify traffic type of flow between hosts using Logistic Regression
3) Classify using Unsupervised Machine Learning – classify traffic type of flow between hosts using K-Means Clustering.

Simulating Traffic Flows

The Distributed Internet Traffic Generator (D-ITG) application was used to generate the traffic flow data used for training the Machine Learning models. D-ITG is described as ‘a platform capable to produce IPv4 and IPv6 traffic by accurately replicating the workload of current Internet applications. D-ITG can generate traffic following stochastic models for packet size (PS) and inter departure time (IDT) that mimic application-level protocol behavior. D-ITG is able to replicate statistical properties of traffic of

different well-known applications (e.g. Telnet, VoIP – G.711, G.723, G.729, Voice Activity Detection, Compressed RTP – DNS, network games' [1]. For the purposes of this proof of concept traffic classification, the following traffic types were used: Ping, Telnet, DNS, Voice (G.711). The choice of traffic classes was due to limitations in simulation tools and issues faced in using D-ITG as discussed further in the Limitations and Future Work section.

D-ITG describes the traffic used as follows:

- Telnet - Generates traffic with Telnet characteristics. It works with TCP transport layer protocol. Different settings will be ignored.
- DNS - Generates traffic with DNS characteristics. It works with both UDP and TCP transport layer protocols.
- VoIP (voice) - Generate traffic with VoIP characteristics. It only works with UDP transport layer protocol. Different settings will be ignored. The emulation of G.711 codec is used.

For Ping traffic, a simple 'ping' command was run with the overlay IP of the destination host.

The process to collect training data for the models is as follows: First simulate the flow of the specific traffic between a certain pair of hosts using D-ITG or other tools. Second, start the traffic_classifier.py script with the appropriate options for training the traffic type. The script starts the Ryu controller and simple_monitor_AK.py (the modified version of simple_monitor_13.py). The data generated from simple monitor script is collected and transformed to update the attributes of a Flow object. The attributes of the Flow object are then periodically outputted to a CSV file. After the CSV files for each traffic type are generated, they are combined in a complete Pandas Dataframe object used for the model training and testing.

Supervised Learning - Logistic Regression

The first Machine Learning algorithm used was a supervised Logistic Regression model. It is used to predict categorical target variables. Mostly, the outcome is a binary value but in the case of multiple targets, it picks the target with the highest probability of occurring. In our program, Logistic Regression performed exceptionally well with an accuracy of over 99%. It is clear to see from the decision boundaries, Figure 2, formed with the first two Principle Components why the accuracy is so high.

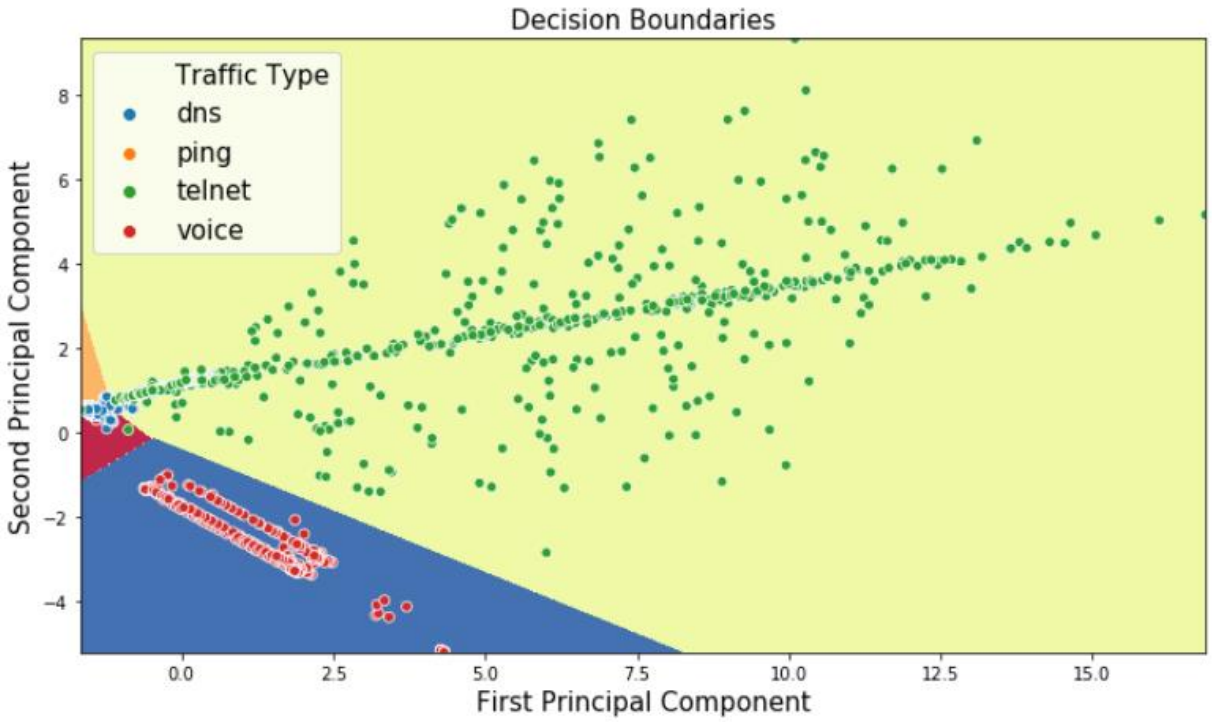

Figure 2: Decision Boundaries for Logistic Regression

A Confusion Matrix can help pinpoint where the model is failing to accurately decide the target. It is a matrix with predicted labels on the y-axis and true labels on the x-axis. If a model is tending to predict a certain traffic class as another class more often, then it will be clearly evident in the Confusion Matrix. However, we see, in Figure 3, that for Logistic Regression, there is almost no failure.

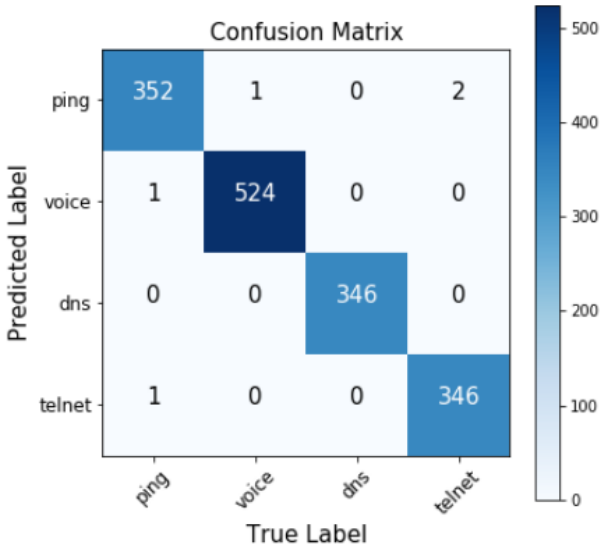

Figure 3: Confusion Matrix for Logistic Regression

Unsupervised Learning – K-Means Clustering

K-Means Clustering is an unsupervised Machine Learning model that groups data points in multi-dimensional space together to form clusters and label each data point in the cluster the same. The initial desired number of clusters are chosen – in our case of four traffic types, four clusters were used as an input parameter to the model. In Figure 4 below, we can attempt to visualize where each centroid of the four clusters are. Each square represents a dimension. The darker the color, the higher the value in that dimension. This shows where in the 12-dimensional vector space, the cluster centroids lie.

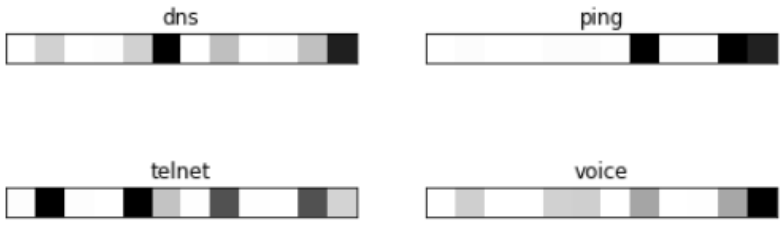

Figure 4: Visualization of Cluster Centroids

To visualize model performance, we again use Principle Component Analysis to reduce the number of dimensions from 12 to 2 and then plot in two-dimensional space. We can clearly understand why model performance for this algorithm was only around 30%. It is not very good at clustering non-circularly groups of data.

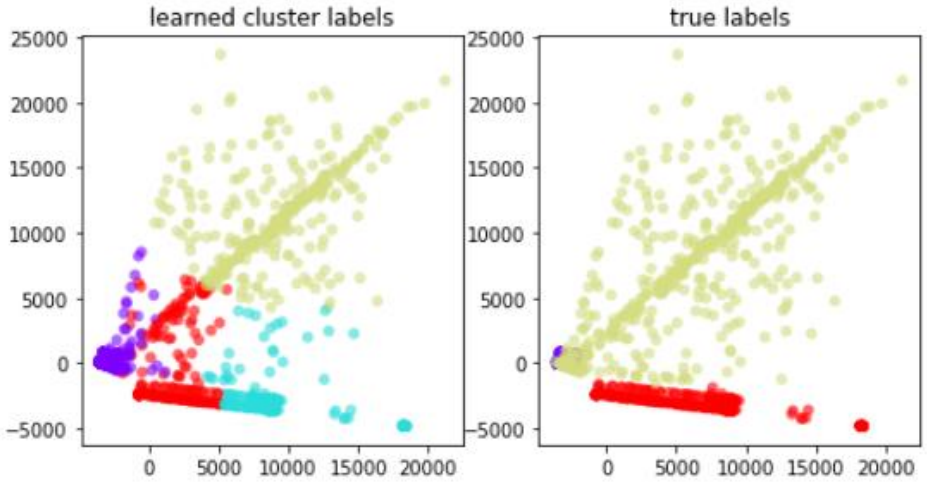

Figure 5: Visualizing K-Means Clustering Performance

Finally, we confirm our understanding by looking at the confusion matrix and seeing a very disbalanced matrix.

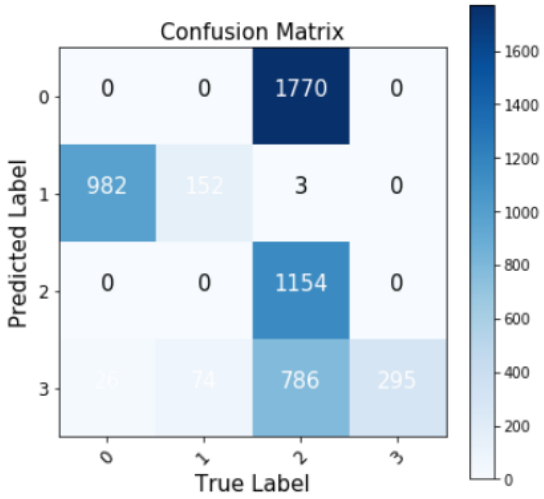

Figure 6: Confusion Matrix of K-Means Clustering

Limitations and Future Work

Although the proof of concept design of this problem works well, it is limited in the following cases:

1) It does not classify multiple flows between the same source and destination nodes. For example, if we start a ping command from host1 to host2, then add voice traffic from host1 to host2, it will assume the ping and voice are 1 unique flow and classify as such.
2) If a flow is started and stopped, the algorithm will not delete the old flow but instead update it with the latest flow statistics. This is a problem because the flow classifier considers average packet size and average number of Bytes. If the flow is stopped for a significant amount of time, these two features will be reduced, and the resulting label of traffic will be inaccurate.
3) The simulation tool (D-ITG) was unable to generate flows for game play or other video traffic. With a lot of internet traffic being video nowadays, this would have been helpful for real world classification situations.
4) We see that unsupervised learning using K-Means clustering performs very poorly. Perhaps this model needs to be further tuned using Hyperparameter tuning. We can also consider other models such as DBSCAN (Density Based Scanning) or CNN (Convoluted Neural Networks) for future work on this project.

In addition to the above, the following items will also help improve the quality of the project.

- Create GUI to visualize flow classifications on different SDN controllers
- Use visual analytics to point out 'hot' areas of high bandwidth and QoS traffic
- Connect ML application to actual SAVI testbed controllers and visualize real traffic flows

References

[1] Distributed Internet Traffic Generator
http://traffic.comics.unina.it/software/ITG/manual/D-ITG-2.8.1-manual.pdf

[2] Classifying Network Traffic Flows with Deep-Learning
https://www.eleceng.adelaide.edu.au/students/wiki/projects/index.php/Projects:2017s1-101_Classifying_Network_Traffic_Flows_with_Deep-Learning#Padding_style

[3] QoS -aware Traffic Classification Architecture Using Machine Learning and Deep Packet Inspection
https://reader.elsevier.com/reader/sd/pii/S1877050918307129?token=481A2C61587FE2C6743C320DDC7612381E99A6BAD5300B53B6C87D475380BC37D4A801FA23FA6B514383FD67EA186865

[4] A Survey of Traffic Classification in Software Defined Networks
https://hoticn.com/files/hoticnPapers/032-paper%20101.pdf

[5] Identification and Selection of Flow Features for Accurate Traffic Classification in SDN
https://ieeexplore.ieee.org/document/7371715?ALU=LU1046369

[6] Unsupervised Learning with Python
https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03