A PRELIMINARY INVESTIGATION OF DIFFERENT FEATURE SETS FOR SKYPE TRAFFIC CLASSIFICATION USING MACHINE LEARNING

by

Aaron Atwater

Submitted in partial fulfillment of the
requirements for the degree of
Bachelor of Computer Science, Honours

at

Dalhousie University
Halifax, Nova Scotia
April 2011

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Computer Science for acceptance a thesis entitled "A PRELIMINARY INVESTIGATION OF DIFFERENT FEATURE SETS FOR SKYPE TRAFFIC CLASSIFICATION USING MACHINE LEARNING" by Aaron Atwater in partial fulfillment of the requirements for the degree of Bachelor of Computer Science, Honours.

Dated: April 28, 2011

Supervisor: _____
N. Zincir-Heywood

Reader: _____
M. Heywood

DALHOUSIE UNIVERSITY

DATE: April 28, 2011

AUTHOR: Aaron Atwater

TITLE: A PRELIMINARY INVESTIGATION OF DIFFERENT FEATURE SETS FOR SKYPE TRAFFIC CLASSIFICATION USING MACHINE LEARNING

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: B.C.Sc. (Honours) CONVOCATION: May YEAR: 2011

_____

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

The purpose of this work is to investigate using machine learning algorithms to classify encrypted Skype traffic. Using real data collected from Dalhousie University's campus network, it compares the performance of the C4.5, AdaBoost, and NaiveBayes algorithms in distinguishing between Skype and non-Skype traffic. A comparison is also made between two different feature sets explored in previous works, as applied to Skype traffic: one intended for encrypted SSH traffic, and another intended as a minimalist Voice over IP (VoIP) identification feature set.

An important aspect of these feature sets is that they forego usage of IP address, port, and payload information among the captured traffic. Traffic classification methods have traditionally made heavy use of these features alone for identification. However, IP address and port numbers are easily changed by application developers wishing to make their programs more difficult to detect and/or block. Skype, for example, can use ports 80 and 443 for communication, which are typically reserved for HTTP and HTTPS traffic, respectively. Payload information is easily obscured by encrypting the contents of the application's data packets. Ignoring payload contents also allows our identification method to avoid privacy concerns raised by the use of Deep Packet Inspection (DPI) techniques.

Another goal of the research is to generate easily reproducible results. With the exception of the dataset itself, all tools employed in generating its results are freely available over the internet and open-source.

Acknowledgements

I wish to thank the National Science and Engineering Research Council for the Undergraduate Student Research Award Program which provided much of the funding for this work.

Great thanks also go to my supervisor, Dr. Nur Zincir-Heywood, for her guidance and direction over the past two years.

Thanks also to my reader, Dr. Malcolm Heywood, and the Honours Coordinator, Dr. Thomas Trappenberg.

Chapter 1

Introduction and Background

Various network management and network forensics tasks rely on the ability to correctly distinguish one type of application traffic from another. Bandwidth management and quality of service functions are essential in most large networked environments. Network usage policies often rely on application detection for their creation or enforcement. Understanding bandwidth usage allows network managers to know what infrastructure to upgrade and when. The ability to correctly detect malicious applications is also invaluable in the field of network security, as it allows vulnerabilities in the network to be found and eliminated, or even prevented in the first place.

Traditional network traffic classification methods have depended heavily on the explicit features of source and destination IP addresses, source and destination ports, and payload contents contained within packets to uniquely associate them with an application. However, numerous developments in the field of networking itself are quickly making traffic classification algorithms that rely solely on these features unreliable at best, and deceptive at worst. For example, the use of Virtual Private Networking, and the establishment of public anonymous VPN servers, allows a user to encrypt all of these features for transit between themselves and an endpoint that may lie beyond the purview of an officiating party. Indeed, any sufficiently strong form of payload encryption is enough to thwart most appliances relying on deep packet inspection (DPI) for traffic identification.

The rising popularity of peer-to-peer network architectures for applications such as file sharing, video streaming, and voice communication, means identification based solely on comparing IP addresses will not remain feasible for very long. Skype connects to login.skype.com only once, at the beginning of a user's session, for authentication and updating purposes before switching to a peer-to-peer networking model. The BitTorrent file sharing protocol has incorporated distributed hash tables (DHT)

in order to remove the need for centralized servers for coordination between peers. Some forms of malware [12, 16] have also incorporated complex date-based algorithms for generating domain names into their command and control procedures, making a blacklist of IP addresses for such an application generally impossible. The other traditional method of identifying and blocking network traffic is via the use of port numbers. However, many applications have long since caught on to this and taken advantage of the practice for their own benefits. Skype will use ports 80 and 443 for communication unless explicitly configured by the user not to do so, whereas these ports have traditionally been reserved for HTTP communication. Since port 443 is widely accepted as the port for encrypted HTTP communication, a traffic identification method looking only for known port numbers and HTTP headers will incorrectly identify any such Skype communication as being HTTPS traffic.

One alternative to depending upon the above-mentioned features is to instead use characteristics of the network traffic itself. For example, while Voice over IP (VoIP) clients may send encrypted data over port 443, it will generally exhibit a sustained traffic pattern that is not present in the case of simple web page downloads. This thesis will present two such sets of features, and evaluate their use in classifying actual network data against known applications. This will be done using the machine learning algorithms C4.5, AdaBoost, and Naïve Bayesian, which are selected based upon their variability and general success in previous work by researchers in the Dalhousie NIMS Lab. The algorithms themselves are explained in more detail in Section 3.1.3. It will also attempt to classify the traffic using only the packet header information available from the traffic, without using statistics taken from the network flows. This is done by looking at packets on an individual basis, as well as using a sliding window technique that is commonly implemented in hardware for efficiency in analyzing online traffic [15].

Chapter 2

Literature Survey

There is a substantial body of work examining how the Skype protocol itself works without attempting to decrypt the encrypted portion of its traffic [5, 6, 11]. Skype performs user authentication through a central server, but then uses a peer-to-peer network architecture for actual voice communication. Clients with high-quality connections tend to be promoted to act as "supernodes" in the Skype network, helping to route calls and other information for their peers.

Some work proposes actively tracking the IP addresses of promoted supernodes in order to classify Skype traffic by looking for connections to these addresses [18]. Although this may be successful in the general case, it requires a significant investment in active infrastructure for tracking these supernodes, and does not suit our goal of ignoring IP address information.

Another detection method proposed in [11] requires payload information (obtained when an unencrypted HTTP GET request is sent from the Skype automatic updater when a user logs in), port number (specifically TCP port 33033) and IP address information. It also requires logging and performing calculations on any particular connection starting right from the initial user authentication procedures. It also relies on checking exact packet byte-lengths, which is extremely vulnerable to changes in the Skype protocol as a detection method. A similar method is employed in [17] which, although able to successfully prevent users from logging in through an ordinary PC Skype client, would be unable to detect Skype traffic if the unencrypted initial packets are not captured for analysis, and is again vulnerable to subtle changes such as value rearrangement in the Skype protocol.

Some work such as that in [8] attempts to distinguish Skype traffic running on ports 80 or 443 from ordinary HTTP or HTTPS traffic. The aim of this thesis, however, is to be able to classify Skype traffic from any other type regardless of the port it is running on.

A group at the University Politecnico di Torino in Italy presents a detailed analysis of Skype traffic and uses a machine learning-based approach to classifying it [5]. However, their method uses several bytes of information extracted from the payload of unencrypted packets sent near the beginning of a Skype session, and also attempts to actively connect to TCP port 33033 of suspected Skype hosts to check their response to an HTTP GET request. This thesis aims to classify Skype traffic in a completely passive manner, ignorant of any payload or IP/port information. However, [5] also makes available a significant data set containing Skype traffic which is used for testing robustness through the next chapters (see Section 3.1.2 for more information on this).

Researchers in Budapest [14] used a machine learning approach to classifying Skype network traffic based on low-level characteristics. Their classification method was replicated and used to benchmark the results obtained in this thesis, and the results are presented in Section 3.2.3.

Chapter 3

Skype Flow-Level Classification

This chapter presents the experiments performed on classifying Skype network traffic at the network flow abstraction level. An overview of the methodology and data sets used is given in Section 3.1. The results of various flow level experiments are presented in Section 3.2, and the chapter closes in Section 3.3 with an analysis of the best classification models obtained throughout the experiments.

3.1 Overview

This section outlines the software tools used for preprocessing and processing the network traffic data in subsection 3.1.1, and then introduces the two different data sets used and how they were obtained in subsection 3.1.2.

3.1.1 Process

All network traffic data is collected and stored in raw packet form using the tcpdump format. It is then processed using NetMate and the NetAI flowstats module to extract the desired feature vectors in the Attribute-Relation File Format (ARFF) for passing off to the machine learning suites used. Slight modifications were made to the open-source NetAI flowstats module to enable it to calculate all the features required for this experiment (specifically, the number of ACK packets in a flow and the mode of packet interarrival times). As packet header fields are already parsed into an indexed data structure by NetAI, these two additional statistics were added alongside the calculation of similar ones. Once processed, the datasets were then used with the WEKA machine learning suite [9] for resampling and the actual training/testing of the machine learning algorithms.

WEKA is a free and open source program written in the Java programming language for processing data sets and using them to train and test a large number of machine learning algorithms. Training refers to the process of giving a machine

Table 3.1: Minimalist Feature Set

Duration	bytes/s, forward
packets/s, forward	bytes/s, backward
packets/s, backward	# ACK packets
Protocol ID	Maximum packet length observed
Minimum packet length observed (not counting ACK, RST, SYN packets)	

Table 3.2: Encrypted Traffic Feature Set

min_fpktl	std_fiat	max_bpktl	proto
mean_fpktl	min_biat	std_bpktl	total_fpackets
max_fpktl	mean_biat	min_fiat	total_fvolume
std_fpktl	max_biat	mean_fiat	total_bpackets
min_bpktl	std_biat	max_fiat	total_bvolume
mean_bpktl	duration		

learning algorithm an initial subset of data from which it attempts to extract patterns which can be used, in this case, to partition the data into a set of classes known *a priori*; the collection of these patterns is then referred to as the model. Testing is the process of then applying this model to the superset of data to see if the same patterns are exhibited.

The particular feature sets extracted for this work are shown in table 3.1 and table 3.2. They were derived from previous investigations done by the researchers at Dalhousie NIMS Lab into finding effective feature sets for traffic classification [3, 4]. The feature set of table 3.2 was designed with the intent of classifying encrypted traffic (and SSH in particular), whereas the feature set of table 3.1 was designed to be a minimalist feature set for classifying VoIP traffic.

3.1.2 Data Sets

The primary set of data for this experiment was provided by Dalhousie University's University Computing & Information Services (UCIS). The dataset was created by recording traffic leaving and entering the campus network for one day in 2007. Due to privacy concerns, the traffic was first anonymized by scrambling the IP addresses, dropping the payload, and setting any checksum fields to zero (as this could potentially be used to recover the contents of short packets). It is important to note,

however, that the length header fields of all packets were left intact. This dataset consists of approximately 44 million flows.

Before this anonymization, the recorded traffic was run through a commercial application-layer classification program called PacketShaper. However, at the time of the data set's creation, PacketShaper was able to identify Skype traffic running over UDP, but unable to correctly identify TCP Skype traffic. To help make up for this deficiency, a second dataset was introduced.

The Politecnico di Torino (PoliTo) in Italy makes publicly available a set of Skype traffic recorded on their own testbed network. This dataset contains both TCP and UDP Skype traffic, and is further subdivided into various useful categories such as ingoing or outgoing, Skype-to-Skype or Skype-to-landline, voice codec used during the call, etc. [6]. A codec is an algorithm for encoding and decoding analog waveforms (audio, in this case) to prepare them for digital transmission [7].

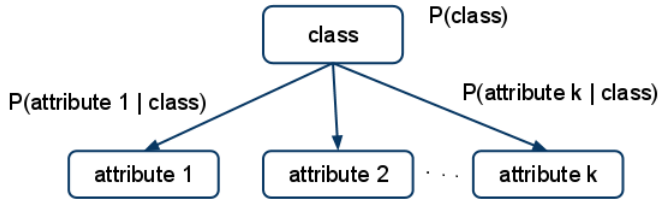
The primary PoliTo dataset consists of approximately 390 thousand flows. There is also a set of ten individual traces used, each of which corresponds to a call made exclusively using a specified voice codec and consists of approximately 200 flows.

3.1.3 Machine Learning Algorithms

The Naïve Bayesian classification algorithm uses a standard Bayesian network, in which a node X is connected to a node Y if the value of X has a direct influence on the value of Y. Training data is examined to iteratively update the probabilities of each node, and Bayes' theorem is applied to establish a *diagnosis* by inverting the dependencies. In the Naïve form of this classifier, it is assumed that the individual attributes of an instance are independent, leading to a Bayesian network similar to that shown in Figure 3.1.3. For more details on the Naïve Bayesian algorithm the reader is referred to [1].

The AdaBoost machine learning algorithm, short for "Adaptive Boosting", is a modification of the boosting process for machine learning designed to allow boosting to be performed with only a single set of training data, as opposite to the three or more partitions that training data must be divided into with ordinary boosting. AdaBoost repeatedly select a base learner from a pool of weak learners and evaluates it against training data; if it performs well, the probability of selecting it again is reduced so that

Figure 3.1: Naïve Bayesian Classification Network Example [1]



the poor classifiers generally receive more training. The testing method for AdaBoost is then to take a weighted vote from the results of all learners in the final pool. For these experiments, the WEKA defaults of using decision stumps as the base learner and reweighting (as opposed to resampling the learners) are used. For more details on the AdaBoost algorithm the reader is referred to [1].

C4.5 is a decision tree classification algorithm. This means that its final decision model is in the form of a (in our case) binary tree, in which each branch contains a threshold value for a single attribute, and the tree is traversed recursively with the values of a testing instance until a leaf node is reached, contained only the output of the decision function. The tree itself is generated recursively, by searching for an attribute threshold that partitions the training data into two sets with minimum possible classification entropy amongst the instances in each set. For more information on the C4.5 algorithm the reader is referred to [1].

3.2 Training and Testing Results

This section presents the results from all the experiments performed at the network flow abstraction level. Section 3.2.1 outlines the results from experiments performed solely on the Dalhousie data set. Section 3.2.2 presents the results obtained by combining the Dalhousie and PoliTo data sets, and Section 3.2.3 presents the results of a classification method derived by another research group for comparison and benchmark purposes.

WEKA automatically performs 10-fold cross-validation during training, and the results from this are included in all results tables. N-fold cross-validation is when training instances are divided into N subsets, and each subset is used as the testing data for a model trained by all others. Although the final model used for testing

is trained across all the training exemplars, cross-validation can also be used as a prediction of the classifier's performance against unseen data.

3.2.1 Isolated Dalhousie Dataset

As the Dalhousie dataset is quite large, a 1% sample (consisting of approximately 440 thousand flows) was taken from it for training purposes. Training models were developed using the machine learning algorithms C4.5, Naïve Bayesian, and AdaBoost. It was determined that C4.5 clearly performed best for the given task, as outlined in the results summary tables 3.3 through 3.5. The Naïve Bayesian algorithm was found to run extremely slowly for both training and testing–for a dataset as large as the Dalhousie data set used in these experiments, it was deemed unfeasibly slow in exchange for the classification rates obtained and thus it was rejected as a viable candidate for further experimentation. In light of this decision and time constraint, some testing results have been omitted from Table 3.4.

Table 3.3: Isolated Dalhousie Dataset with C4.5

	Minimalist Feature Set		Encrypted Traffic Feature Set	
Training	TP	0.979	TP	0.985
	TN	0.993	TN	0.995
	FP	0.007	FP	0.005
	FN	0.021	FN	0.015
Training Cross-Validation	TP	0.972	TP	0.981
	TN	0.992	TN	0.994
	FP	0.008	FP	0.006
	FN	0.028	FN	0.019
Testing Dal Dataset	TP	0.9724	TP	0.9806
	TN	0.9917	TN	0.9938
	FP	0.02761	FP	0.01943
	FN	0.008280	FN	0.006203
Testing PoliTo Dataset	TP	0.4719	TP	0.6426
	FN	0.5281	FN	0.3574

3.2.2 Combined Dataset

As tables 3.3-3.5 demonstrate, the C4.5 machine learning algorithm appears to provide the best classifier performance for Skype traffic classification, and so the C4.5 training model was chosen to be evaluated across the independent Politecnico di

Table 3.4: Isolated Dalhousie Dataset with Naïve Bayesian

	Minimalist Feature Set		Encrypted Traffic Feature Set	
Training	TP	0.976	TP	0.934
	TN	0.810	TN	0.969
	FP	0.190	FP	0.031
	FN	0.024	FN	0.066
Training Cross-Validation	TP	0.975	TP	0.930
	TN	0.806	TN	0.968
	FP	0.194	FP	0.032
	FN	0.025	FN	0.070
Testing Dal Dataset	TP	omitted	TP	omitted
	TN	omitted	TN	omitted
	FP	omitted	FP	omitted
	FN	omitted	FN	omitted

Table 3.5: Isolated Dalhousie Dataset with AdaBoost

	Minimalist Feature Set		Encrypted Traffic Feature Set	
Training	TP	0.596	TP	0.702
	TN	0.948	TN	0.986
	FP	0.052	FP	0.014
	FN	0.404	FN	0.298
Training Cross-Validation	TP	0.596	TP	0.701
	TN	0.948	TN	0.986
	FP	0.052	FP	0.014
	FN	0.404	FN	0.299
Testing Dal Dataset	TP	0.5956	TP	0.7017
	TN	0.9480	TN	0.7227
	FP	0.4044	FP	0.2983
	FN	0.05197	FN	0.2773

Torino dataset. The model obtained near 100% detection rate for UDP traffic, but anywhere from 0-66% detection rate for TCP Skype traffic. This was due to the absence of TCP Skype traffic present in the Dalhousie dataset, so the training sample was augmented with TCP traffic taken from the PoliTo dataset to improve its performance. The results from this are summarized in table 3.6. (Note that the 2.3% / 1.8% false-negative rate is most likely a result of the presence of misclassified Skype TCP flows present in the Dalhousie dataset.)

Table 3.6: Combined Dataset with C4.5

	Minimalist Feature Set		Encrypted Traffic Feature Set	
Training	TP	0.996	TP	0.997
	TN	0.992	TN	0.994
	FP	0.008	FP	0.006
	FN	0.004	FN	0.003
Training Cross-Validation	TP	0.995	TP	0.997
	TN	0.990	TN	0.993
	FP	0.010	FP	0.007
	FN	0.005	FN	0.003
Testing Dal Dataset	TP	0.9727	TP	0.9815
	TN	0.9907	TN	0.9928
	FP	0.00935	FP	0.00716
	FN	0.02737	FN	0.01847
Testing PoliTo Dataset	TP	0.9757	TP	0.9624
	FN	0.0243	FN	0.0376

3.2.3 Budapest University of Technology & Economics Experiment Replication

The method for Skype traffic identification presented in [14] was replicated, as it provided the most similar approach to classification of all the work reviewed in Chapter 2 without incorporating IP address, port number or payload information in the feature set used. This was done to provide a classification benchmark and comparison against pre-existing proposals of methods for classifying Skype network traffic. It was done using source-code modifications to the NetAI flowstats module to compute extra flow statistics, and an AWK script was employed for testing against the Dalhousie and Politecnico di Torino datasets in ARFF format.

The flow-level features used by the group at Budapest University (BME) were

Table 3.7: Testing Results against PoliTo Callset

	Dal-only		Dal+PoliTo		
Proto+codec	Minimalist	Encrypted	Minimalist	Encrypted	BME
SVOPC	1.000000	0.968750	1.000000	1.000000	0.000000
UDP_EG711A	0.505435	0.521739	1.000000	0.994565	0.000000
UDP_EG711U	0.000000	0.392405	0.772152	0.886076	0.000000
UDP_G729	0.598684	0.585526	1.000000	0.993421	0.000000
UDP_iLBC	0.611111	0.950617	1.000000	1.000000	0.000000
UDP_IPCMWB	0.006667	0.986667	0.993333	0.986667	0.000000
UDP_PCMA	0.505618	0.516854	1.000000	0.988764	0.000000
UDP_PCMU	0.500000	0.512195	1.000000	0.987805	0.000000
TCP_ISAC	0.000000	0.000000	1.000000	0.795699	0.000000
UDP_ISAC	0.991071	0.991071	0.991071	0.991071	0.991150

the mode of interarrival times (mode_iat), the number of packets send in the forward direction (fpkt_persec), the average packet length within the flow (mean_pktl), and the forward traffic bandwidth (fwd_bit_persec).

The exact method presented in the paper [14] performed very poorly against our datasets (0.5% detection rate against the Dal dataset and 0% detection rate against the Politecnico di Torino dataset), most likely due to the provided threshold values being constrained to traffic artificially limited to using a single voice codec for data transmission. A codec is a method of compressing/decompressing a digital data stream for transmission, with different codecs providing different loss (and therefore transmission) rates. The codecs used for voice transmission by Skype are elaborated in [5].

This theory was verified by testing all current models against the PoliTo callset, and the results are shown in Table 3.7. All models generated in the previous section gave comparable performance to the replicated model, and the results demonstrate that this machine learning method may be more generalizable to identifying Skype traffic in an unconstrained environment.

3.3 C4.5 Decision Tree Analysis

The number of occurrences of each feature in a typical C4.5 decision tree is shown in table 3.8. The C4.5 decision tree generated using the encrypted traffic feature set shown in Table 3.2 uses interarrival times to make the largest differentiations,

Table 3.8: Encrypted Traffic Feature Occurrence Count

Encrypted Traffic Feature Set				Minimalist Feature Set	
min_fpktl	172	min_fiat	14	max_pktlen	327
min_bpktl	107	mean_bpktl	12	duration	157
duration	82	total_bpackets	12	min_pktlen	148
mean_fpktl	32	total_bvolume	11	fwdbyte_persec	132
total_fpackets	31	proto	10	bkwdbyte_persec	116
max_fpktl	29	std_biat	5	fwdpkt_persec	24
total_fvolume	27	max_fiat	3	bkwdpkt_persec	8
max_bpktl	22	std_fiat	3	proto	3
min_biat	19	mean_fiat	2	ack_cnt	1
std_bpktl	17	mean_biat	1		
std_fpktl	15	max_biat	1		

using them few times in total but very early on in the decision tree structure. After that, packet length information is used heavily to further narrow down the decision-making process. Incoming flow statistics are generally used more frequently than their outgoing counterparts.

The decision tree generated with the minimalist feature set shown in Table 3.1 used protocol and ACK-count information to make the earliest distinction between Skype and non-Skype traffic. Of note here was the tree's heavy use of maximum packet lengths (in both directions) to create packet-size intervals for classification.

3.3.1 Comparison

The encrypted traffic model performed slightly better on the Dalhousie dataset, but the minimalist feature set proved better on the PoliTo dataset. This might suggest that the minimalist feature set is more suited to classifying both TCP and UDP Skype traffic. Looking at the resultant decision trees themselves, there are a few common themes. Both use packet length information heavily to make final classifications. The encrypted traffic feature set primarily uses packet interarrival times for timing information, whereas the minimalist feature set uses byte/packet transmission rates; however, the encrypted traffic decision tree uses its timing information very sparsely and early on in the tree, whereas the minimalist model makes use of its timing information on deeper levels of the tree. Combined with the above observations, this may suggest that packet and byte rates are more relevant to classifying generic Skype

traffic than packet interarrival times.

Chapter 4

Skype Packet-Level Classification

This chapter presents the experiments performed on classifying Skype network traffic at the individual packet level. An overview of the methodology and data sets used is given in Section 4.1. A number of different experiments are performed, and their results are presented in Section 4.2. An analysis of the best classification models obtained throughout the experiments is provided in Section 4.3, and these models are further fine-tuned and the results presented in Section 4.4.

4.1 Overview

This section details the exact data sets and processing methods used in analyzing Skype traffic at the packet level. Section 4.1.1 goes into detail on the procedure used to run the experiments, including novel two-tree hierarchy and shift register algorithms used to improve classification rates. The data sets used for these experiments are largely the same as those in the previous chapter, and their handling is briefly touched upon in Section 4.1.2.

4.1.1 Process

Packet level features were extracted using Tshark, a tool bundled with Wireshark (formerly Ethereal [13]). In all cases, the packet characteristics used matched those shown in table 4.1. This feature set was determined through earlier work in [2].

In the "Single Packet" experiments, the training sample was run through WEKA's implementation of the C4.5 machine learning algorithm, J48, to generate a decision tree-model classifier for Skype traffic. The first experiment generated a classifier with a near-zero false positive rate, but only an 86% detection rate, as shown in Table 4.2. To make up for this deficiency, a second experiment was devised in which all traffic identified as being out-of-class was used to train a second model, greatly improving the final overall classification rates. This results in a two-pass decision tree classification

Table 4.1: Packet-level Features Used

frame.time_delta	ip.checksum	tcp.flags.push
frame.pkt_len	ip.checksum_good	tcp.flags.reset
frame.len	ip.checksum_bad	tcp.flags.syn
frame.cap_len	tcp.len	tcp.flags.fin
frame.marked	tcp.seq	tcp.window_size
ip.len	tcp.nxtseq	tcp.checksum
ip.flags	tcp.ack	tcp.checksum_good
ip.flags.rb	tcp.hdr_len	tcp.checksum_bad
ip.flags.df	tcp.flags	udp.length
ip.flags.mf	tcp.flags.cwr	udp.checksum_coverage
ip.frag_offset	tcp.flags.ecn	udp.checksum
ip.ttl	tcp.flags.urg	udp.checksum_good
ip.proto	tcp.flags.ack	udp.checksum_bad

model, in which vectors rejected as being out-of-class by the initial decision tree are instead passed on to a second tree for re-classification. As this is no longer a simple one-pass binary classification system as performed by WEKA, the software is unable to calculate the receiver operating characteristic (ROC) values given in Table 4.2 for these experiments.

In the "Shift Register" experiments, a simple Java program was devised to simulate a shift register or "sliding window" concept that interpreted the packets on an as-they-arrive basis. The shift register is a queue object that stores each packet received as a single entry. When the register is exported, the most recent packet is exported as a record, along with the next $n-1$ packets, skipping $i-1$ packets in between each packet (where n and i are parameters chosen as appropriate for the experiment). The attributes are relabeled as ⟨packet name⟩-t-⟨offset from end of queue / i⟩. This exportation method leads to a new attribute set that contains $n \cdot c$ attributes, where c is the original number of attributes. Encoding temporal information in this manner, in order to generate classifiers using C4.5 that account for said information, is called flattening, and is explored in [10].

For the experiment using "flow discrimination", a different shift register was used for each flow encountered. Flows were identified based on the five-tuple: ⟨source IP address, destination IP address, transport protocol, source port, destination port⟩.

4.1.2 Datasets

These experiments focused solely on the UCIS dataset, as introduced in section 3.1.2. As the UCIS dataset contains a substantially large number of packets, biased subsets were used for both training and testing.

4.2 Training and Testing Results

Table 4.2 summarizes the results of all the experiments outlined above.

Table 4.2: Skype Packet-level Results

No Temporal Information	No Temp. Information, w/ Retraining	No Temp. Information, w/ PoliTo	No Temp. Information, w/ PoliTo & Retraining	Shift Register (i=5, n=4)	Shift Register (i=3, n=4) w/Flow
Training Results					
TP: 0.901 TN: 0.999 FP: 0.001 FN: 0.099 ROC Area: 0.996 # of Leaves: 396 Tree Size: 791 Dataset: 530k packets	TP: 0.9978 TN: 0.9897 FP: 0.0103 FN: 0.0022 ROC Area: ? # of Leaves1: 396+454 Tree Size1: 791+907 Dataset: 730k packets	TP: 1.000 TN: 0.999 FP: 0.001 FN: 0.000 ROC Area: 1.000 # of Leaves: 451 Tree Size: 901 Dataset: 790k packets	TP: 0.9921 TN: 0.9690 FP: 0.0310 FN: 0.0079 ROC Area: ? # of Leaves: 451+454 Tree Size: 901+907 Dataset: 48k packets	TP: 0.997 TN: 0.982 FP: 0.018 FN: 0.003 ROC Area:0.996 # of Leaves: 4110 Tree Size: 8219 Dataset: 280k registers	TP: 0.992 TN: 0.981 FP: 0.019 FN: 0.008 ROC Area: 0.996 # of Leaves: 1585 Tree Size: 3169 Dataset: 160k registers
Training: 10-fold Stratified Cross-Validation					
TP: 0.867 TN: 0.999 FP: 0.001 FN: 0.133 ROC Area: 0.997	TP: 0.9975 TN: 0.9887 FP: 0.0113 FN: 0.0025 ROC Area: ?	TP: 0.998 TN: 0.999 FP: 0.001 FN: 0.002 ROC Area: 0.999	TP: 0.9617 TN: 0.9425 FP: 0.0575 FN: 0.0383 ROC Area: ?	TP: 0.983 TN: 0.950 FP: 0.050 FN: 0.017 ROC Area: 0.971	TP: 0.974 TN: 0.964 FP: 0.036 FN: 0.026 ROC Area: 0.976
Testing across Distinct Sample2					
TP: 0.8676 TN: 0.9988 FP: 0.0012 FN: 0.1324 ROC Area: 0.926 Dataset: 330m packets	TP: 0.9964 TN: 0.9937 FP: 0.0063 FN: 0.0036 ROC Area: 0.998 Dataset: 2.4m packets	TP: 0.8382 TN: 0.9987 FP: 0.0013 FN: 0.1618 ROC Area: ? Dataset: 330m packets	TP: 0.9964 TN: 0.9934 FP: 0.0066 FN: 0.0036 ROC Area: ? Dataset: 2.4m packets	TP: 0.914 TN: 0.951 FP: 0.049 FN: 0.086 ROC Area: 0.926 Dataset: 550k registers	TP: 0.978 TN: 0.966 FP: 0.034 FN: 0.022 ROC Area: 0.978 Dataset: 1.0m registers
Testing Across PoliTo Dataset					
- -	- -	TP: 1.000 FP: 0.000	TP: 1.000 FP: 0.000	- -	- -

Table 4.3: Attribute occurrence counts - single packet, initial decision tree

frame.time_delta	93	frame.cap_len	7
frame.pkt_len	84	tcp.flags	4
ip.ttl	60	tcp.flags.push	4
tcp.window_size	54	tcp.flags.reset	4
tcp.ack	31	ip.flags	3
tcp.seq	27	ip.flags.df	1
tcp.nxtseq	11	ip.checksum_good	1
ip.len	9	udp.length	1

4.3 C4.5 Decision Tree Analysis

A large number of C4.5 decision tree models were produced in substantially differentiated ways in this chapter. This section provides a summary of and comparison between these classification structures.

4.3.1 Comparison

To help understand why the generated decision trees perform as they do for classifying Skype traffic, the number of times each feature is used as a threshold value for a decision branch in each model is tallied and tabulated. These feature occurrence counts for the various decision trees are given in tables 4.3 – 4.6.

Both single-packet decision trees made extensive use of the packet size and timing characteristics available to them. As packet size is the only attribute directly related to the packet's payload (and the only one usable when viewing encrypted traffic), and timing information is directly related to the sustained nature of VoIP applications, this makes sense in the context of Skype traffic classification. The TCP ACK and sequence flags were the most frequently used TCP flags, which may also suggest that C4.5 is spotting a timing pattern in the data fed to it.

The shift register decision trees followed the same trend in regard to characteristics employed. In regards to temporal information, the tree using flow discrimination seems to use the values of each given characteristic at different times in roughly equal amounts; a possible interpretation for this is that C4.5 was attempting to classifify each packet individually as in- or out-of-class, and use the aggregate information in its final decision.

Table 4.4: Attribute occurrence counts - single packet, secondary decision tree

frame.pkt_len	167	tcp.nxtseq	5
frame.time_delta	81	frame.cap_len	4
ip.ttl	63	ip.proto	3
tcp.window_size	41	tcp.len	2
tcp.ack	27	tcp.flags	2
tcp.seq	19	tcp.flags.push	2
ip.len	14	tcp.flags.reset	2
udp.length	13	tcp.flags.ack	1
ip.flags	5	tcp.flags.fin	1

Table 4.5: Attribute occurrence counts - shift register, no flow discrimination

ip.ttl-t-3	323	frame.cap_len-t-1	74	tcp.flags.push-t-2	34	tcp.flags.ack-t	9
frame.pkt_len-t-3	321	frame.cap_len-t	73	frame.cap_len-t-3	32	ip.len-t	9
tcp.window_size-t-3	205	frame.cap_len-t-2	70	tcp.flags-t-2	31	tcp.flags.ack-t-1	8
tcp.ack-t-3	160	tcp.nxtseq-t-1	65	tcp.flags.push-t-1	30	tcp.flags.ack-t-2	8
tcp.seq-t-3	159	frame.pkt_len-t-2	63	udp.length-t-3	28	ip.flags.df-t-1	7
ip.ttl-t	152	tcp.seq-t	59	ip.proto-t-2	27	tcp.flags.reset-t-2	6
frame.time_delta-t	133	tcp.window_size-t-2	58	tcp.flags-t-3	27	ip.flags.df-t	4
frame.time_delta-t-1	122	tcp.nxtseq-t	55	ip.proto-t-1	26	ip.flags.mf-t	4
frame.pkt_len-t	109	tcp.nxtseq-t-2	54	tcp.len-t-2	25	tcp.flags.push-t-3	4
tcp.ack-t	109	tcp.seq-t-1	48	tcp.flags-t	24	ip.len-t-1	3
ip.ttl-t-1	109	ip.flags-t-1	46	ip.proto-t	23	tcp.flags.fin-t-3	3
frame.time_delta-t-3	102	ip.flags-t-2	46	tcp.len-t	22	tcp.len-t-3	2
frame.time_delta-t-2	97	ip.flags-t	37	ip.len-t-3	22	tcp.flags.reset-t	2
ip.ttl-t-2	96	udp.length-t-2	37	ip.flags-t-3	21	ip.len-t-2	2
tcp.ack-t-2	91	tcp.seq-t-2	37	tcp.len-t-1	19	ip.flags.df-t-2	2
tcp.ack-t-1	91	udp.length-t-1	36	tcp.flags.fin-t	16	tcp.flags.reset-t-1	1
frame.pkt_len-t-1	85	tcp.nxtseq-t-3	35	tcp.flags.fin-t-1	15	tcp.flags.reset-t-3	1
tcp.window_size-t-1	82	tcp.flags-t-1	35	tcp.flags.push-t	14		
tcp.window_size-t	75	udp.length-t	35	tcp.flags.fin-t-2	13		

Table 4.6: Attribute occurrence counts - shift register, with flow discrimination

ip.ttl-t-3	190	tcp.ack-t-2	33	tcp.nxtseq-t-1	9	tcp.flags.fin-t-2	3
frame.pkt_len-t-3	139	tcp.seq-t-1	28	udp.length-t	8	ip.flags-t	3
frame.pkt_len-t-2	114	tcp.flags-t	27	tcp.flags.push-t-2	8	tcp.len-t-2	2
frame.pkt_len-t	110	frame.cap_len-t-3	26	udp.length-t-1	8	tcp.flags.ack-t-3	2
frame.pkt_len-t-1	105	tcp.flags-t-3	25	tcp.flags.push-t-3	8	tcp.flags.push-t-1	2
tcp.window_size-t	73	tcp.flags-t-2	24	frame.cap_len-t	7	ip.flags-t-1	2
tcp.window_size-t-3	70	ip.ttl-t-2	19	frame.cap_len-t-2	7	tcp.len-t	2
tcp.window_size-t-1	58	tcp.nxtseq-t	16	ip.ttl-t	7	ip.flags.df-t-2	1
tcp.ack-t-3	56	tcp.flags-t-1	16	ip.len-t-2	6	tcp.len-t-1	1
tcp.window_size-t-2	54	ip.flags-t-3	16	ip.len-t-1	6	tcp.flags.ack-t-2	1
tcp.seq-t	48	tcp.nxtseq-t-3	13	udp.length-t-2	5	tcp.flags.reset-t	1
tcp.ack-t	48	tcp.nxtseq-t-2	12	tcp.flags.push-t	4	tcp.flags.reset-t-3	1
tcp.seq-t-3	46	frame.cap_len-t-1	10	ip.len-t	4	tcp.len-t-3	1
tcp.seq-t-2	38	udp.length-t-3	9	ip.flags-t-2	3		
tcp.ack-t-1	36	ip.ttl-t-1	9	ip.len-t-3	3		

Table 4.7: Confidence Level Tuning Results

	C=0.15	C=0.20	C=0.25	C=0.30	C=0.35
Tree Size:	699 / 965	765 / 1015	901 / 1175	911 / 1241	945 / 1351
Training:	TP: 0.9985 TN: 0.9956 FP: 0.0044 FN: 0.0015	TP: 0.9987 TN: 0.9957 FP: 0.0043 FN: 0.0013	TP: 0.9989 TN: 0.9956 FP: 0.0044 FN: 0.0011	TP: 0.9987 TN: 0.9958 FP: 0.0042 FN: 0.0013	TP: 0.9989 TN: 0.9959 FP: 0.0041 FN: 0.0011
X-Validation:	TP: 0.9982 TN: 0.9941 FP: 0.0059 FN: 0.0018	TP: 0.9983 TN: 0.9941 FP: 0.0059 FN: 0.0017	TP: 0.9984 TN: 0.9941 FP: 0.0059 FN: 0.0016	TP: 0.9983 TN: 0.9941 FP: 0.0059 FN: 0.0017	TP: 0.9983 TN: 0.9940 FP: 0.0060 FN: 0.0017
Testing:	TP: 0.9986 TN: 0.9938 FP: 0.0062 FN: 0.0014	TP: 0.9986 TN: 0.9939 FP: 0.0061 FN: 0.0014	TP: 0.9987 TN: 0.9934 FP: 0.0066 FN: 0.0013	TP: 0.9987 TN: 0.9935 FP: 0.0065 FN: 0.0013	TP: 0.9987 TN: 0.9937 FP: 0.0063 FN: 0.0013

4.4 Parameter Tuning

Table 4.7 shows the final results of the C4.5 Skype parameter tuning experiments, with comparison based on the confidence level parameter. The confidence level C defaults to $C = 0.25$, and the results from this can be viewed in Table 4.2. The Tree Size row is presented as "size of the first tree / size of the second tree", where the second tree is the result of re-training on records identified as "not Skype" by the first decision tree (as outlined in section 4.1.1). The remaining rows present the true positive, true negative, false positive and false negative statistics on the final two-pass classification results.

Table 4.8 shows the final results of the experiments with comparison based on the minimum instances required per branch parameter. The minimum instance M defaults to $M = 2$, the results from which can be viewed in table 4.2.

A snippet of a typical decision tree model generated from using $C = 0.15$ is given in Figure 4.1.

Table 4.8: Minimum Instances Required Per Branch Tuning Results

Parameter Value:	M=3	M=4	M=6	M=8	M=10
Tree Size:	753 / 1063	635 / 1049	495 / 793	415 / 741	349 / 723
Training:	TP: 0.9987 TN: 0.9936 FP: 0.0064 FN: 0.0013	TP: 0.9987 TN: 0.9933 FP: 0.0067 FN: 0.0013	TP: 0.9985 TN: 0.9935 FP: 0.0065 FN: 0.0015	TP: 0.9985 TN: 0.9934 FP: 0.0066 FN: 0.0015	TP: 0.9985 TN: 0.9934 FP: 0.0066 FN: 0.0015
X-Validation:	TP: 0.9973 TN: 0.9911 FP: 0.0089 FN: 0.0027	TP: 0.9971 TN: 0.9909 FP: 0.0091 FN: 0.0029	TP: 0.9969 TN: 0.9910 FP: 0.0090 FN: 0.0031	TP: 0.9970 TN: 0.9906 FP: 0.0094 FN: 0.0030	TP: 0.9969 TN: 0.9908 FP: 0.0092 FN: 0.0031
Testing:	TP: 0.9979 TN: 0.9920 FP: 0.0080 FN: 0.0021	TP: 0.9979 TN: 0.9918 FP: 0.0082 FN: 0.0021	TP: 0.9974 TN: 0.9918 FP: 0.0082 FN: 0.0026	TP: 0.9973 TN: 0.9913 FP: 0.0087 FN: 0.0027	TP: 0.9974 TN: 0.9916 FP: 0.0084 FN: 0.0026

Figure 4.1: Example C4.5 Decision Classifer with C=0.15

```
tcp.seq <= 12.400969
|  tcp.window_size <= 8.66406
|  |  udp.length <= 5.743003
|  |  |  ip.ttl <= 4.290459
|  |  |  |  tcp.nxtseq <= 8.869258
|  |  |  |  |  tcp.seq <= 5.379897
|  |  |  |  |  |  tcp.ack <= 7.097549
|  |  |  |  |  |  |  udp.length <= 4.430817
|  |  |  |  |  |  |  |  ip.ttl <= 4.110874
|  |  |  |  |  |  |  |  |  tcp.seq <= 1.098612
|  |  |  |  |  |  |  |  |  |  udp.length <= 4.290459
|  |  |  |  |  |  |  |  |  |  |  udp.length <= 3.401197
|  |  |  |  |  |  |  |  |  |  |  |  udp.length <= 3.332205
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.cap_len <= 4.060443
|  |  |  |  |  |  |  |  |  |  |  |  |  |  ip.len <= 3.496508: SKYPE (3.0/1.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  ip.len > 3.496508: NOTSKYPE (25.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.cap_len > 4.060443
|  |  |  |  |  |  |  |  |  |  |  |  |  |  ip.ttl <= 3.89182: SKYPE (4.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  ip.ttl > 3.89182: NOTSKYPE (4.0/1.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  udp.length > 3.332205: SKYPE (14.0)
|  |  |  |  |  |  |  |  |  |  |  |  udp.length > 3.401197: NOTSKYPE (392.0/18.0)
|  |  |  |  |  |  |  |  |  |  |  udp.length > 4.290459
|  |  |  |  |  |  |  |  |  |  |  |  ip.ttl <= 3.89182
|  |  |  |  |  |  |  |  |  |  |  |  |  ip.ttl <= 3.871201: NOTSKYPE (2.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  ip.ttl > 3.871201: SKYPE (17.0)
|  |  |  |  |  |  |  |  |  |  |  |  ip.ttl > 3.89182: NOTSKYPE (13.0/1.0)
|  |  |  |  |  |  |  |  |  |  tcp.seq > 1.098612
|  |  |  |  |  |  |  |  |  |  |  ip.flags <= 2: NOTSKYPE (12.0)
|  |  |  |  |  |  |  |  |  |  |  ip.flags > 2
|  |  |  |  |  |  |  |  |  |  |  |  ip.ttl <= 3.871201: SKYPE (45.0)
|  |  |  |  |  |  |  |  |  |  |  |  ip.ttl > 3.871201
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.pkt_len <= 4.248495: SKYPE (6.0/1.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.pkt_len > 4.248495: NOTSKYPE (3.0)
|  |  |  |  |  |  |  |  |  ip.ttl > 4.110874
|  |  |  |  |  |  |  |  |  |  frame.pkt_len <= 4.762174
|  |  |  |  |  |  |  |  |  |  |  frame.pkt_len <= 4.60517
|  |  |  |  |  |  |  |  |  |  |  |  udp.length <= 3.688879
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.pkt_len <= 4.110874
|  |  |  |  |  |  |  |  |  |  |  |  |  |  udp.length <= 2.397895: SKYPE (6.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  udp.length > 2.397895: NOTSKYPE (7.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  frame.pkt_len > 4.110874
|  |  |  |  |  |  |  |  |  |  |  |  |  |  frame.pkt_len <= 4.290459
```

Chapter 5

Discussion and Future Work

This research showed that it is possible to achieve very high detection rates for Skype traffic with very low false positive rates, and that of the machine learning algorithms compared, C4.5 is the most suited to the task. This classification can be done at the packet level instead of abstracting to the network flow level, but at the cost of having a much larger decision tree model. If network flows are preferred, better performance can be achieved using the encrypted traffic feature set provided in Table 3.2 over the minimalist feature set provided in Table 3.1. More attributes must be calculated using this feature set, but these can be calculated using NetAI without the need for modifications as outlined in Section 3.1.1. Weaker performance was observed when using a shift register of packet features, shown in Table 4.2. The reasons for this are somewhat counter-intuitive, but it is possible that irrelevant or over-specific information is causing the machine-learning algorithm to overfit the classifier to the training data.

In the future, it would be good to evaluate the robustness of the generated classifiers by testing them against data recorded on an entirely different network. This task is made somewhat daunting by the privacy issues involved in recording actual network traffic, especially on as large a scale as that of a university campus network.

In considering the robustness of the generated classification models, it is important to consider the effect of specific threshold values embedded in the classifiers. For example, attributes related to specific timing characteristics such as bytes per second are extremely dependent upon the network environment in which the traffic was recorded; a slower network will have generally lower numbers for the same type of traffic. An extremely beneficial area of research would be to compare the traffic of a single application as it unfolds on networks with different characteristics, both topologically and technologically. If specific statistical distributions can be associated

with each attribute of a network flow regardless of the characteristics of the encompassing network, it would go a long way toward providing a generic classifier that can be employed on any network without needing to be retrained.

Bibliography

[1] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[2] R. Alshammari, P. Lichodzijewski, M. Heywood, and A. Zincir-Heywood. Classifying ssh encrypted traffic with minimum packet header features using genetic programming. *Genetic And Evolutionary Computation Conference 2009*, pages 2539–2546, 2009.

[3] R. Alshammari and A.N. Zincir-Heywood. Generalization of signatures for ssh encrypted traffic identification. *IEEE Symposium on Computational Intelligence on Cyber Security*, 2009.

[4] D. Angevine and A.N. Zincir-Heywood. A preliminary investigation of skype traffic classification using a minimalist feature set. *Third International Conference on Availability, Reliability and Security 2008*, pages 1075–1079, 2008.

[5] D. Bonfiglio, M. Mellia, M. Me, and D. Rossi. Detailed analysis of skype traffic. *IEEE Transactions on Multimedia*, 11(1):117–127, 2009.

[6] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: When randomness plays with you. *Proceedings of the ACM SIGCOMM'07*, pages 37–48, 2007.

[7] F. Fluckiger. *Understanding Networked Multimedia*. Prentice Hall, 1995.

[8] E. Freire, A. Ziviani, and R. Salles. Detecting skype flows in web traffic. *Network Operations and Management Symposium*, pages 89–96, 2008.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[10] K. Karimi and H. Hamilton. Finding temporal relations: Causal bayesian networks vs. c4.5. *Foundations of Intelligent Systems*, 1932/2009:266–273, 2000.

[11] C. Leung and Y. Chan. Network forensic on encrypted peer-to-peer voip traffics and the detection, blocking, and prioritization of skype traffics. *Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 401–408, 2007.

[12] P. Lin. Anatomy of the mega-d takedown. *Network Security*, 2009:4–7, December 2009.

[13] J. Matthews. *Computer Networking: Internet Protocols in Action*. Wiley, 2005.

[14] M. Pernyi, A. Gefferth, T. Dang, and S. Molnr. Skype traffic identification. *Global Telecommunications Conference*, pages 399–404, 2007.

[15] L. Peterson and B. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, second edition, 1999.

[16] P. Porras. Inside risks: Reflections on conficker. *Communications of the ACM*, 52, October 2009.

[17] P. Renals and G. Jacoby. Blocking skype through deep packet inspection. *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–5, 2009.

[18] U. Yu, D. Liu, J. Li, and C. Shen. Traffic identification and overlay measurement of skype. *International Conference on Computational Intelligence and Security*, 2:1043–1048, 2006.