AN INVESTIGATION OF USING MACHINE LEARNING WITH DISTRIBUTION BASED FLOW FEATURES FOR CLASSIFYING SSL ENCRYPTED NETWORK TRAFFIC

by

Daniel Arndt

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2012

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "AN INVESTIGATION OF USING MACHINE LEARNING WITH DISTRIBUTION BASED FLOW FEATURES FOR CLASSIFYING SSL ENCRYPTED NETWORK TRAFFIC" by Daniel Arndt in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: August 13, 2012

Supervisor: A. N. Zincir-Heywood

Readers: M. I. Heywood

S. Sampalli

DALHOUSIE UNIVERSITY

DATE: August 13, 2012

AUTHOR: Daniel Arndt

TITLE: AN INVESTIGATION OF USING MACHINE LEARNING WITH DISTRIBUTION BASED FLOW FEATURES FOR CLASSIFYING SSL ENCRYPTED NETWORK TRAFFIC

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc. CONVOCATION: October YEAR: 2012

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

Encrypted protocols, such as Secure Socket Layer (SSL), are becoming more prevalent because of the growing use of e-commerce, anonymity services, gaming and Peer-to-Peer (P2P) applications such as Skype and Gtalk. The objective of this work is two-fold. First, an investigation is provided into the identification of web browsing behaviour in SSL tunnels. To this end, C5.0, naive Bayesian, AdaBoost and Genetic Programming learning models are evaluated under training and test conditions from a network traffic capture. In these experiments flow based features are employed without using Internet Protocol (IP) addresses, source/destination ports or payload information. Results indicate that it is possible to identify web browsing behaviour in SSL encrypted tunnels. Test performance of ˜95% detection rate and ˜2% false positive rate is achieved with a C5.0 model for identifying SSL. ˜98% detection rate and ˜3% false positive rate is achieved with an AdaBoost model for identifying web browsing within these tunnels. Second, the identifying characteristics of SSL traffic are investigated, whereby a new tool is introduced to generate new flow statistics that focus on presenting the features in a unique way, using bins to represent distributions of measurements. These new features are tested using the best performers from previous experiments, C5.0 and AdaBoost, and increase detection rates by up to 32.40%, and lower false positive rates by as much as 54.73% on data sets that contain traffic from a different network than the training set was captured on. Furthermore, the new feature set out-preforms the old feature set in every case.

Acknowledgements

First and foremost, I'd like to thank Dr. Nur Zincir-Heywood, and Dr. Heywood for putting up with me as long as you did. You have both helped me to become a much more knowledgeable person, and I'm all the better for that. Thank-you for your countless pieces of advice, encouragement, and dedication to my success.

I'd like to thank anyone who has spent time reviewing my work, and provided their feedback. A special thanks goes out to those anonymous reviewers who will never get the credit that is due. A big thank-you to Jazz for being someone I could bounce ideas off of, and for pretending to listen to me as I ramble about whatever the frustration of that day may be.

Thanks to all the free and open source developers out there. The tools provided and the support given, not just in this thesis – but throughout my life – have been unimaginably invaluable.

Of course, none of this would be possible without my parents. Your guidance and support has affected every aspect of my life, even my research. Thank-you for all the years you dedicated to putting me on the right track, and for helping me out as I tried to figure out what to do with my life. Thanks for the Amiga, and the P5 that first ignited my interest in computers.

Finally, my three brothers all deserve a big acknowledgement. Each one of you has been there to keep me in line and push me to do my best. You three provide the perfect balance of love, criticism, humour, and acknowledgement to my life, among other things. You always keep me thinking critically when I should be, and make me laugh when I need to relax.

Chapter 1

Introduction

When the Internet was first born, it was most widely used as a way to share academic research, resources, and communications. In these very early days, security was not a large concern. Most protocols introduced as the Internet began to pick up speed – such as HTTP, SMTP, FTP, and Telnet – did not provide encryption techniques. As the popularity of the Internet increased, network traffic identification became a specific point of interest for administrators. Network traffic identification is important for traffic engineering, network reliability, security, and quality of service. In the early days, when the Internet was a relatively new technology, one could simply identify traffic by the port numbers being used. Applications generally conformed to the values assigned by the Internet Assigned Numbers Authority [1] (IANA). However, identifying what applications are running on a network has become increasingly difficult. The amount of applications that run over the Internet has increased, and non-standard port usage has become common. Sometimes, non-standard ports are even used for protocols that already have a port assigned to them. For example, although port 22 has been assigned for SSH, system administrators will often configure the port to listen on another port instead. This is often done in an attempt to increase security, by avoiding probes for an SSH server on the port. One work notes that in one of their studies, over 100,000 attacks were received on the default SSH port, while none were received on SSH servers listening on non-standard ports [2]. Furthermore, dynamic port usage has become popular in many applications, especially in domains such as P2P [3, 4, 5].

Deep packet inspection (DPI) became the solution to non-standard port usage. With DPI, network administrators could once again identify which particular applications were being used. DPI works by analyzing the payload of the information sent over the physical wire. In other words, the algorithms that are employed read the messages being sent back and forth to determine what applications or protocols

might be running. This method is extremely effective for applications that do not use encryption methods. DPI is inherently computationally expensive, however, as it requires reading the contents of the packets and comparing them to application signatures. Furthermore, because DPI relies on the contents of the packet, encrypted packets are immune to this sort of identification. Security and privacy concerns have led to an increased use in encryption methods. The May 2011 Palo Alto Networks' Application Usage and Risk Report found SSL now accounts for about 23% of the overall bandwidth in the 28 exabytes of data analyzed, and predicts the amount will continue to increase [6]. The report also states that in some cases, SSL is used purely as a means to evade detection.

Encryption allows a user to hide the underlying message of an application from eavesdroppers. Some of these encryption methods perform a plain text initiation of the protocol (such as SSH - Secure Shell, or SSL - Secure Sockets Layer), which allows deep packet inspection to identify the encryption method being used. However, if another application is being tunneled through such a connection, deep packet inspection cannot be used to identify this "hidden" application due to the encryption.

In this thesis, the objective is to detect behavioural models in the SSL tunnels to identify, specifically, web browsing behaviour in a given traffic capture. To this end, C5.0, naive Bayesian, AdaBoost and Genetic Programming (GP) learning models are evaluated under many training and test conditions to find the most suitable method for automatically generating a signature to recognize such behaviour. In these experiments, flow-based features are employed without using the IP addresses, source/destination ports or payload information. Furthermore, this research introduces a new approach to flow features, by considering the distribution of packets within the flow and by investigating how much effect this has (if at all) on the performance of such classifiers to identify different behaviours in SSL tunnels.

The rest of this thesis provides an in-depth investigation into the above stated goal. A literature review including definitions of SSL, TLS, and related work is given in Chapter 2. Next, a technical background is given of both network traffic (including encryption, and flows) and the machine learning techniques used in Chapter 3. The data collection and data generation methods are discussed in Chapter 4. Experiments and results are outlined in Chapter 5. The need for a new set of features is discussed

in Chapter 6. A new tool for generating flow statistics is outlined in Chapter 7. Next, a new set of experiments and their results are given in Chapter 8. Finally, conclusions are given and potential future work is discussed in Chapter 9.

Chapter 2

Literature Review

This chapter summarizes the literature that covers the areas of interest in this thesis. Section 2.1 covers literature that has defined the SSL and TLS protocols. Section 2.2 explains the scenarios in which SSL is being used today. Then, the chapter is finished with a review of related research in Section 2.3.

2.1 History of the SSL/TLS Protocol

The purpose of SSL is to apply the following goals to reliable data communications: data confidentiality, data integrity, authentication, and non-repudiation. By providing a protocol designed to offer these goals, SSL can be used to prevent many network based attacks such as eavesdropping, IP spoofing, and connection hijacking [7]. In this document, with the exception of this section, the abbreviation SSL (Secure Sockets Layer) is used as a shorthand when referring to both SSL and TLS (Transport Layer Security) for which a history is given in the following.

SSL The Secure Sockets Layer, or SSL, was designed by Netscape Communications. The protocol was developed due to the growing concerns of security on the World Wide Web, and was an attempt to provide secure communications over the Internet. SSL was designed to use cryptography to address these concerns and included important features, allowing it to achieve all the goals of cryptography, such as signed server certificates. Furthermore, SSL included the ability to define the available cipher algorithms allowing it to comply with varying cryptography laws around the world. SSL was cleverly implemented at the application level, allowing the protocol to be used not only for secure HTTP transactions, but any reliable transport communications over a network. SSL was released to the public in 1995 as version 2.0. SSL version 1.0 was never made public. The original design of SSL unfortunately contained some security flaws [8]. This led to the entire redesign of SSL, which came soon after in

1996, bringing the protocol to version 3.0. By this point, the Internet was in need of a standard, and so TLS was born.

TLS TLS (Transport Layer Security) was developed using SSL 3.0 as a standard to build on. The protocol is defined in RFC 2246, and was developed by the Internet Engineering Task Force (IETF) Network Working Group [9]. TLS 1.0 is defined to use the version value 3.1 and explains the historical significance of this as "TLS version 1.0 is a minor modification to the SSL 3.0 protocol, which bears the version value 3.0" [9]. For this reason, TLS is often referred to as SSL 3.1. TLS version 1.0 was released in 1999 and quickly became the standard for providing privacy over the Internet. The protocol even contained the ability to downgrade the connection to SSL 3.0, which helped in its adoption, and included several minor changes, including the following [10]:

- The SSL Message Authentication Code algorithm was replaced by the Hashing for Message Authentication Code (HMAC) algorithm.
- Certificates all the way back to the root CA are not always needed. Intermediate authorities are permitted.
- The Fortezza algorithms are not included due to their closed source nature.

In 1995, Phillip Rogaway drafted his comments about problems with cipher block chaining in the IPSec protocol [11]. These comments prompted the realization that these same problems mentioned by Rogaway also existed in the TLS 1.0 protocol when used in the cipher block chaining mode [12]. Later, further problems were also found with the cipher block chaining mode [12], and the protocol was finally updated to TLS 1.1 (SSL 3.2) with RFC 4346 in April 2006 [13]. The new protocol contained minor security enhancements.

The TLS protocol was updated one more time to version 1.2 (SSL 3.3) in August 2008. This version, amoung other changes, removed the reliance on the recently broken MD5-SHA1 for the pseudo-random function and finished message hashes. The algorithm combination was replaced with SHA-256, and options were enabled to allow cipher-suite specific hashing algorithms and pseudo-random functions [14].

2.2 Different Use Cases of SSL

Since SSL operates between the transport layer and application layer, it can be used to secure any application that utilizes a reliable transport protocol such as the Transmission Control Protocol – TCP. Further details of the transport layer can be found in Section 3.1. The protocol works as a tunnel, which can be used to encapsulate any message that is possible over standard TCP/IP communications. This feature of SSL makes it one of the most useful protocols for encryption, and makes it difficult for an observer to identify the underlying application. Deep packet inspection provides network administrators with the ability to detect SSL, but since DPI involves inspecting the packets contents – and the contents messages from the application running under SSL are encrypted – it will not give the administrator any insight into what the SSL tunnel is being used for.

Most commonly, SSL is known for its use in secure communications between a web browser and a web server. This is exactly the case when you do e-commerce such as shopping, or any sort of on-line banking. It is also common for private browsing communications such as e-mail [10, 15], and has become popular in other on-line applications such as voice over IP (VoIP) [16, 17], and VPN connections [18]. The extensive use of SSL has resulted from its flexibility and relatively low overhead. A Google employee cited (with reference to Gmail):

> "On our production front-end machines, SSL/TLS accounts for less than 1% of the CPU load, less than 10KB of memory per connection and less than 2% of network overhead." [15]

Modifications of SSL even exist so that it may be used over other protocols such as User Datagram Protocol (UDP). Since SSL does not have any specific application requirements to be used, the possibility of any application running within it makes it very difficult to identify what its particular application usage is on a network.

2.3 Related Work

Research on application identification in network traffic has been a focus for several institutions for quite some time. Before the turn of the century, signature based

application identification programs began to show up in literature such as Bro [19] and Snort [20]. It has become widely accepted among researchers that port based identification is not a viable approach to application identification [3, 4, 21, 22, 23, 24, 25, 26, 27]. Port based identification is limited in its ability due to dynamic port usage, such as those used by P2P applications [3, 4], and port masquerading [24, 27]. The programs mentioned above rely on well known signatures, instead of ports, to identify the applications. Therefore, these programs are able to detect applications that use dynamic ports, non-standard ports, or port masquerading.

However, work by Moore et al. [21] acknowledges that tunneling is used by those aiming to evade port based and even signature based application detection and proposes a deep packet inspection solution. Furthermore, it has been acknowledged by several researchers that signature based identification, even deep packet inspection, is either no longer a viable solution, or is increasingly less effective [22, 23, 24, 25, 26, 27, 28]. The reasons these works present are many, including high CPU cost of deep packet inspection, privacy concerns and legal issues, encryption thwarting their effectiveness, and the daunting task of maintaining a database of application signatures.

Early work by Williams et al. [4] identifies the need for classification techniques to be considered in terms of their computational performance, not just their accuracy, and therefore supports the notion to move away from DPI techniques. Wright et al. [29] take a deeper look into tunnels that contain multiple applications. This is possibly the first work that attempts to address these concerns. Following the emerging idea of using packet-level statistics such as packet size, timing, and direction, this research investigates identifying the number of applications running within a tunnel to 20% accuracy. Furthermore, when a single application is being tunneled, Wright et al. are able to identify the program with up to 80% accuracy using these simple measurements.

Yamada et al. [30] do make an attempt to focus on SSL encrypted communications. Perhaps one of the earliest studies that makes this a particular focus, Yamada et al. use IP addresses to help identify web clients, and considers the notion of *activities* within a communication instead of a network flow. These activities represent an action such as clicking on a button and waiting for a page to load. The system

makes some use of DPI by decoding the SSL protocol in an attempt to estimate actual volume transfer. However, their system focuses on intrusion detection instead of classification.

In the same year, Bernaille et al. [31] studied SSL, and was possibly the first to involve real SSL traffic for identification. Bernaille et al. only use packet lengths, and only considers the first few packets in identifying the underlying applications for SSL traffic. Since the first few packets are used, this methodology is not reasonable when tunnels are used for multiple applications within a single tunnel. Furthermore, this approach is not scalable when the compression features of SSL are used. Crotti et al. [32, 33] used a similar method, whereby statistically-based classification techniques were used to detect applications using what they call application *fingerprints*. Using probability distribution functions, they were able to achieve promising results, however, these results again relied on packet ordering and were susceptible to the same problems as Bernaille et al. [31]. Later, other research tackled the tunneling problem but focused on SSH tunnels [28, 34]. These approaches used fingerprinting techniques again to detect non-standard applications running within tunnels, but were not used to identify the applications explicitly and focused more on the intrusion detection front.

Erman et al. [22] also approach the classification problem using flow based classifiers, but introduce the use of semi-supervised techniques to get around the fact that labeled traffic is hard to come by. These techniques can use unlabeled traffic to increase their accuracy. In response to concerns about flow based techniques being unreasonable for real-time classification, Erman et al. use a layered approach, allowing their classifiers to make best guesses at particular stages in a flow's life-cycle.

At this point, NetMate and other flow based techniques were becoming more popular [4, 22, 29, 31]. However, these studies often ignore encryption as a major case [4, 22], or simulate encryption [29]. When encryption is made a focus, their techniques do not take into account multiple applications within a tunnel or compression techniques [31].

Researchers began to acknowledge the importance and rising usage of SSL. Many reasons are cited for SSL's increased adoption such as the rise of e-commerce and privacy or authentication concerns. On the flip side of identification, some began

developing techniques to obfuscate traffic within these tunnels [35, 36]. These works altered the timing and data volumes to avoid leakage of information.

Newer research points towards the possibility that different applications may each benefit from different techniques, parameters, or tuning of training data [23]. Although research continued in the area of fine tuning techniques [25, 26, 27, 37, 38, 39, 40, 41], some research began to combine techniques [42, 43, 44, 45, 46, 47] with some promising results. Unfortunately, these works can complicate the process, especially in training or configuration [38, 42]. Perhaps some of the most interesting techniques are those that combine DPI or signatures with packet-level statistics [45].

Flow based techniques continue to be a focus in recent research [27, 43, 44, 45, 48, 49, 50, 51], and although encryption is becoming a more popular focus, SSL is seldom the focus [43, 44, 48, 49, 50].

Two recent and significant works in identifying SSL and their underlying applications are the flow-based machine learning approaches used by McCarthy et al. [51] and the use of a hybrid method of signatures and flows statistics by Sun et al. [45]. The work by McCarthy et al. [51] uses an entirely generated data set to set up the theoretical foundation for SSL classification using these methods, and Sun et al. [45] were able to achieve promising SSL detection results on their experiments using data from a network monitor.

In this thesis, novel contributions include an investigation of SSL identification and tunneled applications using only packet-level statistics (ie. flows) on real traffic. Furthermore, this work introduces the use of a genetic program and tests its performance in the domain of SSL classification. Initial results of the genetic program are not great, but lead to further discoveries into the importance of the features used for identifying SSL. This work provides a novel investigation into the features that best help identify SSL traffic, and further extend researcher's capabilities by introducing a new flow statistic generating tool. Unlike many previous works [27, 45, 46], this thesis does not neglect to include both same-network and robustness tests, which become an integral part of discovering SSL's identifying characteristics.

Chapter 3

Background

3.1 Overview of Network Traffic

Network traffic contains considerably more information than the general user is aware of. To most users, the Internet and other networks "just work" and that is the most they will ever care to know. However, behind the scenes, many things must be calculated and transmitted on the physical wire to allow a device to communicate with other devices. Network traffic is generally conceptualized in the form of a layered model. A layered model describes the many different headers and encapsulations used by different computer networking technologies, both hardware and software. The TCP/IP protocol stack, which is the Five-Layer Model (Figure 3.1), is one common model used to describe the Internet.

In this model, Layer-1 is the *Physical Layer*, which is the layer many people are actually aware of. This is because it involves the physical hardware that connects the devices on the network. The Physical Layer is often described as the means by which bits can be sent over a network. Layer-1 differs from the other layers because there is no concept of a *packet* at this point. A packet consists of the next four layers in the Five-Layer model.

Layer-2 is the *Data Link Layer*. The Data Link Layer is used to allow two interfaces on the same link (often no more than a wire) to send messages back and forth. The Data Link Layer is often implemented by a software driver for an ethernet card, which tells the interface how it needs to "wrap" the data frame in order to communicate over the wire. It includes things such as the Media Access Control (MAC) addresses of the source and destination interfaces as well as the type of network frame being wrapped (most commonly, *Internet Protocol*). Since the Data Link Layer is dependent on the structure of the network, it provides the necessary instructions for transmitting a frame over the physical wire.

The *Network Layer*, which is Layer-3, is often called the *Internet Layer*. It provides

the network addresses required for correctly routing packets in the right direction. This layer is directly responsible for enabling your ISP to know where to send your request when visiting a web page, or how to connect to a FTP server, and does so by attaching a small header to the data frame. The Internet Protcol (IP) works on this layer. By defining a a common address space (ie, the IP address), the Internet Protocol is able to let different network types to be interconnected and transmit messages between them. This is also the layer that allows the Internet to work on almost any type of network.

Next, the data is encapsulated with another header, this time called the *Transport Layer*, Layer-4. The transport layer header is designed to provide "end to end" message transfer capabilities. In other words, this layer is responsible for ensuring the message is forwarded to the correct application on an end system. The header will include information such as the source and destination port numbers. Furthermore, the transport layer may also be responsible for error control, segmentation, flow control, congestion control, or port numbers depending on the Layer-4 protocol used. Common transport layer protocols include Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Finally, a packet begins and ends its journey as a simple application message. The data at this point is called the *Application Layer* message. It is nothing more than the message (or part thereof) that an application wishes to send across a network. This is also the layer that SSL takes advantage of in providing secure communications, although SSL can also be viewed as a process that creates its own layer before the Application Layer. Some examples of common protocols that run on this layer and their descriptions are given in Table 3.1.

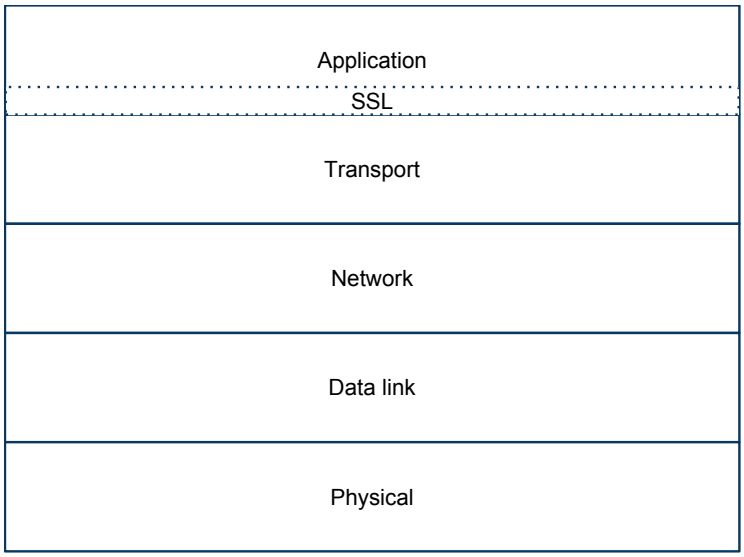

Figure 3.1: Internet - Five Layer Network Model

Table 3.1: Common Application Protocols

Protocol	Encryption	Description
HTTP	none	Most well known for its use in data communtication for the world wide web.
HTTPS	SSL	The same protocol as HTTP, but encrypted with an SSL tunnel. Commonly used for banking, e-mail, and other confidential transmissions.
FTP	none	A protocol for transferring files.
FTPS	SSL	The FTP protocol encrypted with an SSL tunnel.
IMAP	none	Used for accessing e-mail.
Secure IMAP	SSL	The IMAP protocol encrypted with an SSL tunnel.

3.1.1 Technical Details of SSL

The primary focus of network traffic in this thesis is on Secure Sockets Layer (SSL) traffic. As mentioned in Section 2.1, the term SSL is used in this document to refer to both SSL and TLS traffic. The reason for this is that TLS is largely considered a standardization of SSL, and shares many features with it. SSL uses *cryptography* to provide security to any application that runs on a reliable transport protocol. Cryptography is the study of techniques for secure communications. Repeating from

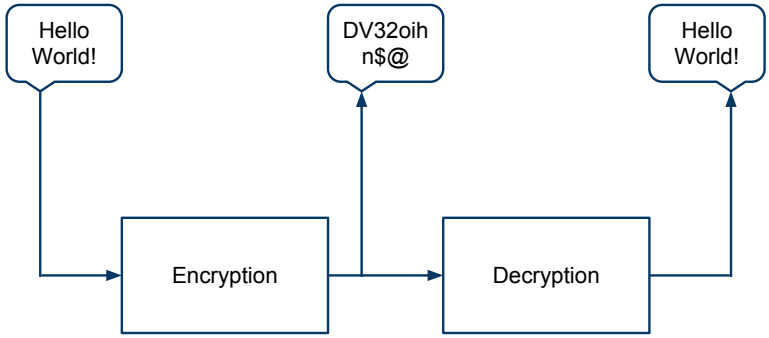

Figure 3.2: Encryption - An example

Section 2.1, SSL attempts to provide data confidentiality, data integrity, authentication, and non-repudiation to these connections through the use of a standardized protocol. A key part to providing all these goals is the use of *encryption*.

Encryption is the transformation of a message (called the *plain-text*) to a message that is unreadable by those who do not have the necessary knowledge to decrypt the message (called the *cipher-text*). An eavesdropper can easily interpret the contents of plaint-text messages being sent between computers over a medium such as the Internet, but when encryption is used, the usefulness of this practice is thwarted. In the case where encryption is used, the eavesdropper will only be able to read the cipher-text, which does not give them the application message unless they know how to decrypt it (Figure 3.2). It is usually ensured (to the best of ability) that only the intended recipient(s) are able to decrypt the message. It is important to note here that since SSL is used at the application layer, the headers for the previous layers (in particular, the transport layer header containing packet size and TCP flags) are left unencrypted. This is important and becomes the basis for gathering information about encrypted communications for this thesis.

Though the name Transport Layer Security is somewhat misleading, SSL/TLS run at the application layer, and encrypt everything above the transport layer. This effectively creates a new layer, the SSL layer, within the application layer (illustrated with a dotted border in Figure 3.1). By running as a "virtual" layer between the transport layer and the application layer, the SSL protocol has independence from the underlying application protocol. This gives SSL the ability to be used with any

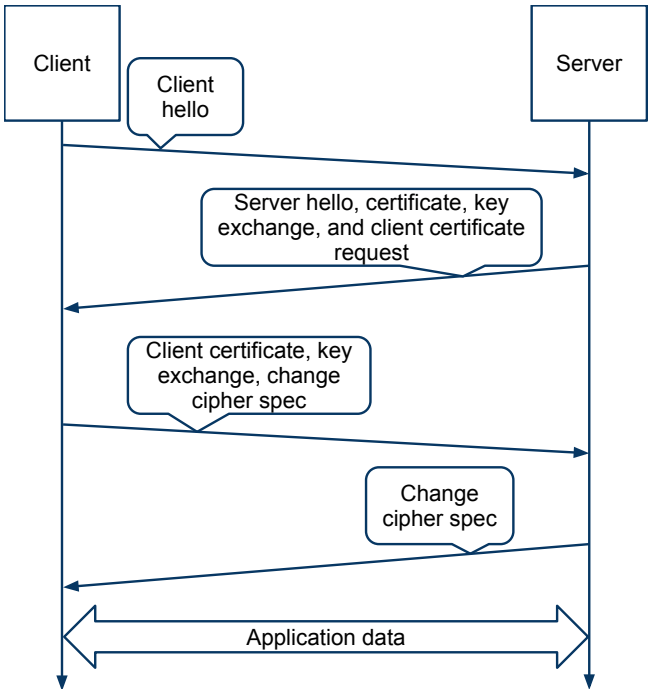

Figure 3.3: Simple SSL handshake

application[1].

Figure 3.3 shows the basic interactions involved in a simple SSL handshake. This represents the communications that occur before a client can communicate securely with the server. The figure shows the first several packets of an SSL tunneled communication and is roughly the same for any application that implements it. The protocol communication is initiated in plain text. The simple SSL handshake is typically used for banking and other scenarios where the client is not authenticated with a certificate. The client will usually be required to identify themselves with a username and password, however, this is not part of the SSL protocol and instead is something that is tunneled through SSL at the application layer. The handshake begins with the client sending a message to the server indicating that it would like to begin a secure session with SSL. This is called the client "hello", and contains information such as supported protocol versions (ie. 3.0 for SSL, or 3.1+ for TLS) and the encryption algorithms (*ciphers*) available. The server will then respond with a server "hello" which includes a chosen encryption method (that it deems strongest), protocol information,

[1] Though it is possible to tweak SSL to work on applications that use protocols other than TCP, the standard protocol only allows the use of reliable protocols such as TCP.

and also includes its signed certificate. The certificate is the server's proof that they are who they say they are, and is often verified by a third party (called a *Certificate Authority*, or CA). The CA will verify the server's certificate (which contains the server's public key) by signing the certificate with their private key. The client then validates this certificate using the public key of the CA, which is stored locally by the client for CAs that it trusts. Now, the two devices can securely communicate a key exchange with the server using the server's public key, with which the client can transfer the *secret key* (essentially a random number), which then will be used as the initialization vector for the encryption method chosen.

3.1.2 Flows

Network traffic flows – or just *flows* for short – are used extensively in this work to provide measurements about the communications over a network. A flow is an artificial concept, and is used to encapsulate the idea of connectedness between two devices. Flows are defined in RFC 2722 [52] where they are explained as:

> "an artificial logical equivalent to a call or connection. [...] a portion of traffic, delimited by a start and stop time."

RFC 2724 [53] gives a more detailed description as:

> "a set of packets between two endpoints (as defined by their source and destination attribute values and start and end times), and as bi-directional."

Within this document, the notion of a flow is described as identifiable by a 7-tuple:

$$(IP_{source}, PORT_{source}, IP_{destination}, PORT_{destination}, t_{start}, t_{end}, PROTO)$$

where IP_x is the ip address of x, $PORT_x$ is a port used by x, t_x is the time at x, and $PROTO$ is the protocol used. A flow is bi-directional, and therefore must have at least one packet in each direction to be considered a valid flow. Only the UDP and TCP protocols are considered since most applications on the Internet use one of these two protocols. UDP flows end after a timeout of 600 seconds, ie. when no packet has been sent or received on the $(IP_{source}, PORT_{source}, IP_{destination}, PORT_{destination}, PROTO)$ combination for 600 seconds. TCP flows require a successful TCP connection to be

considered a valid flow, and the flow ends after a successful TCP connection teardown or 600 second timeout. The 600 second timeout was chosen because it is the suggested timeout value for inactive flows in RFC 2720 [54].

3.2 Machine Learning

Machine learning is an application of artificial intelligence that uses algorithms to enable a computer to learn from input data. The data given to a machine learning algorithm is used to generate an understanding of the data, based on what the algorithm can "learn" from the input it is given. The goal for any machine learning technique is to identify patterns (often very complex ones) in large amounts of data, patterns that even a very skilled human would not be able to recognize due to the enormous amount of information that is often presented. These techniques take advantage of a computer's ability to process massive amounts of data quickly and efficiently. In this work, machine learning is used to provide *classification*. In classification, the data is given to a machine learning algorithm as multiple *instances*, each consisting of a set of values. These values can take multiple forms, including strings (a sequence of characters, such as a word), an integer, a real number, or a selection of one of multiple discrete values (such as true or false). Each one of these values is called an *attribute*, or a *feature* (both terms are used interchangeably in the literature).

The formation of this understanding is done in what is called the *training* phase, and is used to form a *model*. The model can then be used to identify new cases in the *testing* phase. The goal is to identify to which *class* (also called the *label*) each instance in the test set belongs to. This thesis investigates the training and testing of four machine learning approaches – naive Bayes, C5.0, AdaBoost with J48, and Symbiotic Bid-Based Genetic Programming – in order to identify SSL traffic. Furthermore, these same techniques are also tested for their ability to identify applications, specifically web browsing, running within SSL tunnels from a given network traffic trace.

3.2.1 Naive Bayes

The naive Bayes machine learning technique forms a statistical model of the data that is given in the training phase. The algorithm relates each feature to the probability that feature will result in a particular outcome based on the entire training set. To preform testing, the probability of each possible outcome is calculated based on the features each test instance has. Naive Bayes gets its name because it makes the (naive) assumption that each feature is independent, and uses Bayes rule of conditional probability. The mathematical equation for how naive Bayes calculates this probability is represented in Eq. 3.1.

$$P[C_i|F] = \frac{P[F_1|C_i] \times P[F_2|C_i] \times \cdots \times P[F_f|C_i]}{P[F]} \tag{3.1}$$

where C is the set of classes, i is the index of the class being evaluated, F is the set of features, f is the total number of features, and $P[C_i|F]$ is the probability of resulting class C_i given features F. The denominator, $P[F]$, is for normalizing the probabilities, but otherwise the equation is simply a product of the probabilities given the features. The class with the greatest probability based on these calculations is ultimately chosen by the classifier to be the predicted result.

Since naive Bayes creates a statistical model from all attributes available, the resulting model considers each class as a single Gaussian representation across the full attribute space.

3.2.2 C5.0

C5.0 is a decision tree classifier. C5.0 is a commercial implementation and improvement upon C4.5 [55] designed by the same author. In general, the C5.0 algorithm works the same as C4.5, however C5.0 cites better speed, memory usage, and smaller sized trees as some of its benefits [56].

Decision trees are built by repeatedly splitting the training set on the feature (attribute) which "best" splits the data. There are multiple methods for deciding which feature is best, but C5.0 uses a measure of information entropy. The exact criterion for splitting the training set is the *normalized information gain*, which is the difference in entropy caused by choosing a specific attribute for splitting the data.

The attribute that has the highest normalized information gain is ultimately selected to be the one on which the training set is split. It first calculates the entropy of input data S as defined in Eq. 3.2.

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{3.2}$$

where c is the total number of classes, and p_i is the proportion of exemplars that belong to class i. If the split is not pure, then the exemplars are divided to minimize impurity. The next step is to reduce the entropy by calculating the information gain for each attribute, A, in the input data, S, as determined by Eq. 3.3

$$G(S, A) = E(S) - \sum_{v \in (A)} \frac{|S_v|}{|S|} E(S_v) \tag{3.3}$$

where $E(S)$ is the entropy of the input defined in Eq. 3.2, and $|S_v|$ is the number of instances that have value v for the attribute A. In other words, when a tree is constructed, at each step the split that results in the largest decrease in impurity is chosen. A more detailed explanation of the decision tree can be found in [57].

The resulting model of C5.0 is, in effect, a series of IF/THEN statements which do not necessarily employ all attributes. Given this structure, there may be multiple paths for the same outcome (class).

In this thesis, the boosting feature for C5.0 is not used. Instead, AdaBoost with J48 is used separately as described in Section 3.2.3. Moreover, attribute winnowing is left unexplored in this work.

3.2.3 AdaBoost with J48

AdaBoost, which stands for Adaptive Boosting, is a machine learning *meta-algorithm*. AdaBoost is called a meta-algorithm because it is only used to support another machine learning algorithm, and is not a machine learning algorithm on its own. AdaBoost, therefore, can be used with a variety of other machine learning algorithms, and attempts to boost their performance. The specific method used in this work is termed *AdaBoost.M1*, but herein will be referred to as just AdaBoost.

AdaBoost gets the 'Ada' part of its name from the word 'adaptive'. AdaBoost is adaptive because it increases the *weight* of instances that are incorrectly classified

with each round of its execution, adapting the input to create expert classifiers. An increase in weight means that the importance of classifying this particular instance correctly is greater. Due to this feature, AdaBoost can be very sensitive to outliers or incorrectly labeled data, and great care should be taken to ensure that the training data is as accurate as possible in order to generate a model that performs well on unseen data (ie. the test data set).

The 'boost' part of AdaBoost is in reference to the machine learning technique, *boosting*, which attempts to use multiple models (as opposed to just one) to increase classification performance. With each iteration of AdaBoost, after the weights are adjusted, a new model is created. This ensures a model more specialized at classifying these previously unsolved instances. All models are combined at the end to form one boosted model, allowing each sub-model to make a *bid* on each instance it is asked to classify. A bid is essentially a measure of confidence: how certain that particular model is that it can correctly classify the instance.

In this work, AdaBoost is used with J48. J48 is a Java implementation of C4.5 found in Weka [58].

Essentially, AdaBoost works on the assumption that several classifiers, which are experts in their particular domain, are better than just one general classifier that is used for all test cases. In this spirit, several J48 trees are created, each an expert at a certain set of instances. A more methodical explanation of the training algorithm is given below in Algorithm 1.

Algorithm 1 gives a generalization of how the multiple J48 classifiers are built using AdaBoost. The algorithm begins with instances that all have equal weights, and builds a J48 tree in the same form a C4.5 tree would normally be built. Then, if every instance was correctly identified, or error is greater than 50%, the algorithm terminates. The algorithm re-assigns all weights in the data set by first multiplying the correctly identified instances by $e/(1-e)$. Then, the weights of all instances are normalized to have a sum equal to the sum of weights before this updating step. The weight updating process effectively increases the weight of the incorrectly identified instances, and decreases the weight of the correctly identified instances. These steps are repeated until either all t iterations are completed, or a model is built that has either zero error or greater than 50% error. In the case of the termination clause

Algorithm 1 Algorithm for training AdaBoost with J48 tree

```
for all Instances i do
    Weight(i) ← 1
end for
for 1 → Max iterations do
    m ← J48(Instances)
    e ← Error(m)
    if (e = 0) or (e >= 0.5) then
        break
    end if
    Add m to resulting model
    for all correctly classified instances i do
        Weight(i) ← Weight(i)×(e/(1 − e))
    end for
    Normalize all instance weights
end for
```

based on 50% error, the model is not used: only models built in previous iterations are in the final model.

When the models are tested, their confidence is affected by a weight as well. Specifically, the weight $-log(e/log(1-e))$ is given to the class predicted by each sub-model, where e is the error value calculated for that sub-model while being trained.

The final model produced by AdaBoost is much more complex than the algorithm it is boosting. Multiple sub-models are combined to maximize the statistical properties of the solution. As a result, the solution transparency – or human-readability – is likely to be lost.

3.2.4 Genetic Programming

In this work, a specific type of Genetic Programming is employed, namely Symbiotic Bid-Based Genetic Programming. The symbiotic co-evolutionary model, which is applied in the Symbiotic Bid-Based (SBB) Genetic Programing approach, uses a combination of competitive co-evolution and symbiotic co-evolution to evolve a set

of programs to solve a problem. The architecture of the system is relatively complex compared to the above techniques.

SBB has 3 sets of *populations*, shown in Figure 3.4. These include the *point population*, *team population*, and *learner population*. In the case of an explicit action problem domain, such as classification, the point population consists of a subset of the instances from the training set. The team population is evolved, and contains individuals that index learners from the learner population. Each team indexes at least two learners, and these learners evolve what is called a *bidding program* in association with their *action*. The bidding program accepts the input of an instance (ie. a member of the point population) and returns a bid value, between 0 and 1. This bid is used to determine which learner in the team may apply their *action*. In the case of a classification problem, this action is just simply a class label. Each of these populations is described in further detail below.

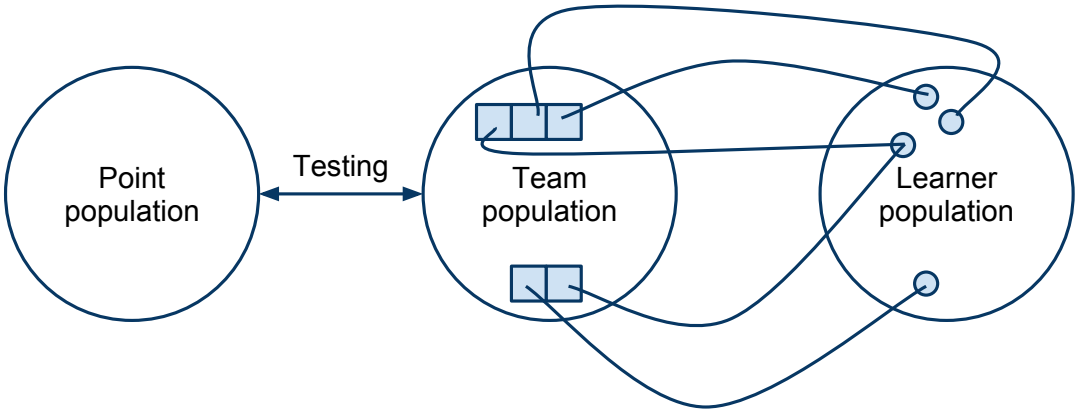

Figure 3.4: Basic SBB Framework

The point population contains P_{size} instances from the training set. This population is constantly refreshed: P_{gap} of these points are removed with each generation of the algorithm, and replaced with P_{gap} new random points. The points chosen to be swapped out are those that score the lowest on a measure of fitness. The fitness measure is calculated as:

$$f_k = \begin{cases} 1 + \frac{1-c}{M_{size}} & \text{if } c > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where c is the number of teams which can correctly identify the point, and M_{size} is the number of teams in the team population. The function is basically a linear fitness function, with highest fitness for a point that is only classified by one team and lower fitness for points which are classified by multiple teams. The point population decouples the cost of fitness evaluation from the size of the training set. By preforming the evaluation on the smaller point population, the algorithm can more quickly complete an iteration.

The team population consists of M_{size} teams, which index from 2 to ω learners from the learner population. A team is nothing more than a set of indices. The team population follows a similar replacement policy as the point population: with each generation M_{gap} teams are removed, and replaced with new teams. To decide which teams are replaced, each team is given a fitness measure defined as:

$$s_i = \sum_{p_k} \left(\frac{G(m_i, p_k)}{\sum_{m_j} G(m_j, p_k)} \right)^3 \tag{3.5}$$

where p_k iterates over each point, m_i is the current team, m_j iterates over every team, and $G(x, y)$ returns 1 if x can correctly classify y. In other words, for each point a team can classify, that team gets an additional fitness of one divided by the number of other teams that can classify it. The team population is responsible for selecting multiple learners which can cooperate to form a classifier. This provides task decomposition and allows learners to focus on separate parts of a problem.

The learner population is where the programs are evolved. A learner has two elements to it: a bidding program and an action. The action is simply the class label when dealing in the classification domain (other problem domains are possible, but not discussed here). The bidding program is a sequence of instructions from Table 3.2 where $R[y]$ is the value of the register at index y, and $I[y]$ is the value of the feature at index y. It is the job of the bidding program to place a bid on each instance. This bid, after having the time to evolve, should become analogous to the learner's confidence that its action is the right action to take. By evolving a high bid for points

for which its action is correct, and a lower bid for points that are incorrect, the learner is able to increase the score of any team that indexes it. Learners are removed when no more teams reference the learner. In short, the learner population provides a pool of programs which may appear in a classifier.

When the team population is replaced with M_{gap} new teams, the new teams are made by first copying an existing team. The copy then has learners removed, added, and mutated according to probabilities represented by p_{md}, p_{ma}, and p_{mm} respectively. There is also a possibility for learner action change at this stage, p_{mn}. A more detailed description of the algorithm can be found in [59].

SBB is able to derive new features from the feature set. This trait allows SBB to take advantage of relations between features in a way that is not possible by the aforementioned algorithms. Models produced by SBB may have multiple programs associated with the same class. The evolved programs can be quite short, and therefore solution transparency is possible.

Table 3.2: Instruction set

Function	Register-Register definition	Immediate value definition
Addition	$R[x] \leftarrow R[x] + R[y]$	$R[x] \leftarrow R[x] + I[y]$
Subtraction	$R[x] \leftarrow R[x] - R[y]$	$R[x] \leftarrow R[x] - I[y]$
Multiplication	$R[x] \leftarrow R[x] \times R[y]$	$R[x] \leftarrow R[x] \times I[y]$
Division	$R[x] \leftarrow R[x] \div R[y]$	$R[x] \leftarrow R[x] \div I[y]$
Cosine	$R[x] \leftarrow \cos(R[y])$	$R[x] \leftarrow \cos(I[y])$
Logarithm	$R[x] \leftarrow \ln \lvert R[y] \rvert$	$R[x] \leftarrow \ln \lvert I[y] \rvert$
Exponential	$R[x] \leftarrow e^{R[y]}$	$R[x] \leftarrow e^{I[y]}$
Conditional	if $R[x] < R[y]$: $R[x] \leftarrow -R[x]$	if $R[x] < I[y]$: $R[x] \leftarrow -R[x]$

The final output of SBB is one team. The team that scores the highest on the last set of points is chosen to be the best solution. This final result works the same way a team works in the training stage: a team is given as input the instance to be tested on, its learners bid, and the highest bidder gets to preform its action (in classification, this action is assigning the class label).

Chapter 4

Data Collection and Data Generation

The data used in this study is gathered from network traffic capture files. These captures consist of the raw data that traveled over the physical wire at a given point on a network. The captures contain headers, timing information, and the application data (if available). The data sets used in this study are outlined in Table 4.1.

Table 4.1: Data Sets Used

Data set	Date captured	Total flows	Availability
UCIS3	Jun 10, 2010 - Jun 12, 2010	27,782,845	Private
UCIS2	Jan 25, 2007	4,195,245	Private
NIMS4	Jan 31, 2011 - Feb 7, 2011	1,259,724	New/Public
MAWI2010	Jun 6, 2010 - Jun 15, 2010	3,536,058	Public
MAWI2011	Jan 1, 2011 - Jan 11, 2011	4,006,610	Public

4.1 UCIS2 & UCIS3

These data sets are traffic captures from the Dalhousie campus network. The Information Technology Services of Dalhousie University captured and labeled the packets using a commercial product called PacketShaper [60]. PacketShaper uses deep packet inspection to identify up to 700 applications. In this work, the focus is on applications that run within an SSL tunnel. Since SSL has a plain text initiation (as described in Section 3.1.1), the ground truth is known in the case of UCIS3. Unfortunately, the UCIS2 data set does not have SSL labeled.

Both UCIS2 and UCIS3 captures were taken on a full-duplex T1 fibre link at a point between the Dalhousie campus network and the Internet. The captures therefore contain all of Dalhousie's interaction with the outside Internet during these time periods, and as such, represent a real-world network usage scenario. The UCIS2 data set consists of 22 GB of raw IP header data captured on January 25th 2007. On the other hand, the UCIS3 data set consists of 122 GB of raw IP header data

captured from June 10th - June 12th, 2010. It should be noted here that due to privacy concerns, the IP addresses have been mapped to new random values, the checksum set to zero, and the captures have the payload removed and therefore only contain header information. However, the packets were labeled before removing this information, allowing for deep packet inspection to be used for this purpose.

4.2 Public Captures: MAWI2010 and MAWI2011

The public captures used in this work consist of those gathered by the Measurement and Analysis on the WIDE Internet (MAWI) Working group [61]. Each day MAWI captures 15 minutes of traffic on a trans-Pacific 150Mbps link, truncates the captures so that only header information is available, and publicly distributes them on-line. For this work, several adjacent days have been combined into two new data sets. The MAWI2010 data set consists of data captured from June 6th, 2010 to June 15th, 2010. The dates were chosen to overlap the time when the UCIS3 data set was captured. Another data set, MAWI2011, was created using captures from the beginning of 2011, from January 1st, 2011 to January 11th, 2011. This data set was made to provide further robustness tests, as well as understand the effect of training models in a different time period than they are trained on. In other words, I aim to answer the question "will a previously generated model work effectively today?"

4.3 Generated Traffic Captures

In order to obtain traffic traces for which the absolute truth about the underlying applications running in SSL tunnels is known, network traffic was generated in the Network Information Management and Security lab at Dalhousie University. The traffic generated is intended to be representative of regular SSL tunneled HTTP traffic. In the real world, most of this traffic would be generated during secure browsing activities such as banking, social networking, and web-based communications such as e-mail. Unfortunately, these activities are rather difficult to mimic. The approach taken in this thesis involves using an SSL tunnel to wrap random "regular" (ie. non-encrypted) web browsing activity. This is the same procedure that would be used to securely visit a bank website or securely access an e-mail through a web browser.

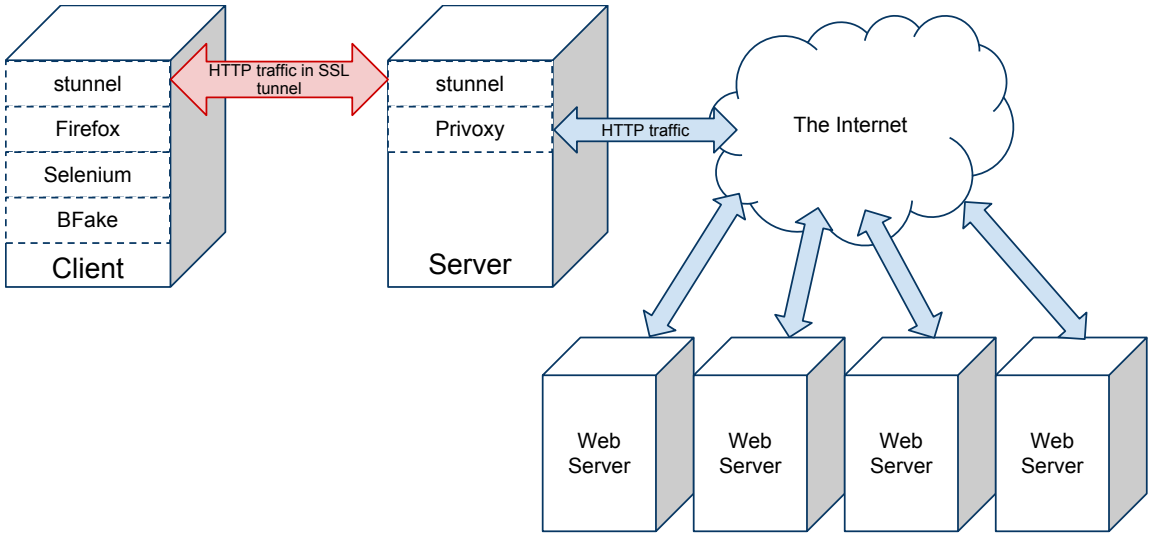

Figure 4.1: BFake Setup

Since the goal is to identify patterns within SSL when tunneling the HTTP protocol, this should provide a reasonable compromise.

The web browsing traffic generation was done by a simple program written in Python and created specifically for this scenario, called *BFake* [62]. Figure 4.1 shows the setup of BFake, whereas Table 4.2 shows the client and server machines software setup employed in these experiments.

The BFake program was run in a virtual machine on the client machine with browser settings set to send all traffic to a listening *stunnel* [63] port. The stunnel proxy was set to create a persistent connection with the proxy machine. The proxy machine was configured to receive the data, remove the SSL tunnel wrapping the HTTP request, and forward this on to a *Privoxy* [64] proxy server running on the same machine. The proxy server was configured to keep connections alive for up to 5 minutes with the HTTP servers if the remote server allowed it. The captures were made on the proxy server, allowing us to capture both the SSL encrypted requests, as well as the corresponding HTTP traffic. Thus, SSL flows that may contain several page views within the same tunnel were obtained. Over the course of many days, 33GB of full capture web browsing traffic was generated.

BFake operates by controlling a standard browser. This is very important to note, because the traffic generated needs to be as similar as possible to how traffic would look if it were captured by a person browsing the web doing their regular business.

Table 4.2: Client and server software setup.

	Server	Client
	Debian 5.0	Ubuntu 10.10 VM running in Gentoo Linux
Services used	tshark 1.2.11 OpenSSL 0.9.8o stunnel 4.29 Privoxy 3.0.16	OpenSSL 0.9.8o stunnel 4.29 Firefox 3.6.13 Selenium server 2.0b1

To this end, Firefox 3 [65] is employed in all our experiments.

The general procedure followed by the program is described as follows. First, the program randomly chooses how many websites it will visit as "seeds" (starting pages) in the current iteration. The program then picks a random web-site, from a list of choices, where it will begin at. The Alexa Top Sites [66] are used for the program to choose from. BFake then pre-determines the number of links it will follow from that web-page. After every page load, the program will wait a random (log-normally distributed) amount of time before clicking a random link on the current page. After fulfilling the amount of links to follow, which was previously determined, BFake will then do one of two things:

- If the number of "seed" web-pages decided in the first step has been achieved, the program will close all browser instances, remove any browsing history or cookies, and terminate.

- If the number of "seed" web-pages has not been achieved, the program chooses a new seed web-site to begin at, opens that page in the current browser, and repeats the process.

The sequence of steps is described in pseudo code in Algorithm 2.

BFake does have a few limitations. There is no user input in BFake. The lack of user input means the program cannot use the POST HTTP request method as intended, such as writing a message on a forum or typing into an interactive text-box. Furthermore, there is no bias in how BFake chooses the links to follow, and no bias on the amount of time spent on a web-page given the information it contains. These are all important characteristics of web browsing activity that could cause some minor differences in the generated traffic when compared to real-life traffic captures.

Algorithm 2 BFake process

```
S ← List of websites
loop
    visit(random(S))
    C ← random number
    for 1 → C do
        L ← links on webpage
        wait(random time)
        visit(random(L))
    end for
end loop
```

Chapter 5

Experiments and Results: With Standard Flow Statistics

In this section, experimental methodology and the results achieved are presented using standard flow statistics provided with a popular open source tool for the task: *NetMate*. The outline of experimental methodology includes all the steps necessary to reproduce the results, given the original data captures. The data sets used in these experiments are large enough that results should be similar with any sufficiently large (and realistic) data set.

5.1 Pre-processing of the Data

First, an explanation is given of the steps taken to ensure that the final data sets employed are accurate and useful. This is called pre-processing the data. It should be obvious that pre-processing of the data has an effect on everything that will follow, and therefore may have a great effect on the performance of the resulting models. Great care should be taken to ensure that the flow statistics generated and labeling are performed as accurately as possible.

5.1.1 Merging Captures

Most captures that span a large length of time are separated into several smaller files. In order to preserve flows, which may span across several of these files, the captures should be merged into one larger file that contains all the packets from each capture in chronological order. This is achieved using *mergecap*, a software tool distributed with Wireshark [67]. Merging the files will allow the flow generator (NetMate) to process the captures as one continuous piece of data rather than several smaller pieces.

5.1.2 Converting Traffic Captures to Flows

To convert raw traffic captures into flow statistics, a program called *NetMate*[68] and a module called *netAI* [69] are used. The combination of the aforementioned programs will be referred to as just NetMate for the rest of this thesis. NetMate reads raw captures, and calculates a variety of statistics about each flow contained in the capture. The software used in these experiments is a slightly modified version, which includes a module to export 40 features of a flow as well as a module to export these features for only the first second of a flow. Table 5.1 shows the features that are calculated and exported. The flows that contain only the first second of data are referred to as *1st-second* flows. 1st-second flows have their attributes calculated only for the first second beginning with the first packet in the flow. The intention is to discover whether application identification is possible early in the stages of a communication between two devices, rather than waiting for the two devices to finish communicating. Furthermore, 1st-second flows will show if classifying encrypted network traffic is possible with a window of data as opposed to the entire flow. This becomes important for network throttling or identifying encrypted applications within long, continuous flows, which may contain several applications.

Each data set is run through NetMate with two different modules. The first module exports all information about the entire connection of the flow. The second exports only statistics about the first second of every flow. A breakdown of the effect this has on the UCIS3 data is shown in Table 5.2, which essentially shows the number of flows that exceed one second for each type of traffic considered in this research. Following this, it is important to mention that UDP flows that contain less than one packet in each direction are removed (via a script) from the data sets. This is essential to conform to the definition of a flow given in Section 3.1.2.

5.1.3 Labeling the Data and Removing IP/port information

The approach for labeling the data differs depending on the data set. For the UCIS3 data set, the labels are provided by the Dalhousie ITS team. As mentioned in Section 4.1, the packets are labeled using DPI, and the label is stored in the IP header's Differentiated Services Code Point (DSCP) field. It is important to note that no labeling is done by ITS for the underlying application inside an SSL communication,

Table 5.1: Features exported by NetMate. (f&b) indicates that a feature is calculated separately for both the forward and the backward directions of the flow.

(a) Data based features

Name	Description
proto	protocol
total_fpackets, total_bpackets	total packets (f&b)
total_fvolume, total_bvolume	total bytes (f&b)
min_fpktl, min_bpktl	minimum packet length (f&b)
mean_fpktl, mean_bpktl	mean packet length (f&b)
max_fpktl, max_bpktl	maximum packet length (f&b)
std_fpktl, std_bpktl	standard deviation of packet lengths (f&b)
sflow_fpackets, sflow_bpackets	average sub-flow packets (f&b)
sflow_fbytes, sflow_bbytes	average sub-flow bytes (f&b)
fpsh_cnt, bpsh_cnt	push flag count (f&b)
furg_cnt, burg_cnt	urg flag count (f&b)
total_fhlen, total_bhlen	header length (f&b)

(b) Time based features

Name	Description
duration	duration
min_fiat, min_biat	minimum inter-arrival time (f&b)
mean_fiat, mean_biat	mean inter-arrival time (f&b)
max_fiat, max_biat	maximum inter-arrival time (f&b)
std_fiat, std_biat	standard deviation of inter-arrival times (f&b)
min_active	minimum active time
mean_active	mean active time
max_active	maximum active time
std_active	standard deviation of active times
min_idle	minimum idle time
mean_idle	mean idle time
max_idle	maximum idle time
std_idle	standard deviation of idle times

Table 5.2: Effects of 1s module on UCIS3 data set

Type	Flows Affected	% Affected
TOTAL	7,688,120	27.67%
OTHER (HTTP)	6,253,072 (2,142,078)	25.7% (46.46%)
SSL	803,459	45.44%
SSH	34,798	97.67%
SKYPE	280,680	23.90%
P2P	316,103	66.34%

Table 5.3: Flow counts in each data set. * indicates port based labelling.

	UCIS3	UCIS2	MAWI2010	MAWI2011	NIMS4
HTTPS	1,750,048*	843,204*	93,198*	88,797*	337,135
HTTP	4,610,264*	1,714,694*	856,857*	1,211,051*	922,589
OTHER	19,717,811	1,615,265	2,582,895	2,703,468	0
P2P	476,491	N/A	N/A	N/A	0
SKYPE	1,174,358	N/A	N/A	N/A	0
SSH	35,622	15,863	N/A	N/A	0
SSL (non-web)	18,250	6,219*	3,108*	3,294*	0

and therefore all SSL communications running over the standard port for web browsing (443) was labeled as HTTPS. Given that the flow has already been identified as SSL, and port 443 is reserved by most machines for SSL, this labeling method should be reasonably accurate. For the generated captures (the NIMS data set), the ground truth is known, and can easily be labeled using a priori knowledge. For public captures, the flows are labeled by port the number alone, because no other suitable alternative is available to use without payload. The UCIS2 data set does not have SSL labeled, and therefore uses the same labeling method as the public data sets.

The last step in pre-processing the data is to remove the IP and port number information from the flows. This information is removed since both IP address and port information are not useful in identifying the contents of encrypted tunnels. The information may provide an unfair bias by helping to identify IP addresses that frequently use SSL (such as HTTPS servers) or commonly used ports for SSL tunnels. However, the goal is to identify arbitrary flows that may be intentionally obfuscated using proxies or non-standard ports. The final count of each labeled data set is shown in Table 5.3. It is important to note that a small fraction of these flows are removed in pre-processing the 1-second flows because a response has not yet been received in the backward direction. This has a small (about 3%) effect on the overall number of flows in the 1-second test set.

5.2 Usage of the Machine Learning Techniques

Each machine learning technique is trained on a training set, and tested on the resulting test sets (the remaining flows from the data sets). Training sets are chosen

by randomly selecting a pre-determined amount of each application type. To aid in selecting the distribution of the flows, an analysis of several select features is done to identify what the distribution of the features looked like for particular applications. It is quite obvious that the flows that most closely resemble SSL in terms of features are flows on port 80, as shown in Figure 5.1, which gives violin plots showing the density of some selected features. The assumption is made that this is due to HTTPS being the most common use of SSL. This hypothesis is given further merit when considering 98.97% of SSL flows traveled through port 443 – the port number assigned to HTTPS by the Internet Assigned Numbers Authority (IANA) [1]. To complement this, work by Este et al. reported that SSL was classified as HTTP by their classifier which had never seen SSL traffic [38]. From this, an increased amount of flows from port 80 are used to aid the machine learning algorithms in creating a distinction.

Naive Bayes and AdaBoost are implemented using Weka [58]. Weka is an open source machine learning toolkit consisting of implementations of several different machine learning algorithms and techniques. Regarding parameters, none are needed for naive Bayes, and AdaBoost is configured with the values in Table 5.4. C5.0 models are built using the commercial software [55], no options are used (ie. the default tree is made), and the parameter for *confidence factor* is varied from 1 to 50. The confidence factor sets a threshold for the pruning of the tree. The C5.0 algorithm compares the error of the children of this node against the resulting error if these nodes were removed and the node was made a leaf with the label of the majority class. The confidence factor is used to decide when pruning should be preformed based on this comparison. After varying this value, the best preforming model is chosen. The confidence factor is varied to explore the effects of pruning the C5.0 tree in different ways. SBB models are evolved using the source code developed here at Dalhousie University by Peter Lichodzijewski and John Doucette [70]. The code is further extended to allow an evolved team to be applied to several test sets and re-run at a later execution time. SBB is run with 50 different seed values due to its stochastic nature, and the best preforming model is chosen. Table 5.5 shows the parameters used for SBB.

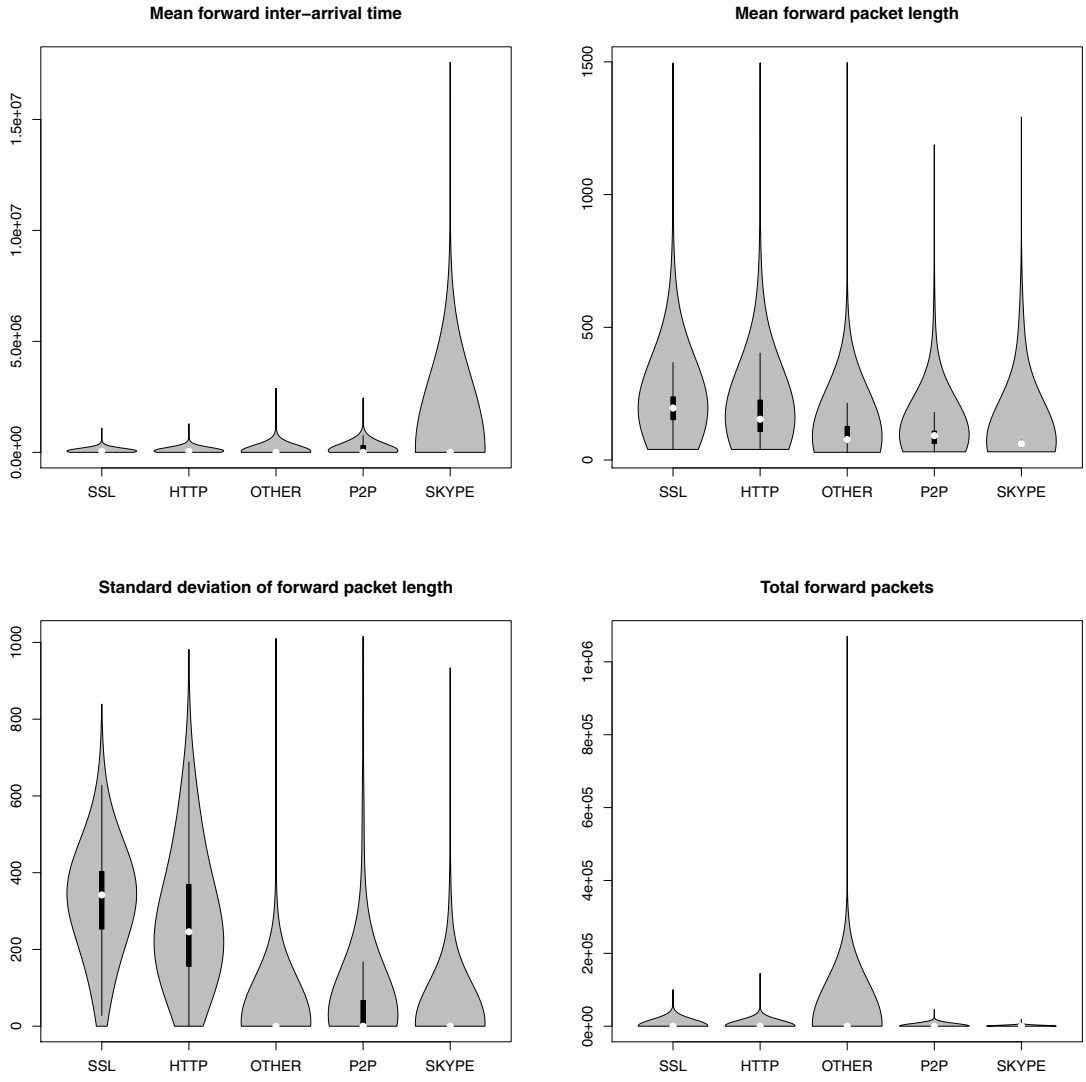

Figure 5.1: Violin Plots of Select Features Using Full Flows

Table 5.4: AdaBoost Parameters

AdaBoost		
Parameter	Description	Value
I	Number of boost iterations	10
P	Percentage of weight mass used to build classifiers	100
J48		
Parameter	Description	Value
C	Confidence threshold for pruning	0.25
M	Minimum number of instances per leaf	2

Table 5.5: SBB parameters used

Parameter	Description	Value
P_{size}	Point population size	120
M_{size}	Team population size	120
P_{gap}	Point population to replace with each generation	20
M_{gap}	Team population to replace with each generation	60
p_{md}/p_{ma}	Probability of learner deletion/addition	0.7
p_{mm}	Probability of learner mutation	0.2
p_{mn}	Probability of learner action change	0.1
ω	Maximum size of a team	10
$maxProgSize$	Maximum size of a program in instruction count	48
$p_{bidmutate}$	Probability of a bid program mutating two instructions	1
$p_{bidswap}$	Probability of a bid program swapping two instructions	1
p_{bidadd}	Probability of adding a new instruction	1
$p_{biddelete}$	Probability of deleting an instruction	1
$numLevels$	The number of levels of teams to use.	1

5.2.1 Training and Test Sets

The training sets are quite possibly the most important part of any machine learning algorithm. A model created by a machine learning algorithm can only learn the information that was given in the training phase. The training set, therefore, puts an upper bound on the quality of the resulting model's performance (ie. it defines the absolute best a machine learning algorithm can do). Unfortunately, there are no clear rules defined for what size training set is best, or how to best choose the training set instances. The choice is sometimes made more difficult by the availability of labeled data, or requirements emphasizing quick training times (smaller sets are usually faster). This experiment analyzes the results of 6 different training sets, sampled once for both types of flows used (ie. the same flows are used to create sets for full flows as are used for 1-second flows). These training sets are designed to explore the possible options and contrast differences between these choices. The training sets range in size from 12000 to 70000 flows.

The training and test sets are split up into two levels. At the first level, training sets are designed for identification of SSL traffic. These training sets are comprised entirely of flows from the UCIS3 data set, since these flows are very reliably labeled and show realistic patterns. The test sets contain all remaining flows from the data

sets employed. Thus, I refer to these SSL-identifying learning models as level 1 solutions, designed to identify SSL traffic from a given traffic file without using IP addresses, port numbers, or payload information. The second level is designed to distinguish SSL tunneled browsing behaviour from non-browsing SSL behaviour. The training sets at this level consist of different configurations of HTTPS and non-HTTPS SSL flows from the UCIS3 and NIMS4 data sets. The test sets contain the remaining SSL flows from each data set.

A uniform random selection is preformed and the amount of data given in Table 5.6 is sampled. Amounts that have an asterisk (*) beside them in these tables are sampled from the artificially generated data set. All other flows are sampled from the UCIS3 data set, which contains the most realistic and reliably labeled data. The bias given to SSL and HTTPS flows is for the obvious reason: this is the protocol that I am interested in detecting in this research. Uniform samples of the remaining labeled applications are used in order to give a diversity to the training set, with the exception that there is a bias to HTTP flows as it is concluded that HTTP flows would be the most difficult to distinguish from SSL flows. To provide a bit more diversity in the training sets, some more flows are sampled from the "other" category (unlabeled flows). An explanation of the justification and intention of each training set is given below.

L1_UCIS3 This is a general training set comprised of entirely UCIS3 labeled flows. This training set should be able to provide enough training data for the reliable identification of SSL.

L1_UCIS3_LARGE This is a large training set designed to investigate the impact of additional flows in the training set.

L2_UCIS3 A data set containing equal amounts of SSL on port 443 from the UCIS3 data set and SSL not on port 443 from the UCIS3 data set. This balanced training set will be used to train models that are able to identify tunneled web browsing among SSL flows.

L2_UCIS3_LARGE This training set is similar in intention to L1_UCIS3_LARGE in that it aims to investigate the use of additional flows. However, since SSL

flows that do not run on port 443 are scarce in numbers, the addition of non-SSL flows to increase performance is investigated.

L2_NU This training set combines the NIMS4 HTTPS flows (flows generated in the lab) with UCIS3 non-SSL flows. This is the only training set that contains HTTPS flows where the ground truth is 100% certain. Because of this, results will rely heavily on how similar the NIMS4 HTTPS flows are to the SSL flows on port 443.

L2_NU_LARGE This training set aims to investigate whether the NIMS4 HTTPS flows are similar enough to UCIS3 HTTPS flows that the inclusion of extra flows from it will increase performance. The training set includes the same training flows as the L2_UCIS3_LARGE training set and includes 20000 additional HTTPS flows from NIMS4.

Table 5.6: Training sets. Amounts with * indicate sampling from an artificially generated data set.

(a) Level 1 Training Sets

	L1_UCIS3	L1_UCIS3_LARGE
SSL	6,000	20,000
OTHER (HTTP)	6,000 (3,000)	20,000 (10,000)
SSH	1,000	5,000
SKYPE	1,000	5,000
P2P	1,000	5,000
TOTAL	15,000	55,000

(b) Level 2 Training Sets

	L2_UCIS3	L2_UCIS3_LARGE	L2_NU	L2_NU_LARGE
HTTPS	6,000	10,000	6,000*	10,000 + 20,000*
Other SSL	6,000	5,000	0	5,000
OTHER (HTTP)	0	20,000 (10,000)	6,000 (3,000)	20,000 (10,000)
SSH	0	5,000	1,000	5,000
SKYPE	0	5,000	1,000	5,000
P2P	0	5,000	1,000	5,000
TOTAL	12,000	50,000	15,000	70,000

5.3 Results

When evaluating network traffic classification, one usually considers two measures of a classifier: detection rate (DR) and false positive rate (FPR). To facilitate in describing these two measures, I will refer to *in class* and *out of class* flows. An in class flow is a flow that is labeled as SSL when referring to level 1 classification, or HTTPS when referring to level 2 classification.

The detection rate is a measure of how often the classifier can correctly identify the target (in class) application or protocol. In this work, the detection rate would be equivalent to the percentage of in class flows that were flagged correctly. In mathematical terms, detection rate can be described as:

$$DR = \frac{tp}{tp + fn} \tag{5.1}$$

where tp is the number of detected in class flows (true positives), and fn is the number of in class flows incorrectly identified as out of class (false negatives). $tp + fn$ is therefore just the total number of in class flows.

False positive rate is a measure of how often a classifier incorrectly identifies an out of class flow as being in class. The false positive rate can be expressed in words as the probability that an out of class flow will be labeled as in class and becomes significant in this field when considering how often an administrator would be bothered by false alarms. In mathematical terms, false positive rate is described as:

$$FPR = \frac{fp}{fp + tn} \tag{5.2}$$

where fp is the number of out of class flows incorrectly labeled as in class (false positives) and tn is the number of out of class flows correctly identified as out of class (true negatives).

It is important to note that detection rates of 100% are not achieved despite clear text initiation of the SSL handshake because of the tactic used. All algorithms only have packet-level information available to them, and no payload, port, or IP address information is given. These results, then, are a best guess based on the associated flow attributes.

Table 5.7: L1_UCIS3 - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	97.30%	96.25%	93.86%	80.34%
UCIS2	99.42%	98.09%	97.18%	49.12%
NIMS4	97.55%	75.41%	63.20%	15.66%
MAWI2010	96.17%	86.36%	82.19%	61.19%
MAWI2011	94.71%	83.29%	77.79%	65.02%

(b) False Positive Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	27.36%	1.96%	2.49%	3.35%
UCIS2	58.39%	5.08%	7.92%	9.69%
NIMS4	98.44%	6.72%	5.99%	7.36%
MAWI2010	40.36%	1.62%	1.95%	2.32%
MAWI2011	37.74%	1.61%	1.65%	1.77%

5.3.1 Full flows, level 1

The performance in terms of detection rate and false positive rate is shown in Table 5.7 and Table 5.8.

Subjectively, the best performing model above was the C5.0 solution trained on the L1_UCIS3_LARGE training set. While AdaBoost provides a solution with slightly better results, the trade-off for a more simple model gives C5.0 a particular advantage when identifying SSL traffic. Not only does C5.0 have lower computational cost when evaluating, but the model is also easier to understand and interpret by an administrator.

The C5.0 solution considers the forward push count (fpsh_cnt) as the first attribute on which to split. The branch for a fpsh_cnt ≤ 1 is a relatively small branch and most commonly results in a decision of NON-SSL. The branch for fpsh_cnt > 1 is much larger and seems to be the more difficult part for C5.0 judging by the numerous branches. After considering the forward push count, C5.0 creates many fine tuned sub-trees and does not appear to have an easy time breaking off groups of either class (a simple problem would produce smaller branches, where leaves produced high yields of specific class). The top 5 used attributes in evaluation are forward push

Table 5.8: L1_UCIS3_LARGE - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	98.09%	96.92%	95.71%	81.12%
UCIS2	99.63%	98.14%	97.71%	50.40%
NIMS4	98.41%	74.41%	76.25%	18.15%
MAWI2010	97.31%	89.47%	89.16%	62.40%
MAWI2011	92.22%	86.54%	80.50%	66.30%

(b) False Positive Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	32.27%	1.44%	1.98%	3.16%
UCIS2	66.71%	5.50%	6.58%	10.18%
NIMS4	99.29%	5.85%	3.63%	8.32%
MAWI2010	43.90%	1.81%	2.07%	2.30%
MAWI2011	41.20%	1.51%	1.72%	1.83%

count, standard deviation of forward packet length, maximum forward packet length, maximum backward packet length, and mean backward inter-arrival time.

5.3.2 Full flows, level 2

The performance in terms of detection rate and false positive rate is shown in Table 5.9 through Table 5.12.

An investigation of these tables show that AdaBoost performs best in terms of only these measures (detection rate and false positive rate). Once again, the slightly lower detection rate and slightly higher false positive rate that C5.0 gives is considered. In this case, however, a low false positive rate is more vital. At the first level, an investigator can verify results using the plain-text SSL handshake; however, at the second level this option is no longer available. Furthermore, the negative effect of higher processing time is mitigated because the first level classifier will have removed any non-SSL flows. The model generated from the L2_UCIS3 training set provides the best balance of low false positive rate and high detection rate on all test data sets.

The AdaBoost model is much more difficult to analyze due to the effect of boosting.

Table 5.9: L2_UCIS3 - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	97.69%	97.85%	96.53%	81.73%
UCIS2	99.72%	98.25%	98.03%	76.72%
NIMS4	94.16%	74.16%	66.45%	35.64%
MAWI2010	97.84%	85.54%	86.40%	61.04%
MAWI2011	96.48%	80.20%	85.19%	62.00%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	90.98%	3.19%	5.31%	10.13%
UCIS2	89.31%	3.68%	4.79%	5.23%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	88.00%	6.21%	8.43%	0.87%
MAWI2011	88.59%	5.40%	7.56%	1.15%

Table 5.10: L2_UCIS3_LARGE - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	98.11%	93.53%	91.38%	80.29%
UCIS2	99.56%	96.31%	94.05%	47.11%
NIMS4	97.56%	60.02%	59.05%	13.85%
MAWI2010	97.04%	77.34%	74.19%	61.81%
MAWI2011	96.13%	66.74%	65.41%	66.57%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	90.67%	2.57%	4.43%	12.44%
UCIS2	88.78%	3.97%	7.11%	3.34%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	82.59%	4.38%	3.76%	1.48%
MAWI2011	73.47%	3.92%	1.97%	1.43%

Table 5.11: L2_NU - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	10.16%	0.82%	1.65%	21.34%
UCIS2	4.47%	1.58%	4.03%	52.21%
NIMS4	97.97%	99.86%	99.73%	99.91%
MAWI2010	28.42%	24.92%	26.76%	89.02%
MAWI2011	31.82%	4.49%	4.70%	15.41%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	7.01%	0.98%	1.25%	18.62%
UCIS2	7.67%	2.65%	8.73%	67.31%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	19.69%	13.42%	1.99%	71.72%
MAWI2011	10.90%	0.55%	0.79%	25.26%

Table 5.12: L2_NU_LARGE - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	96.70%	93.76%	91.07%	90.41%
UCIS2	99.24%	96.63%	95.29%	98.01%
NIMS4	98.90%	99.93%	99.77%	99.64%
MAWI2010	96.89%	80.64%	71.94%	85.53%
MAWI2011	95.91%	69.14%	66.98%	85.09%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	90.25%	3.49%	4.96%	18.35%
UCIS2	89.23%	6.98%	6.34%	21.02%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	87.84%	7.92%	1.99%	8.40%
MAWI2011	80.75%	5.46%	2.00%	12.17%

Table 5.13: L1_UCIS3_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	92.47%	96.04%	94.60%	90.01%
UCIS2	96.32%	98.09%	97.08%	95.96%
NIMS4	88.59%	91.49%	80.85%	99.91%
MAWI2010	91.66%	89.15%	88.14%	85.66%
MAWI2011	90.81%	82.20%	82.60%	86.81%

(b) False Positive Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	18.36%	2.03%	2.10%	4.11%
UCIS2	46.82%	5.03%	5.76%	9.54%
NIMS4	70.41%	6.07%	5.77%	19.69%
MAWI2010	22.33%	2.85%	3.61%	2.73%
MAWI2011	25.30%	1.88%	1.40%	2.57%

Because it essentially provides several decision tree models combined into one, the effects of each decision made by the tree are much more difficult to understand. Looking at the first feature considered by the first tree built, the maximum forward packet length is used to split the flows (C5.0 used standard deviation of forward packet length). From this point, both branches have sub-trees with a great span and therefore showed great complexity in identifying the underlying process. The second and fourth trees in the AdaBoost model use the forward push count as the feature on which to split the initial set of data. This implies that this feature is also important for identifying the underlying application in SSL, not just for identifying SSL itself as discovered when analyzing level 1 models.

5.3.3 1st-second flows, level 1

The performance in terms of detection rate and false positive rate is shown in Table 5.13 and Table 5.14.

For the first time in these tests, analysis of the models show that SBB trained on the L1_UCIS3_1S training set provides a simple yet effective solution. The SBB solution employs 14 of the features, and identified an instance of SSL in only 51

Table 5.14: L1_UCIS3_LARGE_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	95.14%	96.96%	95.09%	88.19%
UCIS2	97.54%	97.94%	97.42%	94.88%
NIMS4	96.58%	90.33%	91.23%	99.57%
MAWI2010	93.16%	91.17%	88.83%	84.28%
MAWI2011	93.20%	89.04%	87.71%	86.21%

(b) False Positive Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	22.01%	1.33%	1.51%	5.48%
UCIS2	52.75%	5.52%	5.81%	7.37%
NIMS4	93.89%	3.88%	2.67%	18.04%
MAWI2010	28.85%	2.14%	2.49%	2.09%
MAWI2011	30.53%	1.35%	1.43%	1.99%

instructions. SBB most commonly employs the mean forward packet length feature, followed by mean idle time, then minimum active time, standard deviation of the idle time, and backward push count. It is interesting to note that SBB focuses on time based features while the other models so far do not. SBB, unlike the other models, has the ability to make more complex relations between features, which may be the reason why time based features are more useful for SBB.

The best model, however, appeares to be C5.0 trained on the L1_UCIS3_1S training set. C5.0 generates a relatively simple decision tree, which will be analyzed later in Section 5.3.5.

5.3.4 1st-second flows, level 2

The performance in terms of detection rate and false positive rate is shown in Table 5.15 through Table 5.18.

Analysis of the above models once again shows that AdaBoost trained on the L2_UCIS3_1S training set is able to perform best in the second layer identification (web browsing vs non web browsing). The maximum forward packet length is the first considered attribute in both the AdaBoost and C5.0 implementations. This model is

Table 5.15: L2_UCIS3_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	93.70%	97.53%	96.51%	80.59%
UCIS2	96.08%	96.39%	96.96%	53.96%
NIMS4	66.98%	87.05%	89.56%	19.68%
MAWI2010	77.06%	79.64%	83.06%	59.61%
MAWI2011	77.76%	77.43%	81.44%	64.59%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	15.06%	3.13%	4.94%	11.80%
UCIS2	21.98%	2.01%	12.18%	3.93%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	37.48%	5.73%	9.98%	0.03%
MAWI2011	36.89%	5.23%	17.15%	0.06%

Table 5.16: L2_UCIS3_LARGE_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	96.04%	94.13%	91.44%	80.25%
UCIS2	98.15%	94.28%	95.08%	49.28%
NIMS4	99.84%	72.47%	74.11%	16.43%
MAWI2010	93.16%	77.82%	74.81%	59.94%
MAWI2011	92.07%	68.40%	73.35%	64.34%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	94.28%	2.49%	4.02%	11.79%
UCIS2	89.40%	8.58%	8.70%	4.55%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	60.56%	14.07%	12.23%	0.52%
MAWI2011	67.58%	15.72%	14.93%	0.67%

Table 5.17: L2_NU_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	10.70%	1.30%	1.42%	11.58%
UCIS2	4.53%	0.64%	0.54%	30.32%
NIMS4	99.39%	99.79%	99.62%	99.86%
MAWI2010	28.18%	26.01%	25.12%	16.04%
MAWI2011	32.28%	2.17%	4.34%	7.11%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	6.45%	0.55%	2.17%	11.75%
UCIS2	7.14%	0.98%	4.54%	50.86%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	22.86%	8.63%	7.34%	1.26%
MAWI2011	19.43%	0.30%	3.77%	3.32%

Table 5.18: L2_NU_LARGE_1S - Results quoted are independently calculated with respect to the test set specified

(a) Detection Rate

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	95.74%	94.43%	92.51%	87.47%
UCIS2	97.96%	96.71%	93.67%	95.09%
NIMS4	99.98%	99.92%	99.75%	98.32%
MAWI2010	93.93%	85.55%	83.23%	75.55%
MAWI2011	93.02%	74.54%	76.81%	77.08%

(b) False Positive Rate. Note: out of class flows do not exist in the NIMS4 data set.

	Naive Bayes	AdaBoost	C5.0	SBB
UCIS3	94.47%	3.18%	4.81%	15.69%
UCIS2	92.97%	16.77%	12.15%	10.36%
NIMS4	N/A	N/A	N/A	N/A
MAWI2010	80.10%	21.54%	16.32%	5.34%
MAWI2011	80.84%	17.76%	13.44%	7.66%

discussed further in Section 5.3.5.

5.3.5 Further Analysis and Best Solution

Unfortunately, models that are trained using NIMS4 flows at level 2 do not perform well on any other data sets other than itself, which suggests that the NIMS4 data set includes different behaviours than the other data sets. However, NIMS4 flows still seem to contain accurate HTTPS flows, as shown by the detection rate of models trained on the UCIS3 data set when tested on NIMS4. It is likely that the NIMS4 data set simply do not exhibit enough diversity in behaviours. The classifiers trained on a subset of the NIMS4 data preform exceptionally well when tested on the remaining NIMS4 data set, which further enforces the theory that there is not enough diversity in the flows. Large training sets were sometimes found to be detrimental to the performance of a machine learning algorithm in these tests.

The above results from Section 5.3.1 through Section 5.3.4 hint towards many interesting details. To no surprise, naive Bayes performs poorly in every case, having very high false positive rates. The results also show that by only considering the first second of a flow, it is possible to achieve very comparable results as flows that contain statistics about the entire duration of the flow. Furthermore, using only the first second of a flow has positive impacts on the performance of the classifiers evaluated on the NIMS4 data set. Models that are trained on UCIS3 have higher detection rates and lower false positive rates when tested on the NIMS4 data set in almost every case for both level 1 and 2 classification. The 1st-second flows also increased overall performance on robustness tests (testing the trained models on data from a network entirely different than the network they are trained on, ie. trained using UCIS3, but tested on MAWI2010 or MAWI2011) as well as the UCIS2 data set. 1st-second flows increased SBB performance drastically; however, SBB still has trouble performing well relative to AdaBoost and C5.0.

Herein, a couple of the best solutions using 1st-second flows are analyzed. At the first level, I have a model created by the C5.0 algorithm trained on the L1_UCIS3_1S training set with a confidence factor of $c = 3$, which preforms exceptionally well (see Section 5.2 for a description of confidence factor). As discussed in Section 5.3.3, I consider the C5.0 model over the AdaBoost model due to the decreased complexity

Table 5.19: Level 1: Per-application Results for C5.0 using the L1_UCIS3_1S training set

(a) Hit and Miss Chart by Application

		UCIS3	UCIS2	NIMS4	MAWI2010	MAWI2011
SSL	HIT	1,663,381	824,339	272,505	84,560	75,823
	MISS	94,972	24,777	64,563	11,379	15,971
HTTP	HIT	4,299,457	1,567,064	863,989	782,652	1,131,057
	MISS	297,155	138,744	52,875	61,041	33,780
OTHER	HIT	18,864,520	1,477,413	N/A	2,469,966	2,601,389
	MISS	219,884	48,110	N/A	60,764	19,168
P2P	HIT	389,682	N/A	N/A	N/A	N/A
	MISS	1,011	N/A	N/A	N/A	N/A
SKYPE	HIT	1,153,957	N/A	N/A	N/A	N/A
	MISS	11,414	N/A	N/A	N/A	N/A
SSH	HIT	33,937	14,697	N/A	N/A	N/A
	MISS	675	128	N/A	N/A	N/A

(b) Detection rates by application

		UCIS3	UCIS2	NIMS4	MAWI2010	MAWI2011
HTTPS	HIT	1,697,207	812,453	293,408	73,935	68,533
	MISS	42,911	30,448	43,660	18,898	19,973
Other SSL	HIT	11,852	6,090	N/A	2,928	3,116
	MISS	1,842	125	N/A	178	172

of C5.0 while still achieving comparable results. The results by application are shown in Table 5.19. A summary of the detection rate and false positive rate can be found in Table 5.13.

19 features are used by C5.0. The percentage of times each feature is used for the training set is shown in Table 5.20. Forward push count is a popular choice among all trained models, and appears here as the first node. Interestingly, C5.0 uses the forward push count twice in a row to begin the tree. It would be reasonable to assume that this feature is very important in effectively detecting SSL flows. Following this, C5.0 has used many of the packet length features, as well as the protocol to further separate flows. At the lower levels of the tree, it seems the most difficult to solve instances have made C5.0 increase the use of inter-arrival time features rather than relying on packet length based features.

Table 5.20: Feature usage in C5.0 decision tree

100%	fpsh_cnt	6%	total_fhlen
73%	max_fpktl	4%	mean_biat
59%	max_bpktl	3%	total_bpackets
51%	proto	3%	std_biat
39%	std_fpktl	2%	mean_fpktl
37%	std_bpktl	2%	bpsh_cnt
36%	total_bhlen	2%	min_fpktl
34%	min_biat	1%	total_bvolume
15%	min_fiat	1%	mean_bpktl
10%	total_fvolume		

For the second level, as discussed in Section 5.3.4, the AdaBoost model is rated as the best performer. The AdaBoost model is definitely much more complex and more difficult to analyze, however there are a couple reasons why this is the case at the second level:

1. The second level payload cannot be inspected by an investigator to check for accurate results. This means a low false positive rate is crucial.

2. The second level is much more difficult for the machine learning algorithm to distinguish since both web browsing and non-web browsing activities will contain the similar traits of the tunnel, SSL.

This model is built using the L2_UCIS3_1S training set. For differentiating between web browsing and non-web browsing traffic, AdaBoost has made use of many of the same features C5.0 used to classify SSL traffic. The results of testing the AdaBoost model are given, separated by application, in Table 5.21. A summary of the detection rate and false positive rate can be found in Table 5.15.

Because of the complexity of AdaBoost, it is much more difficult to analyze the results. In this case, there are essentially ten J48 decision trees in the model that are all evaluated for each instance, and the highest bidder is given the chance to apply its label. What can be gathered is the first attribute used by the first J48 tree was maximum packet length at this level, instead of forward push count, which was used at level 1. Forward push count is, however, used as the second attribute by one branch of the J48 tree. Furthermore, forward push count is also used by the first

Table 5.21: Level 1: Per-application Results for AdaBoost using the L2_UCIS3_1S training set

(a) Hit and Miss Chart by Application

Data Set		HTTPS	NON-WEB SSL
UCIS3	HIT	1,697,207	11,852
	MISS	42,911	1,842
UCIS2	HIT	812,453	6,090
	MISS	30,448	125
NIMS4	HIT	293,408	N/A
	MISS	43,660	N/A
MAWI2010	HIT	73,935	2,928
	MISS	18,898	178
MAWI2011	HIT	68,533	3,116
	MISS	19,973	172

(b) Detection rates by application

Data Set	SSL	NON-WEB SSL
UCIS3	97.53%	96.87%
UCIS2	96.39%	97.99%
NIMS4	87.05%	N/A
MAWI2010	79.64%	94.27%
MAWI2011	77.43%	94.77%

node in the second, third, and fifth J48 trees in the AdaBoost model. This confirms its importance in identifying not only SSL, but differentiating between web browsing and non-web browsing activities within an SSL tunnel. The majority of each tree is comprised mostly of packet length based features.

Chapter 6

A Need for a Different Feature Set

The preliminary investigation of identifying SSL traffic brought forth some interesting discoveries. As discussed in Section 5.3.5, the models trained on 1st-second flows actually performed better on robustness tests. Furthermore, 1st-second flows increased SBB performance, and overall caused only a small decrease in detection rate for other algorithms on the UCIS3 data set.

When SBB performed best, the algorithm used many of the timing features, which were much less commonly used by C5.0 and AdaBoost. One major difference between SBB and the other algorithms was that SBB was able to derive more complex relations between features. For example, SBB could divide one timing feature by another, and create a ratio. It seems logical that using the timing features in this way would be more useful: timing is likely to change depending on the network architecture, configuration, congestion, and the applications being used. The exact forward inter-arrival time may not be an identifying characteristic, but perhaps the forward inter-arrival time is important relative to backward inter-arrival time.

These discoveries lead to the investigation of what information about a flow is most useful in aiding a machine learning algorithm to properly detect SSL traffic. NetMate calculates several different features for each flow, but these features are somewhat arbitrary and often will not help in uniquely representing an application. For example, the maximum packet size for a flow is a rather questionable attribute. Often, when sending a large message, the maximum packet size for a flow is very independent of the application. Instead, the maximum packet size will likely be governed by the maximum size allowable over the network medium (1500 bytes for Ethernet, which limits most of the Internet). An important question to ask is whether the necessary features to most effectively identify SSL traffic are extracted and made available by NetMate. The discoveries outlined above, and discussed in further detail in Section 5.3, hint toward a couple points:

1. The timing data can be useful, especially for robustness. However, timing data seems to be represented by NetMate in a format such that the C5.0 and AdaBoost algorithms cannot directly make the best use of the information.

2. Identifying SSL flows is possible using a window of data. This implies that characteristics of SSL are seen throughout the entire connection. Perhaps, many of the identifying characteristics are contained within relations. "Total", "minimum", or "maximum" values of packet lengths or inter-arrival times are likely less important.

From these points, a new set of experiments are designed to test these ideas and provide a basic investigation in the direction of these ideas. Although the experiments are far from a complete investigation into these topics, the following work lays the foundation for further research into the topics discussed.

Chapter 7

New flow statistics tools

One of the major ways in which the following experiments contribute to research is in the development of a new tool for gathering flow statistics. I believe that in order to properly identify SSL flows, the it is necessary to give the machine learning algorithms the ability to make use of relations of features within the flows.

So to investigate this, a new tool is designed from the ground up, called *flowtbag* [71]. Flowtbag is made publicly available and developed as an open source tool for generating flow statistics. I designed and developed this new tool using my experience gained from working with NetMate. From a high level perspective, Flowtbag operates very similar to NetMate. The program accepts a network traffic capture file, or a live stream of network data as input. Flowtbag then iterates through each packet, assigning it to a flow based on the criteria given in Section 3.1.2. Flowtbag parses the header and calculates any values needed for the features it will export. When a flow is considered complete (as per the criteria in Section 3.1.2), flowtbag calculates the final outputs and prints them to a file along with identifying flow criteria. An interface allows a researcher to easily define new features for the program to extract. Using this interface, flowtbag can be used to create a set of features which describe relations such as a more granular view of the distribution of packet lengths and packet timing.

In order to investigate the aforementioned ideas, a new set of very simple features is created for these experiments. These new features are described in detail in Section 7.1.

7.1 New Features

Using the interface developed for flowtbag, the program is configured to calculate a new set of features. Instead of calculating minimum, mean, maximum, total, and standard deviation, flowtbag is set to calculate ratios (percentages) for packet lengths

and inter-arrival times. A similar technique was used successfully by [72, 73] for identifying web pages in HTTP connections.

The ratios describe the percentage of packets in a flow that fall into a particular range. First, a window is defined. In these experiments, a somewhat arbitrary window of the first 200 bytes is used for packet length, and the first 200000 microseconds for inter-arrival time. Within these windows, a resolution is set. In this case, flowtbag calculates the percentage of packets that are in this window, with a resolution of 4. In other words, the packet lengths for the first 200 bytes (the first 200000 microseconds for inter-arrival time) are binned in ranges of 50 (50000 for inter-arrival time). The percentage of packets that land in each bin is calculated. A final (5th) feature counts the percentage of packets that did not land within the window. These features are used to create a new set of flow statistics for the data sets described in Section 5.2.1. An example of the sort of data this features would represent is given in Figure 7.1.

This approach compliments the earlier investigation into applications running within encrypted tunnels. It provides some initial investigation into flow measurements that can be made at any point during a tunnel. The features are duration and packet count agnostic. This would allow us to take "snapshot" measurements within long running tunnels that may contain multiple applications, such as those used by applications like tor [74]. It is likely that features that represent distributions such as these, if they are identifying characteristics, would be similar for a particular application over any sufficient length of time.

Table 7.1: Features

New features (expressed in total packet counts)	Replaced Features
1 to 50 bytes	minimum packet length
51 to 100 bytes	mean packet length
101 to 150 bytes	maximum packet length
151 to 200 bytes	standard deviation of packet length
over 200 bytes	
10000ms to 50000ms inter-arrival	minimum inter-arrival time
50001ms to 100000ms inter-arrival	mean inter-arrival time
100001ms to 150000ms inter-arrival	maximum inter-arrival time
150001ms to 200000ms inter-arrival	standard deviation of inter-arrival time
over 200000ms inter-arrival	

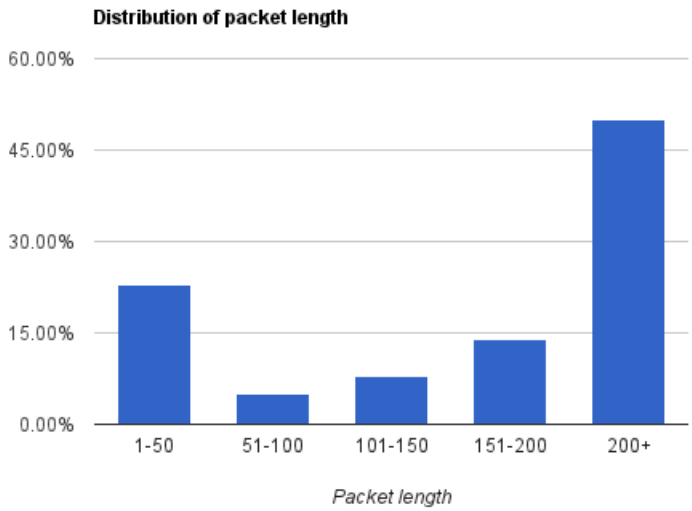

Figure 7.1: An example of the sort of data that is captured by the new features.

Chapter 8

Experiments & Results: New Flow Features

In the following sections, a new set of experiments are described and the results are presented and examined in detail. These experiments use the new set of features described in Section 7.1, and investigate the effects that these features have on the AdaBoost and C5.0 algorithms. The experiments contrast the results with the features used in the previous experiments. These experiments are preformed entirely disjoint of the experiments described in Chapter 5.

8.1 Pre-processing of the Data

The pre-processing of the data is done in a similar fashion to the first set of experiments, as it was described in Section 5.1. The data is first merged using mergecap. At this point, however, rather than using NetMate to process the data captures into flow statistics, flowtbag is used. Three sets of processed data (flows) are generated from each data set: a set containing the original features, a set containing the new features, and a set containing both the new and the old set of features. Furthermore, for this set of experiments, the 1st-second flows are not investigated. UDP flows that do not contain at least one packet in each direction are automatically filtered out by flowtbag. Labeling of the data sets is dealt with in the same manner as the preliminary investigation.

8.2 Usage of the Machine Learning Techniques

In the following set of experiments, a similar technique is used in creating appropriate test and training sets. Training sets were limited to the L1_UCIS3 and L2_UCIS3 training sets described in Table 5.6 since these seemed to provide the most useful combination of results and size. The training and test sets do not contain the same flows as were in the first set of experiments, but rather are entirely new, randomly

generated sets. However, each of these new training sets do contain the same flows, with the only difference being the features represented.

AdaBoost, and C5.0 are used in these experiments. SBB and naive Bayes are not used due to their performance observed in experiments discussed in Chapter 5. Furthermore, although the confidence factor of C5.0 was varied in the experiments discussed in Chapter 5, the results did not warrant doing this again – the confidence factor did not seem to change the overall performance in any identifiable pattern. So, in many ways, the following experiments are a simplified version of the earlier experiments, focusing on the effect of these new features rather than achieving the most optimal performance.

8.3 Results

The results of the following experiments are considered once again in terms of their detection rate (Eq. 5.1) and false positive rate (Eq. 5.2). A quick investigation of feature usage is also presented.

8.3.1 Results Summary

Table 8.1a shows the detection rate of each of the feature sets against each of the 5 data sets. Table 8.1b shows the false positive rate of each of the feature sets against each of the 5 data sets. The results of the first column under each algorithm, the original feature set, show very similar trends to the ones observed in Section 5.3.3. Using an entirely new randomly sampled training set in these experiments, and using a new program to generate these statistics, it is apparent that the new combination has not introduced any inconsistencies. The same drastic drops in detection rate experienced before is once again present on the NIMS4 data set. The detection rates and false positive rates are relatively close to the original set of experiments.

The results from Table 8.1 are most interesting, however, when comparing the original feature set results against the results using the new set of features. The effects can be more easily observed in Table 8.2.Table 8.2a shows the detection rate in terms of the point change (i.e. new percentage minus old percentage), as well as the actual percentage change that this had on the original value. The most noticeable change with the new features is the major increase in detection rate for both the

	C5.0			AdaBoost		
	ORIG	BINS	ALL	ORIG	BINS	ALL
UCIS3	95.02%	95.58%	96.32%	96.30%	96.56%	96.92%
UCIS2	95.78%	98.28%	97.10%	97.81%	98.45%	98.80%
NIMS4	66.51%	88.06%	72.43%	75.80%	97.17%	89.73%
MAWI2010	86.73%	87.81%	88.61%	88.60%	89.47%	90.02%
MAWI2011	75.91%	85.94%	88.21%	84.05%	87.03%	82.16%

(a) Detection rates

	C5.0			AdaBoost		
	ORIG	BINS	ALL	ORIG	BINS	ALL
UCIS3	2.62%	2.58%	2.40%	2.00%	1.90%	1.37%
UCIS2	13.05%	8.70%	8.97%	6.64%	4.97%	4.78%
NIMS4	11.42%	8.41%	6.81%	6.55%	3.56%	2.25%
MAWI2010	2.41%	1.87%	2.74%	2.96%	1.34%	1.89%
MAWI2011	2.55%	1.81%	2.21%	1.59%	1.16%	1.15%

(b) False positive rates

Table 8.1: Results of the original feature set, new feature set, and all features on each of the data sets

NIMS4 data set and the MAWI2011 data set. A closer look will reveal that using the new feature set, both C5.0 and AdaBoost have increased detection rate on all data sets. Furthermore, the C5.0 and AdaBoost algorithms also benefit from a decreased false positive rate on all data sets. While only minor effects are experienced for detection rate and false positive rate on the UCIS3 data set, the robustness tests show very noticeable positive effects. It is certainly positive to see that there were no negative effects from the use of these features. The new features help all tests achieve a detection rate of over 85%. This is a drastic change for some tests, an increase in detection rate of up to 32.40%. Moreover, false positive rates drop as much as 33.33% for C5.0 and 54.73% for AdaBoost. It is without a doubt that these features have been more beneficial to the machine learning algorithms. Most notably, the features have made a significant improvement on the robustness tests.

Evaluating the new features against using all features, one can observe even more interesting details. One of the more noteworthy details is that for robustness tests, there is actually an increase in false positive rate for about half of the tests. One can also see a decrease in detection rate on about half of the tests. It is clear that the

	C5.0		AdaBoost	
	Point Change	% Change	Point Change	% Change
UCIS3	0.56	0.59%	0.26	0.27%
UCIS2	2.50	2.61%	0.64	0.65%
NIMS4	21.55	32.40%	21.37	28.19%
MAWI2010	1.08	1.25%	0.87	0.98%
MAWI2011	10.03	13.21%	2.98	3.55%

(a) Detection rate

	C5.0		AdaBoost	
	Point Change	% Change	Point Change	% Change
UCIS3	-0.04	-1.53%	-0.10	-5.00%
UCIS2	-4.35	-33.33%	-1.67	-25.15%
NIMS4	-3.01	-26.36%	-2.99	-45.65%
MAWI2010	-0.54	-22.41%	-1.62	-54.73%
MAWI2011	-0.74	-29.02%	-0.43	-27.04%

(b) False positive rate

Table 8.2: Point change and percentage change using the new features over the old features for each data set.

combination of the features has caused some interfering information.

Based on these summary results, one could speculate that the combination of minimum, mean, maximum, and standard deviation for a particular characteristic provides useful information, but can be misleading, especially during robustness tests for SSL tunnel identification. As discussed in Chapter 6, this could be due to a number of things including different network configurations, congestion, and the applications being used on the network. These types of network conditions, which are unrelated to the way SSL operates and presents itself, will likely shift many of the original features such as minimum, maximum, and mean values.

8.3.2 Feature Usage

Another way to examine these results is to take a deeper look into the solutions developed by these algorithms and investigate their choices of features. Feature usage is easiest to determine from the C5.0 algorithm. The number of times that a feature is used during training can be counted and can provide a percentage that gives an indication of how important this feature is to the model. AdaBoost, on the other

hand, is more difficult to quantify due to the nature of AdaBoost being a boosting algorithm. Multiple evaluations are done for each test instance by AdaBoost, and analysis is further complicated by weights for each sub model. For this reason, the features used by C5.0 are analyzed.

Table 8.3 shows the top 5 most used features of the C5.0 algorithm for this set of experiments. In three sub-tables, the data is given for using NetMate's original feature set, bins, and all features.

It is immediately apparent that using the original set of features provided by NetMate, the packet length is important. Packet length attributes consist of 4 of the top 5 attributes, with forward push count making its way in as the only non-packet length related feature. Surprisingly, this domination of packet length features is not seen when using the binning method. Instead, the algorithms focus more on time based features. With the bins, inter-arrival time based features make up 3 of the top 5. Packet length is still used as the first feature to split on, and once again forward push count keeps its place on the list. Using all features, there is no obvious point of focus for C5.0. Packet length and time based features make up 2 of the 5 each, and as always forward push count stays on the list of the top 5.

The analysis is promising for many reasons. These feature usage tables combined with the test results agree with early speculation that time based features are more useful for SSL robustness tests. The results indicate that it is possible to achieve comparable or better detection and false positive rates, and increase robustness of the learning models by using time based features. It seems likely there are better ways of representing the time based features, since this is just one way of representing these features. Through analysis, it is likely an investigator could define better values for binning, or even better ways of representing the features.

Table 8.3: C5.0 Feature Usage

NetMate		Bins		All	
Usage	Attribute	Usage	Attribute	Usage	Attribute
100%	max_fpktl	100%	fpktl5	100%	max_fpktl
78%	std_fpktl	59%	fpsh_cnt	59%	fpsh_cnt
56%	fpsh_cnt	48%	biat5	54%	fiat1
54%	max_bpktl	46%	biat1	47%	std_fpktl
48%	min_fpktl	44%	fiat1	43%	std_idle

Chapter 9

Conclusion & Future work

In this work, four machine learning techniques are used to investigate the problem of identification of applications within an SSL tunnel without using payload, port numbers, or IP addresses. Specifically, web browsing behaviour is explored within SSL tunnels as a case study, and the machine learning algorithms naive Bayes, AdaBoost with J48, C5.0, and Genetic Programming are evaluated to identify such a behaviour. Results indicate that by using only the first second of a network traffic flow, SSL can be identified by obtaining a 94.60% detection rate and only 2.10% false positive rate with C5.0 when tested on the same network as the training data is from. The results for robustness testing (ie. testing on a different network than the model is trained on) are also promising with detection rates from 82.60%–88.14% and false positive rates from 1.40%–3.61%. In identifying web browsing within these tunnels, the AdaBoost model has the best performance with a detection rate of 97.53% and a false positive rate of 3.13%. Robustness tests give detection rates between 77.43% and 79.64%, and false positives rates from 5.23%–5.73%. The push count is found to be an important factor in identifying both SSL and web browsing within SSL tunnels, and packet length based features are also very important.

The research is further extended by investigating the features used, based off the discoveries made. While investigating SSL, SBB preformed well on robustness tests using time based features, but other machine learning algorithms did not seem to make much use of these features. Furthermore, it was obvious that an entire SSL flow is not necessary in order to be detected. Therefore, a new tool is created to experiment with features that compliment some of the questions left. While time based features seem very important for robustness tests, it is also apparent that packet length based features have an important role to play in detecting SSL. The new experiments show promising results for both C5.0 and AdaBoost. Detection rate is increased by up to 32.40% on robustness tests, and false positive rates drop as

much as 54.73%. These changes can easily be attributed to the new features, as they drastically increased the usage of time based features, as speculated. Furthermore, the new features out-preformed the old feature set in every case.

Identifying useful features for flow based statistics has always been an area of investigation and debate. Many machine learning algorithms such as decision trees are not able to consider relations among features such as the ratio of forward packet length to backward packet length, and may need to be explicitly given to machine learning algorithms in the form of new features. Going forward, further investigation needs to take place into what measurements are important for identifying applications. Moreover, the set of important features may be specific to the application. An investigation into what makes these features important is warranted to fully understand how to best utilize or model the characteristics present in particular applications.

Further investigation in identifying encrypted web browsing traffic might look in more detail at anonymization networks such as the Tor Project [74], which make web browsing easy to encrypt and disguise. The Tor Project uses a virtual circuit of encrypted tunnels and bounces encrypted communications around the world. Tor allows a user to run many sorts of applications within these tunnels including peer-to-peer and web browsing applications.

More work needs to be done on properly generating realistic SSL flows, including those containing web browsing activity. Traffic generation is a difficult task, but is necessary because data sets that contain labeled traffic are very rare. It is difficult to know the ground truth of captures that do not contain payload information, and payload information is rarely, if ever, given.

Future work might also explore the effect that observing a wireless network may have on the techniques described in this thesis. Wireless networks employ different protocols than standard wired connections, and often use an additional set of encryption techniques due to the nature of wireless communication. Furthermore, IP version 6 provides another avenue in which the patterns of the SSL protocol may be altered. IP version 6 will contain a lot of the same information as IP version 4, however the identifying attributes in which applications such as SSL display themselves may change.

Comparing the measurable benefits and disadvantages of using hybrid approaches

instead of the purely flow-based approach described in this thesis would provide some more insight into the feasibility of such classifiers. Hybrid approaches offer a trade off by allowing the use of some payload information to boost performance at the cost of increased resources.

In this thesis, SBB version 3 was adopted to avoid the $O(n^2)$ cost of training (where n is the size of the team population) associated with version 1 of SBB. However, under tasks with discrete outcomes such as classification, Pareto co-evolution is significantly more accurate. Future research may also consider using SBB version 1 under a streaming context (limited to training exemplars selected from within a sliding window). This is significantly faster than SBB version 1 and, as long as label information is reliable, at least as accurate (and therefore more accurate than the SBB version used in this thesis) [75].

Bibliography

[1] "Internet assigned numbers authority," http://www.iana.org/.

[2] J. Owens and J. Matthews, "A study of passwords and methods used in brute-force ssh attacks," *Department of Computer Science Clarkson University*, 2008.

[3] T. Karagiannis, A. Broido, N. Brownlee, k. claffy, and M. Faloutsos, "Is P2P dying or just hiding?" in *Global Internet and Next Generation Networks*. Dallas, Texas: Globecom 2004, Dec 2004.

[4] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *SIGCOMM Comput. Commun. Rev.*, vol. 36, pp. 5–16, October 2006.

[5] A. Madhukar and C. Williamson, "A longitudinal study of p2p traffic classification," in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, ser. MASCOTS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 179–188.

[6] P. A. Networks, "The application usage and risk report (7th edition)," http://www.paloaltonetworks.com/literature/forms/aur-report.php, May 2011.

[7] J. Viega, M. Messier, and P. Chandra, *Network security with OpenSSL*. OReilly Media, 2002.

[8] D. Wagner and B. Schneier, "Analysis of the ssl 3.0 protocol," in *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996, pp. 29–40. [Online]. Available: http://www.usenix.org/publications/library/proceedings/ec96/full_papers/wagner/wagner.pdf

[9] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246 (Proposed Standard), Internet Engineering Task Force, Jan. 1999, obsoleted by RFC 4346, updated by RFCs 3546, 5746. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[10] Microsoft, "What is tls/ssl?" http://technet.microsoft.com/en-us/library/cc784450(WS.10).aspx, Mar 2003.

[11] P. Rogaway, "Problems with proposed ip cryptography," Feb 2002, http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt.

[12] P. A. Networks, "The application usage and risk report (7th edition)," http://www.openssl.org/~bodo/tls-cbc.txt, Feb 2002.

[13] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346 (Proposed Standard), Internet Engineering Task Force, Apr. 2006, obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746. [Online]. Available: http://www.ietf.org/rfc/rfc4346.txt

[14] ——, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[15] N. M. Adam Langley and W.-T. Chang, "Overclocking ssl," http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html.

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), Internet Engineering Task Force, Jun. 2002, updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[17] "Enabling ssl in ibm lotus sametime," http://www.ibm.com/developerworks/lotus/documentation/sametimessl/.

[18] "Why openvpn uses tls," http://openvpn.net/index.php/open-source/337-why-openvpn-uses-tls.html.

[19] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*. Berkeley, CA, USA: USENIX Association, 1998, pp. 3–3.

[20] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*, ser. LISA '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.

[21] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *In PAM*, 2005, pp. 41–54.

[22] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Perform. Eval.*, vol. 64, no. 9-12, pp. 1194–1213, 2007.

[23] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[24] A. B. Mohd and D. S. bin Mohd Nor, "Towards a flow-based internet traffic classification for bandwidth optimization," 2009.

[25] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: identifying ssh and skype," in *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, ser. CISDA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 289–296.

[26] R. Yuan, Z. Li, X. Guan, and L. Xu, "An svm-based machine learning method for accurate internet traffic classification," *Information Systems Frontiers*, vol. 12, pp. 149–156, April 2010.

[27] M. Soysal and E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Perform. Eval.*, vol. 67, pp. 451–467, June 2010.

[28] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Comput. Netw.*, vol. 53, pp. 81–97, January 2009.

[29] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *J. Mach. Learn. Res.*, vol. 7, pp. 2745–2769, December 2006.

[30] A. Yamada, Y. Miyake, K. Takemori, A. Studer, and A. Perrig, "Intrusion detection for encrypted web accesses," in *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 01*, ser. AINAW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 569–576.

[31] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Proceedings of the 8th international conference on Passive and active network measurement*, ser. PAM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 165–175.

[32] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Detecting http tunnels with statistical mechanisms," in *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*. IEEE, 2007, pp. 6162–6168.

[33] ——, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 5–16, January 2007.

[34] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Detection of encrypted tunnels across network boundaries," in *Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008*. IEEE, 2008, pp. 1738–1744.

[35] N. Schear and D. M. Nicol, "Performance analysis of real traffic carried with encrypted cover flows," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 80–87.

[36] D. M. Nicol and N. Schear, "Models of privacy preserving traffic tunneling," *Simulation*, vol. 85, pp. 589–607, September 2009.

[37] A. Dainotti, W. de Donato, A. Pescape, and P. Salvo Rossi, "Classification of network traffic via packet-level hidden markov models," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, 2008, pp. 1 –5.

[38] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for tcp traffic classification," *Computer Networks*, vol. 53, pp. 2476–2490, September 2009.

[39] R. Alshammari and A. N. Zincir-Heywood, "Investigating two different approaches for encrypted traffic classification," in *Proceedings of the 2008 Sixth Annual Conference on Privacy, Security and Trust*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 156–166.

[40] R. Alshammari, A. N. Zincir-Heywood, and A. A. Farrag, "Performance comparison of four rule sets: An example for encrypted traffic classification," in *Proceedings of the 2009 World Congress on Privacy, Security, Trust and the Management of e-Business*, ser. CONGRESS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 21–28.

[41] R. Alshammari and N. Zincir-Heywood, "Generalization of signatures for ssh encrypted traffic identification," in *Computational Intelligence in Cyber Security, 2009. CICS '09. IEEE Symposium on*, 2009, pp. 167 –174.

[42] A. Callado, J. Kelner, D. Sadok, C. Alberto Kamienski, and S. Fernandes, "Better network traffic identification through the independent combination of techniques," *Journal of Network and Computer Applications*, vol. 33, pp. 433–446, July 2010.

[43] C. Bacquet, K. Gumus, D. Tizer, A. N. Zincir-heywood, and M. I. Heywood, "A comparison of unsupervised learning techniques for encrypted traffic identification," *Journal of Information Assurance and Security*, vol. 5, pp. 464–272, 2010. [Online]. Available: https://www.cs.dal.ca/sites/default/files/technical_reports/CS-2009-09.pdf

[44] C. Bacquet, a. N. Zincir-Heywood, and M. I. Heywood, "Genetic optimization and hierarchical clustering applied to encrypted traffic identification," *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp. 194–201, Apr. 2011.

[45] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, dec. 2010, pp. 1 –5.

[46] A. Dainotti, A. Pescapé, and C. Sansone, "Early classification of network traffic through multi-classification," in *Proceedings of the Third international conference on Traffic monitoring and analysis*, ser. TMA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 122–135.

[47] W. Lu and A. Ghorbani, "A multiple-stage classifier for identifying unknown internet traffic," in *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, aug. 2011, pp. 725 –729.

[48] R. Alshammari and A. Zincir-Heywood, "Unveiling skype encrypted tunnels using gp," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, july 2010, pp. 1 –8.

[49] R. Alshammari and A. N. Zincir-Heywood, "Can encrypted traffic be identified without port numbers, ip addresses and payload inspection," *Computer Networks*, vol. 55, no. 6, pp. 1326–1350, 2011.

[50] ——, "Is machine learning losing the battle to produce transportable signatures against voip traffic," in *IEEE Congress on Evolutionary Computation*, 2011, pp. 1543–1550.

[51] C. McCarthy and A. Zincir-Heywood, "An investigation on identifying ssl traffic," in *Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on*, april 2011, pp. 115 –122.

[52] N. Brownlee, C. Mills, and G. Ruth, "Traffic Flow Measurement: Architecture," RFC 2722 (Informational), Internet Engineering Task Force, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2722.txt

[53] S. Handelman, S. Stibler, N. Brownlee, and G. Ruth, "RTFM: New Attributes for Traffic Flow Measurement," RFC 2724 (Experimental), Internet Engineering Task Force, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2724.txt

[54] N. Brownlee, "Traffic Flow Measurement: Meter MIB," RFC 2720 (Proposed Standard), Internet Engineering Task Force, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2720.txt

[55] R. Research, "C5.0." [Online]. Available: http://www.rulequest.com/

[56] ——, "Is see5/c5.0 better than c4.5?" [Online]. Available: http://www.rulequest.com/see5-comparison.html

[57] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[58] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[59] P. Lichodzijewski and M. I. Heywood, "Symbiosis, complexification and simplicity under gp," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 853–860.

[60] (2009) Blue coat packetshaper. [Online]. Available: http://www.bluecoat.com/products/packetshaper

[61] (2006) Mawi working group traffic samplepoint-f. [Online]. Available: http://tracer.csl.sony.co.jp/mawi/

[62] D. Arndt. (2011) Bfake. [Online]. Available: http://dan.arndt.ca/projects/bfake/

[63] M. Trojnara. stunnel. [Online]. Available: http://www.stunnel.org/

[64] P. Developers. Privoxy. [Online]. Available: http://www.privoxy.org/

[65] Mozilla. Firefox. [Online]. Available: http://www.mozilla.org/

[66] Alexa. Top sites. [Online]. Available: http://www.alexa.com/

[67] W. Foundation. Wireshark. [Online]. Available: http://www.wireshark.org/

[68] S. Zander and C. Schmoll. (2009) Netmate 0.9.5. [Online]. Available: http://www.ip-measurement.org/tools/netmate/

[69] S. Zander and N. Williams. (2009) netai 0.1. [Online]. Available: http://caia.swin.edu.au/urp/dstc/netai/netai.html

[70] P. Lichodzijewski and J. Doucette. Symbiotic bid-based (sbb) gp source code. [Online]. Available: http://web.cs.dal.ca/~mheywood/Code/SBB/

[71] D. Arndt. (2011) flowtbag. [Online]. Available: http://dan.arndt.ca/projects/flowtbag/

[72] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 255–263.

[73] G. Bissias, M. Liberatore, D. Jensen, and B. Levine, "Privacy vulnerabilities in encrypted http streams," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, G. Danezis and D. Martin, Eds., 2006, vol. 3856, pp. 1–11.

[74] Tor project: Anonymity online. [Online]. Available: https://www.torproject.org/

[75] A. Atwater, M. I. Heywood, and N. Zincir-Heywood, "Gp under streaming data constraints: a case for pareto archiving?" in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, ser. GECCO '12. New York, NY, USA: ACM, 2012, pp. 703–710.