ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

Classification of encrypted traffic using deep learning

[1]Kausar Mia, [2]Md. Tamjeed Monshi, [3]Ms. Shanta Rani Saha, [4]Sonjoy Prosad Shaha, [5]Arnab Saha, [6]Angkur Dhar, [7]Md. Tasnim Forhad Rimon, [8]Sadid mahamud

Daffodil International University

Dhaka, Bangladesh

DOI: https://doi.org/10.5281/zenodo.7654314

Published Date: 19-February-2023

***Abstract*: In this paper we examine and evaluate different ways of classifying encrypted network traffic using neural networks. For this purposes we create a dataset with a streaming/non-streaming focus. The dataset comprises seven different classes, five streaming and two non-streaming. The thesis serves as a preliminary proof-of-concept for Napatech A/S. The most desirable approach for Napatech is to classify individual packets upon arrival. Our experiments show that this approach is infeasible due to the content being encrypted and therefore without any distinguishable information. We therefore propose a novel approach where the unencrypted parts of network traffic, namely the headers are utilized. This is done by concatenating the initial headers from a session thus forming a signature datapoint. In experiments using the header-based approach we achieve very promising results, showing that a simple neural network with a single hidden ii layer of less than 50 units, can predict the individual classes with an accuracy of 96.4% and an AUC of 0.99 to 1.00 for the individual classes. The thesis hereby provides a solution to network traffic classification using the unencrypted headers**

Keywords: **Deep Learning, Encrypted Traffic, PCA, t-SNE, Neuron Network.**

1. INTRODUCTION

1.1. Technological advances has lead to an increase in overall network traffic. The many streaming services, expansion of network coverage and the major increase in the amount of devices connected [8], calls for optimal techniques for network bandwidth management. Even though bandwidth technology evolves in the attempt to serve the high demand, its current growth rate cannot compete with that of global internet traffic [25]. Network bandwidth management techniques can help ensure that the "most important" traffic on a network is prioritized. Some traffic may be prioritized higher if for example it is related to a service that has a quality of service (QoS) agreement e.g. some tasks can be more important than others (gaming, VoIP etc.). An important aspect of network bandwidth management is classification of the traffic on the network. Being able to determine which applications network packets belong to, will allow for a policy specification that can prioritize the right traffic. A traffic classification task should be automated and be able to divide network traffic into several classes (application specific or predefined) based on characteristics available in the individual network packets or by a collection of such. The nature of this task thus seems very suitable to solve using a machine learning classification algorithm. Our author's consider a NN in order to solve this problem.

1.2. Classical Approaches

Different approaches have been utilized for the task of classifying network traffic. The most simple is port based classification, where the port number can inform about the protocol used for transmission. With the rise of Peer-to-peer (P2P) network traffic in which some protocols use ephemeral ports, that might fall in the range of other applications e.g. a P2P service might use port 80 which is normally associated with HTTP traffic. This might be done in order to obfuscate the

traffic [19], hence port based classification has fallen out of favour. A limitation of only looking at the port number is that it does not say anything about the content, but rather give a suggestion on the protocol (HTTP, FTP, SSH etc.) used by the application. Deep Packet Inspection (DPI) has shown to be superior to the port based approach when trying to classify P2P network traffic as shown by Sen et al. [43].

1.3 Napatech A/S

Napatech is a global company founded in Denmark in 2003, they entered the Oslo Stock Exchange in 2013 [38]. Napatech have developed a suite of products all in the area of high speed network accelerators. They are designed for real-time network monitoring and analysis from 1Gbps to 200Gbps with zero packet loss. At the core of Napatech's product range is their SmartNICs, which is a range of network accelerators. They are designed to be used in ordinary servers, as such they are typically sold to OEMs. Napatech's SmartNICs are used in a wide range of industries such as cyber security, telecom operators, financial services, cloud and data center, network management, infrastructure and defence. No matter the industry Napatech's SmartNICs provide insights into the performance and operation of a network. In order to ensure that zero packets are dropped Napatech's products are built around an FPGA that can scale to handling the millions of packets flowing through the network.

1.4 Thesis Objective

This thesis aims at exploring the possibilities of classifying network traffic by use of machine learning. Our main fouces at distinguishing between streaming and nonstreaming network traffic. As both streaming and non-streaming network traffic can be transmitted by a protocol such as HTTPS which use en-cryption, the solution has to predict the content type of packets and not only distinguish between protocols used for transmitting the packet.

1.5 Related Work

Some of the work done by others concerning classification of network traffic using machine learning. The majority of the papers work with network flows or network sessions or some features derived from those sessions/flows. In a very recent paper, Michael et al. [34] use a very simple NN consisting of a single hidden layer with 12 units, achieve a very good accuracy of 99% on a dataset containing 12 different classes. They create a 10-attribute feature vector based on statistics gathered from TCP sessions. One of the features they use is the port number which is a huge discriminator given the type of traffic they are dealing with. Furthermore they test the temporal stability of their model by training on a dataset generated in one day and testing on a dataset generated a year later on the same machine. They find that the model only perform slightly worse on the temporally displaced data as their accuracy drop from 99% to 96:7%. However when testing on data collected three years later than the training data they see a drop in performance to 89:6%.

1.6 Source code

The relevant code created throughout this project can be found on our group member.

1.7 Thesis Structure

This section outlines the structure of the thesis and provides a brief introduction to the content of each chapter. The structure is as follows: • Chapter 2: Machine learning outlines the theoretical foundation from which we will try to construct a solution to the posed task. • Chapter 3: Network theory outlines some of the theory needed to gain a better understanding of the network traffic domain. • Chapter 4: Dataset describes how the dataset came to be, as well as describes some of the tools and practises behind the generation of the dataset. • Chapter 5: Experiments describes the different experiments made in the process of classifying network traffic, as well as interpreting neural network decisions. • Chapter 6: Discussion reflects on some of the findings and decisions made along the way. • Chapter 7: Conclusion outlines the key findings as well as provide an overall conclusion to the project and suggests some interesting directions of further work.

2. MACHINE LEARNING

In the previous chapter we introduced the problem and setting of classifying network traffic with the focus of separating streaming from non-streaming traf1fic. Some of the theoretical and mathe-matical foundations on which machine learning is based on. In the first part of the chapter, methods of dimensionality reduction and visu1alization is presented. The 2 nd part of the chapter is theory behind neural networks and presents a technique for visualizing what part of an input that is most relevant for a specific prediction made by the NN.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

2.1 Data exploration

When facing a machine learning task, one of the first steps towards a positive result is to obtain knowledge about the data. This can include many things, such as summary statistics of the different attributes. These statistics describe things like the minimum, maximum, mean and standard deviation of each individual feature, e.g. illustrated with box plots. Another important step can be to explore different plots of the dataset. Often the dataset in question is of very high dimensionality, which can be very hard to visualize and comprehend for humans. Equations.

2.1.1 PCA

A Principal Component Analysis is a mathematical tool invented by Pearson [41] in 1901 but was independently developed and named by Hotelling [18] in 1933. The idea behind a principal component analysis is to take a lot of correlated variables and reduce them to a smaller set of uncorrelated variables called principal components. As the principal components are uncorrelated it means that they are orthogonal to one another. The direction of the first principal component is the direction that accounts for as much variance in the data as possible, each succeeding principal component accounts for as much of the remaining variance in the data as possible under the constraint that it is orthogonal to the previous principal components. As the principal components account for an increasing amount of the variance in the data, they can be used for dimensionality reduction. It might be that a dataset of high dimensionality have all of its variance in only two directions. It is then possible to describe all datapoints using a linear combination of the first two principal components. It is however important to note that the dimensions that the principal components represent might not be interpretable. In order to find the principal components we calculate the eigenvectors and eigenvalues of the covariance matrix. Let A be a matrix of size n m and n m for which m is the number of dimensions or features and n is the number of rows/datapoints. By subtracting the mean from each column we obtain the matrix B:

$$\mathbf{B} = [b_{i,j}] = \left[a_{i,j} - \frac{1}{n}\sum_{i=0}^{n} a_{i,j}\right]$$

Then we calculate the covariance matrix $\mathbf{C} = \frac{1}{n}\mathbf{B}^{\top}\mathbf{B}$

2.1.2 t-SNE

t-distributed Stochastic Neighbour Embedding (t-SNE) is a visualization technique that works well on datasets of high dimensionality. t-SNE can be used as a nonlinear dimensionality reduction tool, but is particularly well suited for visualization of high dimensionality data-sets. t-SNE was invented by Maaten and Hinton [32] in 2008 and builds upon SNE which was invented by Hinton and Roweis [17] in 2002. t-SNE works in three stages: 1. Compute the matrix of pairwise similarities pjji on the original highdimensionality dataset. pjji is the conditional probability that a datapoint xi would choose datapoint xj as its neighbour given a Gaussian distribution centred at xi. Mathematically this is expressed in

$$p_{j|i} = \frac{e^{-d_{ij}^2}}{\sum_{k \neq i} e^{-d_{ik}^2}}$$

$$d_{ij}^2 = \frac{\|x_i - x_j\|^2}{2\sigma_i^2}$$

2.2 Neural Networks

ANN are computing systems vaguely inspired by the architecture of the biological neural network found within animal brains. Much is still unknown about the brains capabilities to process information, though it is known that an essential part is its interconnected neurons that output spikes of electrical activity due to electrical activity outputted by other neurons. The performance of the human brain in complex tasks like facial, speech and image recognition is admirable and drives the

interest of exploring ANN for accom-plishing similar learning tasks using computers. Like a biological neural network, the artificial neural network is constructed from interconnected nodes similar to that of a biological neuron. The nodes are arranged in layers by directed links, and a network is usually constructed having an input layer, one or more hidden layers and an output layer.

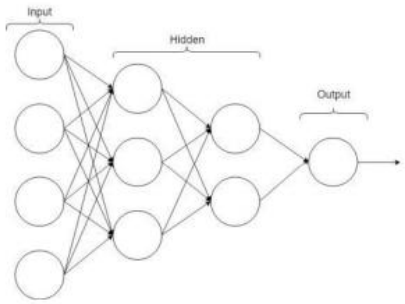


Figure 2.1: A feedforward neural network with two hidden layers.

2.2.1 Neuron

The artificial neuron is the processing unit of a neural network. A neuron consists of inputs, weights, a bias and an activation function that are all used to calculate an output. Each input is multiplied by the weight that expresses the importance of that input to the output of the neuron. A simple neuron is the one known from a Perceptron network [40]. A Perceptron is a single layer feedforward network with a neuron that takes several binary inputs x1; x2; :::; and outputs a single binary value.

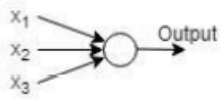


Figure 2.2: A Perceptron network.

We also introduce the dot product operator in the expression for simplification.

output = 1; if w x + b > 0

0; otherwise

2.2.2 Network architecture

When designing an ANN the architecture has a large influence on which functions the network can model. A single layer feedforward neural network is not able to find the optimal solutions for problems with a nonlinear input/output relation. Adding a hidden layer between the input and output layer of an ANN will according to the universal approximation theorem [5] allow the neural network to approximate any continuous function to the desired precision by adding neurons with a nonlinear activation function to that hidden layer. however we can potentially approximate a more complex model with fewer units than a similarly performing network with fewer layers [33].

2.2.2.1 Early stopping

When training a model which is complex enough to overfit the training data, a strategy called early stopping can be applied. Early stopping is a commonly used regularization technique due to its simplicity and effectiveness. When monitoring the training error and the validation error during training of a model, one might experience that the training error continuously decreases while the validation error starts increasing again. This is a typical pattern seen when the model is overfitting to the training data. we must expect that the model would generalize better if we stopped training at the point where the validation error was at its lowest. This is exactly what early stopping accomplishes.

2.2.2.2 Dropout

Dropout is an efficient technique that approximates that of combining expo-nentially many different neural network architectures [45]. The technique is called dropout as it is achieved by eliminating the output of a number of the non-output units within the network.

2.2.3 Evaluating a classifier

Evaluating the performance of a neural network used in a supervised learning task is highly dependent on the nature of the task. As training of the neural network progresses, both loss and accuracy is often calculated on a validation set. Accuracy can however not be used as the only performance measure due to the nature of some classification tasks. Solving a binary classification task with an unbalanced dataset where 70% of the observations belongs to class 1, a naive neural network would achieve 70% accuracy by always predicting 1, even though this would clearly not be a good model.

2.2.4 Confusion matrix

A confusion matrix is a way to visualize the performance of a classifier. A confusion matrix is a c c matrix, where c is the number of classes. Each column of the confusion matrix displays the instances of a predicted class while each row shows the actual class instance.

		Prediction		
		Positive	Negative	Total
Ground truth	Positive	T P	F N	TP +FN
	Negative	F P	T N	FP +TN
	Total	TP +FP	FN +TN	N

3. DATASET

In the previous chapter we outlined the theory of network traffic. We have used this to create a dataset. In this chapter we will describe some of the thoughts and work that went into creating a dataset for a supervised learning task.

3.1 Existing datasets

Existing datasets for classification of network traffic are widely available, how-ever they all seem to have a different focus than what we were interested in. Some of them relies on extracted statistical features (e.g. mean time between packets, average packet size, etc.) from flows or sessions. Other existing datasets that are not relying on extracting features have a different focus, such as malware detection [27], VPN-nonVPN [13] classification and intrusion detection [44].

3.2 Classes

As mentioned in a previous section, the focus of this thesis is classification of network traffic in order to identify video and audio streaming. As most streaming services use the same application-layer protocols (HTTP/HTTPS), we have simplified the problem by only including classes that utilize these. Protocols such as SMTP, FTP etc. is left out as they are not of particular interest in this streaming/non-streaming setting. The following will present the chosen classes and their characteristics:

HTTP: Regular unencrypted web traffic will be labelled http. This type of traffic is usually regular web browsing where a server is serving content on TCP port 80 [20]. Different types of web-browsing is needed to ensure diversity in the amount and length of network packets. Even though the traffic itself is not encrypted the TCP payload might still be encrypted at the application layer.

HTTPS: Regular web traffic, encrypted using SSL/TLS will be labelled https. This type of traffic is usually regular web browsing where a server is serving content on TCP port 443 [20]. Approximately half of the Internet traffic is using this protocol [12].

Netflix: Streaming data from Netflix will be labelled netflix. This is the actual video and audio stream from Netflix's CDN and not the browsing of the Netflix homepage. The video/audio streams are served via HTTPS using TLS v1.2 encryption as described by Stewart and Long [46]. The streams are served on TCP port 443. Netflix use a JavaScript player to buffer and assemble fragments which appears to be in ISO BMFF Byte Stream Format.

DR TV: Streaming data from DR TV will be labelled drtv. The stream is served by Akamai CDN via HTTPS using TLS v1.2 over TCP port 443. DR TV uses a JavaScript player to buffer and assemble MPEG-2 TS-files/fragments which are

found in a m3u8 manifest file which is a plain text file format. Some DR TV streams are encrypted on the application layer while others are not.

3.3 Generating network traffic

In order to label network traffic correctly, we needed a controlled environment. In this environment we needed several different frameworks, libraries and differ-ent techniques in order to create a correctly labelled dataset. In this section we will describe the most important ones.

We have used Python, because this allows for utilization of many different libraries. As the project includes capturing of network traffic, calculations of general statistical measures, visualizations and machine learning, Python allows us to include all of this functionality within the same project. The following sections will describe the most important aspects of how we captured network traffic.

3.3.1 Selenium

Selenium is a framework that can automate tasks within different browsers and is mainly used for web application testing purposes. Utilizing the key interface of the framework called Selenium WebDriver, allows for opening a browser instance as a user would, and perform interactions within the browser by use of locating elements within the DOM.

3.3.2 HTTP(S) getter

In order to generate HTTP and HTTPS traffic in a controlled way, we used a traffic generator written in Python called PyTgen however the code was written in Python 2 and we therefore made the necessary adaptions to make it run in

Python 3. PyTgen works by sampling randomly every 10 seconds from a list of host addresses (see appx. B), requesting the site and repeating for a fixed amount of tries, we chose only one try in order to not cause any suspicion. Had we chosen more retries then PyTgen would wait for a random amount of time bounded by an upper limit which can be specified by the user. This process continues in a fixed time window in order to switch between traffic types for easier labelling.

3.3.3 Wireshark

Wireshark is an open-source tool to analyze network packets. Such a tool is often referred to as a network analyzer or sniffer. Its main functionality is to capture network traffic as data flow across a network interface.

Capturing traffic with Wireshark requires the user to specify which interface it should listen to, this can be a wireless network card, an Ethernet port, USB port, Bluetooth port, etc. Wireshark displays the captured packets' information from the data link layer and up (without the preamble and frame checksum). Wireshark provides filtering functionality for extracting pack-ets with certain characteristics, for example if one wants to locate the packets coming from a specific source, a filter on the source ip can be applied. Wire-shark can export a collection of packets in different file formats, among those is the pcap file format. Wireshark is also available as a terminal oriented version (tshark) that allows for capturing and displaying packets when a GUI is not needed. Tshark has been utilized to capture network traffic.

3.3.4 Pcap

Pcap is an API30 for capturing network traffic. On Unix systems pcap comes with the library libpcap, while it on windows is most commonly known as WinPcap. As libpcap is the "de facto" standard library for network capturing on Unix systems, its file format has also become the standard. The extension of the file format is .pcap.

3.4 Post Processing

In order to process the captured network traffic we have applied several methods to extract the desired information. This section will outline the most important methods.

Scapy- Scapy is a Python library which can manipulate, forge, decode, send and capture network packets. It can be used for reading pcap files and display the content of such, sorted by distinct network traffic flows. This tool is therefore of great use in a process that involves pcap files and the task of extracting specific information from those.

Custom session extractor Scapy comes with a built in "session" extractor, unfortunately it extract flows and not sessions as we pursue. Thus we created our own session extractor that groups packets flowing in both directions identified by the five tuple:

(source IP, source port, destination IP, destination port, protocol type)

where the source and destination IP and ports are interchangeable and packets transferred in either direction will be grouped together by our session extractor.

Correct labelling - threshold In order to ensure that only the correct streaming sessions were selected and labelled as their proper class, we decided to set a minimum threshold for how large the session should be in order to be labelled as the relevant streaming service. By manually analyzing several pcap files in Wireshark we found a threshold of 5000 packets in a session to be reasonable. The threshold is large enough to not include the browsing of the web pages done by the Selenium automation framework, but small enough to include most streaming even if the stream was slow to start.

Packet representation- As the packets come in the pcap format as a byte string we decided to interpret the individual bytes as integers in the range [0::255]. This way we get a vector of integers in the same length as the original packet.

Packet anonymizer In order to ensure that our machine learning model did not base its classification decisions on for instance UDP/TCP port numbers we decided to anonymize each frame. By setting the first 12 bytes of a packet to zero we removed both source and destination MAC address, as each address is 6 bytes long. Setting bytes 26 to 33 (inclusive) to zero removes the source and destination IP-addresses as each IPv4 address is 4 bytes long. Setting bytes 34 to 37 (inclusive) to zero we remove the port numbers. The port numbers are described using 2 bytes giving a port range from 0 to 65535. As described in chapter 3 the checksum of the IP header and the TCP/UDP header is calculated as a ones' compliment on 16bit words. As the checksum is calculated on the headers, the IP addresses and port numbers are used in the calculation. Since ones' compliment is a simple operation, it might show some correlation to the different IP addresses and port numbers and we therefore decided to remove the checksum from the IP header (bytes 24 and 25) and TCP/UDP header (TCP: bytes 50 and 51, UDP: bytes 40 and 41) of each packet. Figure 4.2 provides an overview of anonymized the header bytes.

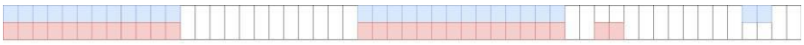


Figure 4.2: The anonymised header bytes. Blue indicates which bytes are anonymised in a packet from a TCP session. Red indicates which bytes are anonymised in a packet from a UDP session.

3.5 How to streamline the process

As we quickly discovered, capturing streaming traffic was a very slow process. Each streaming service needed to stream for 60 seconds in order for the sessions to be large enough to stand out from all other sessions.

Pandas and HDF format Hierarchical Data Format (HDF) is a file format designed to store large amounts of data. Experiencing that opening pcap-files using Scapy could take more than an hour, we extracted the relevant fields from the pcap files and stored them in Pandas DataFrames, which we then saved in the HDF file format. By doing so we were able to open a HDF file using Pandas in approximately one second instead of an hour. The fields that we chose to extract and store were the following:

Time the frametime as recorded by tshark.

• IP destination the IPv4 address of the destination host.

• IP source the IPv4 address of the source host.

• Protocol the protocol used in the session, TCP or UDP.

• Port destination the port used on the destination host.

• Port source the port used on the source host.

• Bytes the raw frame bytes as captured by tshark.

• Label the class-label of the session that the packet belongs to.

By storing those field we were able to uniquely identify and group each flow by its five-tuple and get the corresponding raw bytes and true label.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

4. EXPERIMENTS/METHODOLOGY

In the previous chapter we presented some of the thoughts and methods used when creating the dataset used in this thesis. The purpose of this chapter is to describe the methods, experiments and results we have made in the process of classifying network traffic.

4.1 Payload experiments

In this section we will explore the possibility of solving the classification task using transport layer payloads. This is a highly desirable solution for Napatech as a hardware implementation of such a classifier could be implemented using limited memory, making it applicable for real-time classification. The payload is either a TCP payload or a UDP payload and might be encrypted either using SSL/TLS or on the application layer. We know from investigating the developer tools in the Google Chrome browser that Netflix sends a somewhat similar sequence in the beginning of each video segment being transferred. An example of this behavior is shown in figure 5.1, it should however be noted that this byte string has been decrypted by the receiver before being displayed in Google Chrome.

```
···{moof····mfhd·······G···ctraf
···{moof····mfhd·······I···ctraf
···{moof····mfhd·······Q···ctraf
···{moof····mfhd·······S···ctraf
```

Figure 5.1: Example of beginning sequences taken from a Netflix video stream.

4.1.1 Data extraction

To pursue this solution further, we extracted the payload from two h5 files where the traffic was captured on the same PC. One file containing Netflix sessions and another containing HTTPS browsing. As the classification should be based solely on the individual payload without any temporal information or information about previous packets, all maximum-length payloads were extracted, thereby all having the same length of 1460 bytes. By extracting maximum-length payloads we ended up with 45000 and 47000 payloads from Netflix and HTTPS respectively. The individual bytes values was then converted to integers in the range [0::255]. This way we get datapoints situated in a 1460-dimensional space.

4.1.2 Data exploration

In order to explore the data in greater detail, we perform dimensionality reduction through PCA.

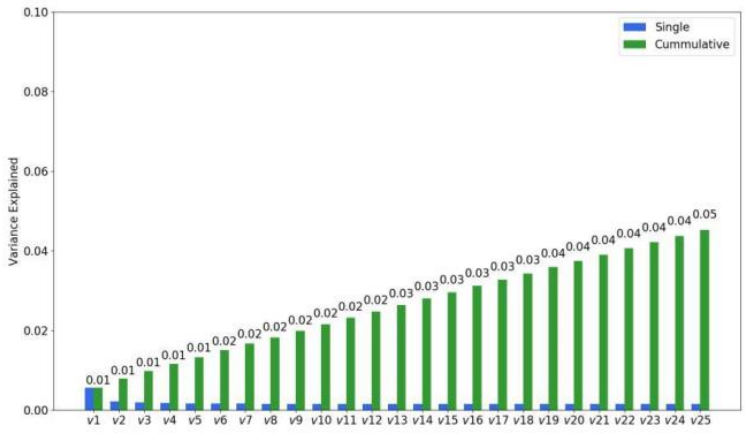


Figure 5.2: Cumulative plot of 25 first principal components when performing PCA on payloads taken from HTTPS and Netflix.

Figure 5.2 shows that the share of variance explained by the 25 first principal components is just about 5% and that each principal component only explains a minor fraction of the variance within the constructed dataset. This indicates that the datapoints are spread out in all dimensions. In figure 5.3 we see the projection of the data onto the first two principal components from which we see that the majority of the two classes in question are situated in the same cloud of datapoints.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

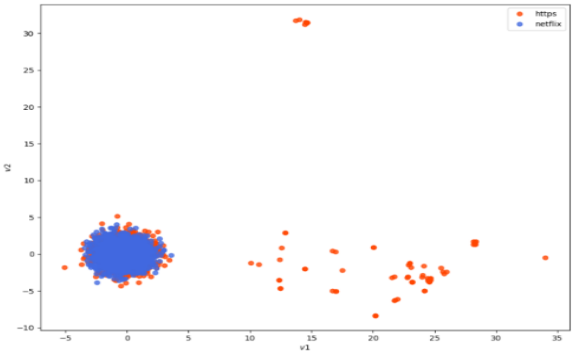


Figure 5.3: Plot of payloads taken from HTTPS and Netflix projected onto first two principal components. We see that the majority of https datapoints coincide with the netflix datapoints.

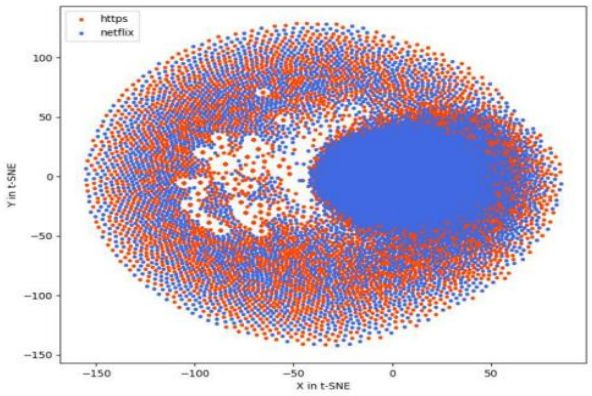


Figure 5.4: t-SNE plot with perplexity 10 over 5000 iterations of payloads taken from HTTPS and Netflix.

As can be seen in figure 5.4 there does not seem to be any clusters even with perplexity set to 10 which should capture even small clusters. We ran t-SNE with different perplexities in the range [2::100] and still no clusters seemed to be present. Therefore, we performed dimensionality reduction by PCA down to 50, 100 and 1000 dimensions and then ran t-SNE on the dataset of reduced dimensionality. All three runs showed similar results. The plot made from running t-SNE on the 50-dimensional dataset can be seen in figure 5.5, from which it is clear that the datapoints are distributed somewhat evenly equivalent with how the t-SNE technique would model random distributions according to Wattenberg et al. [51].

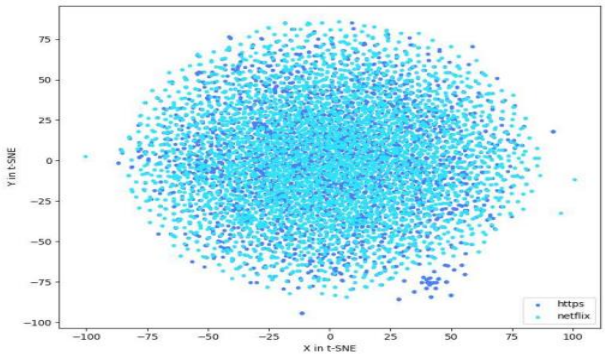


Figure 5.5: t-SNE plot with perplexity 10 over 1000 iterations of payloads taken from HTTPS and Netflix after PCA have been used to re-duce number of dimensions to 50.

As the visualizations from PCA and t-SNE did not reveal any discriminative structure within the data, we looked into how the individual attribute values are distributed in the two classes. In figure 5.6 we see that in the Netflix payloads, the occurrence of the values [0::255] seem close to uniformly distributed, meaning that no value occurs more frequently than

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

others. The heatmaps are shown in figure 5.7a and 5.7b for which the index within the payload is mapped to the x-axis and the numerical value of the corresponding byte is mapped to the y-axis. Looking at figure 5.7a we see that the values of Netflix payloads are distributed evenly across the payload, as anticipated from the results in figure 5.6. As for https shown in figure 5.7b, we do see that some values e.g. 48 occurs more often than other values, however it appears that the value occur more often in all positions of the payload and is not bound to a specific location, this is indicated by the horizontal blue lines.

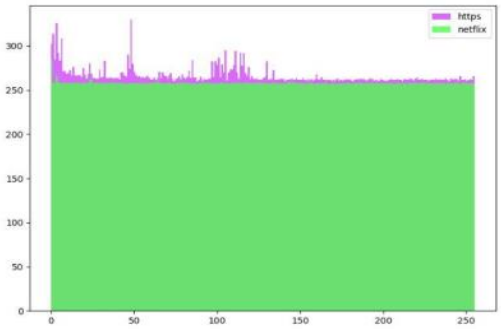
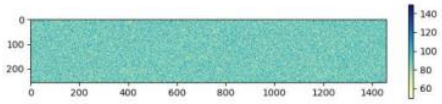
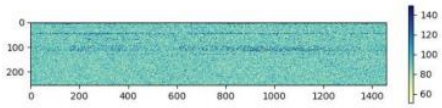


Figure 5.6: Histogram showing the amount of times a value in the range [0::255] occurs in a given class (y-axis scaled down by 1000).



(a) Netflix heatmap.



(b) HTTPS heatmap.

Figure 5.7: Heatmaps showing how byte values in the range [0::255] (y-axis) occur at index [0::1459] (x-axis) of the payloads for both the netflix class (a) and https class (b).

4.1.3 Exploratory Results

Even though the initial explorations of the dataset for the payload-based approach showed poor indications as a foundation for a classification task, we created small experiments using TensorFlow to train a neural network to classify individual packets from the two classes HTTPS and Netflix. In the first experiment we created a simple feed forward neural network with one hidden layer consisting of 100 units with a ReLU activation function and an output layer using the softmax function. Even though this is a relatively simple model it still has more than 146000 trainable parameters:

1460 100 = 146000 (Weights)

146000 + 100 = 146100 (Add bias)

146100 + 100 2 = 146300 (Add output weights)

146300 + 2 = 146302 (Add output bias)

The loss function used was cross-entropy and the Adam optimizer was used with a learning rate of 0.001. A batch size of 100 and 100 epochs was used. Of the 92000 datapoints we started with a training set of size 9200 in order to see if the network would fit the data at all. In that case we saw that the network was able to overfit the training data with a training accuracy of 1.0 after 50 epochs using early stopping, but the validation accuracy was just around 0.5 and the test accuracy was 0.506. The confusion matrix is shown in figure 5.8 where we see that the network does indeed not generalize to unseen data as it appears to be randomly guessing.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

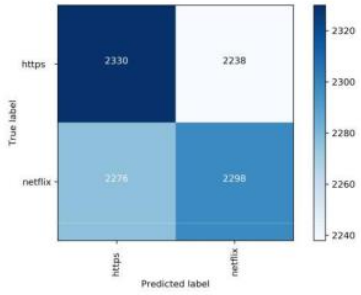


Figure 5.8: Confusion matrix after 50 epochs with accuracy of 0.506.

We then increased the amount of data to avoid overfitting. We found that when the amount of training data was larger than 40000 samples there were no signs of overfitting, but also no signs of any real learning, the validation and test accuracy would not go above 0.51 in any of our exploratory tests. We tried with more complex models having additional hidden layers each with up to 1000 hidden units, these did however not demonstrate any indications of actually learning the task.

4.2 Header experiments

In the previous section we described our findings when looking at a pure pay-load based classification approach. As described, we did not find any promising structure or other features in an encrypted payload. In order to progress our classification task, we investigated the unencrypted parts of the network packets: The individual headers of OSI layers 2,3 and 4, that is the headers of the data link layer, the network layer and the transport layer. We suspect that there might be a "signature" in the header that can be used to distinguish between the individual classes. Wang et al. [49] has shown that it is possible to perform classification of end-to-end encrypted network traffic by using the first 784 bytes of a session. Wang et al. include the payload in which we found no information useful for a classification task, therefore we choose to leave it out of our header-based experiments.

4.2.1 Data extraction

In order to create the dataset for the experiments we extracted the headers from the hdf files that we saved during the data generation process. From each session we extracted the headers of the data link layer, network layer and transport layer of the first 16 packets. This was accomplished by extracting the first 54 or 42 bytes for TCP and UDP packets respectively. Since there is a misalignment with the header length, the UDP header was zero padded up to 54 bytes before concatenating them. This results in input vectors of length 54 16 = 864 for both TCP and UDP as shown in figure 5.9.

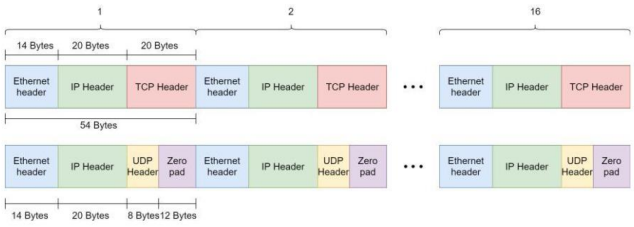


Figure 5.9: Datapoint creation by concatenation of headers from first 16 packets.

4.2.2 Data exploration

The dataset we created initially was generated on a PC provided by Napatech. The PC was running Ubuntu 17.10 and the browser used when streaming was Google Chrome version 64. Therefore we name the dataset Linux Chrome (LC). It consists of 19901 individual datapoints which distributes among the different classes in the following way:

http: 2313 datapoints

https: 2194 datapoints

netflix: 3134 datapoints

twitch: 2970 datapoints

As we see the dataset is not perfectly balanced, but if we want to, we can balance it by randomly sampling a given amount from each class, either oversampling the underrepresented classes or by removing datapoints from classes that contain the most samples. In general, we do not want to remove data, given the quality is good, as the general consensus within machine learning tasks is the more the better. When we look at the amount of datapoints, assuming a minute of streaming pr. datapoint for all classes except http and https, we get a total of 15394 minutes of stream or more than 10 days of streaming "watched".

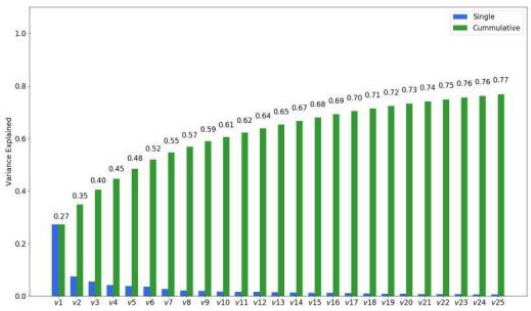


Figure 5.10: Plot of variance explained by different principal components from LinuxChrome dataset of the first 16 headers.

Performing PCA on the LC dataset showed a much more promising picture than what we saw in the payload analysis. Figure 5.10 shows that the first 25 principal components account for more than 77% of the variance in the data. Looking at figure 5.11 in which the 864-dimensional dataset has been projected onto the first two principal components, we see that the classes are more separated than what we saw with the payload-based approach. Furthermore, the non-streaming classes http and https in particular are separated from the majority of the other classes. In order to investigate further, t-SNE was performed on the LC dataset. Figure 5.12 shows that the t-SNE algorithm is able to find a low dimensional mapping of the high dimensional dataset that looks very promising for our classification task. The classes are grouped in clusters and even though the individual classes are divided into several clusters, they seem to be somewhat grouped in the same area of the plot.

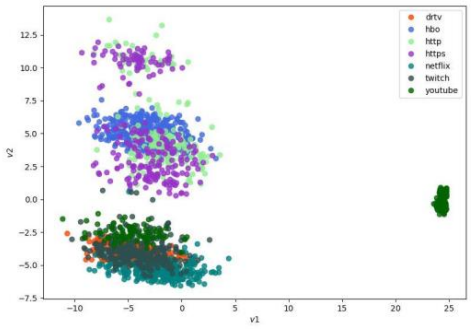


Figure 5.11: LinuxChrome dataset of the first 16 headers projected on v1 and v2.

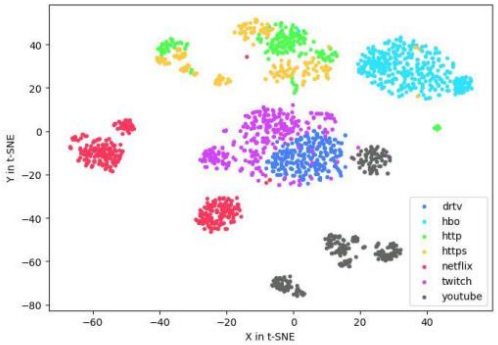


Figure 5.12: t-SNE plot with perplexity 30 over 1000 iterations of the first 16 headers from the LinuxChrome dataset.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

Based on this analysis we believe that the approach taken with the LC dataset provides a good basis for a classification task. In fact, by looking at the t-SNE plot it seems possible to perform classification of individual classes, and not only classification in the streaming/non-streaming context.

4.2.3 Initial architecture

In order to test if the t-SNE plot was indicative of whether the approach taken with the LC dataset was suited for our learning task, the dataset was split into train, validation and test by using an 80/10/10 split. A simple neural network, consisting of a single hidden layer with 50 units having ReLU activation functions, was implemented. The output layer comprised a 7 (one for each of the defined classes) neuron layer using the softmax activation function. As a loss function, cross-entropy was chosen. In order to minimize cross-entropy a gradient based approach was used. As the consensus is that gradient decent methods with momentum seem to converge faster, Adam [24] with a learning rate of 0.001 was used. By implementing early stopping with a patience of 20 epochs and a min-delta of 0.05, training was stopped if the cross-entropy calculated on the validation set did not decrease by 0.05 in 20 epochs. The batch size was set to 100.

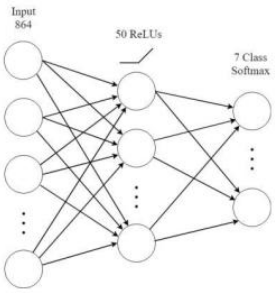
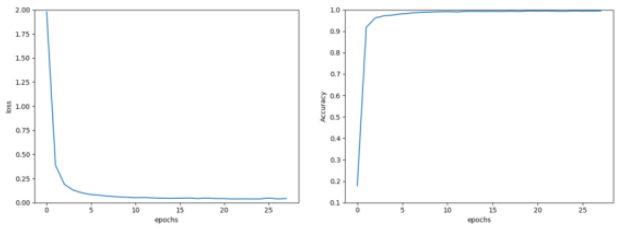


Figure 5.13: Brief overview of the architecture of the neural network with a single hidden layer.

4.2.4 Initial results

When training the aforementioned model on the LC dataset, we see from figure 5.14a that the validation loss decreases very quickly and figure 5.14b shows that the validation accuracy increases accordingly.



(a) Epochs vs. loss. (b) Epochs vs. accuracy.

Figure 5.14: (a) Plot showing how the validation loss decreases when training on the LinuxChrome dataset (b) Plot Showing how the validation accuracy increases when training on the LinuxChrome dataset.

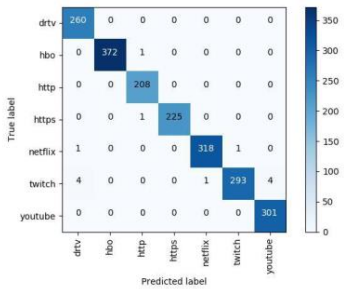


Figure 5.15: Confusion matrix when training and testing with a 80/10/10 split on data from LinuxChrome dataset of the first 16 headers. Accuracy of 0.993.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 11, Issue 1, pp: (52-67), Month: January - March 2023, Available at: www.researchpublish.com

When testing the trained model on the test-split of the LC dataset we get the confusion matrix shown in figure 5.15. Looking at the confusion matrix in figure 5.15 we see a very impressive result, especially in the streaming/non-streaming context. Only a single streaming (hbo) session has been labelled as non-streaming (http) and we see that not a single time has a non-streaming session (http/https) been labelled as a streaming session. This result is made clear by the streaming/non-streaming confusion matrix shown in figure 5.16.

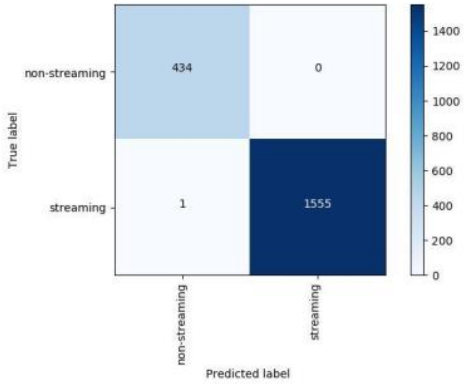


Figure 5.16: Confusion matrix of the streaming/non-streaming context cre-ated from results shown in figure 5.15. Accuracy of 0.999.

From the streaming/non-streaming confusion matrix in figure 5.16 the following performance metrics can be obtained:

$$\text{Non-streaming precision} = \frac{434}{434+1} = 0{:}998$$

$$\text{Non-streaming recall} = \frac{434}{434+0} = 1$$

$$\text{Streaming precision} = \frac{1555}{1555+0} = 1$$

$$\text{Streaming recall} = \frac{1555}{1555+1} = 0{:}999$$

The fact that not a single non-streaming session has been misclassified seems very promising. From Napatech's point of view it is better to save a streaming session that should not have been saved than to discard a non-streaming session that should have been saved. Thereby the recall of the non-streaming class is most important. A perfect recall of the non-streaming class would also yield a perfect precision of the streaming class.

5. DISCUSSION

In the previous chapter we presented the experiments and the results of classifying network traffic, with special emphasis on distinguishing between streaming and non-streaming.

In this chapter we will discuss and reflect on the results and findings. Furthermore, we will discuss some of the choices involved in the process of starting a machine learning project from having no data to a proof-of-concept model.

5.1 Dataset discussion

A machine learning project is only as good as the data that is available. A major challenge of this project has been that no dataset was readily available which satisfied our needs. As the overall task was to investigate whether or not it was possible to classify network traffic such that streaming could be distinguished from other types of traffic, we needed to create our own dataset that could serve this purpose.

5.1.1 Choice of classes

The first task was to choose the classes that should be included in the dataset. We chose to include five different streaming services that are all commonly known in Denmark. To obtain diversity within the data we chose streaming services of different kinds, i.e. Netflix, HBO and DR TV contains longer shows, where Twitch and YouTube can have both short, long

and live shows. We chose the largest streaming services that we know of, as we assumed that a lot of the traffic Napatech would like to identify would come from one of those services. To include network traffic that is not streaming, we chose to create two classes called http and https that contain network traffic transmitted by use of the HTTP and HTTPS protocols. We chose to include both encrypted and unencrypted web browsing as classes. Encrypted because it was important to investigate whether or not it was possible to distinguish between encrypted streaming and encrypted non-streaming. Unencrypted because it accounts for a large fraction of the internet traffic. In retrospect having the classes hbo and netflix makes the process of capturing data much slower as we wanted to obtain balanced classes. HBO and Netflix requires a login, and they are thereby able to impose a limit of one stream at a time to a user. YouTube, Twitch, DR TV does not impose such limits and thus can stream in ten or more tabs at the time. The consequence of this is that it takes 10 times longer to collect the same amount of Netflix and HBO sessions as it does for one of the other classes. For all classes different nuances exist, e.g all streaming services offer streaming in different resolutions, effectively changing how the datapoints might look. We know from Dubin et al. [7] that it is possible to classify the resolution of a streaming service, which leads us to believe that the datapoint associated with different resolutions will be situated in different places on the manifold of the true distribution. For the classes http and https different types of browsing exist, from light browsing to heavy browsing on webpages that include a lot of additional resources e.g JavaScript files, stylesheets and pictures etc. In our case only light browsing is included in the dataset by use of the PyTgen tool. We expect that a file download would be represented differently than light browsing.

5.1.2 Type of data

The dataset is only created by looking at regular Ethernet traffic. Cellular network traffic works in a very different way than Ethernet in which more frag-mentation is seen because the different technologies might have different MTU sizes. In order to make a truly robust dataset, different types of network traffic should be included, but since data capturing is very time consuming this should only be considered if deemed necessary.

5.2 Machine learning discussion

Within this project our focus has been on exploring how classification of network traffic could be solved as a supervised learning task using neural networks. The following will go through some of the important aspects for the solution and discuss some potential alternatives.

6.2.4 Hyperparameter search

We have not used an extensive grid or random search over hyperparameters. This is due to the fact that this project serves as a proof-of-concept, and the performance of the model has been quite good from the start. As mentioned, we have looked into the number of units in the hidden layer, but not only in order to increase performance, but rather to investigate how small we could make the network while still maintaining acceptable performance.

6. CONCLUSION

Throughout this project we have investigated a solution to network traffic classification using neural networks. The problem posed by Napatech is of high relevance as the increase in overall network traffic [25], calls for optimal solutions in order to obtain good network management. The classical approaches to network traffic classification are being challenged by the major focus on privacy and security in network communication, e.g. this is somewhat achievable by use of protocols that utilize encryption, which hinder an approach like DPI. Aiming to solve the problem as a supervised learning task made the data generation process very demanding and it has therefore been a large part of the work. The workflow for capturing and labelling the sessions has been optimized such that additional datasets have been and easily can be generated. Expanding the solution to a larger scale in which more services would be considered by the classifier might, as previously discussed, benefit from turning towards a semi-supervised learning task, as this could ease the time-consuming process of creating a dataset. Having captured and created a structured dataset, we have examined two different approaches of learning a solution to the problem by use of neural networks. The payload-based approach and the header-based approach. As shown in section 5.1, a payload-based approach is highly doubtable due to encryption. We demonstrate by use of both t-SNE, PCA and other plots of the dataset, that we cannot distinguish between streaming and non-streaming. A conclusion which is also reflected in the classification experiments using a neural network.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster,

[2] J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non linear classifier decisions by layer-wise relevance propagation. PLOS ONE, 10(7):1–46, 07 2015. doi: 10.1371/ journal.pone.0130140. URL https://doi.org/10.1371/ journal.pone.0130140.

[4] R. Bar Yanai, M. Langberg, D. Peleg, and L. Roditty. Realtime Classification for Encrypted Traffic, pages 373–385. Springer Berlin Heidelberg,Berlin, Heidelberg, 2010. ISBN 978-3-642-13193-6. doi: 10.1007/978-3-642-13193-6_32. URL https://doi.org/10.1007/978-3-642-13193-6_32.

[5] Cisco VNI. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. https://www.cisco.com/ c/en/us/solutions/collateral/ serviceprovider/visual-networking-index-vni/complete-white-paper-c11- 481360.html, 2017. [Online; accessed 13-Jun-2018].

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4):303–314, 1989.

[7] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. 2016.

[8] R. Dubin, A. Dvir, O. Pele, O. Hadar, I. Richman, and O. Trabelsi. Real time video quality representation classification of encrypted http adaptive video streaming-the case of safari. arXiv preprint arXiv:1602.00489, 2016.

[9] Ericsson. Internet of Things forecast. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast, 2018. [Online; accessed 04-Jun-2018].

[10] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. Performance Evaluation, 64(9-12):1194–1213, 2007.

[11] ESA. How many stars are there in the universe. https://www.esa.int/Our_Activities/Space_Science/Herschel/How_ many_stars_are_there_in_the_Universe, 2018. [Online; accessed 01- Jun-2018].