



**CLASSIFICATION OF ENCRYPTED WEB TRAFFIC USING MACHINE
LEARNING ALGORITHMS**

THESIS

William Charles Barto

AFIT-ENG-13-J-11

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

**DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED**

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-13-J-11

CLASSIFICATION OF ENCRYPTED WEB TRAFFIC USING MACHINE LEARNING
ALGORITHMS

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Engineering

William Charles Barto, B.S.C.E.

June 2013

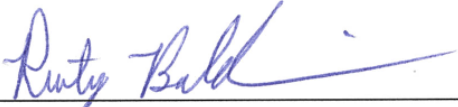
**DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED**

AFIT-ENG-13-J-11

CLASSIFICATION OF ENCRYPTED WEB TRAFFIC USING MACHINE LEARNING
ALGORITHMS

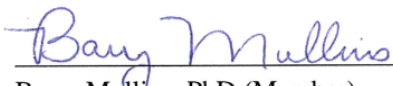
William Charles Barto, B.S.C.E.

Approved:



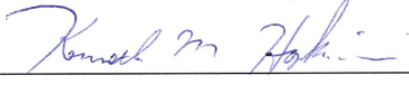
Rusty Baldwin, PhD (Chairman)

20 May 13
Date



Barry Mullins, PhD (Member)

20 May 13
Date



Kenneth Hopkinson, PhD (Member)

20 May 13
Date

Abstract

The increase in secure web services and encrypted network communication makes the network analysis of encrypted web traffic of utmost importance. This research evaluates the use of Machine Learning (ML) algorithms to classify web services within encrypted Transport Layer Security (TLS) flows. The ML algorithms are compared primarily based on classification accuracy. The execution time of the classifiers, however, are also considered as classifiers must be able determine labels quickly to be used in near real-time network protection devices.

The first 12 packets of a flow are analyzed using the following five ML algorithms: Naïve Bayes, NBTree, LibSVM, J4.8, and AdaBoost+J4.8. Every experiment is run using 10-fold cross validation and 16 distinct web services as labels. With the exception of Naïve Bayes the algorithms perform with an accuracy greater than 96%. AdaBoost+J4.8 and J4.8 produce the best accuracies and runtimes. While NBTree and LibSVM both perform marginally worse than AdaBoost+J4.8 and J4.8 in accuracy, their runtimes are each at least an order of magnitude greater.

Additional experiments show the accuracy and runtimes of J4.8 and AdaBoost+J4.8 initially increase as the flow lengths analyzed increase. J4.8 reaches a peak accuracy of 97.99% at 14 packets. AdaBoost+J4.8 peaks later at 18 packets with 98.41% accuracy. AdaBoost+J4.8 requires 21.55 microseconds to classify a single flow at peak accuracy, while J4.8 requires only 2.37 microseconds to classify at peak accuracy. The quick runtimes and high accuracies of the J4.8 and AdaBoost+J4.8 indicate that these ML algorithms are good choices for near real-time classification of web services within an encrypted TLS flow.

Table of Contents

	Page
Abstract	iv
Table of Contents	v
List of Figures	vii
List of Tables	viii
List of Acronyms	ix
I. Introduction	1
1.1 Problem Definition	1
1.2 Goals	1
1.3 Overview	2
II. Background	4
2.1 Introduction	4
2.2 TCP/IP	4
2.3 Network Analysis	6
2.4 Network Encryption	9
2.5 Machine Learning	11
2.5.1 Statistical Classifiers	12
2.5.2 Decision Trees	12
2.5.3 Metalearning Algorithms	15
2.6 Encryption and Network Analysis	16
2.6.1 Deep Packet Inspection	16
2.6.1.1 Current DPI Solutions	17
2.6.2 Shallow Packet Inspection	19
2.7 Summary	24
III. Methodology	25
3.1 Problem Definition	25
3.1.1 Goals and Hypothesis	25
3.1.2 Approach	25
3.2 System Boundaries	26

	Page
3.3 Workload	28
3.3.1 Workload Generation	31
3.4 System Services	32
3.5 Performance Metrics	33
3.6 System Parameters	33
3.6.1 J4.8 Parameters	34
3.6.2 AdaBoost Parameters	35
3.6.3 Computer System Parameters	36
3.7 Factors	36
3.8 Evaluation Technique	37
3.9 Experimental Design	38
3.10 Methodology Summary	38
IV. Results and Analysis	40
4.1 Algorithm Comparison	40
4.2 Flow Length Comparison	44
4.2.1 J4.8	44
4.3 AdaBoost+J4.8	49
4.4 Summary	58
V. Conclusions	59
5.1 Conclusions	59
5.2 Contributions	60
5.3 Future Work	61
5.4 Summary	62
Appendix: Additional Experiments	63
Bibliography	72

List of Figures

Figure	Page
2.1 TCP/IP Packet Structure	5
2.2 Sample Decision Tree	13
3.1 Flow Identification System	28
4.1 Violin Plot Showing Effects of 10-Fold Cross Validation on Distribution	42
4.2 Violin Plot Showing Effects of 10-Fold Cross Validation on Distribution	43
4.3 Violin Plot Showing (Lack of) Normality for J4.8 Runtimes	48
4.4 Plot of Accuracy by First N Packets for J4.8 and AdaBoost+J4.8	51
4.5 Plot of Runtime by First N Packets for J4.8 and AdaBoost+J4.8	52
4.6 Zoomed in Plot of Accuracy by First N Packets for J4.8 and AdaBoost+J4.8 . .	53
4.7 Ratio of Time Training and Runtime of AdaBoost+J4.8 to J4.8	55

List of Tables

Table	Page
3.1 Features	30
3.2 Labels	32
3.3 Factor Levels	37
4.1 Algorithm Comparison	41
4.2 Statistical Difference in accuracy for All 5 Algorithms	41
4.3 Statistical Difference in Runtime for All 5 Algorithms	43
4.4 Means for Flow Length Comparision J48	46
4.5 Statistical Difference of Accuracies in J48 by Flow Length Analyzed	47
4.6 Statistical Difference of Runtimes in J48 by Flow Length Analyzed	47
4.7 Means for Flow Length Comparison AdaBoost+J48	50
4.8 Statistical Difference of Accuracies in AdaBoost+J48 by Flow Length Analyzed	54
4.9 Statistical Difference of Runtimes in AdaBoost+J48 by Flow Length Analyzed	54
4.10 Statistical Difference of Accuracies and Runtimes	57
A.1 J4.8 Performance with Flow Length as Packets	63
A.2 J4.8 Performance with Flow Length as Mircoseconds	64
A.3 Features with Timing	65
A.4 J4.8 Performance with Flow Length as Packets and Timing Features	66
A.5 J4.8 Performance with Flow Length as Microseconds and Timing Features . . .	67
A.6 Means for Flow Length Comparison J48 No Google Docs	69
A.7 Means for Flow Length Comparison AdaBoost+J48 No Google Docs	70

List of Acronyms

Acronym	Definition
ARFF	Attribute-Relation File Format 26
CF	Confidence Factor 15
DoS	Denial of Service 8
DPI	Deep Packet Inspection 6
FFT	Fast Fourier Transform 29
FIS	Flow Identification System 26
FN	False Negative 33
FP	False Positive 33
FTP	File Transfer Protocol 21
GMU	George Mason University 23
GP	Genetic Programming 22
HMM	Hidden Markov Model 23
HTTP	HyperText Transfer Protocol 7
IANA	Internet Assigned Numbers Authority 21
IDS	Intrusion Detection System 6
IP	Internet Protocol 4
ISP	Internet Service Provider 1
MITM	Man-in-the-Middle 10
ML	Machine Learning 1
NBTree	Naïve Bayes Tree 13
P2P	Peer-to-Peer 6
PKI	Public Key Infrastructure 17
PPTP	Point-To-Point Tunneling Protocol 21

Acronym	Definition
QoS	Quality of Service 1
RAM	Random Access Memory 36
RFC	Request for Comments 10
RIPPER	Repeated Incremental Pruning to Produce Error Reduction 22
SaaS	Software as a Service 10
SACK	Selective Acknowledgment 30
SPI	Shallow Packet Inspection 6
SSH	Secure Shell 22
SSL	Secure Socket Layer 9
SUT	System Under Test 26
SVM	Support Vector Machine 12
TCP	Transmission Control Protocol 4
TLS	Transport Layer Security 1
TN	True Negative 33
TPR	True Positive Rate 33
TP	True Positive 32
UDP	User Datagram Protocol 20
VOIP	Voice Over IP 23
VPN	Virtual Private Network 21

CLASSIFICATION OF ENCRYPTED WEB TRAFFIC USING MACHINE LEARNING ALGORITHMS

I. Introduction

1.1 Problem Definition

Network analysis is a useful and sometimes critical tool in network administration. It is used by Internet Service Providers (ISPs) to monitor the traffic on their networks and provide a consistent Quality of Service (QoS) to their customers [9]. Network administrators can employ network analysis tools that detect malware intrusion and data exfiltration, enabling automatic blocking and reporting of such communication [8]. However, encryption interferes with analysis of this communication [8]. Many traditional network analysis devices cannot effectively analyze encrypted network traffic.

More and more applications are moving to the cloud and using web-based services [10, 18]. These web based services often use encrypting network protocols such as Transport Layer Security (TLS) [16, 44, 50]. As the amount of TLS traffic increases, the ability to examine TLS flows for security risks becomes more important.

1.2 Goals

This research provides a method of quickly classifying a web service within an encrypted network traffic flow. Previous research has analyzed the ability of Machine Learning (ML) algorithms to classify applications and protocols associated with network traffic [33]. Further research shows the feasibility of using ML algorithms to determine whether a single application is associated with an encrypted traffic flow from a mix of other traffic [5]. ML algorithms are also useful in near real-time classification of network

traffic [11]. This research builds upon previous research by measuring the efficacy of ML algorithms in classifying 16 types of web services within TLS flows. It is hypothesized that the tested ML algorithms will be able to successfully classify web services associated with encrypted network flows. It is further hypothesized that the accuracy and runtime of the ML algorithms will increase as the amount of the network traffic flow analyzed increases.

The high-level motivating goal of this research is to improve the ability of network analysis devices to reduce malware intrusion and data exfiltration via encrypted network communications. While the technique analyzed in this research does not attempt to identify particular instances of intrusion or exfiltration, it does enable the enforcement of broader policies to mitigate these risks. By classifying encrypted web services, network administrators can detect and prevent the use of unapproved web services. This motivating goal drives two low-level quantifiable research goals.

The first goal is to assess which ML algorithms are likely to perform well in classifying encrypted web traffic. Initial experiments are run to test each of five ML algorithms over an identical set of traffic. The algorithms that perform the best with respect to accuracy and runtime are used for the remainder of the experiments.

The second goal of this research is to determine how many packets are necessary to reach an acceptable classification accuracy. The runtime and accuracy are hypothesized to increase as the number of packets analyzed increases. While a solution that optimizes both metrics probably does not exist, this research quantifies each metric individually for increasing flow lengths analyzed, starting from the beginning of the flow. The beginning of the flow is used due to success in previous research.

1.3 Overview

This chapter defines the goals of this research and the problem it sets out to solve. Chapter 2 describes the basics of network analysis, network encryption, ML, and how they interact before reviewing the current literature in this field of study. Chapter 3 discusses

the experimental setup, including the generation of the test data, and how the performance of the ML algorithms is measured. Chapter 4 reports the findings of the experiments. Chapter 5 discusses the conclusions drawn from this research and the suggestions for future work.

II. Background

2.1 Introduction

This chapter presents the theory and background information necessary to understand the following chapters. Section 2.2 briefly reviews network protocols. Section 2.3 discusses the basics of network analysis. Section 2.4 discusses the current state of network encryption. Section 2.5 defines ML and the ML algorithms used later in this research. Section 2.6 outlines the current solutions for analyzing encrypted network traffic and the past research done in the field.

2.2 TCP/IP

Transmission Control Protocol (TCP) and Internet Protocol (IP) are the principal protocols of the Internet [28]. IP is a networking protocol used for communication between nodes in the Internet [28]. IP sends data messages in small bursts known as packets. These packets consist of two parts, the header and the payload [39]. The header contains enough information for the packet to reach its destination [39]. TCP is used to provide reliable communication and is usually encapsulated as the IP payload [28]. TCP also uses a header and payload, with header flags for connection management, a checksum for validation, and fields for verifying all packets were received [40]. If an Ethernet connection is used between two nodes in the Internet, the IP packets are encapsulated within an Ethernet frame [27]. Figure 2.1 shows the fields within Ethernet, TCP, and IP headers.

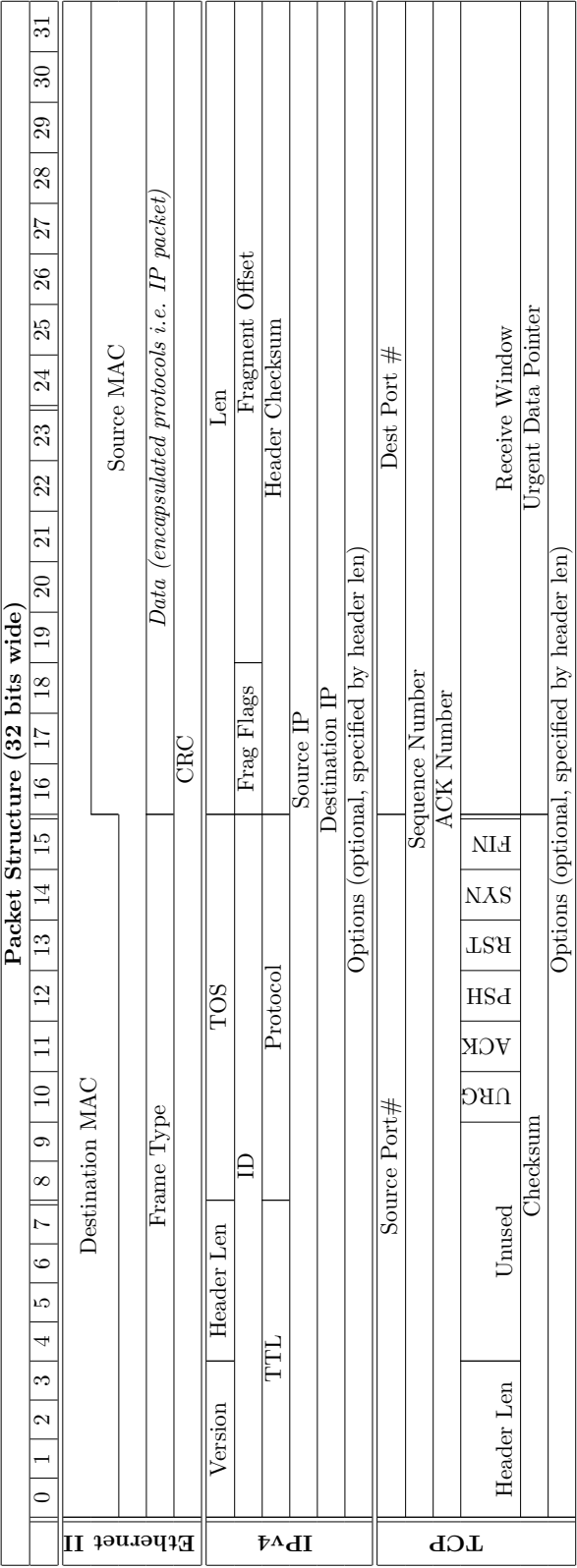


Figure 2.1: TCP/IP Packet Structure modified from [34]

2.3 Network Analysis

Network analysis plays an important role in network security. With the threat of malware increasing, many network administrators have deployed tools to perform network analysis, with the intent of preventing intrusion. These tools, known as Intrusion Detection Systems (IDSs), inspect all incoming traffic to determine if any is malicious [9]. Organizations whose network contains sensitive information have also deployed systems that monitor traffic leaving their network to detect and prevent sensitive data from being stolen over their network [8]. Rather than concerning themselves with malicious or sensitive data, ISPs use network analysis to maintain QoS for their customers [35]. They maintain QoS by prioritizing network traffic based on the type of traffic, or the application creating and using the traffic. Some network applications require a constant stream of packets to function properly, while others, such as Peer-to-Peer (P2P) applications, download large amounts of data, but do not have a strict timing requirement [35]. An estimated 50%-70% of Internet traffic is P2P [25]. By prioritizing the time sensitive traffic and throttling bandwidth intensive, time-insensitive traffic ISPs can better serve their customers. Network Analysis applications use either Shallow Packet Inspection (SPI) or Deep Packet Inspection (DPI) (or some combination of the two). These two distinct forms of network analysis use different analytical tools to determine the type of information traversing the network.

The network analysis tools designed to control malware intrusion and data exfiltration are primarily concerned with analyzing the network communication between a set of controlled computing systems on a protected network and uncontrolled computing systems external to the protected network [11, 30]. ISPs face a different problem in that they do not control the end-point machines internal or external to their network. Instead, ISPs seek to monitor communication internal to their network, in addition to communication coming and going from their network, for the purpose of providing QoS for their customers [35].

SPI is a type of network analysis that only inspects the headers of packets, including the information in the TCP/IP headers used to route traffic [14]. This involves analyzing information such as: source transport layer address (source port), destination transport layer address (destination port), source IP address, destination IP address, and the transport layer protocol [14]. Depending on the transport layer protocol, other information in the header, such as flags, could also be used. The source and destination IP addresses can be used to validate certain rules. Some government organizations might block or log all traffic that comes or goes to foreign IP addresses [30]. Many protocols have assigned ports, and comparing the port number to a known list may indicate which application generated the traffic [19]. Timing characteristics can also be used to analyze the traffic [32, 33]. For example, HyperText Transfer Protocol (HTTP) traffic: “80% of HTTP requests occur within three seconds of each other. 95% occurs within a minute and a half” [8], HTTP traffic that does not follow the expected pattern can be flagged as abnormal. Typically HTTP traffic occurs in short bursts as pages load, with timing gaps while the endpoint user consumes the information [8]. Continuous streaming of HTTP traffic, depending on the direction of traffic, could indicate uploading or downloading files larger than traditional web pages such as media or large files on disk.

DPI analyzes the entire packet, including the header and the payload [14]. By analyzing the payload of the packet, the inspection tool can analyze all information that is traversing the network, not just the type of information and where it is going. This analysis compares the contents of messages against a list of predetermined signatures [23]. These signatures are typically produced from parts of known documents or files that the administrators of the DPI device want to block or log [19]. For example, DPI devices that are configured to inspect incoming traffic can compare the traffic to a list of signatures generated from a collection of known malware. The DPI device would then be able to detect if any of the malware on the list is in an incoming message. To prevent data from evading

detection by being segmented into multiple messages, DPI devices cache all messages per communication stream and reassemble the full stream [20]. A similar setup is used to prevent and/or detect sensitive data exfiltration. Sensitive files and documents can be used to form signatures and the signatures are stored in a collection. The DPI device just needs to reassemble outgoing messages and compare them to the signatures of sensitive data. If sensitive data is found in an outgoing communication stream, the DPI device can end the communication stream and/or log the breach. Research has been done supporting the use of DPI devices to block certain websites; Yu, Cong, Chen, and Lei [52] suggest hashing the domains of pornographic and illegal websites so ISPs can eliminate access to them.

DPI tools face a number of challenges to successful operation. Since the DPI tool sits between endpoints within a protected network and the outside world, it adds delay to all traffic that passes through it [20]. The DPI device needs to take delay into account and implement a solution that adds a tolerable delay to the traffic delivery. The slower the DPI device, the more traffic it must handle at a given time [20]. Consider the scenario of one message arriving every second: if the device can process each message in less than one second it will continue to operate correctly. If the device takes longer than one second to analyze each flow the messages will arrive faster than it can process, and the tool will quickly have more traffic cached than it can handle. If the amount of traffic passing through the DPI tool exceeds the amount of traffic the tool can handle, it either blocks excess traffic, or allows uninspected traffic through. In the first scenario users inside the protected network or users attempting to communicate with systems inside the protected network will experience a dropped connection, and suffer effects equivalent to a Denial of Service (DoS) attack. If the latter happens, the DPI tool has been effectively neutralized and cannot protect the network, allowing potentially malicious traffic onto the network and sensitive data out of the network. In order for a DPI tool to be effective it must be able to process network communications as fast as it can receive them [14]. To mitigate

these throughput and delay concerns for DPI tools, researchers have investigated hardware-based solutions, such as those utilizing FPGAs [15, 37, 43, 45, 52]. The hardware-based solutions increase the throughput of DPI devices by eliminating the extra overhead of DPI tools on generic hardware. Researchers are also looking at accelerated data structures, such as bloom filters [37, 52]. SPI can also face these performance concerns.

Another method for increasing DPI throughput is to only analyze the beginning of a communication stream [9]. Research done by Braun, Munz, and Carle suggests that this is almost as effective as inspecting the entire communication stream [9]. Inspecting the first ten packets of the stream can identify P2P traffic and malware identification is usually successful using only the first 1000 bytes of a communication stream [9]. They also suggest randomly varying the size of the stream that is inspected to prevent a malicious endpoint from padding the beginning of a communication stream to avoid detection [38]. Restricting the length of the inspected data will only be effective if the DPI device inspects more than the padding done by an attacker. While this technique is useful for increasing DPI device throughput, it can be circumvented by clever attackers who pad their malicious communications with sufficient irrelevant data.

Another concern when using DPI is privacy. Since DPI involves analyzing entire packets, including the contents of packets, users who have DPI performed on their communication have no privacy [38]. In corporate environments where those in charge of network administration also own and control all computing systems on the network, privacy is not as big a concern. ISPs do not control all the computing systems on their networks, as customers can connect their own personal computers. ISPs performing DPI on customers' communication is a complicated legal issue [38].

2.4 Network Encrpytion

TLS is replacing Secure Socket Layer (SSL) as the security protocol used to encrypt the connections between internet browsers and web servers [3, 46]. Use of TLS/SSL is

becoming more prevalent [16, 44, 50]. In fact some application vendors are switching to Software as a Service (SaaS) business models, which would increase the amount of applications relying on the Internet and its security protocols [10, 18]. TLS consists of two separate protocols, the TLS Record Protocol and the TLS Handshake Protocol both of which are defined in Request for Comments (RFC) 5246 [21]. The TLS Record Protocol uses symmetric cryptography and requires a reliable connection, such as that provided by TCP. The symmetric key for the TLS Record Protocol is typically negotiated by the TLS Handshake Protocol [21]. The TLS Handshake Protocol has three basic properties: the endpoints' identity *can* be authenticated via asymmetric cryptography; the negotiation of the symmetric key is secure, unavailable via eavesdropping or Man-in-the-Middle (MITM) attacks; and no third party can modify the negotiation without alerting the original endpoints [21]. TLS is designed to encapsulate other layers of traffic such as HTTP.

The TLS Handshake Protocol includes six steps listed below from RFC 5246 [21]:

- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and exchanged random values.
- Provide security parameters to the record layer.
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

The TLS Handshake Protocol includes the option for adding extensions to the Client Hello messages [21, 24]. These extensions provide information not required by the protocol that will aid with opening the connection. One such extension is server name, which is especially useful when a single physical host is running multiple virtual servers [24]. The server name allows the physical host to respond with the correct certificate and initiate a connection with the desired virtual server.

2.5 Machine Learning

Before discussing modern approaches to analyzing encrypted network traffic, a quick background on ML is presented as it is used in many of the approaches [7, 11, 13, 22, 23, 29, 32, 33, 35, 36, 38, 47, 49, 50]. Witten et al. [48] describe Machine Learning (ML) as changing behavior based on input to improve performance. Russel et al. [42] describe ML as “learning to adapt to new circumstances and to detect and extrapolate patterns.” In the papers noted above, ML was used to do exactly that: detect and extrapolate patterns. This research focuses on using ML algorithms to detect patterns in the packet sizing information to extrapolate and predict the type of traffic contained within a TLS flow. ML algorithms are given inputs called samples with each sample having a set of features. These features are either discrete—boolean is a discrete with two values—or numeric. A learning algorithm attempts to use the samples’ features to infer something about the samples [48].

Witten et al. [48] describe four types of machine learning: association learning, clustering, numeric prediction, and classification learning. In association learning the ML algorithm attempts to find associations between samples. Clustering algorithms attempt to divide the samples into clusters, or categories, based upon their features. Numeric prediction algorithms attempt to predict a numeric value given the samples [48]. Classification algorithms, the kind used in this research, take samples—consisting of features and an associated label—and attempt to predict the label of future unlabeled samples. Classification ML algorithms are a subset of supervised learning algorithms because they are trained with

previously labeled samples as opposed to clustering algorithms which are just given unlabeled samples to discern distinct groups automatically [48]. Classification algorithms have two stages, training and classifying [48]. Training occurs when the algorithm is given the labeled samples, classifying occurs when labels are predicted for the new unlabeled samples. There are many kinds of classification algorithms; this research focuses on the J4.8 decision tree, a variant of C4.5 [48].

2.5.1 Statistical Classifiers.

Naïve Bayes is a classification algorithm that relies on Bayes' rule of conditional probability [48]. The mathematics will not be delved into here as this algorithm is only briefly discussed, but Naïve Bayes is used extensively in this field [13, 22, 32, 36, 47].

Support Vector Machines (SVMs) are another type of ML algorithm. The SVM uses mathematical methods to represent the feature set in a higher dimensional space, where different labels are linearly separated [17]. This is an oversimplified explanation as a full description would be too verbose for this discussion. For a full definition of SVM and other kernel-based ML algorithms refer to [17].

2.5.2 Decision Trees.

Decision trees use a divide-and-conquer method to classify inputs [48]. Decision trees consist of two types of structures, leaves and decision nodes [41]. A leaf indicates which label—or class—is used as the final classification for the decision tree [41]. A decision node has a condition and a sub decision tree for each possible outcome of the condition [41]. Figure 2.2 shows a very simplistic decision tree. The diamonds are decision nodes and the ovals are leaves. When using a decision tree for classifying, the features are compared at each subsequent node and a new branch is taken; the input features are classified when a leaf is reached [41]. Decision trees can be efficiently implemented as nested if-else blocks [11]. They can also be simplified as rules [41]. The rule for finding a duck in the decision tree given in Figure 2.2 would be: if and only if it walks like a duck and quacks like a duck

is it a duck. Different decision tree algorithms use different techniques to create the final decision tree, as finding the ideal decision tree is NP-complete [41]. The creation of the decision tree is the training part of the ML algorithm.

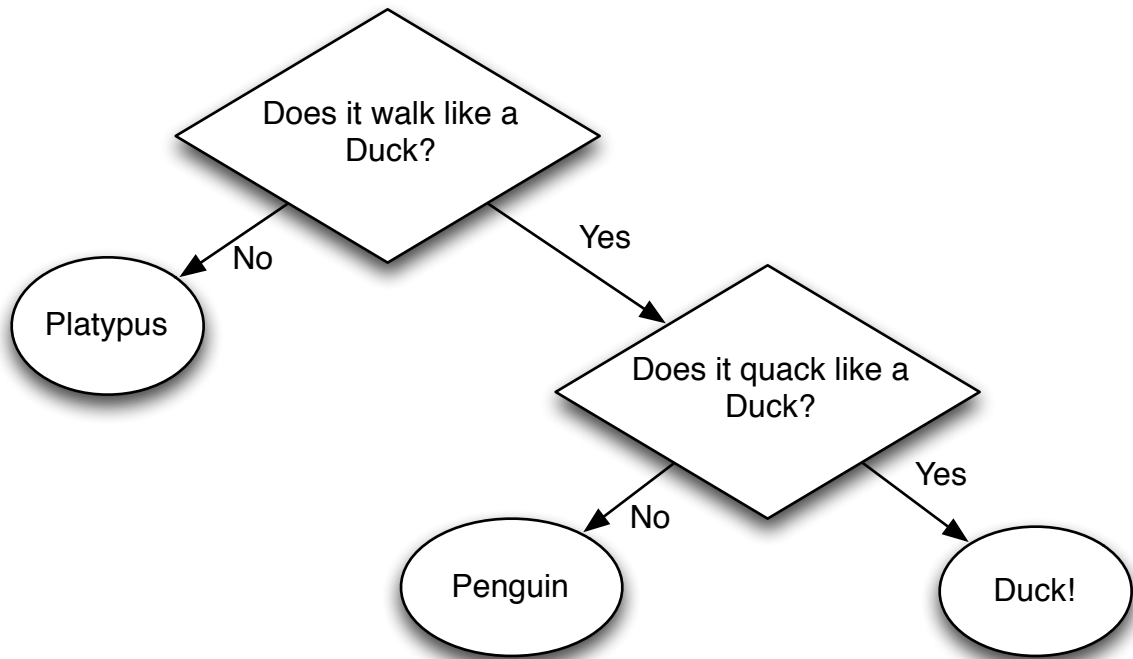


Figure 2.2: Sample Decision Tree

A Naïve Bayes Tree (NBTree) is a combination of Naïve Bayes and decision trees, where the leaves are Naïve Bayes classifiers [48]. Again, the full description of the NBTree is outside the scope of this research, but they have been used with promising results in this field [47].

J4.8 is the decision tree algorithm used in this research. It is an open source Java implementation of C4.5 [48], which is widely used [7, 11, 29, 33, 35, 36, 47]. C4.5 was created by Ross Quinlan [41] and is completely defined, with source code, in his book. Only a quick summary of C4.5 is given here, for the full description refer to Quinlan's

book. C4.5 builds the decision tree by iterating over the training samples and making modifications to the tree. As with most decision trees, deciding what node to insert is determined by maximizing some gain criterion of inserting that node [41]. ID3, the predecessor to C4.5—also written by Quinlan—used information gain [41]. Quinlan’s definition of information is similar to entropy and is measured in bits. In the initial publication of C4.5 Quinlan points out an inherent flaw in using information gain: it favors conditions with more outcomes [41]. C4.5 instead relies on the gain ratio, which is defined as the information gain divided by the potential information of dividing the tree into the resulting number of outcomes. In addition to maximizing the gain ratio when creating decision nodes, C4.5 further imposes the following constraint: “the information gain must be large—at least as great as the average gain over all tests examined” [41].

There are two possible types of features in the standard form of C4.5: discrete and numeric. If a condition of a decision node is based on a discrete feature then the decision node will have a split for each possible value of the feature [41]. If the feature is numeric then the tree will split with a less-than-or-equal and a greater-than condition. When choosing the split value for a numeric condition, C4.5 uses the largest feature value that is less than the median [41]. Since all samples must share the same set of features, a common problem in ML is unknown features [41, 48]. That is to say, if height is a feature of an object, all samples must have a listed value for height, though the values may differ or be listed as unknown. If the current condition is testing an unknown feature, C4.5 takes the previously most common branch. Decision nodes are inserted into the tree to maximize gain ratio and creation of nodes stops when the gain ratio requirements of adding a decision node cannot be met [41].

One common problem with ML algorithms is overfitting [41, 48]. Overfitting occurs when the trained algorithm too closely resembles the training dataset and is not general enough to accurately classify samples not used in training [48]. C4.5 uses pruning to reduce

the problem of overfitting [41]. Although multiple pruning methods are discussed by Quinlan, C4.5 uses reduced-error pruning [41]. The pruning method begins by examining leaves and working its way up the tree. If replacing a parent node with a leaf or child produces an acceptable error rate, then the replacement is made. Quinlan defines the error rate for pruning as the number of items incorrectly classified by that branch divided by the total number of items classified by that branch [41]. C4.5 uses an input parameter of Confidence Factor (CF). A higher CF indicates a higher probability that data used in future classifications is similar to the training data. The more similar future data will be to the training data, the less pruning required [48]. Upper and lower limits on the error rate are calculated based on the CF. Quinlan describes his method of determining when to prune as doing “violence to statistical notions of sampling and confidence limits” [41]. He further states, “Like many heuristics with questionable underpinnings, however, the estimates it produces seem frequently to yield acceptable results” [41]. Witten et al. define this method of replacing parent nodes with child nodes as subtree raising [48].

The final characteristic of C4.5 to discuss is windowing. A window is a subset of the samples used to train. The size of the windows are controlled by input parameters. C4.5 builds a decision tree using the initial window. The next window is built by adding approximately half of the samples incorrectly classified when classifying the samples not in the original window [41]. This new window is used to create a new tree. C4.5 ensures the windows chosen have the same distribution of labels as the entire training dataset [41]. Additional windows are considered until a tree is found that has no incorrect classifications on samples outside the training windows, or the newly created trees are worse than the previous trees [41]. For the complete definition of C4.5 please see Quinlan’s book [41].

2.5.3 *Metalearning Algorithms.*

Metalearning algorithms are used to increase the learning power of other ML algorithms [48]. AdaBoost, an example of a metalearning algorithm, *boosts* other ML

algorithms by iterating over multiple created classifiers of the boosted algorithm in an attempt to lower their error [48]. Many ML algorithms use weights, including C4.5—which uses them for unknown features—when creating classifiers. AdaBoost assigns equal weights to all samples then uses the boosted algorithm to create a classifier and calculate the classifier’s error. On subsequent iterations AdaBoost modifies the weights in favor of the samples which are incorrectly classified and creates new classifiers, again calculating their error. This cycle is repeated until the error of the new classifier is zero or is greater than 0.5 [48]. When used to classify data, Adaboost iterates over all generated models and returns the classification with the highest weight [48]. AdaBoost has been used successfully in this field to improve classification accuracy [29].

2.6 Encryption and Network Analysis

2.6.1 Deep Packet Inspection.

Deep packet inspection relies on the ability to compare the communication on the network to known signatures. When the traffic being analyzed has been altered, whether by obfuscation or encryption, simply scanning the traffic for known signatures will not successfully detect the signatures [8]. When faced with encrypted traffic, a DPI device has four options. First, the device can block all encrypted traffic from entering or leaving the network. This will prevent malicious traffic from entering the network and sensitive data from leaving the network. Unfortunately secure traffic is often required and this option severely limits the capabilities of the network. Second, the DPI device can allow all encrypted traffic through. While this would not interfere with the normal use of the network or applications requiring secure communication, it would hinder the DPI device’s capabilities to prevent malware intrusion or sensitive data exfiltration. Another possibility is to rely on the information in each packet that is not encrypted, essentially performing SPI on encrypted traffic. This has merits in providing both QoS and application classification, and is a good first step, but as discussed in the next section, it cannot—on its own—restore

the ability of the DPI device to detect malware intrusion and sensitive data exfiltration. The fourth and final option is to circumvent the encryption on the secure communication for the purpose of inspection. If the DPI device can inspect the decrypted data, it can allow the communication to pass through the device with the assurance that the device performed as well as it would have on unencrypted communication [23]. Secure traffic that cannot be decrypted can use any of the other three options depending on the needs of the system. General problems with the fourth option are discussed below. SPI on encrypted traffic is the focus of this thesis.

2.6.1.1 Current DPI Solutions.

Commercial tools currently exist that circumvent the encryption of encrypted network traffic to allow the deployment of an inspection device that analyzes the unencrypted contents of each flow [2]. These devices act as a gateway for all encrypted traffic that enters or exits a network [2]. This is done by making the inspection device a certificate authority that is trusted by all machines on the network. This lets the device replace server certificates—and client certificates if they exist—with a new certificate that allows the device to eavesdrop on communication. When the client views the newly generated certificate it can confirm it is communicating with a trusted entity. This setup enables each endpoint to communicate with the inspection device securely, and allows the inspection device to decrypt, inspect, and re-encrypt all messages before sending them to their original destination. It is also worth noting that this design does not explicitly handle the case of symmetric encryption if the key is shared outside of a viewable network communication. That is to say any secure communication that relies on a shared key to which the DPI device does not have access will not be available for inspection. If a Public Key Infrastructure (PKI)-based encryption scheme is used to share a symmetric key, such as the TLS Handshake Protocol, the DPI device could detect the shared key and use that to decrypt any further encrypted traffic in that communication stream.

This design faces a number of problems that need addressing. First, as with most DPI implementations, performance is a concern [19]. Weaknesses in DPI devices caused by an overwhelming amount of traffic have already been discussed. Adding the requirement to issue and cache certificates for all secure communication streams and handle the decryption and encryption of traffic will add more delay to the DPI device. Not only could this delay cause concerns for the end users of the system, but the increase in delay also prolongs the amount of time connections must be monitored by the DPI device.

A DPI device that doubles as a certificate authority also faces the problem of successfully detecting all encrypted traffic. The DPI device will not easily detect any data that is encrypted on an endpoint before any communication is begun, and then uses a normally unencrypted communication protocol to transmit the obfuscated data [19]. The DPI device will also not be able to recognize signatures or regular expressions in obfuscated communication.

Intercepting and decrypting all encrypted traffic also adds a new vulnerability to the system. Since the DPI device is universally trusted on the network, a compromise to the DPI device compromises the entire network. If an attacker can monitor the memory of the DPI device, either through physical access or malicious code, they can see the contents of all network traffic regardless of encryption. If an attacker gains access to the DPI device's private key, they can issue their own certificates that will be trusted by every system on the network. This makes it possible for the attacker to have every communication sent to the systems trusted. This single point of security failure means that a DPI device acting as a Certificate Authority must have protection, both physically and in the cyber realm. Vulnerabilities have been found in currently-available devices that function this way, as reported by the vendor [1].

A third problem is invasion of privacy [38]. While an organization that owned or controlled all systems on its network could enforce a certificate being installed on every

system, an ISP would have a much harder time forcing the same requirement on its customers. While some private companies and the government have cause to monitor all traffic entering and leaving their controlled networks, an ISP might have trouble convincing customers to install the ISP's certificates, allowing the ISP to view all encrypted traffic directed to or from their customers.

2.6.2 Shallow Packet Inspection.

Encryption changes the characteristics of network traffic, which can have negative impacts on SPI. Multiple applications can use the same protocol—or lower level application—for encryption, resulting in the traffic from all the applications using the same port [38]. Many P2P applications use encryption and random ports to evade network analysis tools [26]. The use of proxy servers can also prevent a network analysis tool from correctly determining the source or destination of network traffic. Timing and sizing analysis can be used to gather information about encrypted network flows. If ISPs have the ability to accurately determine what applications are communicating on their network—even if the communication is encrypted—then ISPs can limit applications as necessary to preserve QoS [38]. Research has been done using ML algorithms to identify applications utilizing both unencrypted and encrypted network traffic flows [7, 11, 29, 33, 35, 36, 47]. Some of the research in this field cites the privacy concerns created by DPI as a main reason for using SPI [38]. While these solutions cannot accurately detect malware or sensitive data traversing a network, they can provide other information, such as the application within the network traffic flow. This is ideal for ISPs looking to preserve QoS and gives commercial and government organizations the ability to enforce application restrictions or to further inspect specific application traffic using another method. Some of the techniques and results of SPI are discussed below.

In 2005, Moore and Zuev [33] published a list of 249 formally defined features—called discriminators—designed to be used in statistical analysis of traffic flows. Timing

and sizing information from a bi-directional traffic flow are used to calculate most of the features. This set of features, or a subset, are frequently used [29, 32, 38] and this research uses a subset, which is defined in Section 3.3. They also defined a network traffic flow by the 5-tuple of source IP address, destination IP address, transport layer protocol—often TCP or User Datagram Protocol (UDP)—source port number, and destination port number. In 2005, Moore and Zuev [32] also published a paper using Bayesian analysis techniques with this feature set. They used 377,526 manually classified flows, which had 10 distinct labels [29]. The majority of the traffic flows analyzed were unencrypted. Using port numbers as a feature, they were able to achieve accuracy of 96.29% with a combination of fast correlation-based filter, Naïve Bayes, and kernel density estimation. The paper also reports a drop in accuracy ranging from 1% to 56% depending on classification method when using samples captured months apart for training and classifying. The best classifier—defined by highest accuracy—only dropped from 96.29% to 93.73%. This indicates that any classification process relying on network statistics should be updated regularly especially when the network changes.

In 2006, Williams et al. [47] published a comparison of five ML algorithms—BayesNet, C4.5, two variations of Naïve Bayes, and a Naïve Bayes tree—with four methods of feature reduction when classifying unencrypted traffic flows. The labels for the samples were determined by port number. In addition to classification accuracy, Williams et al. measured the build and classification times of the ML algorithms. C4.5 was the fastest algorithm for every feature set, with 54,700 classifications per second for the full feature set. It was also the most accurate when given the complete feature set. NBTree had comparable accuracy, but was approximately ten times slower classifying and sixty times slower when training, for the full feature set. BayesNet and one of the Naïve Bayes variations had only slightly lower accuracy, but were still at least 20% slower than C4.5 when classifying. And although C4.5 was the second slowest of the ML algorithms to train,

the combined highest accuracy and fastest classification time lead the authors to conclude that C4.5 is the best for real-time classification.

In 2007, Moore and Li revisit the same datasets and labels that Moore and Zuev studied in 2005 [29]. This time Moore and Li compare the previously best Naïve Bayes based classifier against C4.5 and C4.5 with AdaBoost. The authors only use the first five packets of the flow to classify the traffic. Once again the data is hand classified and port numbers are used as features. C4.5 is again the most accurate of the ML algorithms and the fastest in classification with an accuracy of 99.8%. The high accuracy is due, at least in part, to the inclusion of port numbers in the feature set. The bulk of the traffic analyzed is Web-browsing, Mail, and File Transfer Protocol (FTP) all of which have well defined Internet Assigned Numbers Authority (IANA) port numbers [7]. In 2009, Moore and Li collaborated with Canini and Zadnik to produce a network edge device that classifies network traffic flows using techniques defined in their early works [11]. The device takes advantage of the increased speeds of using an FPGA and the simplicity in programming a decision tree—specifically C4.5—to fully classify traffic in a full-duplex 1 Gbps line. Their product, the AtoZ automatic traffic organizer, is well detailed in [11] and is a good example of practical uses of this type of research.

Joint research by Osaka City University and the Oki Electric Company outlines a possible implementation that utilizes ML to accurately identify applications communicating securely on a network [38]. They obtained their data by classifying traffic using DPI before it entered a Virtual Private Network (VPN), then encrypting the traffic with either IPsec or Point-To-Point Tunneling Protocol (PPTP). The feature set for this research is a subset of the features defined in [33]. Their experimental results show accuracy as high as 97.4% between four labels, depending on the applications and type of encryption [38].

Alshammari and Zincir-Heywood have published numerous papers in classifying encrypted network traffic flows. In their research they make a case for the importance of

identifying applications within encrypted network traffic flows to IDS and QoS tasks [7]. They also discuss the ease of using non-standard ports in encrypted traffic as a first step in anonymizing data, and the importance of not using port numbers for identifying network traffic flows [4–7]. In 2008 they test ML algorithm effectiveness in detecting Secure Shell (SSH) traffic when presented with a mixture of 12 types of traffic flows [4]. Although there are multiple traffic types, rather than attempt to classify them all, they only label flows as either SSH or non-SSH. The authors come to the conclusion that C4.5 is preferable to Repeated Incremental Pruning to Produce Error Reduction (RIPPER) due to C4.5's detection rate of above 99% [4]. In [5] they attempt to distinguish two encrypted traffic types, SSH and Skype traffic, using C4.5, SVM, and Naïve Bayes. Once again they do not use IP addresses, port numbers, or payload data to detect SSH. C4.5 had the highest detection rate with 99.6%. In [6] the authors continue to use ML algorithms to classify encrypted network traffic flows, without port numbers, IP addresses, or payload information and use four datasets—three captured in the real-world and one simulated—for testing. This research compares SVM, AdaBoost with decision stumps, Naïve Bayes, C4.5, and RIPPER across all four datasets, when trying to detect either SSH or Skype traffic. C4.5 produced the highest accuracy across all captured datasets and SVM produced the highest accuracy for the simulated dataset. They also show a drop from 97% to 83.7% detection rate for the C4.5 algorithm when testing over a different dataset than training. While this is not as good as training and testing on the same set, they claim the 83.7% detection rate shows robustness in C4.5 and demonstrates its feasibility in use across networks. In 2011, Alshammari and Zincir-Heywood compared C4.5, AdaBoost with decision stumps, and team-based Genetic Programming (GP) [7]. They compared the three learning algorithms when attempting to detect SSH traffic in a mixture of other traffic. They also trained on feature sets generated from one dataset and tested on the other networks. C4.5 once again out-performed the other algorithms when detecting only SSH or Skype traffic on

the network on which it was trained. C4.5 and team-based GP performed comparably when classifying traffic on networks not used for training. They report team-based GP outperforming C4.5 when classifying applications within SSH tunnels.

In 2012, Nguyen et al. discuss the need for “timely and continuous” classification [36]. They define timely classification as not relying on the entire flow for classification, but rather using a predefined number of consecutive packets. They also define continuous classification as not relying on the beginning of the flow, but rather the classifier should be able to classify a traffic flow when given a middle subset of the flow. A timely and continuous network traffic flow classifier should be able to classify flow based on any subset of consecutive packets with the predefined length. To test the feasibility of using timely and continuous classification they test C4.5 and Naïve Bayes on a sliding window of consecutive packets, with varying packet lengths. The dataset used for their experiments came from a network capture with two application types for classification, a game—*Wolfenstein: Enemy Territory*—and Voice Over IP (VOIP). Both algorithms had performance degradation when classifying data that did not include the beginning of the flow. The authors attribute this, in part, to the classifier no longer knowing the direction of the flows; i.e., it could not distinguish client from server. C4.5 and Naïve Bayes performed similarly. The authors conclude that when classifying on consecutive packets that do not include the beginning of the flow, the features used to train should include subflows that do not use the initial packets as well. The authors also report that subflows that begin a thousand packets into a flow have better precision and recall than subflows that are between one and ten packets into the subflow. This indicates that flow features reach a steady state over time. The research presented in the following chapters of this paper studies the timely but not continuous classification of many types of TLS traffic.

In 2004, Wright et al. built a Hidden Markov Model (HMM) to classify network traffic [49]. They used a dataset created from real network captures at George Mason

University (GMU). The GMU dataset determined the labels of the flows by port number. They recognize this is not 100% true, but needed a method to automatically classify enough traffic for testing their HMM methods. They had accuracy ranging from 29% to 86.4% with the best model depending on the type of traffic. In 2006, they published another paper yielding better results with the same dataset [50]. They rely solely on packet timing, sizing, and direction information. No data is collected from TCP/IP headers. They report detection rates from 75% to 100% when using eight types of flows with aggregate traffic. When classifying actual traffic flows, the reported accuracy ranges from 57.7% to 96.7% depending on the traffic type. Wright et al. also worked on tracking the number of connections within a single encrypted tunnel.

In 2009, Wright et al. [51] published a technique, called traffic morphing, that made one type of network traffic resemble another with the intent of hiding traffic from ML analyses. They report successfully lowering the accuracy of a Naïve Bayes classifier from 98.4% to 63.4%. In 2012, Dyer et al. publish a report about overcoming traffic morphing and other counter measures that rely on padding [22]. Dyers et al. test multiple classifiers on three datasets and attempt to classify the websites. The authors report Naïve Bayes and SVM perform comparably and return accuracies greater than 60% against all countermeasures.

2.7 Summary

This chapter presents background information to understand the research and contributions described in the following chapters. The need for network analysis—especially of web service traffic—is given, followed by the current solutions and problems in network analysis. TLS and ML algorithms are also discussed with a focus on C4.5 and AdaBoost, due to their prevalence in the following chapters.

III. Methodology

This chapter describes the methodology for testing the effectiveness of five ML algorithms in classifying the web services within encrypted network traffic flows. Section 3.1 discusses the goals of the research, the hypotheses tested, and the approach used. Section 3.2 introduces the system being tested and the inputs and outputs of the system. Section 3.3 describes the dataset used to test the ML algorithms and how it was created. The chapter concludes by describing the factors used in the experiments, how they are evaluated, and the experimental design.

3.1 Problem Definition

3.1.1 Goals and Hypothesis.

The goal of this research is to determine the effectiveness of ML algorithms in classifying a web service associated with a TLS traffic flow. ML efficacy is determined by classification accuracy, as defined in Section 3.5. It is hypothesized that the tested ML algorithms will successfully classify web services associated with encrypted network flows. In addition to determining whether the ML algorithms can classify the data, this research further analyzes how many packets of the flow are necessary to classify that flow. It is expected that shortening the amount of time on which the features are calculated decreases the accuracy. The time necessary to train a ML algorithm and the amount of time required to classify flows using the algorithm is also discussed.

3.1.2 Approach.

The strategic goal of this research effort is to prevent data exfiltration through encrypted communications. Due to the nature of encryption, network security devices that rely *solely* on DPI cannot detect data exfiltration in encrypted traffic. By measuring the effectiveness of ML in classifying encrypted network traffic flows, this research will

increase the ability to detect data exfiltration. While determining the web service is not sufficient to definitively determine a network traffic flow as containing sensitive data, it is a necessary first step. This research focuses on classifying encrypted web services due to TLS's increasing prevalence. Determining the web service can be used for broader policies. For example, some web services such as social networking could be in violation of network policy. Others, such as web mail, could be considered high risk and warrant additional inspection.

This experiment uses real world data to test the efficacy of ML algorithms in classifying the application within an encrypted flow. By using features calculated directly from real-world network traffic flows—where the web service is already known—as input samples to the ML algorithm, the accuracy of the ML algorithm when classifying the endpoint web service can be directly measured.

3.2 System Boundaries

The System Under Test (SUT), shown in Figure 3.1, is the Flow Identification System (FIS). The FIS consists of the ML framework and the computer on which the framework runs. The ML framework includes two components: one that trains—or creates—the ML classifier, and one that uses the ML classifier to classify network traffic flows. Both components of the ML framework are under test as the classifier's accuracy is dependent upon the trainer. Encrypted flows are given to the trainer which generates the classifier used in the second component. The encrypted flows are given to the SUT as an Attribute-Relation File Format (ARFF) file. ARFF is a file format that stores the traffic flow as a label and a list of attributes. The attributes include the sizing information of the traffic flow. These attributes are discussed more in Section 3.3 and are used as the classifying features for the ML algorithm. When the ARFF files are used for training, the label is included for each traffic flow. The ARFF files constitute the workload for the SUT.

In addition to this workload, there are system parameters that affect the accuracy of the ML framework. These parameters control how the ML classifier is created and are discussed in more detail in Section 3.6.

The flows used to train and test the ML algorithm are collected from a single computer network as described in Section 3.3.1. It is expected that different computer networks will produce different results [6, 7]. While a ML algorithm trained on one network is not expected to be as accurate on another network, it is expected that the results from this research effort can still apply to other networks if the training and classification data comes from the same network. However, the accuracy of the algorithms is expected to decrease as differences between the data used to train and test the classifiers increase.

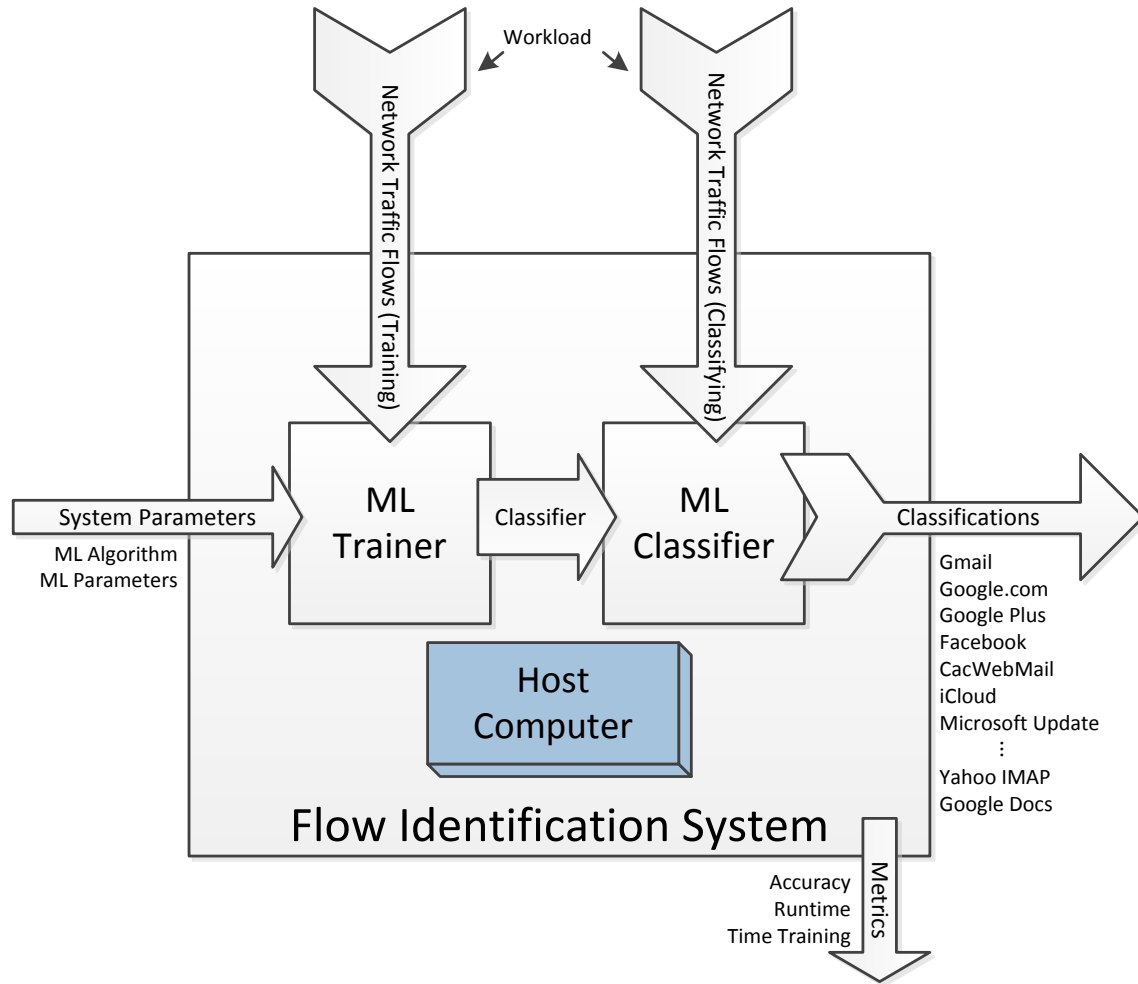


Figure 3.1: Flow Identification System

3.3 Workload

The workload of the SUT consists of the encrypted flows. The amount of time necessary to classify each flow is dependent upon the how the classifier was trained. ML classifiers are entirely dependent upon the network flows used to train the ML algorithm. The following parameters are used when creating the workload.

- Flow length analyzed

The amount of the network traffic flow analyzed when calculating the features is defined as the number of packets from the beginning of the flow, referenced as the First N Packets. Nguyen et al. showed using the beginning of the flow to be more accurate in their research, and using the beginning of the flow allows for the least delay in classifying [36]. The accuracy of the ML algorithm is expected to decrease as N decreases. Connection-based traffic consists of three basic parts: handshake, communication, and teardown. A decrease in N will remove the teardown stage and limit the information the ML classifier has to classify the traffic flow. As the amount of data used continue to decrease, the accuracy is hypothesized to decrease. The same N is used for training and testing the ML algorithm. Appendix A includes the results of alternate experiments where time is used to determine flow lengths. First N packets is chosen as the defining length of flow for two reasons, its ease of use, and the inherit additional information in using time elapsed. By using microseconds as the primary measure, timing data are confounded with the sizing data.

- The characteristics documented and used for the ML features

In previous research Moore et al. used 249 features to describe a network traffic flow [33]. A subset of these 249 features is used to conduct this research. Features pertaining to the sizing and number of packets within in the traffic flows are the focus of this research, with some additional features based on TCP flags. These features are chosen due to their expected speed of calculation when compared to some of Moore's other features, such as Fast Fourier Transforms (FFTs). Little saved state is required and simple arithmetic is sufficient to calculate all used features, making these features efficient for a near-real-time device. The final feature set is listed in Table 3.1. Alternate experiments are run with an additional 21 timing-based features, as discussed in Appendix A.

Table 3.1: Features

Feature	Description
1-21*†	Number of bytes in Ethernet packet
22-42*†	Number of bytes in IP packet
43-63*†	Number of bytes in IP and TCP headers
64-65	Number of packets
66-67	Number of packets with TCP ack flag set
68-69	Number of packets with only the ack flag set
70-71	Number of packets with TCP optional SACK Blocks
72-73	Max number of SACK blocks in a single packet
74-75	Number of packets with ack flag set and SACK information
76-77	Number of packets with TCP payloads
78-79	Number of combined bytes within TCP payloads
80-81	Number of packets with the TCP push flag set
82-83	Number of packets with TCP syn flag set
84-85	Number of packets with fin flag set
86-87	Was a packet sent allowing SACK blocks (Value is Y or N)
88-89	Number of packets with TCP urgent flag set
90-91	Number of combined bytes within packets that have urgent flag set

All values are calculated for both Client to Server and Server to Client

† –Values are also calculated across entire flow

* –Value is collected per packet and the following 7 stats are computed:

Minimum; First Quartile; Mean; Median; Third Quartile; Maximum; Variance

The first three rows have 21 features each: 7 stats for each Client to Server, Server to Client, and entire flow.

Refer to [31] for more information on Selective Acknowledgment (SACK)

- Label

Every encrypted flow has a label. The goal of the system is to correctly label encrypted TLS flows. The label is an important workload parameter since it almost certainly affects the accuracy of the system. Certain labels might be recognized with more accuracy and, even if classification is not successful for every label, accurately identifying particular labels could still help fulfill the goal. The labels used in this experiment are described below and listed in Table 3.2.

3.3.1 Workload Generation.

The flows for the workload are captured from a real-world research network at the Air Force Institute of Technology. The traffic is captured continuously over a seven day period. The seven day period is considered to be representative of a typical week for the network. This traffic is separated into traffic flows using the same 5-tuple as Moore et al. in [48]. 1,527,185 TCP streams are found and 991,198 UDP streams are found. Out of all the TCP streams, 218,814—14.3%—of them use TLS. Due to ML algorithms’ need for sizable datasets when classifying, an automated system for classifying the web services is required [48]. Since TLS traffic has an optional field for using plain text server names when initializing encrypted connections, tests are done to determine the frequency of the field’s usage, and which server names are used. Of the TLS streams, 204,150—93.3%—have the optional server name. Of those only 2,825 unique server names are used. Furthermore, 109,959—53.9% of the TLS flows with server names—are using the 17 most common server names. All but one of those 17 server names are chosen as the labels for the dataset. The most used server name, with 54,711 flows, is an update server. Due to the number and small size of these flows, they are not considered in this research. Human readable forms of the 16 selected labels with the associated number of flows are listed in Table 3.2.

Table 3.2: Labels

Label	Number of Flows
Gmail	10930
Google.com	6671
Google Plus	6502
Facebook	5154
CacWebMail	4701
iCloud	4436
Microsoft Update	4330
Google Safebrowsing	3689
Mesh Accounts	3012
Live Messenger	2028
bitbucket	1067
Metric Static	911
BlackBoard	599
Google IMAP	585
Yahoo IMAP	403
Google Docs	230
Total	55248

3.4 System Services

The FIS provides one service: classification of traffic flows. Each traffic flow is classified by the ML algorithm as one of the labels used to train the ML algorithm. This label is the output of the system. Table 3.2 provides a complete list of labels. For each classification, the system is either correct (a True Positive (TP)) or incorrect. If the system

is incorrect then it is a False Positive (FP) for the resulting label and a False Negative (FN) for the true label. Similarly, when a TP occurs it is a True Negative (TN) for all other labels.

3.5 Performance Metrics

The primary performance metric of the system is the accuracy of the traffic flow classifications. The accuracy of each factor set is calculated as follows,

$$Accuracy = \frac{Number\ of\ Correct\ Classifications}{Number\ of\ All\ Tested\ Samples}. \quad (3.1)$$

For a multiclass classifier, such as this system, the overall accuracy is also the combined, weighted True Positive Rate (TPR) across all classifications. The amount of time necessary to train each algorithm is also reported, as given by the ML framework. The average CPU time each algorithm requires to classify a single test flow is calculated by,

$$Runtime = \frac{Time\ Required\ to\ Classify\ Test\ Samples}{Number\ of\ Tested\ Samples}. \quad (3.2)$$

To be feasibly used in a near-real-time classifying system, the runtime should be low enough for the classifying system to maintain network throughput.

3.6 System Parameters

The system parameters include the individual ML algorithm parameters. The specifications of the computer used to host the ML framework also affects the metrics because it affects system performance and the amount of time necessary to train and test the ML being considered. This research focuses on the J4.8 ML algorithm, an open source variation of C4.5. The following parameters affect how the algorithm creates the decision tree. The default values referred to are the values given by the Weka 3.6.7 toolkit. For more detailed information on the Weka toolkit, the J4.8 algorithm, and the possible configurations see [48]. The default parameters for all ML algorithms are used for this research, unless otherwise specified. While tuning the algorithms by attempting to

optimize the parameters is common, the default parameters produce acceptable results and time constraints limited the scope of this research.

3.6.1 J4.8 Parameters.

- Binary Splits

If binary splits are used, the nodes of the decision tree will have at most two subtrees.

This parameter is either true or false. Binary trees are not used for this study.

- Minimum Number of Objects

This number defines the minimum number of training samples associated with any leaf of the tree. Higher numbers lead to a more general tree, but at the possible cost of accuracy. The default value for this parameter is 2; the value must be a positive integer.

- Laplace Estimator

The Laplace estimator is used to eliminate problems that come from dividing by probabilities of zero. This can increase the accuracy of the classifier if not all possibilities are covered in the data used to create the decision tree. The Laplace estimator is not used by default and will not be used herein. The value of this parameter is either true or false.

- Pruning

Pruning reduces the number of nodes in the tree, making it simpler. Simpler trees not only make it easier to analyze the relationships, but also make the resultant decision more general. If pruning is disabled, the following parameters are not used. Pruning is enabled by default and is used in this study.

- Subtree Raising

Subtree Raising occurs when a parent node in the decision tree is replaced with one of its child nodes. While subtree raising simplifies the tree and enhances pruning, it is time consuming and increases the time of tree generation. Subtree Raising is used by default and is used for this study.

- Modified J4.8 Reduced Error Pruning

If this feature is on the ML trainer uses a modified reduced error pruning method instead of the default C4.5 method of pruning. Modified J4.8 Reduced error pruning is not used by default and will not be used for this experiment.

- Confidence Factor

Smaller confidence factors cause less pruning and a less general decision tree. The confidence factor is only used with the C4.5 pruning method, it is not used in conjunction with the J4.8 modified reduced error pruning. The confidence factor is a decimal number between 0 and 1, the default value is 0.25 and is used for this experiment.

3.6.2 AdaBoost Parameters.

- ML Classifier

The ML algorithm that is being boosted. DecisionStump is the default in Weka. This research looks at the effect of using J4.8 as the boosted classifier.

- Number of Iterations

The number of iterations AdaBoost uses to boost the classifier. The default is 10, which is what is used in this research.

- Seed

The number used to seed the random number generator for re-weighting the samples. One is the default and is used in this research.

- Resampling

If this parameter is true the AdaBoost algorithm uses resampling instead of reweighting. The default parameter of false is used.

- Weight Threshold

This is the threshold at which pruning occurs. This is set to 100 by default, which is what is used in this research.

3.6.3 Computer System Parameters.

The system parameters affect the amount of time necessary to train and test the ML algorithms. All experiments are run on a Dell T7500 with 12 gigabytes of Random Access Memory (RAM) and a 2.67 gigahertz six-core Intel Xeon processor. Ubuntu 12.04 is the operating system, and Weka is run using Java 1.6.

3.7 Factors

- Flow Length Analyzed

The flow length used for classification determines the amount of data the ML algorithm has to train and classify each network flow. The flow length analyzed is tested to determine how many packets are necessary to classify the endpoint web services of the network traffic flow. If the required flow length is sufficiently short, ML could be used in real-time detection devices.

- Complete Traffic Flow

The complete traffic flow is classified to determine the best expected accuracy the ML algorithm.

- First N Packets

The first N packets are used to test, with N ranging from one to twenty. Twenty was experimentally determined to be after the performance peak with respect to accuracy.

- ML Algorithm

Five ML algorithms are used in this research: J4.8, AdaBoost+J4.8, Naïve Bayes, NBTree, and SVM. These algorithms are described in Chapter 2 and were chosen due to their previously reported results.

Table 3.3: Factor Levels

Factors	Levels
Flow Length Analyzed (Number of Packets)	All; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19; 20
ML Algorithm	J4.8; AdaBoost+J4.8; Naïve Bayes; NBTree; SVM

3.8 Evaluation Technique

The workload is created from real network data as discussed in Section 3.3. Thus, direct measurement is used to evaluate the SUT. Version 3.6.7 of the Weka Toolkit trains and tests the ML algorithm. Weka’s implementation of the 5 ML algorithms are used for this experiment, with the exception of LibSVM taken from [12]. 10-fold cross validation is used for testing. The workload is divided into 10 equal folds of network traffic flows. Each fold is iteratively used for testing after the remaining nine folds are used for training. The results of the 10 iterations are averaged to produce the mean accuracy for that experiment. Using 10-fold cross validation is standard in ML experiments and increases the fidelity

of the results by guaranteeing every flow is used to both train and test, across separate iterations [48]. Direct measurement is used since the network data and flow features are recorded from real network traffic, and that data is used by the ML framework to test the ML algorithms. By using real network data the results of the research are applicable for use in real networks.

3.9 Experimental Design

Due to time constraints a partial factorial design is used. Refer to Table 3.3 and Section 3.7 for the factors used in these experiments. Experiments are run testing all 5 ML algorithms at an experimentally-determined packet level, for 5 experiments. Pilot studies show classifying the first 12 packets produces high accuracy. Every flow length is tested with the best two algorithms for $21 * 2 = 42$ experiments. Although 10-fold cross validation is used for every experiment, it is still typical to run 10 repetitions of every experiment to measure variance and for a better measure of accuracy [48]. The same set of folds is used within a repetition and the folds are randomly regenerated between repetitions. Thus, $5 + 42 = 47$ unique experiments * 10 replications results in 470 total experiments. The focus of this research is the efficacy of ML algorithms in classifying encrypted flows and the number of packets necessary to correctly classify these flows. The research achieves this goal by focusing the experimental setup on testing the flow length analyzed. 10 iterations of 10-fold cross validation is expected to result in a narrow confidence interval at 95% confidence. This level of confidence is necessary, because misclassifications could lead to data exfiltration. Due to the randomness of generating the folds, the entire workload can be used for each 10-fold cross validation.

3.10 Methodology Summary

Data exfiltration is a growing concern for organizations. With an increasing amount of applications using encrypted web services, tools that can assist in web service classification

would improve network security. This research determines the feasibility of using ML algorithms to classify endpoint web services using encrypted network flows. Furthermore, if classifying encrypted network flows is feasible, then the flow length needed to classify is experimentally determined. If the web service can be classified quickly enough, then this strategy of application identification could be used as a first step in near-real-time network protection devices.

The network data used for testing is real-world data collected from an existing research network. Features for the samples are generated from the sizing information of encrypted network traffic. These samples are fed into the Weka toolkit's implementation of 5 ML algorithms for training and testing. Experiments are run for each ML algorithm at a single flow length and the best two are run for all 21 flow lengths. All experiments are evaluated on the accuracy and runtime of the classifications.

IV. Results and Analysis

This chapter discusses the results from the experiments detailed in Chapter 3. Section 4.1 analyzes the results from the experiment with all five ML algorithms. Section 4.2 analyzes the effects of flow length on the accuracy and runtime of J4.8. Section 4.3 discusses AdaBoost+J4.8 and the difference in results between the two algorithms. Section 4.4 summarizes the results.

4.1 Algorithm Comparison

The accuracy, time training, and runtime are shown below in Table 4.1. Recall the runtime is the average CPU time taken to classify a *single flow*. The time training is reported in seconds, and the runtime is reported in microseconds. The algorithm comparison experiment only uses features calculated from the first 12 packets of the flow. Shapiro-Wilk normality tests confirm that most of the values have normal distributions, and Table 4.1 uses a † to indicate which values are not normal. Recall from Chapter 3 that 10-fold cross validation is used. Figures 4.1 and 4.2 show the effects of averaging the 10 folds on distribution and normality. Figure 4.1 shows a decrease in variance, and Figure 4.2 shows both a decrease in variance and an increase in normality.

A pairwise *t*-test is run across all pairs of algorithms to determine which algorithms produce statistically-different accuracy and runtimes. The p-value from these results are shown in Tables 4.2 and 4.3. A p-value of less than 0.05 indicates that the null hypothesis—that the values are not different—should be rejected within a 95% confidence interval. As the tables show, all algorithms produce statistically-different results. Note the tables show values rounded to six decimal places.

Naïve Bayes performs the worst with a mean accuracy of only 0.6238, and while it classifies faster than NBTree or LibSVM, it is still orders of magnitude slower than AdaBoost+J4.8 and J4.8. LibSVM has a mean accuracy of 0.9656, which, while only

Table 4.1: Algorithm Comparison: Mean Values from 10 repetitions

Algorithm	Accuracy	Time Training (s)	Runtime (μ s)
NBTree	0.974276	2064.1796	601.63216†
NaiveBayes	0.623773	1.2924	341.098358
AdaBoost+J48	0.979286	113.8176	16.724549†
J48	0.978298	14.4324	2.479733
LibSVM	0.965606	216.5916	2188.477515

† - Indicates the values are not normally distributed

Table 4.2: Statistical Difference in accuracy for All 5 Algorithms

P-Values from t -Test

	NBTree	NaiveBayes	AdaBoost+J48	J48	LibSVM
NBTree	1	0	1.5e-05	8.6e-05	0
NaiveBayes	0	1	0	0	0
AdaBoost+J48	1.5e-05	0	1	0	0
J48	8.6e-05	0	0	1	0
LibSVM	0	0	0	0	1

A P-Value <0.05 indicates the samples are not the same with 95% confidence intervals

being 0.0127 less than J4.8, is 91.0348 standard deviations away. This shows the apparent similarity in the accuracy is deceiving, as confirmed by the t -test. Both J4.8 and AdaBoost+J4.8 perform well for accuracy and runtime, with accuracies above 0.978 and runtimes below 16.725 μ s. Tukey tests are not shown, but produce slightly different results. Due to this, one-way tests, which are non-parametric, are used to verify the t -test's results. The slight discrepancies in the Tukey test are suspected to be caused by the fitted model used, and the Tukey test is suspected to be less accurate.

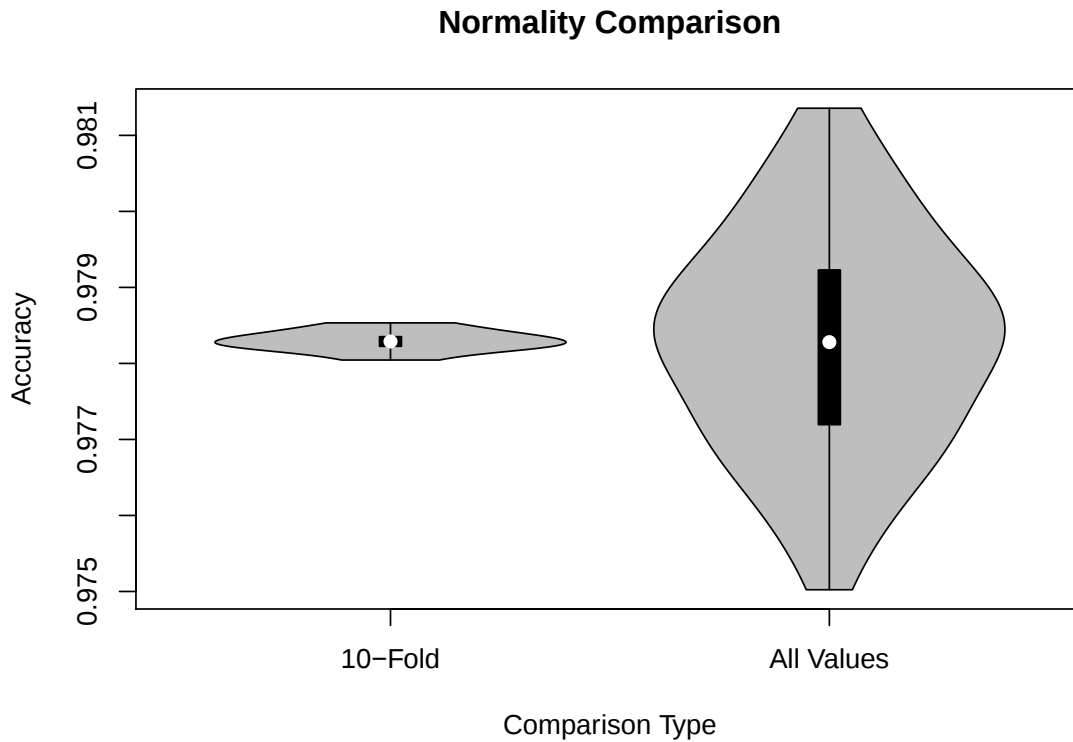


Figure 4.1: Violin Plot Showing Effects of 10-Fold Cross Validation on Distribution

The poor runtime of Naïve Bayes, NBTree, and LibSVM could probably be improved by using feature selection algorithms. Feature selection would eliminate features determined to be of little use in classifying and reduce the computations needed. This especially improves the runtime of ML algorithms that require intensive mathematical calculations, such as Naïve Bayes, NBTree, and LibSVM. Feature selection can also improve accuracy. Tuning the parameters could also increase the accuracy—and maybe the runtime—of the three weakest performers, making them more comparable to J4.8 and AdaBoost+J4.8. Of course, it could also increase the performance of J4.8 and AdaBoost+J4.8, though perhaps less significantly.

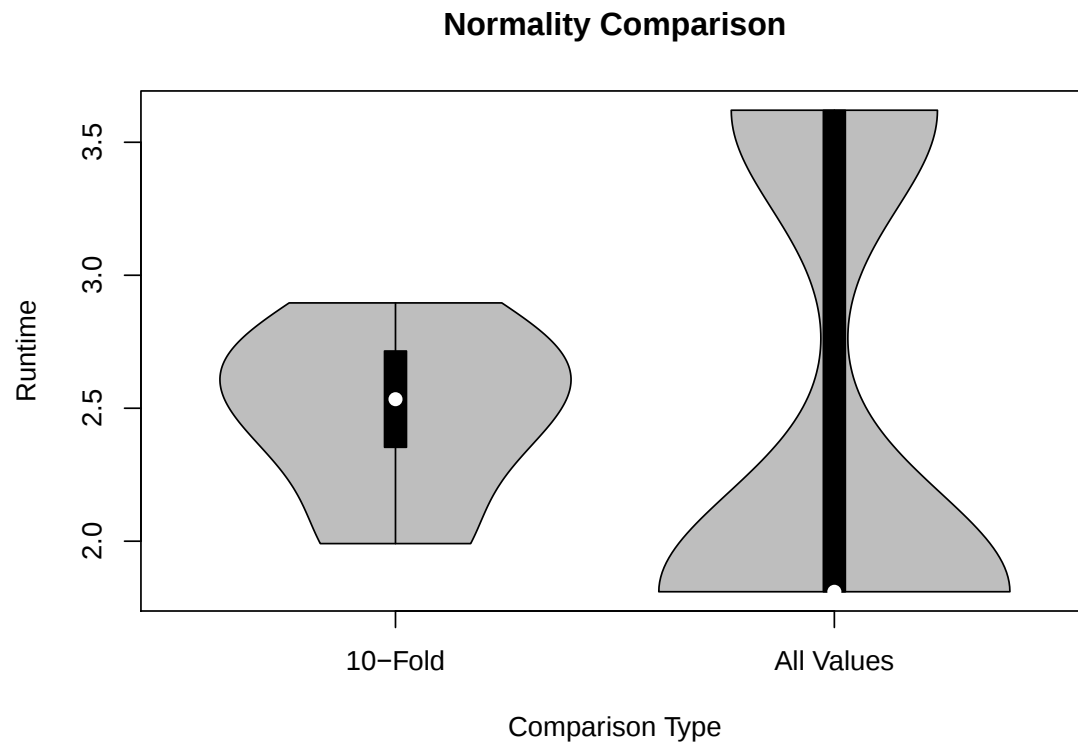


Figure 4.2: Violin Plot Showing Effects of 10-Fold Cross Validation on Distribution

Table 4.3: Statistical Difference in Runtime for All 5 Algorithms

P-Values from t -Test

	NBTree	NaiveBayes	AdaBoost+J48	J48	LibSVM
NBTree	1	0	0	0	0
NaiveBayes	0	1	0	0	0
AdaBoost+J48	0	0	1	0	0
J48	0	0	0	1	0
LibSVM	0	0	0	0	1

A P-Value <0.05 indicates the samples are not the same with 95% confidence intervals

4.2 Flow Length Comparison

This section analyzes the effects of varying the flow length analyzed in the workload generation on J4.8's performance. Section 4.3 analyzes AdaBoost+J4.8 and the differences between the two algorithms across all tested flow lengths.

4.2.1 J4.8.

Table 4.4 shows the mean accuracy, time training, and runtime using J4.8 for all flow lengths analyzed. As expected, the accuracy and the runtime increase as the flow length analyzed increases. The accuracy peaks and eventually decreases as the flow length continues to increase. Figures 4.4 and 4.5 show this increase for both J4.8 and AdaBoost+J4.8. Statistical difference between accuracy is analyzed for every possible combination of flow length pairs. Once again, *t*-tests are used to determine which values are statistically different. Tukey and one-way tests are also run to verify the *t*-test's findings. Although they mostly came to the same statistical conclusions, *t*-tests are used for the final conclusion due to earlier concerns with Tukey's use of fitted models and the inherent randomness of one-way tests.

Table 4.5 displays which flow lengths produce statistically different accuracy for J4.8. The X indicates that the two flow lengths—listed by row and column—are *not* statistically different. As the table shows, most flow lengths are statistically different and the highest accuracy, which is produced by 14 packets, is statistically different from all others. While being statistically the most accurate, a flow length of 14 packets does not have statistically different runtimes than flow lengths of 9-10, or 12-16 packets. The accuracy also decreased beyond 14 packets—including analyzing all available packets—indicating that the flows characteristics change after the first 14 packets. The increased flow lengths are suspected to produce worse accuracy due to the addition of end point specific information. This is further discussed after analyzing the performance of AdaBoost+J4.8. The worst accuracy can be explained by the unequal number of samples. Simply classifying all flows as the

most common label—Gmail—results in a 0.1978 accuracy. Distinguishing between the two most common labels and correctly labeling those—and no others—would result in a 0.3186 accuracy.

Table 4.6 shows which flows produce statistically different runtimes. This table is similar to Table 4.5, as the X indicates the two flow lengths are *not* statistically-different. Two clusters appear and lengths of 9-16 packets and 16-20 packets are not statistically different within the clusters, with the exception of a flow length of 11 packets. The entire flow is also clustered with flow lengths of 16-20 packets. While more packets were included in the feature calculation, the runtime is only the time required for the ML algorithm to classify *not* including time taken for feature calculation. No attempt was made to optimize feature calculation, but the average time taken to calculate and save the features is within an order of magnitude of the runtime for flow lengths of 1-20 packets. This plateau of runtimes would indicate that the features provide enough information for J4.8 to create decision trees of equivalent complexity for flow lengths greater than 15 packets. Figure 4.3 shows a violin plot comparing the distribution of 10-12 packet flow lengths. It is hypothesized that a flow length of 11 packets breaks from the cluster due to its skew towards shorter runtimes. The flow length of 10 packets is skewed toward higher runtimes. Refer to Figure 4.1 for an example of normally distributed values.

Overall, J4.8 performs well with an accuracy above 97% when only analyzing the first 10-20 packets of the flow and runtimes less than 3 microseconds for all tests. At peak accuracy the runtime for classifying a single flow was 2.371126 microseconds, that is equivalent to classifying 421,740 flows *per second*. The relative speed of classifying and high accuracy make J4.8 a good candidate for near-real-time web service classification.

Table 4.4: Means for Flow Length Comparision J48

Num Packets	Accuracy	Time Training (s)	Runtime (μs)
1	0.238524†	2.554	1.194619
2	0.238524†	2.5344	1.067906
3	0.348331†	3.4348†	1.104102
4	0.349491	6.0195	1.466129†
5	0.34903	6.5041	1.484212†
6	0.899841	8.5593	1.882422
7	0.899938	9.3896	2.081517
8	0.916846	10.7701	2.117722
9	0.924989	12.803	2.353026
10	0.973735	10.9558	2.280615†
11	0.973842	11.2326	2.135819†
12	0.978298	13.4345	2.35303
13	0.979348	14.839	2.407332
14	0.979979	14.3684	2.371126†
15	0.979022	15.8042	2.353033
16	0.97834	16.2736	2.497816
17	0.978754	18.186	2.660732†
18	0.978171	19.6132	2.715037
19	0.976392	19.7065	2.715034†
20	0.974513	19.9652	2.805531
All	0.955198	23.5585	2.733133

† - Indicates the values are not normally distributed

Table 4.5: Statistical Difference of Accuracies in J48 by Flow Length Analyzed

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	All
1	-	X																			
2	X	-																			
3			-																		
4				-																	
5					-																
6						-	X														
7						X	-														
8								-													
9									-												
10										-	X										
11										X	-										
12												-				X		X			
13													-								
14														-							
15															-						
16												X				-		X			
17																	-				
18												X				X		-			
19																			-		
20																				-	
All																					-

X - Indicates the values are not statistically different according to the paired *t*-test

Table 4.6: Statistical Difference of Runtimes in J48 by Flow Length Analyzed

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	All
1	-	X	X																		
2	X	-	X																		
3	X	X	-																		
4				-	X																
5				X	-																
6						-	X														
7						X	-	X		X	X										
8							X	-	X	X	X	X									
9								X	-	X	X	X	X	X	X	X					
10							X	X	X	-	X	X	X	X	X	X					
11							X	X	X	X	-	X									
12								X	X	X	X	-	X	X	X	X					
13									X	X		X	-	X	X	X					
14									X	X		X	X	-	X	X					
15									X	X		X	X	X	-	X					
16									X	X		X	X	X	X	-	X	X	X	X	X
17																X	-	X	X	X	X
18																X	X	-	X	X	X
19																X	X	X	-	X	X
20																X	X	X	X	-	X
All																X	X	X	X	X	-

X - Indicates the values are not statistically different according to the paired *t*-test

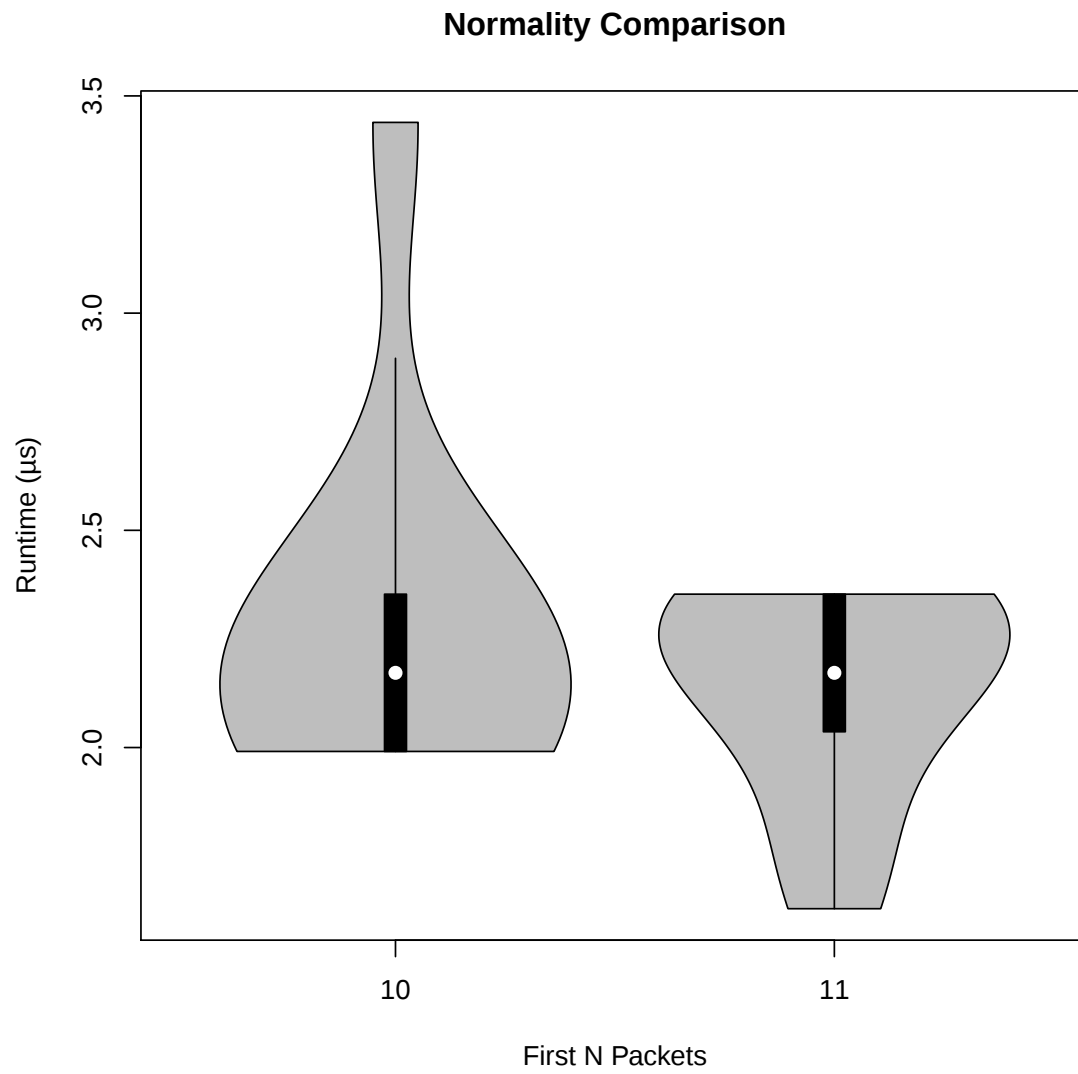


Figure 4.3: Violin Plot Showing (Lack of) Normality for J4.8 Runtimes

4.3 AdaBoost+J4.8

Table 4.7 shows the mean results from using AdaBoost+J4.8 to classify test samples with all flow lengths analyzed. Similar to J4.8, the accuracy increases—at least initially—as the flow length increases. The runtime also increases as flow length increases, although without an apparent plateau. Figures 4.4 and 4.5 plot the accuracy and runtime versus flow length. Figure 4.6 shows accuracy versus flow length, but zoomed in on the final 10 flow lengths to show the differences in accuracy more clearly. Table 4.8 shows which flow lengths produce statistically-different accuracies. As with J4.8, only a handful of the flow lengths produce statistically similar results, and the highest accuracy—this time 18 packets—was statistically-different from all others. Unlike J4.8, few of the runtimes are not statistically-different, as shown in Table 4.9. Flow lengths of 17 and 18 packets are not statistically different from each other in runtime, but are statistically worse than flow lengths 1-16 and better than 19, 20 packets, or the entire flow.

Recall that AdaBoost works by iterating over the boosted algorithm and varying weights to create additional classifiers that have less error for these new weightings. These additional classifiers require more time to train and test. AdaBoost stops iterating if newly created classifiers cannot improve performance. Figure 4.7 shows the ratio of training and runtimes between AdaBoost+J4.8 and J4.8. For flow lengths between 1 and 5 packets the time training, runtime, and accuracy are all very similar or identical; this indicates that AdaBoost is unable to produce any extra classifiers to improve performance. The spike in time training and runtime, beginning at 6 packets, indicates that AdaBoost creates additional classifiers. While the additional classifiers slightly lower accuracy for flow lengths 6-8, AdaBoost statistically improved J4.8's accuracy for the flow lengths 9-20 and when all packets are available.

Table 4.7: Means for Flow Length Comparison AdaBoost+J48

Num Packets	Accuracy	Time Training (s)	Runtime (μs)
1	0.238524†	2.5909	1.176507
2	0.238524†	2.5895†	1.321329
3	0.348331†	3.4874	1.267011†
4	0.349491	6.0863	1.520428
5	0.34903	6.5356†	1.502338†
6	0.899486	80.7932	14.172457
7	0.89947	84.5713	13.900967
8	0.916274	89.3318	13.321739
9	0.926048	106.1684	15.638576
10	0.974605	90.4441	13.339829
11	0.974533	95.3992	13.756144
12	0.979286	105.82	15.457567
13	0.980653	117.322	16.815069
14	0.981362	122.5245	18.136428
15	0.980924	137.6435	19.693045
16	0.981782	140.5467	20.127418
17	0.982906	158.178	21.285831
18	0.984104	160.4537	21.557334
19	0.983078	164.6682	21.99176
20	0.983056	174.626	22.715774†
All	0.969943	249.8702	25.249793

† - Indicates the values are not normally distributed

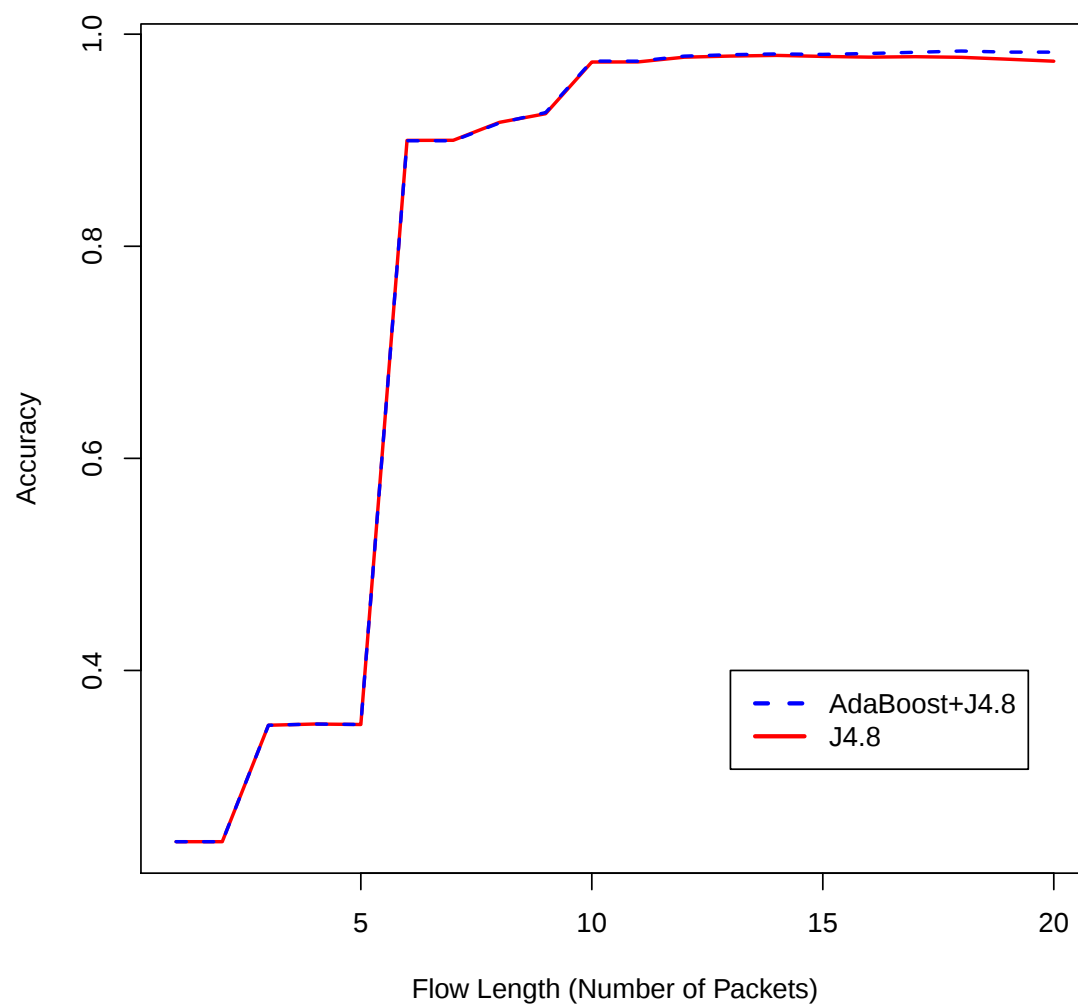


Figure 4.4: Plot of Accuracy by First N Packets for J4.8 and AdaBoost+J4.8

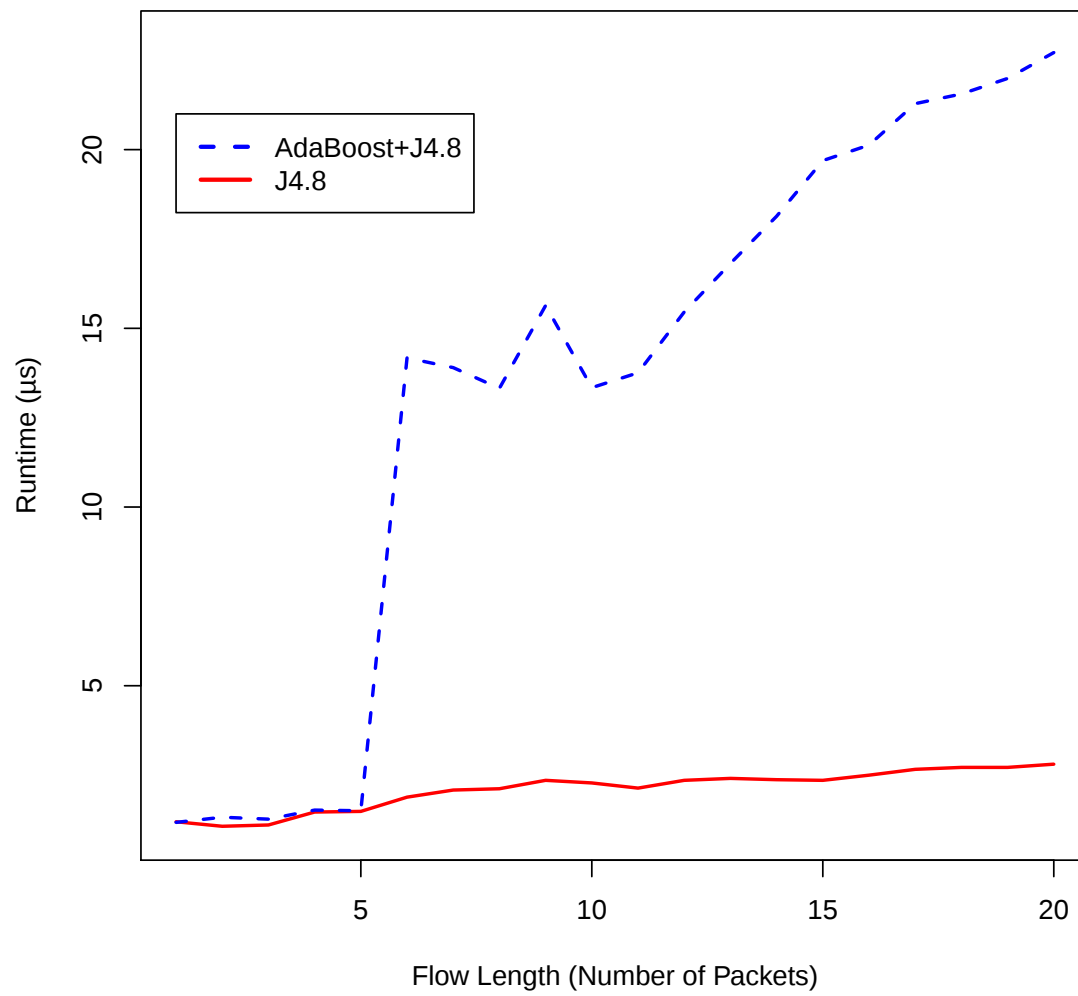


Figure 4.5: Plot of Runtime by First N Packets for J4.8 and AdaBoost+J4.8

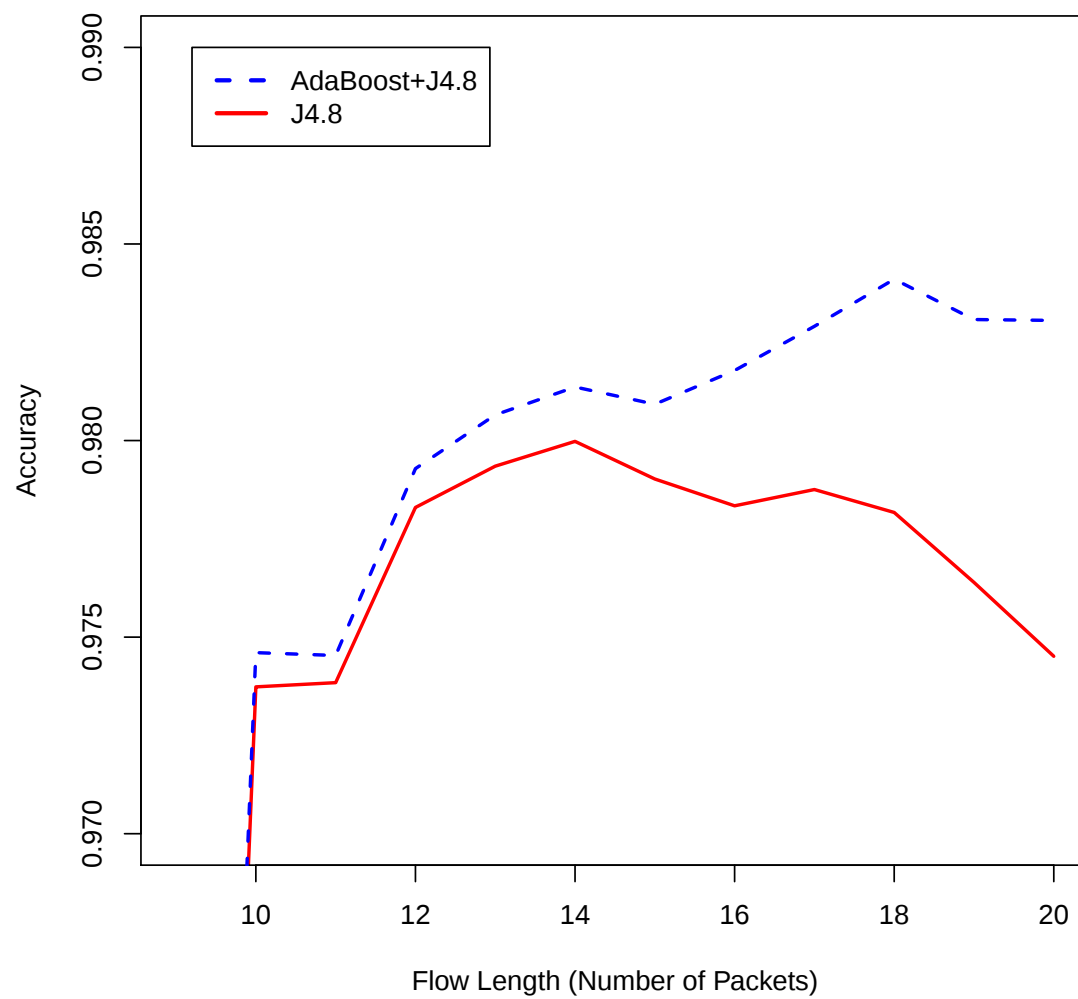


Figure 4.6: Zoomed in Plot of Accuracy by First N Packets for J4.8 and AdaBoost+J4.8

Table 4.8: Statistical Difference of Accuracies in AdaBoost+J48 by Flow Length Analyzed

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	All
1	-	X																			
2	X	-																			
3			-																		
4				-																	
5					-																
6						-	X														
7						X	-														
8								-													
9									-												
10										-	X										
11										X	-										
12												-									
13													-								
14														-							
15															-						
16																-					
17																	-		X	X	
18																		-			
19																	X		-	X	
20																	X		X	-	
All																					-

X - Indicates the values are not statistically different according to the paired t -test

Table 4.9: Statistical Difference of Runtimes in AdaBoost+J48 by Flow Length Analyzed

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	All
1	-	X	X																		
2	X	-	X		X																
3	X	X	-																		
4				-	X																
5		X		X	-																
6						-	X				X										
7						X	-				X										
8								-		X	X										
9									-			X									
10							X			-											
11						X	X	X			-										
12									X			-									
13													-								
14														-							
15															-						
16																-					
17																	-	X			
18																	X	-			
19																			-		
20																				-	
All																					-

X - Indicates the values are not statistically different according to the paired t -test

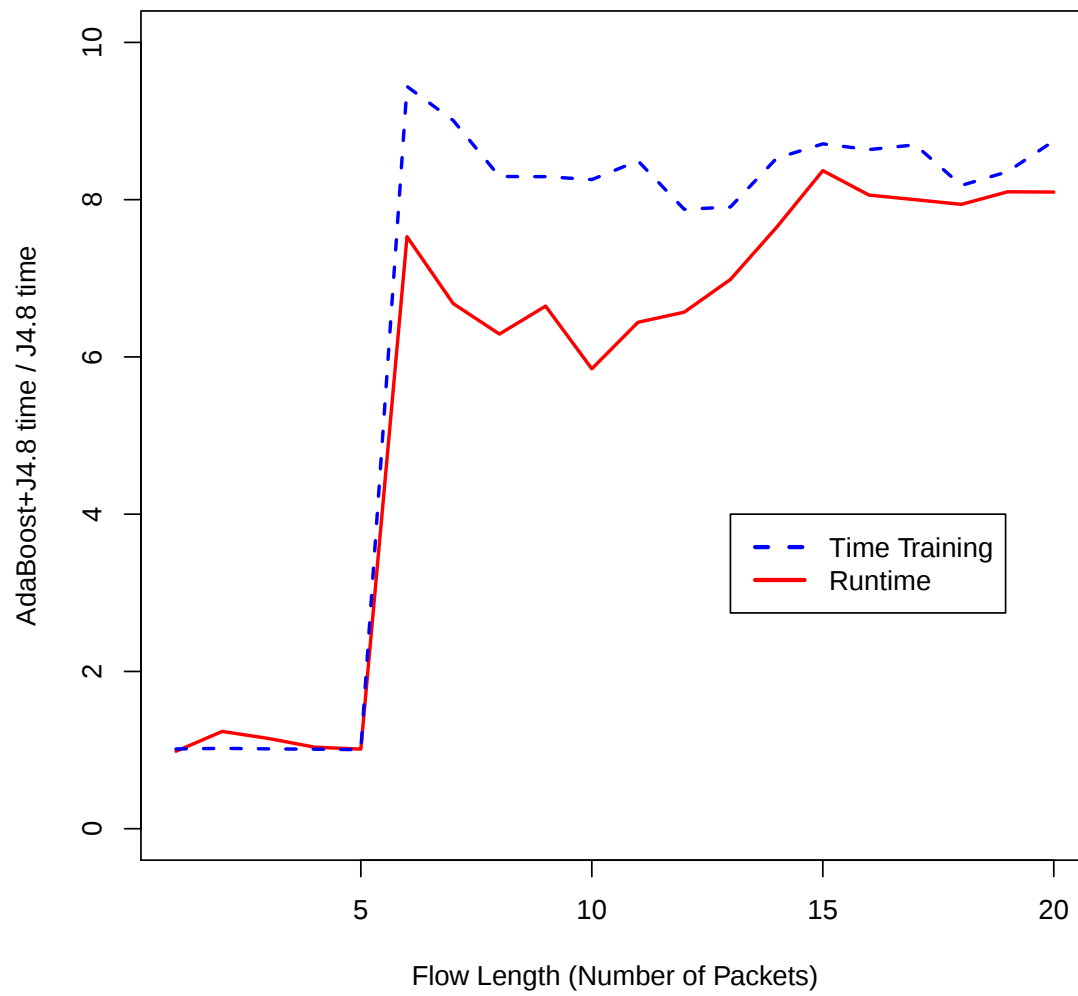


Figure 4.7: Ratio of Time Training and Runtime of AdaBoost+J4.8 to J4.8

Table 4.10 shows the statistical differences between accuracies and runtimes of J4.8 and AdaBoost+J4.8 for every flow length analyzed. Statistical difference is tested with pairwise *t*-tests at a 95% confidence interval. Running Friedman tests—which are non-parametric—also indicate that the J4.8 and AdaBoost+J4.8 perform differently, even taking the different datasets into account. As with the *t*-tests, a 95% confidence interval is used for the Friedman tests. Figures 4.4 and 4.5 show the increase in runtime is much more significant than the increase in accuracy.

The increase in accuracy as flow lengths increase is explainable by the addition of information about each flow as more packets are used to calculate the features. The spike at the flow length of 6 packets is probably due to the resolution of the TCP handshake and the beginning of web service-specific information. The accuracy continues to increase as a higher percentage of the information acquired is web service-specific. This includes TLS handshake, which takes as little as six packets—for a total of nine packets into the flow—to complete. The peak and eventual decrease of the accuracy could be due to the introduction of confounding data. Differences in browsers or user profiles could affect the page content being returned and introduce additional variability. The accuracy of analyzing the full flow is lower than the accuracy of analyzing flow lengths 10-20 for both algorithms. This indicates that flows continue to become more unpredictable after the initial peak. AdaBoost+J4.8's later peak in accuracy is due to AdaBoost's additional classifiers. By comparing the first 5 accuracies it is apparent that AdaBoost did not successfully reduce the error with additional classifiers and used the same classifier that J4.8 produced. This is not surprising as J4.8 is deterministic and will always produce the same decision tree with the same input samples. AdaBoost's additional classifiers most likely increase accuracy after the first 9 packets due to the ability of additional classifiers to more accurately handle less common branches in the original tree.

Table 4.10: Statistical Difference of Accuracies and Runtimes

Number of Packets	Accuracy	Runtime (μs)
1	-	-
2	-	J
3	-	-
4	-	-
5	-	-
6	J	J
7	J	J
8	J	J
9	A	J
10	A	J
11	A	J
12	A	J
13	A	J
14	A	J
15	A	J
16	A	J
17	A	J
18	A	J
19	A	J
20	A	J
All	A	J

J - Indicates J4.8 was statistically different and better.

A - Indicates AdaBoost+J4.8 was statistically different and better.

AdaBoost+J4.8 also performs well with an accuracy above 98% when analyzing only the first 13-20 packets of the flow and has runtimes less than 23 microseconds for all tests. Effectively classifying 46,387 flows a second. If 23 microseconds per flow is fast enough for a near-real time device to classify all web service traffic without becoming overloaded, then AdaBoost+J4.8 is a better decision than J4.8, because of the improved accuracy. It would take AdaBoost+J4.8 less than 5 *seconds* to classify all TLS flows seen in a week on the test network (compared to 0.52 seconds for J4.8), this shows the feasibility of using either algorithm in a near-real-time device for the test network. If runtime is a concern, J4.8 is almost an order of magnitude faster and—with less than 1% drop in accuracy—could be the better choice.

4.4 Summary

With the default parameters J4.8 and AdaBoost+J4.8 outperformed Naïve Bayes, NBTree, and LibSVM in both accuracy and runtime. In a more detailed analysis J4.8 has a faster runtime than AdaBoost+J4.8 for flow lengths of greater than 5 packets, and AdaBoost+J4.8 has a higher accuracy for flow lengths greater than 8 packets. The runtimes of both algorithms increase as the flow length analyzed increases. The accuracy for both algorithms increases initially before peaking, and eventually decreasing. AdaBoost+J4.8 peaks later than J4.8, due to AdaBoost's additional classifiers. The peak and eventual decline is believed to be due to greater variation per user occurring later in the flow. J4.8 and AdaBoost+J4.8 are both feasible for use in near-real-time devices and, depending on the requirements, either could be preferred.

V. Conclusions

This chapter summarizes the results of the research. Section 5.1 discusses the conclusions based on the results in Chapter 4. Section 5.2 lists the contributions of this research, and Section 5.3 describes the recommendations for future work. Section 5.4 summarizes this chapter.

5.1 Conclusions

The workload for the ML algorithms tested is generated as described in Section 3.3.1. A DPI method is used to automatically classify the web services within encrypted network flows using an optional TLS field. While this method can only classify encrypted flows that use the optional field, it labels a sufficient number of flows to test the effectiveness of the ML algorithms.

Five ML algorithms—Naïve Bayes, NBTree, LibSVM, J4.8, and AdaBoost+J4.8—are tested against flow lengths of 12 packets. J4.8 and AdaBoost+J4.8 perform the best for both accuracy and runtime. Naïve Bayes was not further considered due to its low accuracy. LibSVM and NBTree were not further considered due to their high runtimes. J4.8 and AdaBoost+J4.8 are further tested for flow lengths of 1-20 packets and for the entire flow. They produce statistically-different results for both accuracies and runtimes for flow lengths of 5 or more packets, with J4.8 being faster whenever a difference exists. AdaBoost+J4.8 is statistically more accurate for flow lengths of 9-20 packets.

This research found both J4.8 and AdaBoost+J4.8 to be suitable for near-real-time detection devices. For flow lengths of 10-20 packets, J4.8 classifies over 97% of flows correctly in less than 3 microseconds. For flow lengths of 13-20 packets, AdaBoost+J4.8 classifies over 98% of flows correctly with runtimes under 23 microseconds. Both algorithms produce higher accuracies when not using the entire

traffic flow. If AdaBoost+J4.8 can classify flows fast enough to meet a network's throughput requirements, it is recommended above the other algorithms tested. If the device could be expected to classify in less time, then J4.8 is the recommended algorithm, as the peak accuracy decreases by less than 0.5% while the runtime is less than 11% of AdaBoost+J4.8's.

5.2 Contributions

This research presents two ML algorithms for use in classifying the web service within an encrypted TLS flow and shows ML algorithms can achieve acceptable accuracy in this context. This work furthers the current research in the field by contributing the proven capability to classify applications that utilize TLS to communicate through the Internet. As more applications shift to include web or cloud-based services, this becomes increasingly important. In addition to introducing the ability of ML algorithms to classify web services, this research also expands on the number of labels used in ML-based traffic classification research. It also directly analyzes the performance effects of using AdaBoost in conjunction with J4.8 on web service classification.

This research also analyzes the time necessary to classify an encrypted traffic flow on modern systems. The feature set used, with the two algorithms selected shows excellent results with the ability of a ML algorithm to classify a week's worth of real network TLS traffic in less than 5 seconds using the slower of the algorithms. This speed of classification shows the feasibility of this technique for malware intrusion and data exfiltration prevention in near-real-time network analysis devices.

A new method of dataset generation is also described. By using the TLS optional server name, large amounts of captured network data can be automatically labeled for testing. This new method is also network independent, although the specific labels are expected to change per network. A subset of features used in previous work is defined and a software tool is created that calculates these features within an acceptable delay.

5.3 Future Work

The items listed below are suggestions for future work that could expand on the research presented here.

- **Use Web Service Types as Labels**

Other research classifies flows labeled by application type—i.e., Mail or P2P—instead of web service. Flows such as Google Plus and Facebook could be combined into a social networking label. Labels such as Google IMAP and Yahoo IMAP could be combined into an IMAP label. By attempting to combine labels into broader categories ML algorithms could provide insight into similarities between grouped labels.

- **Test On Other Networks**

As discussed in Chapter 3, performance is expected to change for different networks. It would be useful to test how accurate the same ML algorithms are at classifying web services on other networks. Results could be compared when the ML algorithms are trained on the new test networks and when they are trained on the current network then tested on the new network.

- **Test Changes Over Time**

Networks are dynamic and change over time. The accuracy of the classifiers could change as the networks change. Running the same experiment with data collected from the same network at a later date could provide information on the stability of the results reported in this document and on how the network has changed. Furthermore, using classifiers created during this research to classify flows captured at later dates would provide information on how often the classifiers need updating.

- **Vary Number of Training Samples**

As shown in Appendix A, removing the least common label increases accuracy. Further experiments that vary the number of training samples could find a recommended number of samples to use for creating the ML classifier.

- **Using a MITM DPI Device to Label Flows**

As discussed in Chapter 2, some DPI devices intercept and decrypt encrypted flows to inspect the unencrypted traffic. Using one of these devices to inspect the unencrypted traffic and label the flow, while training and testing ML algorithms on features calculated from the encrypted flow would provide a broader range of labels for classification.

- **Test With More Flow Lengths**

Testing with additional flow lengths could reveal an additional peak in accuracy or a plateau in runtimes for flow lengths greater than 20 packets. While these would require the ML algorithm to wait for more packets before beginning analysis, if there is a dramatic change waiting could be beneficial.

- **Test With Additional Algorithms**

Testing with other ML algorithms or performing parameter tuning on the tested algorithms could produce better results for either the measured accuracy or runtime.

5.4 Summary

This chapter discusses the results and conclusions of the thesis. The contributions to the field are presented, and several suggestions for future work are listed.

Appendix: Additional Experiments

A.1 Timing Experiments

The experiments listed below were run with 10 iterations of 10-fold cross validation. They were run before the final experiments analyzed in Chapter 4. Time Testing is listed in seconds and is the total time it took to classify all testing flows.

A.1.1 Results with Original Features.

Tables A.1 and A.2 show the results of classifying the encrypted flows with flow length analyzed defined as either number of packets or microseconds.

Table A.1: J4.8 Performance with Flow Length as Packets

Packets	Accuracy	Time Training (s)	Time Testing (s)
1	0.2385	3.202	0.0074
2	0.2385	3.371	0.0072
3	0.3483	4.304	0.0069
4	0.3495	7.227	0.0083
5	0.349	8.513	0.0087
6	0.8998	10.78	0.0122
7	0.8999	11.21	0.013
8	0.9168	12.61	0.0141
9	0.925	15.41	0.0151
10	0.9737	13.14	0.0131
11	0.9738	13.31	0.0138
12	0.9783	15.42	0.0147

Table A.2: J4.8 Performance with Flow Length as Microseconds

MicroSeconds	Accuracy	Time Training (s)	Time Testing (s)
1000	0.3206	3.943	0.0076
2000	0.4333	5.512	0.008
3000	0.4344	8.714	0.0098
4000	0.4341	8.372	0.0091
5000	0.9	11.5	0.012
10000	0.9	12.13	0.0136
15000	0.9168	12.81	0.0139
20000	0.9249	15.41	0.0144
25000	0.9735	13.63	0.0133
30000	0.9737	13.57	0.0133
35000	0.9783	15.8	0.0143
40000	0.9794	17.43	0.0169
45000	0.9804	17.65	0.0155
50000	0.9797	19.79	0.0177

A.1.2 Results with Timing Features.

Tables A.4 and A.5 use the features described in Appendix A.1.2, which is Table 3.1 with additional timing features added.

Table A.3: Features with Timing

Feature	Description
1-21*†	Number of bytes in Ethernet packet
22-42*†	Number of bytes in IP packet
43-63*†	Number of bytes in IP and TCP headers
64-65	Number of packets
66-67	Number of packets with TCP ack flag set
68-69	Number of packets with only the ack flag set
70-71	Number of packets with TCP optional SACK Blocks
72-73	Max number of SACK blocks in a single packet
74-75	Number of packets with ack flag set and SACK information
76-77	Number of packets with TCP payloads
78-79	Number of combined bytes within TCP payloads
80-81	Number of packets with the TCP push flag set
82-83	Number of packets with TCP syn flag set
84-85	Number of packets with fin flag set
86-87	Was a packet sent allowing SACK blocks (Value is Y or N)
88-89	Number of packets with TCP urgent flag set
90-91	Number of combined bytes within packets that have urgent flag set
92-102*†	<i>Interarrival time of packets</i>

All values are calculated for both Client to Server and Server to Client

† –Values are also calculated across entire flow

* –Value is collected per packet and the following 7 stats are computed:

Minimum; First Quartile; Mean; Median; Third Quartile; Maximum; Variance

The first three rows have 21 features each: 7 stats for each Client to Server, Server to Client, and entire flow.

Refer to [31] for more information on Selective Acknowledgment (SACK)

Table A.4: J4.8 Performance with Flow Length as Packets and Timing Features

Packets	Accuracy	Time Training (s)	Time Testing (s)
1	0.2403	6.305	0.0085
2	0.6228	16.17	0.0147
3	0.7231	19.36	0.0164
4	0.7185	30.18	0.0183
5	0.7198	29.9	0.0195
6	0.9285	21.03	0.0149
7	0.9279	20.33	0.0151
8	0.9257	23.36	0.0139
9	0.9316	22.88	0.0141
10	0.9717	23.9	0.015
11	0.9717	28.11	0.0177
12	0.9783	28.58	0.0165

Table A.5: J4.8 Performance with Flow Length as Microseconds and Timing Features

MicroSeconds	Accuracy	Time Training (s)	Time Testing (s)
1000	0.323	7.259	0.0089
2000	0.7016	16.67	0.0137
3000	0.7234	24.33	0.0158
4000	0.7185	30.82	0.0192
5000	0.9293	20.65	0.0142
10000	0.9293	20.36	0.015
15000	0.9279	23.2	0.0143
20000	0.9308	27.66	0.0165
25000	0.9718	25.34	0.0166
30000	0.9724	28.92	0.0175
35000	0.9761	28.65	0.016
40000	0.9792	28.47	0.0175
45000	0.9793	28.36	0.0175
50000	0.9793	31.54	0.0191

A.2 Most Common Labels

Tables A.6 and A.7 are created from experiments identical to those in Chapter 4 with the least common label—Google Docs—removed. The accuracy increases for all flow lengths. Sample runs of the experiments show Google Docs to be the label with the lowest accuracy. There are not enough samples to draw definitive conclusions, but it is hypothesized that the lower number of training samples causes the lower accuracy.

Table A.6: Means for Flow Length Comparison J48 No Google Docs

Num Packets	Accuracy	Time Training (s)	Runtime (μs)
1	0.239522	2.5337†	1.181425†
2	0.239522	2.5459†	1.181428
3	0.349784†	3.3846	1.254139
4	0.350958†	6.0295	1.544939
5	0.350503	6.5407†	1.363184
6	0.903283	8.6468	1.872113
7	0.903412	8.794	2.035696†
8	0.920375	10.5857	2.108397
9	0.92838	12.0611	2.344678†
10	0.977182	11.1637	2.126585
11	0.977344	11.069	2.217461
12	0.981686	13.3913	2.39922
13	0.982833	14.9458	2.39922
14	0.98352†	14.1251	2.453733†
15	0.982687	15.2254	2.344685†
16	0.98218	16.3038	2.526447
17	0.98216	17.0873	2.617313
18	0.981984	19.5797	2.744549
19	0.980139	19.1842	2.690033†
20	0.977333	20.6291	2.871792†
All	0.958957	22.7547	2.72638†

† - Indicates the values are not normally distributed

Table A.7: Means for Flow Length Comparison AdaBoost+J48 No Google Docs

Num Packets	Accuracy	Time Training (s)	Runtime (μs)
1	0.239522	2.57†	1.126906
2	0.239522	2.5969†	1.181432
3	0.349784†	3.4163	1.272311
4	0.350958†	6.0868	1.544959
5	0.350503	6.5712	1.599481†
6	0.903975†	91.7281	16.503678
7	0.903981	89.835	16.212858
8	0.921068	102.1627	16.249228†
9	0.929585	98.2398	16.849023
10	0.978202	103.8764	16.958091
11	0.978165	102.3776	17.012623
12	0.982766	122.5157	19.084673
13	0.9844	124.457	19.011976
14	0.985103	125.3094	18.921077
15	0.984647	136.2271	19.557225
16	0.985728	138.5325	20.411523†
17	0.986581	144.2625†	20.375149
18	0.987384†	166.9176	21.956452
19	0.986381†	172.6407	22.592617
20	0.985805	195.7394	23.465041
All	0.972507	232.4322	24.319312†

† - Indicates the values are not normally distributed

A.3 Summary

When the flow length analyzed is defined by number of packets, the accuracy increases faster when timing features are included, but the overall accuracy is not improved. As mentioned in Chapter 3, the timing features were not included so the features would better represent the type of flow, and not the time required to communicate to a particular server. The number of packets from the beginning of the flow is used as the definition for flow length instead of microseconds, due to the inherent timing information included in using a time-based definition. Additional experiments also indicate that increasing the number of training samples may increase the accuracy.

Bibliography

- [1] “Multiple Vulnerabilities in Cisco IronPort Encryption Appliance”, February 10th, 2010. URL <http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20100210-ironport>.
- [2] “Securing Web 2.0 and Social Networking for Enterprise IT”. URL http://www.cisco.com/en/US/prod/collateral/vpndevc/ps10142/ps11720/whitepaper_c11-704647_ps10164_Products_White_Paper.html.
- [3] Akhawe, Devdatta, Bernhard Amann, Matthias Vallentin, and Robin Sommer. “Here’s My Cert, So Trust Me, Maybe? Understanding TLS Errors on the Web”.
- [4] Alshammari, R. and A. N. Zincir-Heywood. “Investigating Two Different Approaches for Encrypted Traffic Classification”. *Privacy, Security and Trust, 2008. PST ’08. Sixth Annual Conference on*, 156–166. 2008.
- [5] Alshammari, R. and A. N. Zincir-Heywood. “Generalization of signatures for SSH encrypted traffic identification”. *Computational Intelligence in Cyber Security, 2009. CICS ’09. IEEE Symposium on*, 167–174. 2009.
- [6] Alshammari, R. and A. N. Zincir-Heywood. “Machine learning based encrypted traffic classification: Identifying SSH and Skype”. *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, 1–8. 2009.
- [7] Alshammari, Riyadh and A. Nur Zincir-Heywood. “Can encrypted traffic be identified without port numbers, IP addresses and payload inspection?” *Computer Networks*, 55(6):1326–1350, 4/25 2011.
- [8] Borders, K. and A. Prakash. “Quantifying Information Leaks in Outbound Web Traffic”. *Security and Privacy, 2009 30th IEEE Symposium on*, 129–140. 2009. ISBN 1081-6011.
- [9] Braun, L., G. Munz, and G. Carle. “Packet sampling for worm and botnet detection in TCP connections”. *Network Operations and Management Symposium (NOMS), 2010 IEEE*, 264–271. 2010. ISBN 1542-1201.
- [10] Campbell-Kelly, Martin. “Historical Reflections: The Rise, Fall, and Resurrection of Software as a Service”. *Communications of the ACM*, 52(5):28–30, 05 2009. URL <http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=39363001&site=ehost-live>.
- [11] Canini, Marco, Wei Li, Martin Zadnik, and Andrew W. Moore. “Experience with high-speed automated application-identification for network-management”.

Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '09, 209–218. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-630-4. URL <http://doi.acm.org/10.1145/1882486.1882539>.

- [12] Chang, Chih-Chung and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. *ACM Trans.Intell.Syst.Technol.*, 2(3):27:1–27:27, may 2011. URL <http://doi.acm.org/10.1145/1961189.1961199>.
- [13] Charalampopoulos, I. and I. Anagnostopoulos. “A Comparable Study Employing WEKA Clustering/Classification Algorithms for Web Page Classification”. *Informatics (PCI), 2011 15th Panhellenic Conference on*, 235–239. 2011.
- [14] Chaudhary, A. and A. Sardana. “Software Based Implementation Methodologies for Deep Packet Inspection”. *Information Science and Applications (ICISA), 2011 International Conference on*, 1–10. 2011.
- [15] Clark, C. R. and D. E. Schimmel. “Scalable pattern matching for high speed networks”. *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 249–257. 2004.
- [16] Constine, Josh. “Facebook Could Slow Down A Tiny Bit As It Starts Switching All Users To Secure HTTPS Connections”, November 8th, 2012.
- [17] Cristianini, Nello and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.
- [18] Cusumano, Michael. “Cloud computing and SaaS as new computing platforms”. *Communications of the ACM*, 53(4):27–29, 2010.
- [19] Dainotti, A., A. Pescapé, and K. C. Claffy. “Issues and future directions in traffic classification”. *Network, IEEE*, 26(1):35–40, 2012.
- [20] Degabriele, J. P. and K. G. Paterson. “Attacking the IPsec Standards in Encryption-only Configurations”. *Security and Privacy, 2007. SP '07. IEEE Symposium on*, 335–349. 2007. ISBN 1081-6011.
- [21] Dierks, T. and E. Rescorla. “The Transport Layer Security (TLS) Protocol Version 1.2”. RFC 5246 (Proposed Standard), August 2008. URL <http://www.ietf.org/rfc/rfc5246.txt>. Updated by RFCs 5746, 5878, 6176.
- [22] Dyer, Kevin P., S. E. Coull, T. Ristenpart, and T. Shrimpton. *Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail*, 332; 332–346; 346. 2012 IEEE Symposium on Security and Privacy. IEEE, -05 2012. ISBN 978-1-4673-1244-8.
- [23] Dyer, Kevin P., Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. *Format-Transforming Encryption: More than Meets the DPI*, 2012.

- [24] Eastlake 3rd, D. “Transport Layer Security (TLS) Extensions: Extension Definitions”. RFC 6066 (Proposed Standard), January 2011. URL <http://www.ietf.org/rfc/rfc6066.txt>.
- [25] Goss, R. and R. Botha. “Deep packet inspection Fear of the unknown”. *Information Security for South Africa (ISSA)*, 2010, 1–5. 2010.
- [26] Hirvonen, M. and M. Sillio. “Two-phased method for identifying SSH encrypted application flows”. *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, 1033–1038. 2011.
- [27] Hornig, C. “A Standard for the Transmission of IP Datagrams over Ethernet Networks”. RFC 894 (Standard), April 1984. URL <http://www.ietf.org/rfc/rfc894.txt>.
- [28] Kurose, James F. and Keith W. Ross. *Computer networking*. Addison Wesley, 4th edition, 2008.
- [29] Li, Wei and A. W. Moore. “A Machine Learning Approach for Efficient Traffic Classification”. *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS '07. 15th International Symposium on*, 310–317. 2007. ISBN 1526-7539.
- [30] Liu, Yali, C. Corbett, Ken Chiang, R. Archibald, B. Mukherjee, and D. Ghosal. “SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack”. *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, 1–10. 2009. ISBN 1530-1605.
- [31] Mathis, M., J. Mahdavi, S. Floyd, and A. Romanow. “TCP Selective Acknowledgment Options”. RFC 2018 (Proposed Standard), October 1996. URL <http://www.ietf.org/rfc/rfc2018.txt>.
- [32] Moore, A. W. “Internet traffic classification using Bayesian analysis techniques”. *Performance evaluation review*, volume 33, 50–60. Association for Computing Machinery, 2005. ISBN 0163-59990163-5999.
- [33] Moore, Andrew W., Denis Zuev, and Michael Crogan. *Discriminators for use in flow-based classification*. Technical Report RR-05-13, Department of Computer Science, Queen Mary, University of London, August 2005.
- [34] Morehart, Ryan A. “Evaluating the Effectiveness of IP Hopping via an Address Routing Gateway”, March 2013. DTIC Document.
- [35] Nguyen, T. T. T. and G. Armitage. “A survey of techniques for internet traffic classification using machine learning”. *Communications Surveys and Tutorials, IEEE*, 10(4):56–76, 2008.

- [36] Nguyen, T. T. T., G. Armitage, P. Branch, and S. Zander. “Timely and Continuous Machine-Learning-Based Classification for Interactive IP Traffic”. *Networking, IEEE/ACM Transactions on*, PP(99):1–1, 2012.
- [37] Nourani, M. and P. Katta. “Bloom Filter Accelerator for String Matching”. *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 185–190. 2007. ISBN 1095-2055.
- [38] Okada, Y., S. Ata, N. Nakamura, Y. Nakahira, and I. Oka. “Application identification from encrypted traffic based on characteristic changes by encryption”. *Communications Quality and Reliability (CQR), 2011 IEEE International Workshop Technical Committee on*, 1–6. 2011.
- [39] Postel, J. “Internet Protocol”. RFC 791 (Standard), September 1981. URL <http://www.ietf.org/rfc/rfc791.txt>. Updated by RFC 1349.
- [40] Postel, J. “Transmission Control Protocol”. RFC 793 (Standard), September 1981. URL <http://www.ietf.org/rfc/rfc793.txt>. Updated by RFCs 1122, 3168, 6093, 6528.
- [41] Quinlan, J. Ross. *C4.5 Programs for Machine Learning*. Morgan Kaufman, San Mateo, California, 1993.
- [42] Russell, Stuart Jonathan, Peter Norvig, Ernest Davis, Stuart Jonathan Russell, and Stuart Jonathan Russell. *Artificial Intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ, 3rd edition, 2010.
- [43] Schrader, Karl R. “An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols”, 2009. DTIC Document.
- [44] Sullivan, Danny. “Google To Begin Encrypting Searches and Outbound Clicks By Default With SSL Search”, Oct 18, 2011 2011.
- [45] Thomas, Brennon D. “Performance Evaluation of a Field Programmable Gate Array-Based System for Detecting and Tracking Peer-to-Peer Protocols on a Gigabit Ethernet Network”, 2010. DTIC Document.
- [46] Turner, S. and T. Polk. “Prohibiting Secure Sockets Layer (SSL) Version 2.0”. RFC 6176 (Proposed Standard), March 2011. URL <http://www.ietf.org/rfc/rfc6176.txt>.
- [47] Williams, N. “A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification”. *Computer communication review*, 36(5):7–15, 2006.
- [48] Witten, Ian H., Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Burlington, MA, 3rd edition, 2011.

- [49] Wright, Charles. “HMM profiles for network traffic classification (extended abstract)”. *VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, 9–15, 2004.
- [50] Wright, Charles V., Fabian Monroe, and Gerald M. Masson. “On Inferring Application Protocol Behaviors in Encrypted Network Traffic”. *Journal of Machine Learning Research*, 7(12):2745–2769, 12 2006. URL <http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=23757985&site=ehost-live>.
- [51] Wright, CV, SE Coull, and F. Monroe. “Traffic morphing: An Efficient Defense against Statistical Trac Analysis”. *Proceedings of the Network and Distributed Security Symposium-NDSS*. 2009.
- [52] Yu, Hua, Rong Cong, Luying Chen, and Zhenming Lei. “Blocking pornographic, illegal websites by internet host domain using FPGA and Bloom Filter”. *Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on*, 619–623. 2010.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 13-06-2013		2. REPORT TYPE Master's Thesis			3. DATES COVERED (From — To) Oct 2011–June 2013	
4. TITLE AND SUBTITLE Classification of Encrypted Web Traffic Using Machine Learning Algorithms				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Barto, William Charles				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB, OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-13-J-11	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Greg Sullivan - CEO Global Velocity Inc. 222 S. Central Ave. Suite 400 St. Louis, MO 63105					10. SPONSOR/MONITOR'S ACRONYM(S) GV	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED						
13. SUPPLEMENTARY NOTES This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT The increasing usage of web services and encrypted network communication makes the network analysis of encrypted web traffic of utmost importance. This research evaluates the feasibility of using ML algorithms to classify web services within encrypted TLS flows. The ML algorithms are compared primarily based on classification accuracy. The runtimes of the classifiers are also considered, as classifiers must be able determine labels quickly in order to be used in near real-time network protection devices. Five ML algorithms are initially considered when analyzing only the first 12 packets: Naïve Bayes, NBTree, LibSVM, J4.8, and AdaBoost+J4.8. AdaBoost+J4.8 and J4.8 produce the best accuracies and runtimes and are tested on flow lengths of 1-20 packets. J4.8 reaches a peak accuracy of 97.99% at 14 packets. AdaBoost+J4.8 peaks later at 18 packets with 98.41% accuracy. AdaBoost+J4.8 requires 21.55 microseconds to classify a single flow at peak accuracy, while J4.8 requires only 2.37 microseconds to classify at peak accuracy. The quick runtimes and high accuracies of the J4.8 and AdaBoost+J4.8 indicate that these ML algorithms are good choices for near real-time classification of web services within an encrypted TLS flow.						
15. SUBJECT TERMS Networks, Machine Learning (ML), Encrypted Traffic, Traffic Classification						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU		18. NUMBER OF PAGES 88	
a. REPORT	b. ABSTRACT	c. THIS PAGE				
U	U	U	19a. NAME OF RESPONSIBLE PERSON Rusty Baldwin, PhD			
				19b. TELEPHONE NUMBER (include area code) (937) 255-6565 ext.4445, rusty.baldwin@afit.edu		