Modern methods of Detecting DDoS attacks in SDN

B. Sujeeth Kumar
es19btech11022@iith.ac.in

G. Sai Sidhardha
cs19btech11050@iith.ac.in

Krishn V. Kher
es19btech11015@iith.ac.in

M. Bharat Chandra
es19btech11016@iith.ac.in

M. Sri Akhil
es19btech11014@iith.ac.in

Abstract

In this project, we attempt to develop a prototype framework that efficiently detects DDoS intrusions in real time networks based on modern Deep Learning based methods. We also try to go one step further and explore certain existing implementations for preventing/mitigating certain kinds of DDoS attacks. Finally we also present a new idea (to the best of our knowledge) for converting resource-targeting DDoS attacks to bandwidth-targeting DDoS attacks, so that the prevention/mitigation procedures applied for bandwidth-targeting DDoS attacks can be applied to resource-targeting DDoS attacks as well.

1. Introduction

The Internet has become a large part of our lives ever since it came in existence, in the late 20th century. Ever since, thousands and millions of improvised Internet-based communication systems have been proposed, tested and deployed in real time. Many protocols have been proposed and deployed even upon existing network infrastructures. Both hardware and software based ideas have been proposed for improvement on various Internet-based services.

However, the downside is that despite all these successes, there were also many problems that we have been encountering. Some have got solved elegantly, some have been partially solved or have been solved for specific systems, while there are still many issues that remain unsolved.

One of the major concerns in the last category is the problem of secure communications and services. There is so much importance and variety associated with the domain of network security that there are even whole academic degrees and jobs offered just in the domain of Cryptography and Network Security!

Looking back 40-50 years, we have evolved a lot in this domain; we have defined a whole mathematical framework to define threat models and develop (possibly) complex and secure protocols that provide the service even under those adversarial conditions. This is what is usually studied in Cryptography. A major part of Networking Security involves extending the theoretical threat scenarios studied in Cryptography, in real time and exploring safe and efficient models accordingly. **Now in Networking Security, an attack on a real time client-server kind of system is informally defined as some sort of a malicious attempt to hamper the intended services of the server or any sort of communications between the genuine client and server**.

There are many, many kinds of attacks that have been discovered, since the time the Internet was deployed. Precautionary measures and mitigation strategies have been devised for many of them, but unfortunately there are still many attacks that have no complete anti-attack solution. One may (rightly) wonder on the fact that the large number of security vulnerabilities still present in today's systems seems to indicate that the application of secure protocols and practices in real systems seems to be going at a much slower paces than the corresponding advances in theoretical Cryptographic research.

Part of the reason why this is so is because some security problems are inherently hard and it is a major challenge to actually devise protocols that only defend from an attacker while also providing legitimate services to genuine users. Another part of the reason for this is that when the Internet was initially designed, many of the resulting protocols and services were designed keeping in mind non-malicious users. However, unfortunately, within a matter of a decade it was observed that this assumption was not true!

One such domain of attacks that we concentrate on in this paper is *DoS* attacks. DoS stands for Denial-of-Service. Informally stated, attacks under this domain try to overload the server and/or its resources in some way or the underlying network routing the packets between the client and server, thus making it unable to render legitimate services to

genuine users. There is a subdomain of DoS attacks known as *DDoS* attacks. DDoS stands for Distributed-Denial-of-Service. Attacks under this domain try to achieve the goal of DoS attacks, in a distributed manner, i.e. they try to employ various network nodes and machines to overwhelm the target server/system.

In the recent times of COVID-19, the number of DDoS attack has unfortunately been on the rise as can be seen in Fig.1. Even as recently as January-February 2022, a huge

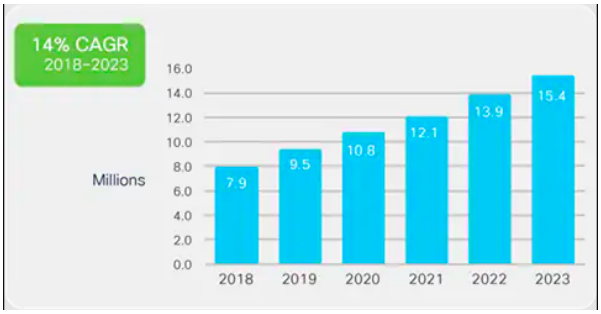

Figure 1. Source: Cisco Annual Report

spike was observed in the number of DDoS attacks in certain parts of the globe due to various reasons. Due to such attacks certain important/sensitive websites were down for extended periods of time, causing much financial and social damage ref: Fig.2. According to a research report published

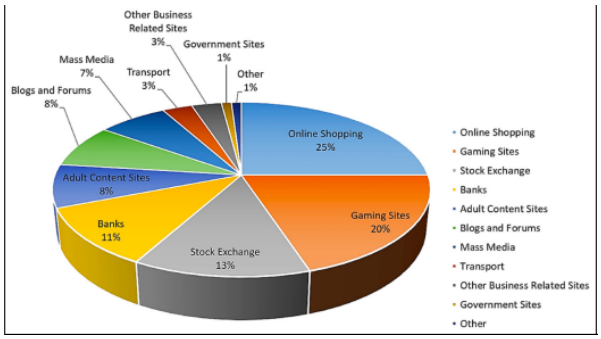

Figure 2. Breakdown of attacked sites in Q2 2011 [7]

by the Ponemon Institute in 2012, the average cost incurs for 1 minute downtime as a result of DDoS attacks was approximately US $22,000 which may range from US $1 to US $100,000. These sort of unwanted discrepancies motivate the strong and urgent need to come up with solutions which can defend against DDoS attacks.

We are focusing on trying to detect and mitigate DDoS attacks in a *SDN* based networking. SDN stands for "Software defined Networking." SDN based networking has become a popular trend, especially ever since the paper that proposed it [5].

2. Literature Review

There are many DDoS detection and prevention measures currently available. This survey paper [7] gives a very detailed report on most of the existing DDoS attacks and the common prevention/mitigation methods currently existing.

Before the age of classical Machine Learning, DDoS attacks used to usually get detected by monitors that would keep on collecting the statistics. Then either a network administrator had to keep an eye on the nature of the statistics, to have any chance of detecting a DDoS attack before it was actually launched.

Things changed for the better after classical ML models were researched upon [9]. However with the rise of Deep Learning based methods and their success in image recognition, speech models, semantic text understandings, etc., researchers have started trying DL based approaches to learn the functions and thresholds from various datasets to detect DDoS attacks [2] [10].

3. Proposed methodology

We first discuss the detection approaches we tried to explore as a part of this project. Later on, we also present the relevant accuracy and efficiency metrics which various models gave. We then discuss our attempts on mitigating and propose a possibly novel idea (proof of concept design) for mitigation.

3.1. System Design

In our system design (ref: Figure 4), we have an SDN controller that controls the functioning of all the switches and routers in the network. These switches and routers provide the network route from the clients to the users. The protocol that it uses for communication with the switches and routers is *OpenFlow*.

The SDN controller installs flow rules in the flow tables of the virtual switches and routers of the network, via flow_mod messages (OpenFlow terminology) depending on various events triggered in the network. Say a certain packet arrives, for which S1 (in Figure 4) has no flow rule stating what action to take on that particular packet/flow of packets. Then a "packet-in" event gets triggered, and the corresponding message is sent to the controller. All such events are kept in an event queue (at least in the Ryu implementation). Based on its programmed logic, the controller then decides whether that particular flow has to be forwarded (via some particular output port of the switch), enqueued (for say processing at a later instant of time), or dropped. The corresponding flow rule is installed by the controller into the flow table of the concerned switch (in this case S1) and then the action is taken up by the controller.

Coming to the design of our application, we realize that in order to detect a DDoS attack, we need to first continu-

Match	Action
To H2	Out from 2

Figure 3. Source: Dr.Kotaro's slides

ously monitor the health of the network system, via some statistical data. This is what the GetStatistics module (ref: Fig.4) performs. This module then passes the information retrieved from various nodes along with the source & destination IP addresses, source & destination ports and timestamps & switch to the Inference module.

The pretrained ML/DL models are used in this module. This module takes the input from the GetStatistics and passes data with the relevant features to the model and retrieves predictions corresponding to each particular switch, source and destination ports and IP addresses, etc.

Depending on the prediction of the Inference module (ref: Fig.4) (benign/attack) the Mitigation module (ref: Fig.4) attempts to mitigate the damage or prevent further damage that has happened on that particular switch. Various mitigation strategies may be applied depending on the particular attack detected. In case of a zero-day attack, some general form of recovery might be applied, such as dropping packets with a certain source IP address (via match/action flow rules (flow_mod message)), etc.

The topology co-ordinator (in this case a Mininet project instance) internally stores the whole topology in a topology database, so we can use it for other kinds of queries/updates on the network topology, which could be dynamic in nature.

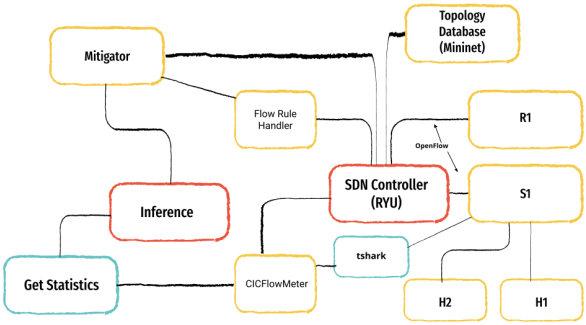

Figure 4. System Design. Interactive version available here.

3.2. Detection

We used supervised and semi-supervised learning based approaches. We also used ensemble methods under supervised learning based approaches. These models classify test samples derived from feature extractor of the SDN framework.

Before we proceed further, we provide an approximate picture of how well-clustered the dataset on which we are training is in Figure 5. This was done via t-SNE [14]. In an unsupervised learning sense, our goal is to finally cluster each of the test points into the correct clusters.

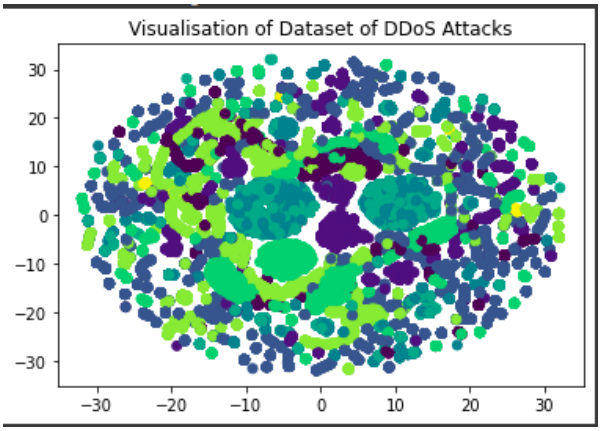

Figure 5. Clustered version of DDoS attack data

3.2.1 Supervised Learning

Multi-Layer-Perceptron: We implemented a deep learning model more specifically an autoencoder attached to a layer which predicts whether there is an attack or not and also classifies it according to the type of DDoS attack ref: ¡link¿. The reason we used autoencoder is because of the fact that the autoencoder can significantly increase the anomaly detection accuracy compared to linear and kernel Principal Component Analysis. It also helps in getting a compact representation of the input data flow. Finally the compact representation of the data is passed onto a softmax layer to predict the type of DDoS attack.

The model receives a 77 dimension vector as input is connected to a FCC (fully connected layer) to give 64 dimensions. It is then reduced to 32,16,8. This is the encoder structure of the model. This is then followed by a decoder which reconstructs the 64 dimensions in the exact reverse manner. All the layers use ReLU activation function. The output from the decoder is then sent to a FCC to give out an 8 dimensional vector(since we are only classifying 8 types of DDoS attacks) and a softmax activation is done on it to predict the classes.

The loss function is a simple categorical cross entropy loss between the output labels and the true labels.

We also tried to use some machine learning models to do the inference and they performed very well compared to the deep learning models.

CatBoost: CatBoost is based on gradient boosted decision trees. During training, a set of decision trees is built consecutively. Each successive tree is built with reduced loss compared to the previous trees. The number of trees is controlled by the starting parameters.We used this model since it is an extremely efficient ensemble method which does very well in classifying large data.

LGBM classifier: LightGBM is a gradient boosting framework based on decision trees to increase the efficiency of the model and reduce memory usage. It uses two novel techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling which fulfills the limitations of histogram-based algorithm that is primarily used in all Gradient Boosting Decision Tree frameworks.

3.2.2 Semi-Supervised Learning

Variational Auto Encoder: A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, it formulates an encoder to describe a probability distribution for each latent attribute. Our hope was that it define a distribution for each type of attack and then classify the type of attack.
The architecture is shown in Figure 6, where it has an encoder and decoder portion (unsupervised learning) followed by a final classifier MLP architecture (supervised learning) to classify the data.
The loss functions used were a combination of categorical cross entropy for reconstruction loss (unsupervised portion), KL divergence loss and categorical cross entropy loss for the classification loss(supervised portion).

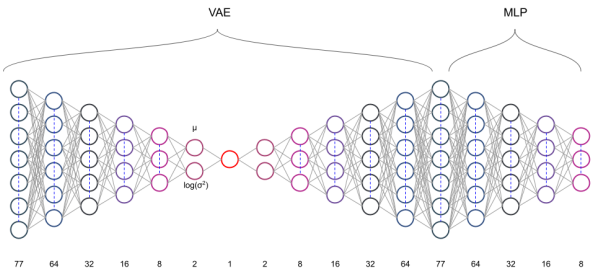

Figure 6. VAE Architecture

4. Implementation

The following modules have been implemented:

1. Network Topology.
2. Ryu Application.
3. Deep Learning and Machine Learning models.

The network topology has been implemented using the ”Mininet” package.

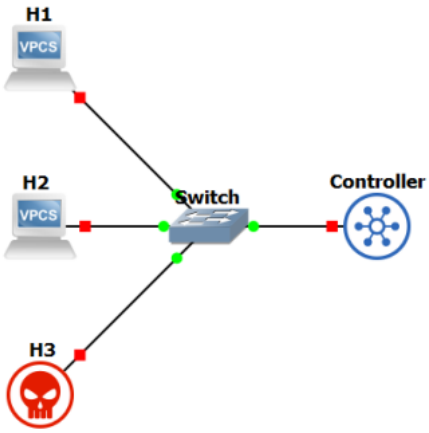

Figure 7. Linear Topology

For preliminary testing, a simple linear topology (Figure 7), with 1 switch, 3 hosts and 1 controller, has been implemented.

The SDN controller has been implemented using the Ryu API. The SDN controller populates the flow rules in the switches and performs flow classification in real time.

The deep learning and machine learning models have been trained independently and saved in pickle files. The pickle files are loaded in the SDN controller to perform inference.

A DDoS attack is generated from one of the hosts to another host belonging to the same subnet. The ”Hyenae” utility has been used for this purpose. Since packets flow through the switches, packets are captured at the switch interfaces using the ”tshark” utility and are saved in a pcap file. The ”cicflowmeter” utility has been used to read the pcap file to generate the features into a csv file.

Multithreading is performed using ”eventlet” threads. The main thread executes the ExampleSwitch13 (the main Ryu application) class and the slave thread is responsible for performing inference.

4.1. Feature Extractor (FE)

The slave thread calls the extract_features method in the Traffic_Classifier() class. The flow features are extracted from the csv files periodically by the slave thread. The extracted features are passed to the inference() method to perform inference.

4.2. Traffic Classifier (TC)

Inference is performed on the features passed to the inference method in the Traffic_Classifier() class. The inference can be performed on different supervised and unsupervised models like XGBoost, CatBoost, Variational Auto-Encoders and Auto-Encoders.

The incoming flows are classified into 8 categories, they are as follows: Benign, LDAP, MSSQL, NetBIOS, Portmap, TCP SYN, UDP, UDPLag.

4.3. Response on Alert (RoA)

Currently we have 2 mitigation approaches in mind; one is a simple drop-packet approach and other is an algorithm that we propose (proof of concept design only, have not implemented it due to lack of time). We describe them below as follows.

1. **Drop-packet**: Once we detect that a certain flow is malicious, we send a flow_mod message to the switches currently receiving that flow to drop any packets belonging to the flow. This is a very brute-force and primitive approach for mitigation, but it still at least partially helps in safeguarding the server from getting overloaded.
2. **Mycielski's Colored Dropout**: The inspiration for this mitigation approach comes from this wonderful paper by Mycielski [8]. The basic idea is that given a positive integer $k \leq 3$, using Mycielksi's general technique, one can generate a graph G such that $\chi(G) = k$, and it also turns out that it is the smallest graph that can be colored using k colors! For a given positive integer k, let M_k denote the graph generated using Mycielski's construction. Then, another property it has is that, for $k \leq 3$, given any two adjacent nodes u and v in the graph G, there exists a Hamiltonian path between u and v.

   Based on these properties, we propose an algorithm for mitigation of DDoS attacks.

   What this algorithm does is to basically convert any resource based attack to a bandwidth-based attack, possibly before the attacker can launch the full attack and alerts the network administrators before hand, and then any bandwidth-based mitigation method can be used. The goal that is achieved through this is that the server is protected from large damages, just because of the structure of the network topology itself and this algorithm, because it penalises nodes that are closer to the centre of the graph (and thus closer to the server/server cluster).

   One can observe that this algorithm is randomized (and this is rather good since there are higher chances that an attacker will not be able to guess which switches will start dropping the packets. Basically once a node has been declared infected and the recoloring of the vertices has been done, there is a higher chance that 2 consecutive switches have the same color and therefore the chance of sending more flows is reduced (because of the invariant).

   There are definitely improvements that can be done to this algorithm, that will try to explore in the future. An exemplary figure has been depicted in Figure 8.

Algorithm 1 Mycielski's Colored Dropout

Require: Parameter $k \geq 4$, Mycielski graph M_k.
Ensure: No packet flows through 2 consecutive switches assigned the same color.

```
S ← S_0                          ▷ S_0: initial (valid) coloring of M_k.
                                 ▷ S: current (can be imperfect)
coloring of M_k.
curr_state ← normal.
T_0                              ▷ Concentric depth of M_k.
C ← {1, 1, ..., 1}/sum(C).       ▷ Coloring probabilities.
while True do
    if curr_state == anomaly then
        V                        ▷ Holds pairs (c, t) which correspond to a node with color c at concentric depth t.
                                   These are nodes that have been hit by a malicious flow.
      for (c, t) in V:
          C[c] += T_0 − t
C ← C/sum(C).
for node in range(|M_k|):
if node.adjacent == infected:

          node.color ← random_toss(C).
(S gets modified).
    else if curr_state == recovery then
        S ← S_0.
        C ← {1, 1, ..., 1}/sum(C).
    else
continue
    end if
end while
```

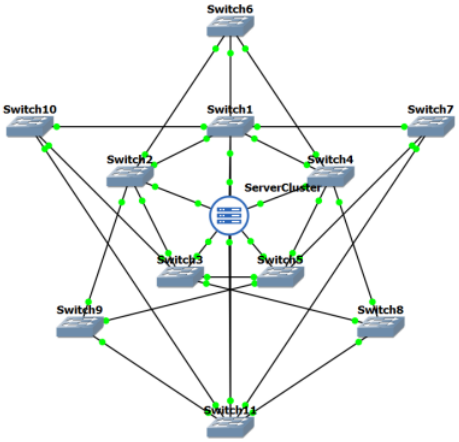

Figure 8. Mycielski Topology

Model can be seen in the below graphs.
The validation accuracy we obtained was **81.43**%.

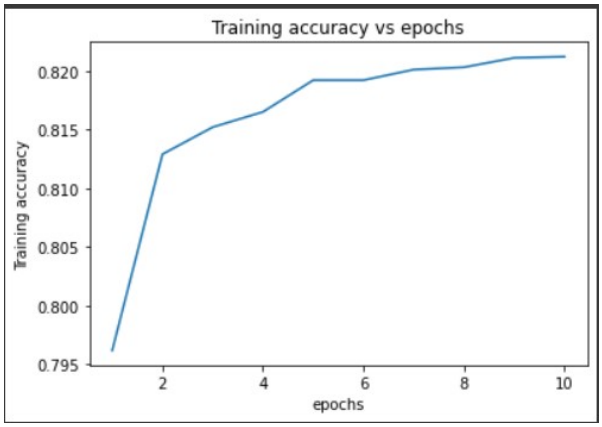

Figure 9. The Auto Encoder's training accuracy vs. epochs

5. Evaluation & Results

5.1. Model Training

In terms of time taken for training, the order was: CatBoost $<$ LightGBM $<$ VAE $\approx$ AutoEncoder MLP. This is expected since CatBoost and LGBM are advanced ensemble models using multiple techniques to learn patterns in the data. The code relevant to this can be found here.

5.2. Dataset

The dataset consists of 1245798 rows and 90 columns. Columns such as Unbound:0,Unbound:0.1,Unbound::0.1.1, source/destination IP/port,Flow ID,timestamp, Inbound and SimilarHTTP were dropped as they had very low correlation to the target feature. Finally we ended up with 77 columns. We dropped all the rows which had nan or inf inputs. We used one hot encoding to represent the target feature. There are 8 types of attacks in the dataset i.e there are 8 unique values in the target column. We also had to drop some columns and rearrange the rest of the columns in the same way as we were getting input of the flow from the switches. Finally we split the data into train and validation using stratified sampling on the target feature. The validation split was 0.3.

5.3. Accuracy & Efficiency

Here we show the accuracies of various models so that we could pick the best model for the DDoS detection.

5.3.1 Auto Encoder Model Statistics

We set the learning rate to 0.002 and decay to 1e-6. We trained it on 10 epochs. The statistics of the Auto Encoder

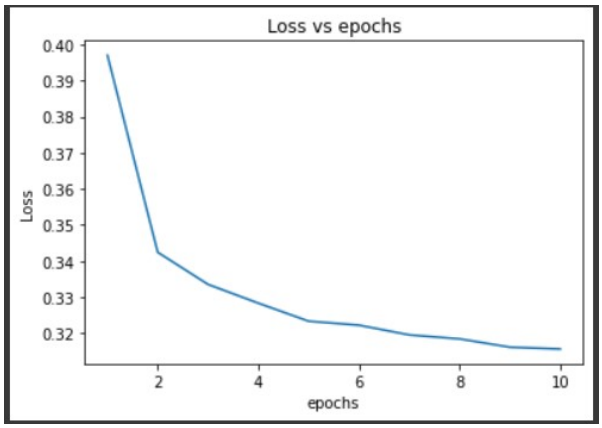

Figure 10. The Auto Encoder's loss vs. epochs

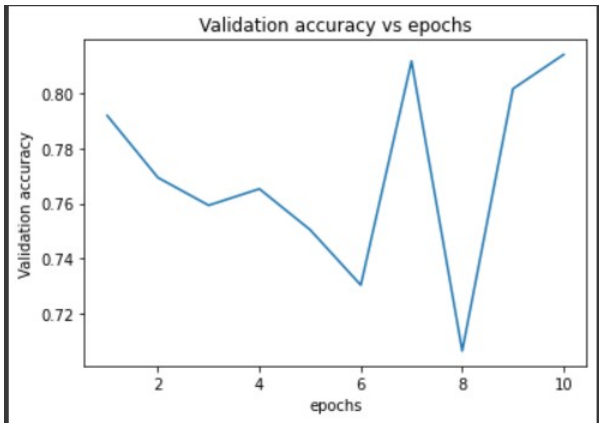

Figure 11. The Variational Auto Encoders validation accuracy vs. epochs

5.3.2 CatBoost Classifier Statistics

We used 20 iterations for training and set the learning rate to 0.3; all other parameters were left as default parameters, as the CatBoost website recommends. The confusion matrix of the CatBoost Classifier Model on validation data can be seen in the below graphs.
The validation accuracy we obtained was **86.42051139521957**%.

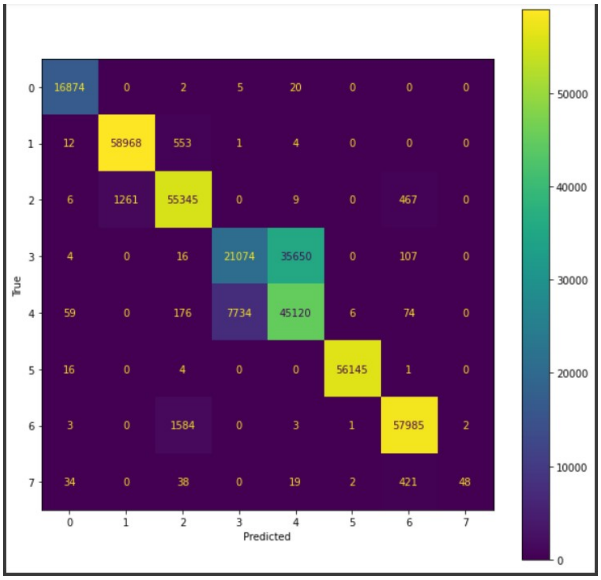

Figure 12. The CatBoost confusion matrix

5.3.3 LGBM Classifier Statistics

All parameters were left as the default parameters, since the accuracy that we got with those parameters itself was pretty high. The confusion matrix of the LGBM Classifier Model on validation data can be seen in the below graphs. The validation accuracy we obtained was **87.00441569196311**%.

5.3.4 Variational Auto Encoder Model Statistics

We set the learning rate to 0.002 and used 10 epochs. The statistics of the Variational Auto Encoder Model can be seen in the below graphs.
The validation accuracy we obtained was **68.40**%.

5.4. Attack Simulation

The DDoS attack (for testing purposes) has been generated using the Hyenae utility. Hyenae is a highly flexible and platform independent network packet generator. There are 14 different types of DDoS attacks attacks that ca be generated using Hyenae, some of the attacks are as follows: Arp-Request, ARP-Reply, ICMP-Echo, TCP, UDP and DNS-Query.

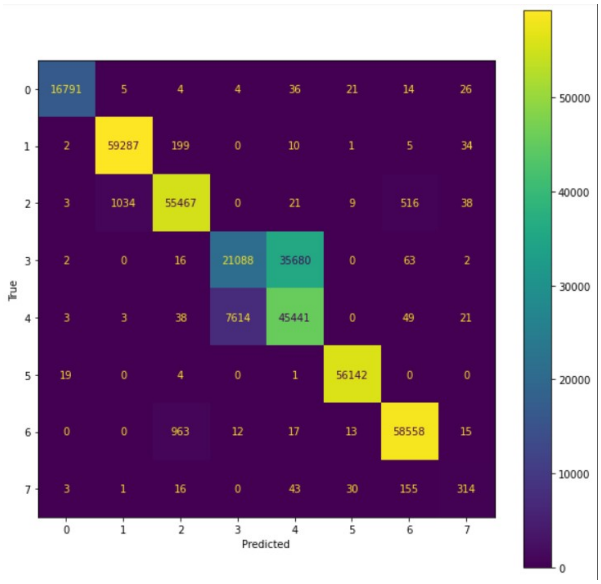

Figure 13. The LGBM confusion matrix

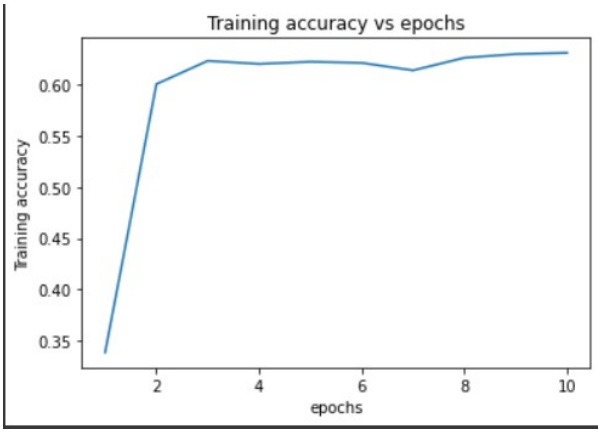

Figure 14. The Variational Auto Encoders training accuracy vs epochs

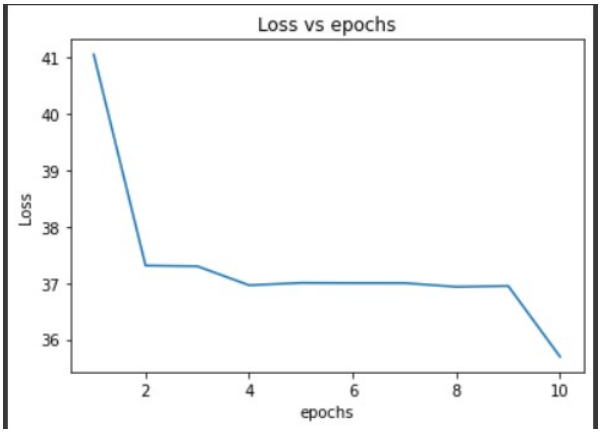

Figure 15. The Variational Auto Encoders Loss vs epochs

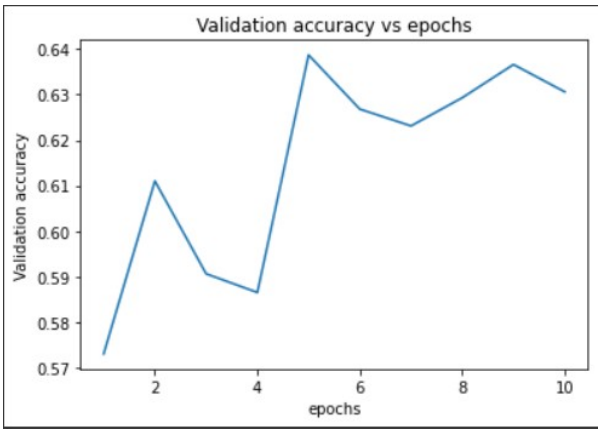

Figure 16. The Variational Auto Encoders validation accuracy vs epochs

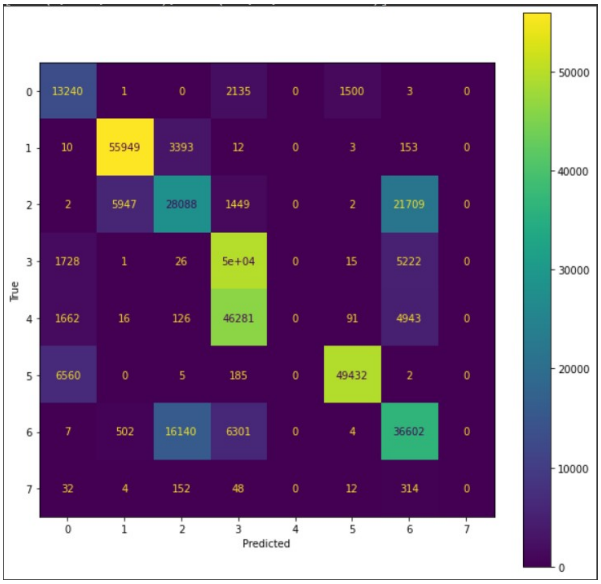

Figure 17. The VAE confusion matrix

We have generated the TCP SYN attack in our network. One of the host is the adversary and some other host belonging to the same subnet is the victim. To generate the TCP SYN attack, the following command has to be given on the adversary: $ sudo hyenae -a tcp -i <Interface Index> -s <Adversary MAC address>-<Adversary IP Address>@<Adversary Port> -d <Victim MAC address>-<Victim IP Address>@<Victim Port> -f <TCP Flag>

5.5. Model Inference

Here is the feature importance graph displayed by the CatBoost. As we can see, it appears that there are very few features that actually play a major role in the detection.

Across models, we observed that response time for inference was roughly the same. The same seemed to be

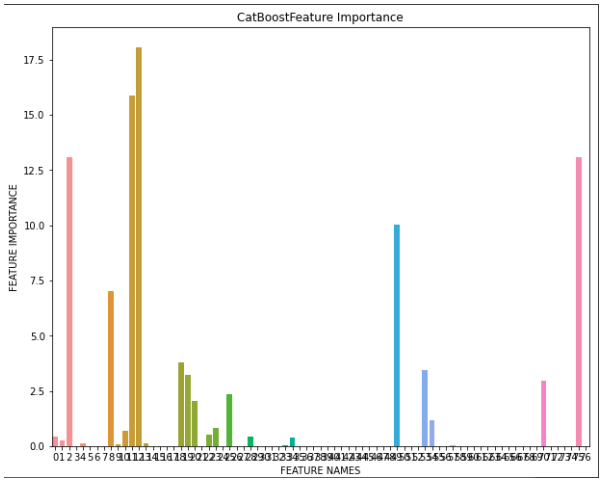

Figure 18. Feature Importance

true for the accuracies as well, though the ensemble results were more reliable. We also note that the overall accuracies that we achieved in the validation sets after pre-processing were comparable to other benchmark papers that have been quoted in this report.

6. Conclusions and Future Scope

From the results we derived, we could conclude that overall system design seems fine, but there are 2 utilities in particular that are currently the bottleneck of the system, and those are:

1. **cicflowmeter**: The cicflowmeter module is currently a bottleneck because currently, we are first using the tshark utility to capture packet data from each of the interfaces on the switches and then we are running cicflowmeter centrally to convert the .pcap files to .csv files which takes a lot of time even for a few MB of packet capture data. We had to adopt this procedure because apparently, cicflowmeter's real time flow retriever wasn't working for us, and other alternatives such as sFlow and NetFlow were not plausible for us to experiment with, at the time of our initial attempt of this project.

   We would like to improve this time without adding too much intelligence into the switch. In [15], proposed an entropy based light-weight DDoS detection system by exporting the flow statistics process to switches. Although the approach reduces the overhead collecting the flow statistics in the controller, it (possibly inadvertently) brings back intelligence into the network devices, which kind of defeats the point of SDN.

2. **Model inference**: it takes some time for the threads to read the flows and then pass the feature vectors to

the model inference. While most of the models predict within 1-5 seconds, the overall process of reading a flow sample and predicting the output and storing them in appropriate data structures can take upto a few minutes even in conditions of less traffic. For this we wish to explore more efficient data structures/libraries such as dask, etc.

Currently our model may be susceptible to model inversion attacks, as we have used a publicly available dataset and haven't taken any specific measures to prevent these kinds of attacks [13] [3]. We would like to improve our framework by training it on some DDoS attack data generated by us as well, along with employing some of the standard anti-model inversion attacks procedures commonly used.

Another possibility we would like to explore more in the future is Deep Reinforcement Learning based approaches. Next, we would also like to try to secure our system from network topology based attacks, since SDN environments are especially vulnerable to these type of attacks [4].

Next, we would like to focus on implementing some ideas sourced from graph theory and combinatorics in networks, getting motivated from the algorithm that we have proposed earlier. Those include routing in $\mathcal{O}$(congestion + dilation) time [11] (uses Lovasz local lemma repeatedly) and improvised face routing algorithms for geographical routing using SDNs [1] and possibly some ideas from papers presented in [6].

In regards to mitigation, we would like to try out other methods of mitigation such as ingress/egress filtering and probabilistic packet matching, among other methods mentioned in [7]. For prevention, we would like to try the Honeypot system!

7. Acknowledgement

We would like to thank Mr.Tahir Ahmed Shaik [12] cs20mtech14007@iith.ac.in for his valuable help in providing us insights on training strategies and deployment of various tools for this project. We thank Dr. Kotaro Kataoka on explaining us different aspects of networking required for the project and also for giving us alternatives when we were stuck at some stages of the project.

References

[1] Yuan-Po Cheng and Ming-Jer Tsai. Face routing on a non-planar graph: Theory and applications to networks. In *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '15, page 97–106, New York, NY, USA, 2015. Association for Computing Machinery.

[2] Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Delia Jurcut. Ddosnet: A deep-learning model for detecting network attacks. *CoRR*, abs/2006.13981, 2020.

[3] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015.

[5] Diego Kreutz, Fernando M. V. Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440, 2014.

[6] Alejandro López-Ortiz and Angèle M. Hamel, editors. *Combinatorial and Algorithmic Aspects of Networking, First Workshop on Combinatorial and Algorithmic Aspects of Networking, CAAN 2004, Banff, Alberta, Canada, August 5-7, 2004, Revised Selected Papers*, volume 3405 of *Lecture Notes in Computer Science*. Springer, 2005.

[7] Tasnuva Mahjabin, Yang Xiao, Guang Sun, and Wangdong Jiang. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, 13(12):1550147717741463, 2017.

[8] Jan Mycielski. Sur le coloriage des graphs. *Colloquium Mathematicae*, 3(2):161–162, 1955.

[9] Tam N. Nguyen. The challenges in SDN/ML based network security : A survey. *CoRR*, abs/1804.03539, 2018.

[10] Quamar Niyaz, Weiqing Sun, and Ahmad Y. Javaid. A deep learning based ddos detection system in software-defined networking (SDN). *CoRR*, abs/1611.07400, 2016.

[11] Thomas Rothvoß. A simpler proof for o(congestion + dilation) packet routing. *CoRR*, abs/1206.3718, 2012.

[12] Tahir Ahmed Shaik and Kotaro Kataoka. Capsaeul: Slow http dos attack detection using autoencoders through unsupervised learning. page 49–55, 2021.

[13] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. *CoRR*, abs/1609.02943, 2016.

[14] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[15] Rui Wang, Zhiping Jia, and Lei Ju. An entropy-based distributed ddos detection mechanism in software-defined networking. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15, page 310–317, USA, 2015. IEEE Computer Society.