COLL 400 Research Project

Eli Jensen
Computer Science
The College of William & Mary
Williamsburg, VA
erjensen@wm.edu

Colton Turner
Computer Science
The College of William & Mary
Williamsburg, VA
cdturner02@wm.edu

Andre Tran
Computer Science
The College of William & Mary
Williamsburg, VA
atran03@wm.edu

Marshall Wright
Computer Science
The College of William & Mary
Williamsburg, VA
fmwright@wm.edu

ABSTRACT

This project aims to monitor and categorize network traffic based on its source website. This service could then be used to improve network management, security, and performance. We have gathered network traffic data from 4 major websites: Amazon, Bing, Discord Voice Chat, and Minecraft servers. Using three basic machine learning models – K-Nearest Neighbors, Decision Trees, and Random Forests, as well as one advanced neural network LSTM model — we were able to effectively classify the traffic. Our assessment revealed that the LSTM model achieved the highest accuracy (**0.91**) and Macro F1 Score (**0.89**), showing its adeptness at handling this intricate data. It is the goal of this project to demonstrate the capabilities of machine learning algorithms when integrated into network analysis.

KEYWORDS

Network Traffic Classification, Machine Learning, Neural Networks.

1 Introduction

Packets are sent too and from computers through socket connections using MAC addresses and IP addresses. In this research we tracked web traffic and attempted to figure out where each packet was coming from without using the IPs, URLs and encrypted data. Computers and servers must send out requests to talk to others to send the information they need to. Understanding web traffic is important as it can lead to optimization and can expose websites' cybersecurity weaknesses.

To do this, we first collected our data in the application Wireshark from four distinct websites. We categorized websites into four types: single download, continuous download, continuous upload, and continuous upload and download. After this we picked what features of our data we wanted to use to classify our data. We ended up selecting the features protocol, length, time delta, and information. Then we used a series of machine learning models to classify our data. Our four models were k-Nearest Neighbor, decision trees, Random forests, and LSTM. Overall, we ended up with over 80% accuracy for all our models with the highest being 91% from LSTM, which was the most complex and the most accurate.

2 PROPOSED METHOD

2.1 Data Preparation

To properly gather website traffic data for later classification, we follow the process of data collection, feature extraction, and model fitting.

2.1.1 *Packet Capturing Tool*. In our project, we use Wireshark [1] (as shown in Fig 2), an extremely versatile network traffic analyzer, to gather and consolidate network traffic data. Wireshark boasts a comprehensive set of functionalities enabling us to capture, decode, and examine network traffic and various levels. Wireshark provides detailed insights into individual packets, allowing us to view information on utilized protocols, packet sizes, and information about the content of each packet. This information generated by Wireshark is crucial to the classification process as it provides the necessary features needed to supply our machine-learning models. The tool also accommodates a broad spectrum of protocols and formats, making it the perfect choice for analyzing diverse traffic. Finally, the ease-of-use of Wireshark is truly unmatched in the world of traffic capturing and it provides a very beginner-friendly user experience. Its interface, equipped with filtering capabilities, real-time packet capture, and comprehensive views of packet content has proved to be a vital point in our classification research.

2.1.2 *Data Collection*. To begin the data collection process, we open Wireshark, the network traffic-capturing tool, set the host filter to our desired website, and begin the capturing of packets. In our research, we selected 4 websites to gather traffic to train, test, and validate our models (outlined in Fig 1). Minecraft, Discord Voice Chat, Bing, and Amazon. We initially used a dataset that contained over 10 different website packet captures and from this set narrowed our set to just the 4. For the

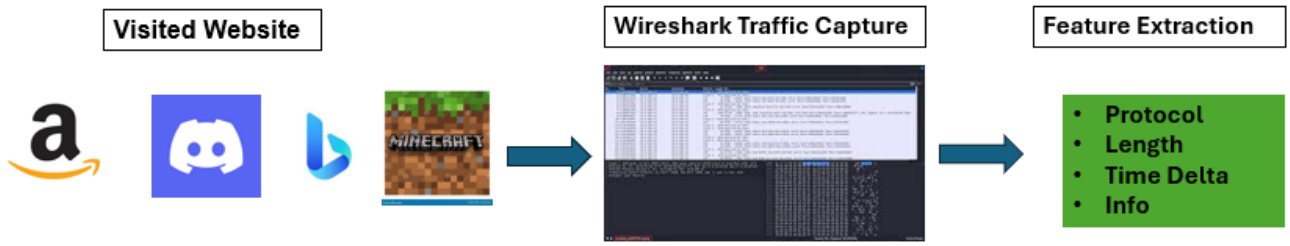

Figure 1: Data preparation method.

collection process, we start a capture through Wireshark and browse a website for a single minute. This process is repeated for each of the 4 sites each being saved to their own CSV file. After each individual record of data is created, we then used a simple Python script to remove any packets that contained incomplete data and then merged each dataset into a single file. The raw record of this data, illustrated in Fig 3, comprises close to 15000 individual packets collected from the 4 target sites.

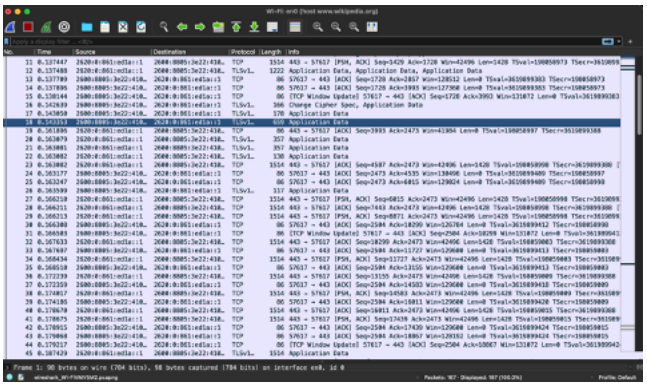

Figure 2: The Wireshark capture interface, demonstrating traffic capture from a specific source.

2.1.3 *Feature Extraction*. After we consolidated the packets into a single dataset, we moved towards extracting the specific features we would need to sufficiently train and test our learning models.

2.1.4 *Data Visualization.* Before selecting our features, to help visualize and understand the underlying patterns in a data set we created a t-SNE graph. T-stochastic neighbor embedding or t-SNE is a form of non-linear dimensionality reduction. It aims to capture the similarities between cells by placing cells from the original data close together on a 2 or 3-dimensional plot. (in this case 2D) It does so in a way where the clustering in the high dimensional space is preserved. Normally graphs of data are only viewable as a function of two variables, but using t-SNE we can view the test and target data in two dimensions with all features. For our t-SNE graph we dropped flow from our data and made "Site" the target data. Then we used standard scaler to standardize the data for better performance, followed by fit-transforming our feature data. Then we created the t-SNE graph with two components for a 2D graph and plotted the fit_transformed data in t-SNE.

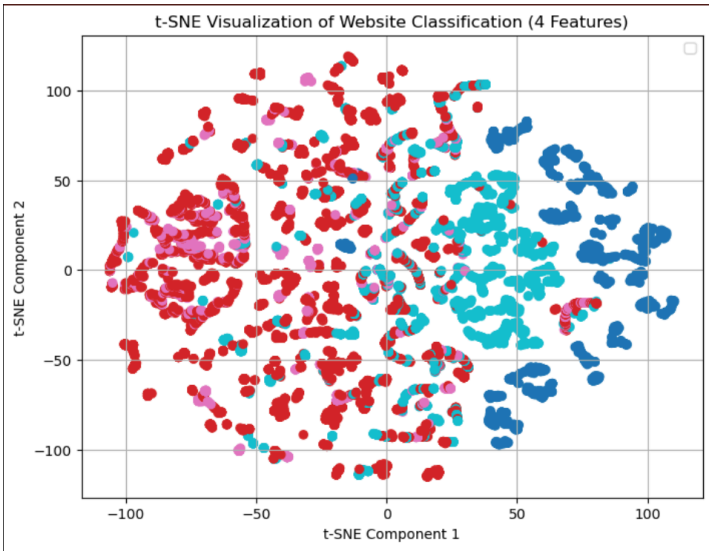

Figure 3: Our t-SNE graph with all features on a 2D-plane

From our graph, we can see how clusters are forming and where our classifiers got classifications wrong and can think about why that might be. For example, in the graph we can see a cluster of data points classified in the first two t-SNE groups but plotted with groups three and four.

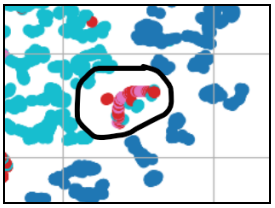

Figure 4: Data points on t-SNE graph

This is a clear indicator that these points were probably misclassified in our data and the impact from that can affect our accuracy score so we need to go back and look at our models to try and mitigate this.

2.1.5 *Feature Selection*. While the raw captures from Wireshark contain many different features we could use, we have selected 4 key features (shown in Fig 3) from each record to begin training our models:

- **Protocol**: This feature indicates the protocol used by each individual packet. Our dataset contains many packets using various protocols such as TCP, UDP, TLSv1.2, and TLSv1.3.
- **Length**: This feature shows the total size in bytes of each individual packet, like the encompassing packet headers and the payload.
- **Time Delta**: This feature is a calculation of the time since the last packet has been received. It provides a better understanding of the nature of the traffic flow from the time it takes packets to arrive from their source.
- **Info**: This feature contains a mapping of various similarities contained in packet headers, created by Wireshark, categorizing info by source/destination ports, sequence numbers, window sizes, and different forms of Application Data.

data

Protocol	Length	Info	Time Delta	Site
1	1514	3	1.00000000013978E-06	bing
1	1514	3	0.0	bing
2	1354	2	1.00000000013978E-06	bing
2	92	2	1.00000000013978E-06	bing
1	56	3	0.0	bing
1	54	3	2.60000000000815E-05	bing
1	54	16	4.69999999994641E-05	bing
2	1126	2	0.0041790000000005	bing
2	92	2	9.999999992516E-07	bing
1	54	3	7.90000000003843E-05	bing
1	56	3	0.0060690000000001	bing
1	56	3	1.00000000013978E-06	bing
2	215	2	0.0089109999999994	bing
1	54	3	8.50000000003348E-05	bing
1	1494	3	0.0072200000000002	bing
1	1494	3	2.99999999953116E-06	bing
2	1198	2	5.10000000000233E-05	bing
1	56	3	0.0133260000000001	bing
2	200	2	0.0082560000000002	bing
1	54	3	8.69999999997262E-05	bing
1	1494	3	0.0171659999999995	bing
1	1494	3	4.00000000055911E-06	bing
2	601	2	4.60000000002125E-05	bing
1	56	3	0.011819	bing
1	56	3	9.999999992516E-07	bing

Figure 5: Sample of the raw dataset containing roughly 15000 packets with our desired features.

2.2 Employed Methods

For our project on monitoring and analyzing network traffic for classification by visited site, we employ a combination of machine learning models including Random Forest (RFC), K-Nearest Neighbors (KNN), Decision Trees (DTC), and LSTM. These models play a pivotal role in our solution due to their performance in handling these large datasets for classification.

2.2.1 *K-Nearest Neighbors.* For one of our simple models of classification we decided to use k-nearest neighbors or kNN classification. This model is a method of classification that predicts where a data point should go based on the data points closest neighbors.

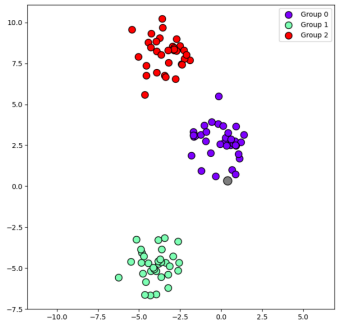

Figure 6: An example graph where the gray point would be classified with kNN classification

We choose to use the kNN model because it is rather simple and can give good accuracy for large amounts of data being input into the model. It also can be made more detailed and specific based on how detailed and specific we want it. We achieved a pretty high accuracy and proved that our data and algorithm can accurately predict which website a new observation belongs to.

Data collection for this model was the same as every model we used. The model requires training and test data for both our regular data and our target data, so split the data we collected into training and testing data with a 70/30 split. We then moved our target data into a separate data frame from the rest of our data. The classifiers we used were *list classifiers*.

To run this model, we first wanted to find the correct number of neighbors to base our model on. First we scaled the data using a standard scaler to give us better accuracy. Then we found the number of neighbors that gave us the highest accuracy for correct predictions. In our code we tested 2 neighbors to 20 neighbors and graphed the accuracy given. As seen in Fig. 7 below we found that the number of neighbors that gave us the highest accuracy was 4 followed closely by 6. This gave us an accuracy of almost 88%.

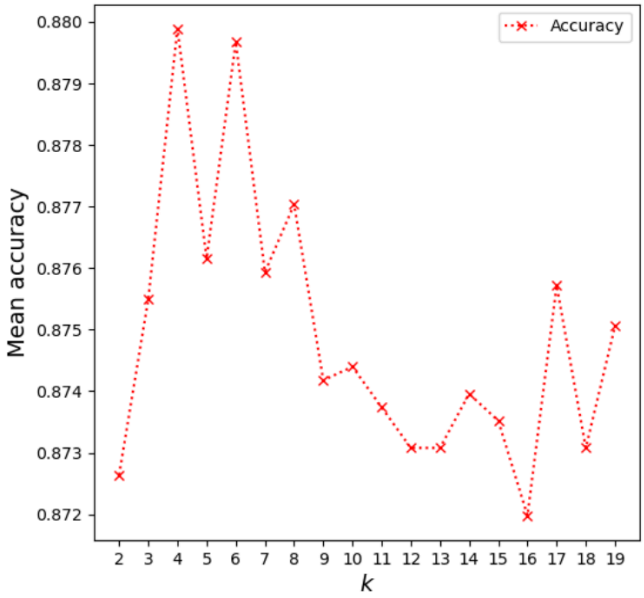

Figure 7: Graph depicting the accuracy of k neighbors

After deciding on 4 neighbors we fitted the training data and predicted the target values. Then we compared this to our actual values in our test data set and our kNN score was 89.9%.

After scoring our predictions we ended with an accuracy of **0.88** and a Macro-F1 score of **0.80**. It can be seen in Fig. 9 that Discord had the best results using kNN with only one overall error. The biggest issue with this model was scaling the data. We were worried about having over fit data, but initially before scaling the data we had a lower accuracy. After scaling the data we had a much better accuracy and also attempted to weight the data, however found no major statistical difference. This model was a good baseline model to use as it gave us a good accuracy, but it may not handle new data well.

2.2.2 *Decision Trees.* One of the models we used to classify the data collected from our network traffic was a decision tree classifier (DTC), which is a simple machine learning model that takes a given set of data and uses features extracted from that data to make decisions on how to classify the data by splitting the data into smaller subsets. When the DTC cannot split the data further, the leaf nodes from the resulting tree represent the DTCs predicted classification of each point or subset of data. In order to make these splits as efficiently as possible within a reasonable amount of time, the DTC uses a greedy approach to forming the tree by looking at each subset of data separately, and choosing a split that will only split that subset of data as cleanly as possible [2]. The way a feature was used to split one subset of data may not split another subset cleanly.

We chose to use a DTC model in our network traffic analysis because this model required very little data preprocessing before it was able to train on the data. Also, this model performs well with large datasets, which was perfect since our dataset contained almost 15,000 individual packets to analyze. One problem with this model is that there is a chance that the model will be overfit for the training data, which is where it can perfectly classify the training data but can struggle to accurately classify anything outside of the training data. We address this complication by also running our data through a random forest classifier as well.

After training the DTC, our model resulted in a **0.8** accuracy score with a **0.73** Macro-F1 score (shown in Table 1) . The model was very accurate at predicting the network traffic coming from Amazon and Minecraft, and only had some trouble distinguishing between Discord and Bing. While this is not an amazing accuracy level of classification, for it being our simplest model we were pleased with the results. Also, this model only classified the packets on a per-packet basis and does not look at the flow of the packets over time.

2.2.3 *Random Forest.* Random Forest is an ensemble learning method primarily used for classification and regression purposes. It operates by building multiple decision trees during the training phase and combining their

outputs through a majority vote to arrive at a final decision [3]. This so-called forest of decision trees (illustrated in Fig 4) is utilized to create feature randomness in an uncorrelated forest of trees. A Random Forest operates as such:

- Building Phase: Multiple decision trees are created and built from bootstrapped data in the training set. Each tree has a subset of features randomly chosen, allowing for a split at each node.
- Majority Voting: As there are many different trees within the forest that each come to a classification decision independent of one another, a majority voting mechanism is put in place to allow the class with the most votes to become the final prediction.

The key benefits of using a Random Forest are:

- Versatility: Random Forest can be applied to both classification and regression tasks, making it applicable to a wide range of predictive modeling problems.
- Robustness: Random Forest is robust to overfitting, a common problem in single decision tree models. By aggregating predictions from multiple trees, it reduces the risk of capturing noise in the data and provides a more stable prediction.
- Parallelization: They can be easily parallelized, allowing for efficient training on large datasets by distributing computation across multiple processor cores.

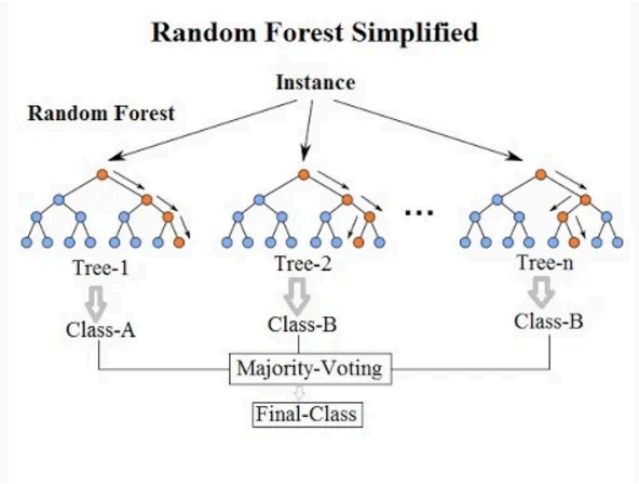

Figure 8: Generic example of a Random Forest classification model.

2.2.4 *Recurrent Neural Network and LSTM.* The recurrent neural network (RNN) is a type of neural network that allows bidirectional flow between its layers. This allows ordered sequential data to be input, where prior inputs may influence current inputs and outputs. RNNs are often used in problems that deal with time series or ordered sequences, such as in natural language processing or prediction problems. However, RNNs are known to suffer from the exploding or vanishing gradient problem, in which the weight parameters for each layer lead to a very large gradient or vanishingly small gradient, resulting in an unstable model or a model that does not learn, respectively. To solve this problem, the long short-term memory (LSTM) model introduces three gates, the input gate, output gate, and forget gate, that help the model determine which information is necessary and helpful to predict the correct output. Because the LSTM model does not suffer from the same vanishing or exploding gradient problems as the basic RNN architecture, it is generally more popular and widely applied for problems that deal with time series and sequences.

2.2.4.1 *LSTM Data*. In order to balance the load of collecting data, time to train the model, as well as sample size, we decided to split each network traffic flow into packet "slices" (20 packets). Each packet slices has the same training features as in the 3 basic models, but we added an "IP source/destination" column that contained information about whether the user or the website was the original sender of that packet. We removed information about the actual IP address, and instead labeled it with 0 or 1 for user and site respectively. Because the LSTM model could learn patterns from each packet sequence, we were able to use this capability to include a new feature which captured the nature of internet flows as bidirectional, whereas the basic per-packet models might not have necessarily found this information useful.

3 Evaluation

In this section, we will provide an in-depth explanation of the performance of our proposed models on the dataset.

3.1 Evaluation Metrics

We use two primary metrics to evaluate the models: Accuracy and Macro F1 score. These metrics provide a comprehensive view of performance across all classes.

Accuracy is defined as:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples}$$

Precision indicates how many predicted positive samples are really positives, and is defined as:

$$Precision = \frac{Total\ Positives}{True\ Positives + False\ Positives}$$

Recall explains how many of the positive samples were correctly identified as positive and can be defined as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The **Macro F1 Score** which we use as a major determining factor in the effectiveness of our models, takes the mean of precision and recall for each class and then averages this, it can be defined as:

$$Macro\ F1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2\ x\ Precision_i\ x\ Recalll_i}{Precision_i + Recall_i}$$

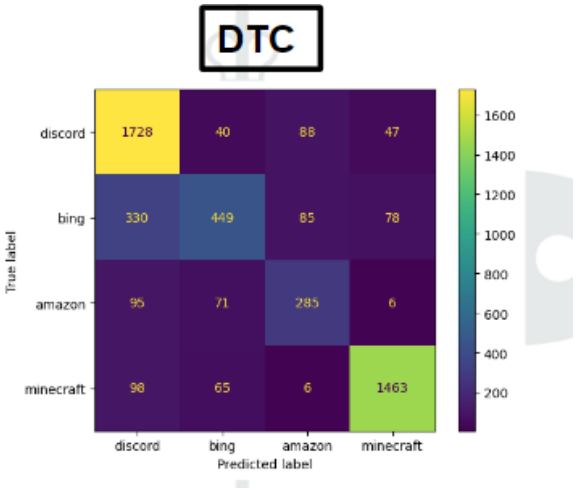

Figure 10: Confusion Matrix from Decision Tree classifier model during testing.

3.2 Evaluation Results

Model	Accuracy	Macro F1 Score
K-Nearest Neighbor	0.88	0.86
Decision Tree	0.80	0.73
Random Forest	0.80	0.74
LSTM	**0.91**	**0.89**

Table 1: The accuracy and Macro F1 Score of each of our proposed models.

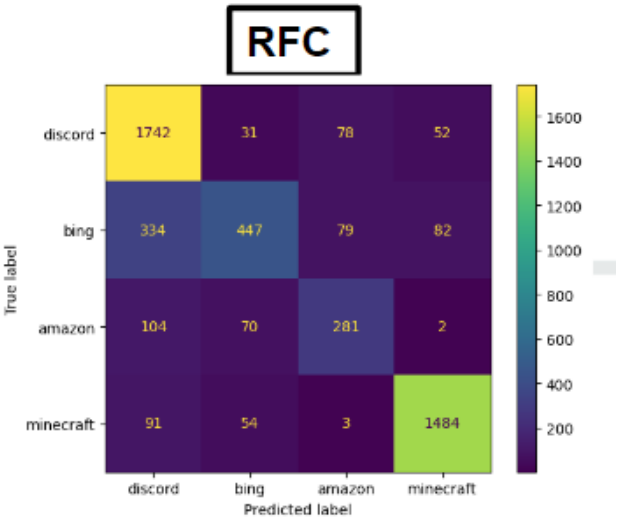

Figure 11: Confusion matrix from Random Forest classifier model during testing.

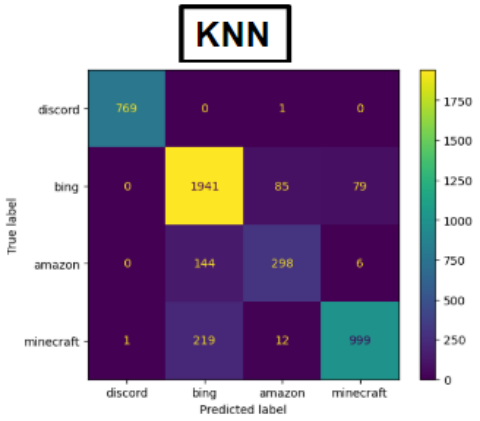

Figure 9: Confusion matrix from the KNN model during testing.

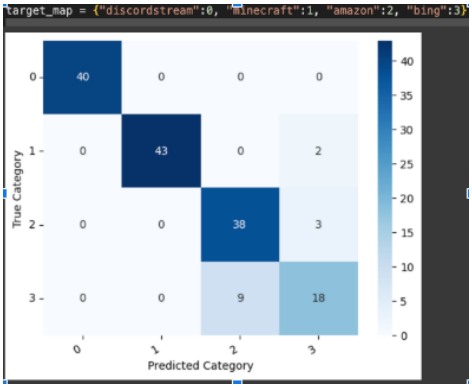

Figure 12: Confusion matrix from LSTM model during testing.

4 DISCUSSION & FUTURE WORK

4.1 Interpretation of Results

Using our data collected from wireshark, we employed the use of a DTC, RFC, and KNN model to classify the packets with the new feature included. These models had decent levels of accuracy, with **0.8**, **0.8**, and **0.87** accuracy scores respectively. However, these models still performed their classifications on a per-packet basis. We felt that an LSTM model would work best to test our hypothesis on the importance of the packet flow due to the LSTMs ability to use time and packet order as a factor in its classification. Our LSTM model had an accuracy of **0.91**, which was significantly higher than the accuracy of the basic models that were run on the same data. This accuracy increase serves as an indication that the flow of the packets can be very helpful in identifying the source of network traffic, as opposed to simply looking at the packets individually.

4.2 Future Work

To expand on the work conducted in our paper, we would make improvements in data size and expand the data set to include more websites, especially those that are more similar to each other. Our model performs well in analyzing traffic to and from four websites that are relatively different in terms of their dominant protocol and between-packet time intervals, however, in the future, we would like to see if we can improve on a model to classify a diverse range of websites that may or may not be similar to each other. Furthermore, future research is needed to find further applications of web traffic analysis using machine learning models in order to implement our proposed methods or discover further areas of work to improve our models.

5 CONCLUSION

Our goal was to analyze and classify network traffic using machine learning models along with the packet information of all the packets going through the network. We collected all of our data using Wireshark and removed the source and destination IPs, and then we found the time since the last packet was sent for each packet and added it as a feature of the dataset. This is because we hypothesized early on that the flow of the packets would be important in classifying the data. When we ran our models on the network data, our basic models had accuracy between 0.8 and 0.87 while only analyzing the data on a per-packet basis, but when the data was run on our LSTM model our accuracy grew to 0.91, which seems to corroborate our hypothesis. The main insight to take away from our findings is that the flow of the packets over the network is very important in classifying the origin of network traffic. Models that can take this time flow into account, such as an LSTM model, will be more accurate at this classification than models that just examine each packet individually.

REFERENCES

[1] Wireshark Foundation. 2024. Wireshark. https://www.wireshark.org/
[2]What is a decision tree?. IBM. (2024, March 26). https://www.ibm.com/topics/decision-trees
[3]]IBM, “What is Random Forest? | IBM,” *www.ibm.com*, 2023. https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly
[4] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. Raymond Choo, and M. Sarsour, “Network traffic classification: Techniques, datasets, and challenges,” *Digital Communications and Networks*, Sep. 2022, doi: https://doi.org/10.1016/j.dcan.2022.09.009.
[5] “9.2. Long Short Term Memory (LSTM) — Dive into Deep Learning 0.14.4 documentation,” *d2l.ai*. https://d2l.ai/chapter_recurrent-modern/lstm.html
[6] IBM, “What are Recurrent Neural Networks? | IBM,” *www.ibm.com*, 2023. https://www.ibm.com/topics/recurrent-neural-networks