Final Report: Network Traffic Classification for Detection of Malicious TCP SYN Flood as a Distributed Denial of Service Attack

Section 1 Introduction:

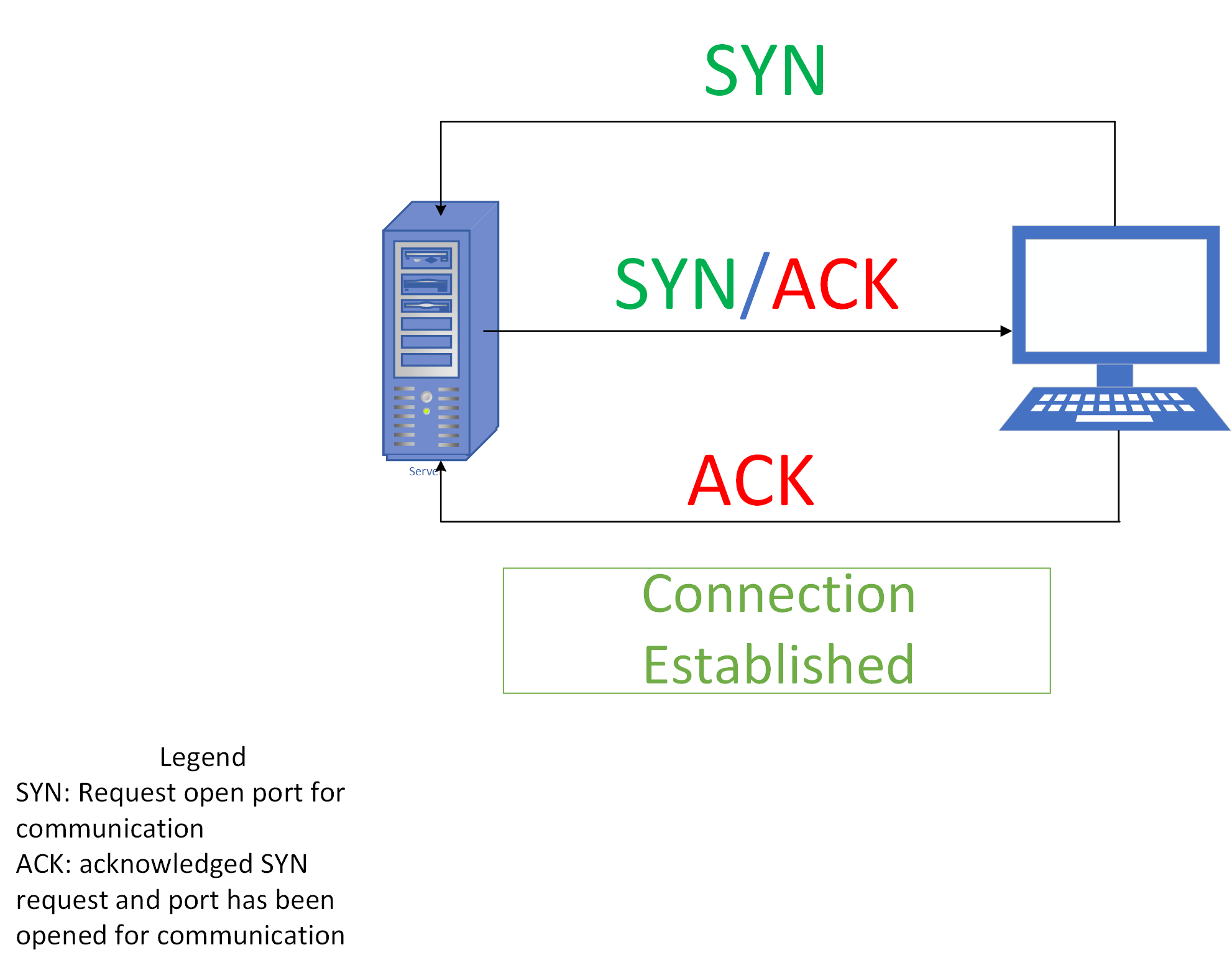
Denial of Service (DoS) attacks are a prevalent threat for any network architecture. This form of cyber attack targets a victim host over the internet by overwhelming it’s resources until it is unable to serve legitimate clients. A Denial of Service attack is commonly executed from a single malicious attack vector and is directed towards a victim over the internet in an attempt to exhaust the victim’s resources. As a result, legitimate users who wish to use the host’s services are disrupted. A more sophisticated form of this attack is known as a Distributed Denial of Service (DDoS) attack instead of a single attack vector, multiple malicious perpetrators attempt to exhaust the resources of a host at the same time. The effect is the same except DDoS attacks require a more complex defense mechanism in order to prevent or stop them.

DDoS attacks can be executed in a myriad of methods. Including, but not limited to:

1. Flooding the network to prevent legitimate network traffic
2. Disrupting the connections between two machines, thus preventing access to a service
3. Preventing a particular individual from accessing a service.
4. Disrupting a service to a specific system or individual
5. Disrupting the state of information, such resetting of TCP sessions

[1]

The primary focus of this piece will pertain to providing an accurate detection method for a specific form of DDoS attack known as a SYN Flood. SYN floods can occur on the application layer (Layer 7) or at the Network layer (Layer 3) of the OSI model. Here we will specify our focus to Layer 3. Such an attack exploits the commonly used Transport Control Protocol (TCP) at layer 3 of the OSI networking model. This specific protocol is often initiated by a client to connect to a host in order to access a service. Typically, a client opens a port to connect to the server, then sends a SYN request to the server so it may open a port for the client to connect over the internet. The SYN request is followed by a SYN/ACK reply from the server which acknowledges the SYN request from the client and opens a port for them. As well, server replies with its own SYN request back to the client for the use of a port on the client side. The process concludes with an ACK reply back to the server from the client acknowledging the use of a port on the client side. This protocol is unique in that both the client and server have now opened up a channel of communication between one another over the internet. (Figure 1)



A SYN flood abuses this protocol by exploiting the finite availability of ports that the victim may have open. This attack is executed by bombarding the victim host with multiple SYN requests at a time. As previously mentioned, the initial SYN request is sent so that the server host may open up a port for communication with the client. However, unlike the typical scenario, when the victim server sends back a SYN/ACK request the client will never reply back with an ACK reply.

By bombarding the victim host with a surplus of SYN requests it begins to open up so many of its ports that legitimate users can no longer connect to the server. Furthermore, having such an excessive amount of ports open can take a toll on the CPU cycles the host has available. As a result, none or few legitimate users may connect to the host for service since all its resources have been exhausted by the attack.

Section 2.1 Related Work:

Recent works have discussed the advent of software defined networking and its contribution to network security. Software Defined Networking (SDN) is a technique to manage the security and properties of a network exclusively using software. The benefits are immense, and it is an emerging practice with potential to provide sustainable network solutions for large scale architecture (Meti, Narayan, & Baligar, 2017). Within the realm of SDN there are a multitude of approaches to detect DDoS attacks, however a commonality is to use machine learning techniques based on historic data. However, machine learning algorithms are plentiful and a thorough survey of the effectiveness of them is necessary to determine the optimal solution (Zhang, Zhang, & Yu, 2017).

For the purposes of simplification, the majority of solutions were evaluated in a testing environment with a single point of entry into the host network. The solutions proposed were commonly implemented at that single point. As in (Degirmencioglu, Erdogan, Mizani, & Yilmaz, 2016) they implemented an adaptive approach where publicly facing firewall rules were dynamically generated to block malicious traffic as it became detected. The machine learning algorithm implemented was Naïve Bayesian. This is a more ‘Active’ solution where malicious threats are immediately dealt with upon detection. Another form of detection is a ‘Passive’ approach where the detection is done over an intrusion detection system or IDS, such as the solution proposed by (Nadiammai & Hemalatha, 2014). Their approach is merely for the detection of malicious traffic without active denial techniques in place. These two approaches accomplish the same task of detecting malicious DDoS traffic, but their implementation differs. Whether an ‘Active’ or ‘Passive’ solution is the optimal approach for malicious traffic detection depends on a variety of factors and is a problem that is out of scope of this piece.

Cloud computing is an emerging field of software engineering where large scale infrastructure is virtualized in an off premise environment. Cloud computing has its own host of security concerns specifically related to DDoS attacks as outlined by (Degirmencioglu, Erdogan, Mizani, & Yilmaz, 2016). As mentioned earlier, software defined networking is a technique which can provide promising solutions in this developing field. Therefore, there is a need for a software defined solution to help mitigate this issue. The cloud environment also provides its own set of challenges, but algorithms are being created to mitigate the issue such as the one proposed by (Al-Hawawreh, 2017). As cloud computing continues to become more widespread there must be careful consideration for the security measure necessary to accommodate this new environment.

Another emerging field of software engineering is ‘Internet of Things’ (IoT) devices. These are small scale devices that provide convenient services over the internet, typically there is a sample of them to service multiple users at the same time. [2]

However, they may also serve as an attack vector, and are especially susceptible to DDoS due to their more finite resources (CPU cycles, power, etc…). (Lee, Baik, Kim, & Yang, 2017) discusses the challenges associated with these devices as well as security measures that can be put in place to mitigate their potential threat. This is another field where software defined networking is a promising technique to implement security solutions because there are typically multiple IoT devices on a network and managing them individually can be a laborious task. Instead, a software defined approach can be implemented to oversee their security.

There are a variety of contexts where a DDoS attacks can pose a serious threat and it is important to implement a compatible solution to provide safe services to users. There are a plethora of solutions to mitigate these issues, but the realm of software defined networking using machine learning provides many possibilities to achieve the intended result.

A brief summary of the machine learning algorithms, datasets used, and more can be found in the “Literature Review” document in the “Summary” subsection located in the “Docs” directory of this project

Section 3 Dataset:

We present a brief discussion on the dataset used to evaluate the model. In order to apply the machine learning algorithms effectively, the dataset was divided into training and testing subsets with a split of 30-70 respectively (See section: Architecture and Tooling). As well, the size of the dataset was considerable enough that performance was a constant concern. There were an estimated 1.5 million samples of data to consider. The data collected was from an ISP, which has access to tremendous amounts of network traffic from a wide variety of internet users in multiple regions. Therefore, it is unlikely that any sort of bias in the dataset would be present.

Section 4 Architecture and Tooling:

We devote this section to a discussion on the pipe and filter architecture used to achieve the intended results. As previously mentioned, the dataset was of considerable size, so performance optimization measures had to be implemented in order to preserve computing time and memory. The architecture involves a number of steps in order to preprocess, generate features, discretize, and construct the machine learning model.

Section 4.1 Preprocessing:

The first step along the pipeline involves preprocessing the data before features can be generated. A number of steps are completed including:

* Changing all timing features were changed to a timestamp data type. This helped to save memory, while also providing a more versatile way to manipulate these features for the next step.
* “TCPFLAGS” and “FLOWS” were reduced to an int8 data type instead of the more commonly used int64 data type. This is another step to reduce memory use when loading the dataset.
* Packet counts and byte counts were another set of features reduced from the commonly used int64 data type to the less memory intensive int32 datatype.
* A new feature was created called “DURATION” which is the time difference between “ENDTIME” and “STARTIME”.
* Finally, the samples were arranged in order based on the “ENDTIME” feature.

This stage in the pipeline can be the most time consuming and memory intensive because it involves loading in the entire data set to effectively execute the sorting. After the preprocessing terminates, a new dataset will be created for use by the feature generator.

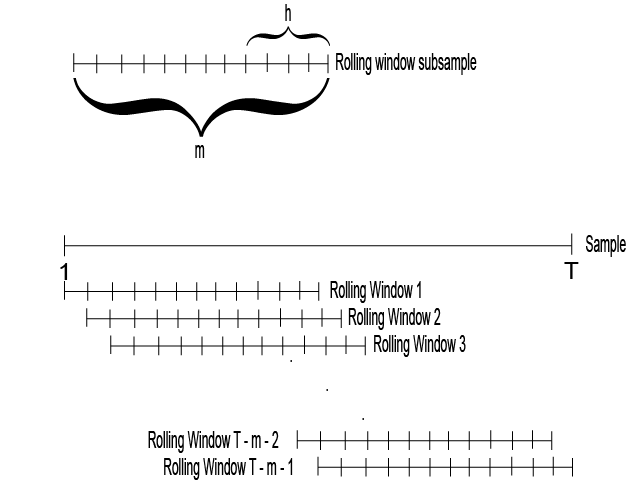
Section 4.2 Feature Generation:

The most important step of the pipeline involves generating new features from the existing ones. These types of features are known as ‘engineered features’ and can be classified into two types:

* Connection based engineered features.
* Time Based engineered features.

Both types of features were generated from a “rolling window” which is a subset of the initial dataset. The connection based features generate their rolling window by fixing a sample row x. Then the subsample is generated by considered all the previous 1000 samples from x where the “SRCADDRESS” is equal to that of x. Furthermore, a similar approach is used to generate another set of features based off the “DSTADDRESS” feature of x, and a combination when “SRCADDRESS” and “DSTADDRESS” features are the same.

The time based features generate their rolling window by looking in the previous ten minutes of the target sample x. Identically to the connection based rolling window, only samples with the same “SRCADDRESS” feature as x are considered. As well, another set of features is generated when “DSTADDRESS” matches that of x’s. This process repeats for every sample in the dataset. Refer to the diagram below for a visualization.



[3]

Using the rolling window the following engineered features were generated:

* Connection based features
  + Address Occurrences
    - The size of the rolling window
  + Distinct destination ports
    - How many unique destination ports were used
  + Distinct source ports
    - How many unique source ports were used
  + Distinct source address (or destination address)
    - For the rolling window based off source address, counts the number of unique destination address. Similarly for a rolling window based off the destination address counts the number of unique destination ports
  + Average packet count
    - Average of the packet count for all samples in the rolling window
  + Average byte count
    - Average of the byte count for all samples in the rolling window
* Time based features
  + Address Occurrences
  + Total occurrences
    - Count of the total number of occurrences in the last ten minutes
  + Distinct destination ports
  + Distinct source ports
  + Distinct source address (or destination address)
  + Average packet count
  + Average byte count

Section 4.3 Data Discretization:

This is a further step to help reduce the memory intensity when manipulating the dataset and simplify calculations when applying the dataset to the machine learning models. The primary technique used to discretize the data involves “binning” the source and address ports.

Assuming that there are 64000 distinct source and address ports, they can be subdivided into 10 ‘bins’. These bins assign the value 1-10 for each of the source or destination ports based on what interval it may lie in. For our case we obtain the following bins:

* Port number 0-6400 is assigned the value 1
* Port number 6401-12 800 is assigned the value 2
* Port number 12 801-19 200 is assigned the value 3
* Port number 19 201-25 600 is assigned the value 4
* Port number 25 601-32 000 is assigned the value 5
* Port number 32 001-38 400 is assigned the value 6
* Port number 38 401-44 800 is assigned the value 7
* Port number 44 801-51 200 is assigned the value 8
* Port number 51 201-57 600 is assigned the value 9
* Port number 57601-64 000 is assigned the value 10

Another step involved in the discretization process involves modifying the type for the “DURATION” feature. Originally the duration is of type pandas.timeDelta [4]. However, this type proved to be difficult to work with during the next phase of our approach so it had to be modified to an integer representing the length of time in milliseconds.

The final step in the discretization process involved rounding off all decimal values to their nearest integer. This step was critical so that only finite integers were considered in the machine learning algorithms implemented in the following section.

Section 4.4 Model Execution:

After the features have been generated and the data discretized the following step along the pipeline is to apply the machine learning algorithms. The three algorithms to be considered are:

* Decision Tree [5]
* Support Vector Models [6]
* Gradient Boost Tree [7]

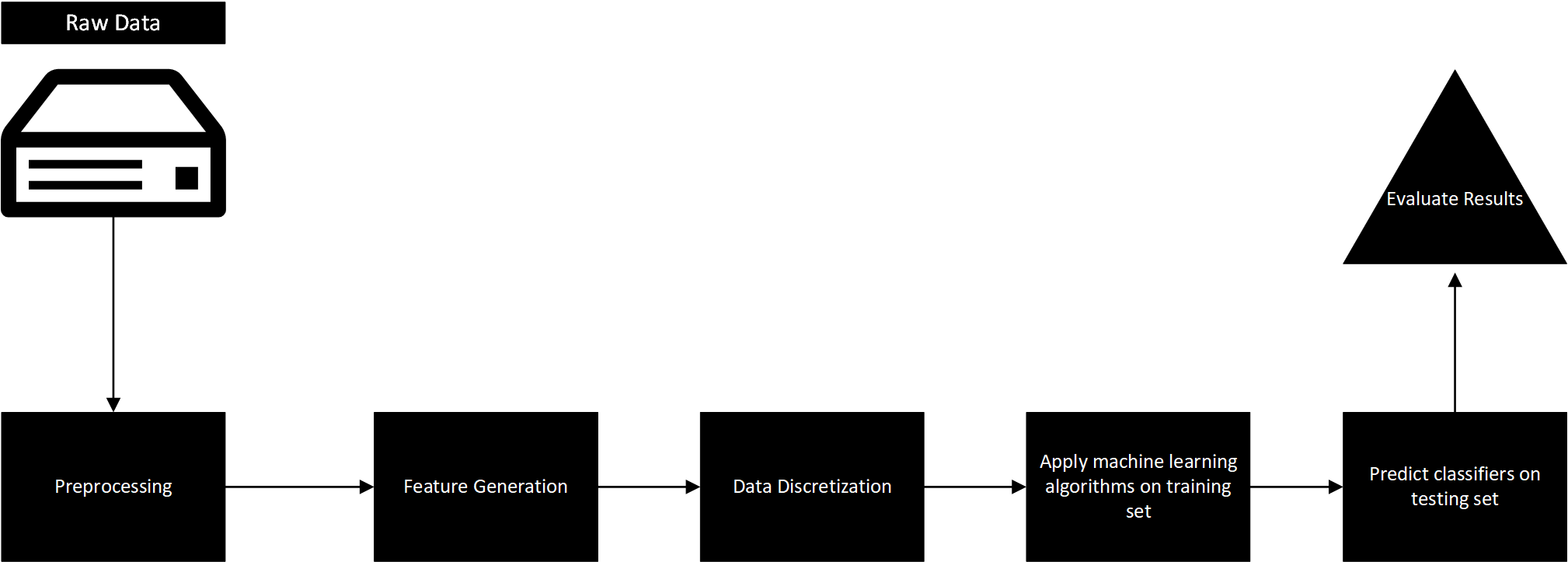
Each of which can be executed using the Sci-Kit Learn library. It must be noted however, that execution time of the feature generation was extremely long, so only a portion of the full dataset was used in this step.

Section 4.5 Results:

A summary of the results is provided below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Decision Tree Classifier | SVM Classifier | Gradient Boost Tree Classifier |
| Accuracy | 0.680485862 | 0.703185982 | 0.680485862 |
| Precision | 0.330232558 | 0.358657244 | 0.330232558 |
| Recall | 0.097353627 | 0.02783491 | 0.097353627 |

In general, the accuracy was relatively high with room for improvement, but the precision and recall were relatively low and require additional tuning in order for precision to increase.



Section 5 Test Model:

Section 6 Future Work:

We commit this section to a discussion on optimization techniques that can improve the quality of the desired results (i.e. accuracy and precision). These techniques pose their own disadvantages which will be discussed in depth, but the purpose is to demonstrate the potential in enhancing the pipe and filter process to produce better results. We will also highlight some key applications that can improve the security posture of any network architecture specifically at the consumer level with their local ISP.

Section 6.1 Multi-Level Classifier:

The models predicted some significant results, but a more optimal approach would be to harness the advantages of each of the machine learning algorithms and combine them into an all encompassing algorithm.

Another approach would be to apply a multi-level classifier to filter through the predicted outcomes and select them based off a desired level of confidence. A multi-level classifier would also benefit from an increased feature set, to potentially increase the accuracy of the detection of malicious traffic.

Section 6.2 Optimized Feature Selection:

Continuing from the discussion in the previous subsection, the drawbacks of such an approach would be the increased computing and memory resources necessary to execute the algorithms and infer the data in a reasonable time. The process of selecting an adequate number of features is another problem out of scope of this piece. The features selected in this process were done in a heuristic way without significant evidence to support the choices. Instead, another algorithm can be employed to choose the correct feature set in a way that optimizes accuracy and precision while conserving computing and memory resources. Such an approach could greatly optimize resource management while increasing the quality of the desired results.

Section 6.3 Tuning, undersampling and oversampling:

The result obtained were promising, but require more work in order for them to improve to a higher quality. Primarily, a more efficient process to generate features needs to be implemented so that the machine learning algorithms has more data to work with. This can be achieved through more powerful computing resources or optimized coding practices.

Oversampling is a technique to correct any bias involved with a dataset. It may be a technique worth considering, but the benefits for this specific process do not seem so significant since the dataset being considered is so substantial. Such a large dataset contains a plethora of information from a variety of sources so bias is insignificant.

Undersampling is an opposing technique that may be useful in this context. Since the dataset is so massive it can be exceptionally difficult and time consuming to process and extract the necessary features. Instead an undersampling technique can be implemented to reduce the large number of samples without inducing a bias in the dataset. Such a technique is plausible, but out of scope of this piece.

Section 6.4 Applications:

This pipe and filter process has impactful applications in a software defined network environment. If a significant amount of data can be compiled by a user, then the machine learning algorithms employed here can classify benign and malicious traffic in a manner that can be dealt with by common network security appliances such as firewalls or proxys. In an active defensive network environment, firewalls can dynamically generate rules to block malicious traffic classified by the machine learning algorithm to increase security. Or a passive defensive network environment can monitor the malicious traffic flows to determine the best course of action when dealing with incoming network traffic.

The most practical application envisioned with this pipe and filter process may lie in the small scale consumer market however. Most consumers of the internet get most of their internet traffic filtered through an ISP provided router. If this router is capable of collecting enough data then it can be implemented in the algorithm to detect malicious traffic for the average internet user. The advent of software defined networking helps to create an inexpensive solution to the increased threats prevalent on the internet. By blocking internet traffic from malicious perpetrators, flagged by the algorithm, a consumer level internet user can browse the internet more safely. Malicious perpetrators can be blocked with software without incurring too much cost from the ISP and providing a safer internet experience.

Of course, this process must be replicated for other forms of cyber attacks in order to be viable.

Section 7 Conclusions:

The results have great potential for improvement and seem to be promising. However, more time must be spent to optimize the process so that the accuracy and precision can increase. If this process can be improved then there may be significant applications for all users of the internet.

Section 8 References:

* Scholarly References
  + Zhang, B., Zhang, T., & Yu, Z. (2017). DDoS detection and prevention based on artificial intelligence techniques. *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. doi:10.1109/compcomm.2017.8322748
  + Meti, N., Narayan, D.G., & Baligar, V.P. (2017). Detection of distributed denial of service attacks using machine learning algorithms in software defined networks. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Doi:10.1109/icacci.2017.8126031
  + Degirmencioglu, A., Erdogan, H. T., Mizani, M. A., & Yilmaz, O. (2016). A classification approach for adaptive mitigation of SYN flood attacks: Preventing performance loss due to SYN flood attacks. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. doi:10.1109/noms.2016.7502971
  + Al-Hawawreh, M. S. (2017). SYN flood attack detection in cloud environment based on TCP/IP header statistical features. *2017 8th International Conference on Information Technology (ICIT)*. doi:10.1109/icitech.2017.8080006
  + Lee, Y., Baik, N., Kim, C., & Yang, C. (2017). Study of detection method for spoofed IP against DDoS attacks. *Personal and Ubiquitous Computing,* *22*(1), 35-44. doi:10.1007/s00779-017-1097-y
  + Nadiammai, G., & Hemalatha, M. (2014). Effective approach toward Intrusion Detection System using data mining techniques. *Egyptian Informatics Journal,* *15*(1), 37-50. doi:10.1016/j.eij.2013.10.003
  + Osanaiye, O., Cai, H., Choo, K. R., Dehghantanha, A., Xu, Z., & Dlodlo, M. (2016). Ensemble-based multi-filter feature selection method for DDoS detection in cloud computing. *EURASIP Journal on Wireless Communications and Networking,* *2016*(1). doi:10.1186/s13638-016-0623-3
* Technical References
  1. <https://www.techopedia.com/definition/24841/denial-of-service-attack-dos)\>)
  2. <https://www.webopedia.com/TERM/I/internet_of_things.html>
  3. <https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html>
  4. <https://pandas.pydata.org/pandas-docs/stable/timedeltas.html>)
  5. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
  6. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm>
  7. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html