Towards better protocol identification using profile HMMs

JHU Technical Report JHU-SPAR051201

Charles V. Wright, Fabian Monrose, Gerald M. Masson
Johns Hopkins University
Information Security Institute
Baltimore, MD 21218
{cvwright,fabian,masson} at jhu dot edu

Abstract

We present improved techniques for the identification of unknown TCP connections in wide-area Internet traffic using Profile Hidden Markov Models. Specifically, we built mixture models using a k-means clustering approach to find component behavior patterns in the traffic traces. These mixture models allow us to better recognize protocols that tend to exhibit more than one characteristic behavioral pattern. Moreover, our models use only those features that remain intact after encryption, namely packet sizes and inter-arrival times. Using a vector quantization approach to combine these features in a single model, we show how to substantially increase recognition accuracy over prior work — in some cases well over 30 percent.

1 Introduction

We present improved techniques for the identification of unknown TCP connections in wide-area Internet traffic. Unlike earlier approaches, we restrict ourselves to using only features of the traffic that remain observable after encryption using, for example, end-to-end cryptographic protocols such as SSL [17] or IPSec [13]. Our approach is highly automated, requiring only a modest set of training data and little or no advance knowledge of the particular protocols' properties.

The techniques we provide may be useful to network administrators in identifying the software related to suspicious traffic discovered on non-standard TCP ports, or to detect policy violations by users running prohibited applications and attempting to circumvent firewalls or avoid detection. For example, many chat and file sharing applications can be easily configured to use the standard TCP port for the World Wide Web in order to pass through common configurations of simple packet-filtering firewalls. Furthermore, recent peer-to-peer file-sharing applications [4] can run entirely on non-standard, user-specified ports, and Trojan horse or virus programs may encrypt their communication to deter the development of effective detection signatures. While our techniques are not robust against a malicious user who carefully modifies the stream of packets she sends, it does make her job significantly more difficult, requiring her to accurately mimic on-the-wire behavior patterns of legitimate traffic.

The approach we present in this paper significantly extends earlier efforts [2] in modeling network connections with Hidden Markov Models and in designing accurate classifiers from HMMs. We present enhancements that span improvements in building the protocol models, building the classifiers, quantization approaches, and clustering TCP sessions — all in an effort to improve the overall accuracy of our design. In particular, we explore the use of a new mixture model topology made up of several component profile HMMs using only packet sizes and inter-arrival times. Mixture models allow us to better recognize certain protocols

that tend to exhibit different behavioral characteristics, such as `SSH` which comprises both file transfer and remote login sessions. Moreover, unlike the naïve classifier presented in [2] our Viterbi classifier is capable of incorporating both timing and size information in a single model. By using realistic data traces, we show that our techniques allow for traffic characterization approaches that provide reasonable accuracy to be of practical value, even when our models are trained and tested under disparate network conditions.

PAPER OUTLINE: The rest of this paper is organized as follows. In §2 we present a high level overview of our system design. We explain the basics of our modeling techniques in §3, and §4 chronicles the enhancements to earlier work and illustrates the corresponding improvements to recognition ability. We present a performance analysis in §5, and review related work in §6. Lastly, we conclude with some closing remarks and directions for future work in §7.

2 High-level Design

Here we present a high-level view of the techniques used to construct and evaluate our classifiers. The overall process is illustrated in Figure 1 and entails (*i*) data collection and preprocessing (*ii*) feature selection, modeling and model selection, and finally (*iii*) the classification of test data and evaluation of the classifiers' performance. In what follows we provide a brief review of the individual modules.

DATA COLLECTION: In [2], a HMM-based classifier was built and evaluated based on packet traces derived from the MITLL Intrusion Detection Evaluation [15]. While this naïve classifier demonstrated recognition accuracy within 5% of other approaches [7] for the non-interactive protocols in the artificial MITLL dataset, its performance on traffic traces from a real Internet link was promising, at best. Given the availability of more realistic data we forgo the use of the MITLL traces in this work, instead concentrating our efforts on a new dataset [10] collected by the Statistics Group at George Mason University in 2003. The traces contain headers for IP packets on their Internet link from the first 10 minutes of every quarter hour for a few months. The dataset contains traffic for a `class B` network which includes several university-wide and departmental servers for mail, web, and other services, as well as hundreds of Internet-connected client machines. For each extracted TCP connection, we record the sequence of ⟨size, arrival time⟩ tuples for each packet in the connection, in arrival order. We encode the packet's direction in the sign bit of the packet's size, so that packets sent from server to client have size less than zero and those from client to server have size greater than zero. A summary of the traffic statistics for a random 10-hour period used for training our models is provided in Appendix A.

DATA PREPROCESSING: To assist in the creation of our models from packet sequences, we first apply a simplistic preprocessing step. To simulate the effect of encryption on the unencrypted traffic in our dataset we assume the encryption is performed with a symmetric block cipher, and round the observed packet sizes up accordingly. We report recognition rates for our size-based classifiers using a block size of 64 bytes, which is larger than most used in practice, yet still affords a nice balance of recognition accuracy and computational efficiency.

MODELLING: Building behavioral models for the various application protocols is the most complex task in our design and evaluation. The general outline is given in Figure 2. Given a set of training sequences, we begin by constructing an initial model (see Figure 3) such that the length of the chain of states in the model is equal to the average length (in packets) of sequences in the training set. Using initial parameters that assigns uniform probabilities over all packets in each timestep, we apply the well-known the Baum-Welch algorithm [3] to itera-

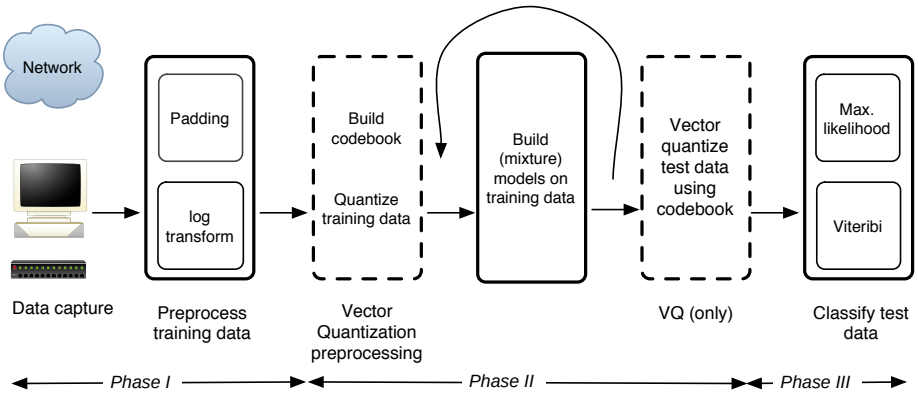

Figure 1: Process overview for construction of our Hidden Markov Model-based classifiers.

tively find new HMM parameters which maximize the likelihood of the model for the training sequences. Additionally, a heuristic technique called "model surgery"[18] is used to search for the most suitable HMM topology by iteratively modifying the length of the model and retraining. Next, to build mixture models (§4.3) that capture protocols which exhibit more than one characteristic behavior pattern, we apply a clustering technique in which we repeat the above process several times to build multiple profile HMMs for a given protocol, and then combine the HMMs into a final mixture model.

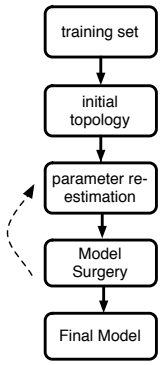

Figure 2: Outline of the steps required for building a profile Hidden Markov Model on the training data.

Classifiers: Given a HMM trained for each protocol, we then construct a classifier for the task of choosing, in an unsupervised manner, the best model—and, hence, the best-matching protocol—for new packet sequences. In our first attempt, we use the principle of Maximum Likelihood [9] to classify sequences as belonging to the protocol whose model assigns them the highest probability. We also explore the effectiveness of a classifier that uses Viterbi [21] paths and assigns sequences to the protocol whose model assigns them the greatest best-case probability.

To evaluate our classifiers' performance, we select traces for a 10-hour period chosen at random and use that to train our models. For each protocol, we select approximately 400 sequences for training. We then randomly select a set of traces from a *different* day and use that for testing. Each protocol in the testing set is then assigned a label by the classifier. For each classifier, we report average classification percentages for the connections in the test sets. We present a confusion matrix for each classifier's results, to illustrate the recognition rates for each protocol as well as the nature and severity of any misclassification. To the standard confusion matrix we add an extra column ("none") that depicts the percentage of sequences for each protocol which failed to match any model. The classifiers and their relative performance are further discussed in §4.1.

3 Model Building

We now we describe in more detail the construction of our Hidden Markov Models. As in [2], we build HMMs with profile topology using packet sizes and inter-arrival times. However, we improve on the results in that work by incorporating some rudimentary changes to the way the HMM is built, as well as some more fundamental enhancements (§4) to its basic design. Before discussing these changes, we briefly review the HMM design in [2].

3.1 Profile HMM Topology

Our models follow a similar design as those used in [14, 8, 18] for protein sequence alignment. The profile HMM (Figure 3) is best described as a left-right model built around two long parallel chains of hidden states, such that each chain has one state per packet in the TCP connection. Each state emits symbols with a probability distribution specific to its position in the chain, and states in these central chains are referred to as `Match` states, because their probability distributions for symbol emissions match the normal structure of packets produced by the protocol.

The main difference between the HMM in [2] and those in [14, 8, 18] is that the HMMs used to model proteins have only a single chain of `Match` states. The addition of a second match state per position was intended to allow the model to better represent the correlation between successive packets in TCP connections. Since TCP uses sliding windows and positive acknowledgments to achieve reliable data transfer, the direction of a packet is often closely correlated — either positively or negatively — with the direction of the previous packet in the connection. Therefore the `Server Match` state matches only packets observed traveling from the server to the client, and the `Client Match` state matches packets traveling in the opposite direction. For example, a transition from a `Client Match` state to a `Server Match` state indicates that a typical packet (for the given protocol) was observed traveling from the client to the server, followed by an similarly typical packet on its way from the server to the client.

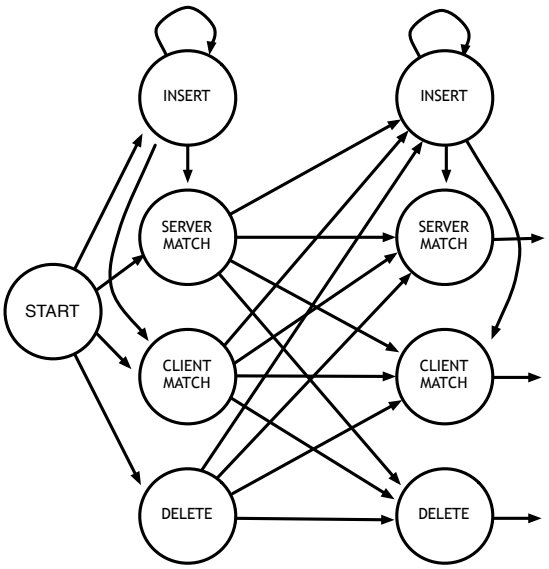

Figure 3: Profile Hidden Markov Model with two match states per position.

To allow for variations between TCP connections for the same protocol, the model has two additional states for each position in the chain. One, called `Insert`, allows for one or more extra packets "inserted" in an otherwise conforming sequence, between two normal parts of the session. The other, called the `Delete` state, allows for the usual packet at a given position to be omitted from the sequence. In practice, the `Insert` states represent duplicate packets and retransmissions, while the `Delete` states account for packets lost in the network or dropped by the detector. Both types of states may also represent other protocol-specific variations in higher layers of the protocol stack.

3.2 Handling Training Error

We implemented our HMMs in C using the open source General Hidden Markov Model Library (GHMM) [18]. Unfortunately, GHMM is unable to handle training error in the HMM parameter reestimation procedure; thus if even a single sequence in the training set is given a near-zero probability under the model, the only option is

to abort the reestimation procedure. However, the noisy nature of wide-area Internet traffic virtually guarantees that some training sequences will not fit in well with the others in the training set, so accommodating for training errors is critical.

We address this issue in our current implementation by excluding zero-probability sequences from training, and using a tunable parameter that controls the maximum reestimation failure rate we are willing to tolerate. This allows us to perform a greater number of iterations of the Baum-Welch algorithm on each model, thereby building better models for all protocols. We find that limiting the failure rate to 5% works quite well in practice for most network protocols and HMM topologies.

3.3 Model Surgery

To iteratively find the correct length of the model, we apply a heuristic technique known as model surgery [18]. Model surgery is based on the idea that all positions in the chain should represent about the same fraction of packets in a sequence, and that the match state(s) should represent the most typical behavior in a given position. We start with a model whose length is equal to the average number of packets per sequence in our training set[1]. In each iteration, we construct and train a model of the current length, and then inspect the likelihoods of the four states in each position in the chain. If the `Insert` state is the most likely state, this implies that the model is missing some essential behavior of the training set that occurs immediately before the behavior represented by the match states at the given position in the chain. To better capture this behavior, we increase the model's length by one. Conversely, if the `Delete` state is the most likely state, we decrement the current length. The process is repeated up to a maximum number of steps, or until the current length goes unchanged in some iteration. Typically, 10 iterations are sufficient for convergence.

Our empirical results show that when using only a single feature, model surgery typically improves recognition rates by roughly 2%, and makes the quality of the classifier much less dependent on the initial choice of model length. Consequently, the results presented for single variables (Tables 1, 2, and 4) all use models built with model surgery. However, we find surgery less effective when modeling both size and interarrival time in the same model, so surgery is not used in that case (§4.2).

4 Enhancements

More fundamental design changes were necessary to significantly improve accuracy rates on wide-area network traffic. For one, unlike [2] we use a Viterbi classifier and a new mixture model topology made up of several component profile HMMs. Additionally, we apply a vector quantization approach to combine both timing and size information into a single model. These enhancements are elaborated on below.

4.1 Classifiers

The task of a model-based classifier is, given an observation sequence $\mathcal{O}$, and a set $\mathcal{C}$ of k classes with models $\lambda = \lambda_1, \lambda_2, ..., \lambda_k$, to find $c \in \mathcal{C}$ such that $c = \texttt{class}(\mathcal{O})$. Our first such classifier, assigns protocol labels to sequences according to the principle of maximum likelihood. Informally, we choose $\texttt{class}(\mathcal{O}) = \underset{c}{\texttt{argmax}}\ (O \mid \lambda_c)$, where $\underset{c}{\texttt{argmax}}$ represents the class c with the highest likelihood of generating the sequences in $\mathcal{O}$.

Our second classifier is similar to the first, but it makes use of the well-known Viterbi [21] algorithm for finding the most likely sequence of states ($\mathcal{S}$) for a given output sequence $\mathcal{O}$ and HMM λ. The Viterbi algorithm can be used to find both the most likely state sequence (i.e., the "Viterbi path"), and its associated probability $P_{viterbi}(\mathcal{O}, \lambda) = \underset{\mathcal{S}}{\texttt{max}}\ \mathcal{P}(\mathcal{O},\ \mathcal{S} \mid \lambda)$.

[1] In practice, we cap the initial length at an arbitrary maximum of 200 packets. This allows us to build reasonable models for protocols such as `Telnet`, which tend to have very long-running connections, while still maintaining computational efficiency in the HMM calculations

Classification Probability									
Protocol	AIM	SMTP-out	SMTP-in	HTTP	HTTPS	FTP	SSH	Telnet	none
AIM	77.9	0.3	0.7	2.9	3.7	0.1	1.9	12.1	0.5
SMTP-out	1.1	82.9	0.9	0.2	0.2	0.4	0.9	10.7	2.8
SMTP-in	1.8	3.6	85.0	0.3	0.4	0.5	2.1	5.3	0.8
HTTP	0.1	0.0	0.0	89.2	5.3	0.0	2.5	1.7	1.0
HTTPS	0.7	0.1	0.1	8.2	76.1	0.2	8.0	2.5	4.1
FTP	2.5	0.8	10.9	0.7	0.4	55.3	2.6	26.7	0.0
SSH	0.8	0.0	0.0	1.2	13.6	1.3	58.5	2.7	21.8
Telnet	2.7	0.0	0.4	1.0	4.7	0.1	10.5	67.2	13.5

Table 1: Maximum Likelihood classifier, packet sizes only

Classification Probability									
Protocol	AIM	SMTP-out	SMTP-in	HTTP	HTTPS	FTP	SSH	Telnet	none
AIM	87.3	0.5	0.8	3.6	3.5	2.3	0.5	1.5	0.1
SMTP-out	3.3	87.7	1.5	0.0	0.3	1.5	0.4	4.7	0.6
SMTP-in	1.6	2.1	90.6	1.4	0.2	1.3	0.9	1.5	0.2
HTTP	0.4	0.1	0.0	93.6	4.2	0.1	0.5	0.8	0.4
HTTPS	0.5	0.1	0.1	7.7	88.8	0.2	1.1	0.1	1.2
FTP	3.9	1.0	22.4	9.4	0.7	57.6	1.3	3.6	0.0
SSH	2.2	0.3	0.0	3.4	8.4	1.9	77.3	1.7	4.8
Telnet	3.4	0.0	1.9	10.1	2.9	2.7	2.3	76.4	0.3

Table 2: Viterbi classifier with profile topology, packet sizes only

Given an output sequence $\mathcal{O}$, our Viterbi classifier finds Viterbi paths for the sequence in each model λ_i and chooses the class c whose model produces the best Viterbi path. We can express this decision policy concisely as

$$\texttt{class}(\mathcal{O}) = \underset{c}{\texttt{argmax}}\ P_{viterbi}(\mathcal{O}, \lambda_c) \qquad (1)$$

Tables 1 and 2 depict the classification results where our features are packet sizes only. The results show that our improvements to the way we build HMMs allow the Maximum Likelihood classifier to significantly outperform that in [2], which recognized SSH and Telnet with only 42% accuracy while training and testing on the same day. The Viterbi classifier further improves recognition rates for all protocols, especially the interactive ones. Recognition for SSH increases by $\approx$ 20%, to over 77%, and the rates for Telnet and AIM increase by more than 9%. Due mostly to a decrease in confusions with SSH, the recognition rate for HTTPS also improves by more than 12%. FTP, the only protocol with a recognition rate still below 75%, and the protocol whose correct classification rate is the least improved by the Viterbi classifier, does see a 23% drop in the rate of its most common confusion. The recognition rates for the other protocols increase by $\approx$ 5%.

We note that while the overall accuracy of the Viterbi classifier is much improved over that in [2], it still frequently confuses an interactive protocol (Telnet) with HTTP and SSH with HTTPS. We show in the following sections how our subsequent improvements to the structure of the HMMs (§4.3) and to the number of features included in our models (§4.2) reduce this error rate.

We note that the protocol whose recognition rates are most improved under the Viterbi classifier is SSH. Unlike our data for the non-interactive, more strictly-defined protocols, our traces of SSH connections are almost certain to contain sequences generated by several differently-behaving remote terminal applications, bulk data transfer (SCP), and possibly tunneled connections for other services (e.g., the X Window System). Therefore we are not surprised to find that, for the packets generated

by SSH connections, some state sequences in the HMM for SSH are much more likely than other state sequences in the same model. In §4.3 we explore whether the interactive protocols might be better represented by mixtures of HMMs rather than single profile models.

4.2 Multiple Features

While these results show surprising success for building with models of network protocols using only a single variable, one would suspect that recognition rates could be further improved by including both size and timing information in the same model. To evaluate that hypothesis, we employ a vector quantization technique to transform our two-dimensional packet data into symbols from a discrete alphabet so that we can then use the same type of models and techniques as used for dealing with timing or size individually.

Our vector quantization approach proceeds as follows: given the data for each packet as a two-dimensional tuple of ⟨inter-arrival time, size⟩, we first apply a log transform to the times to reduce their dynamic range [11, 16]. Next, to assign the sizes and times equal weight, we scale the ⟨log (time), size⟩ vectors into the -1,1 square.

The nature of our models requires that we treat packets differently based on the direction they travel across the wire. We therefore split the packets into two sets: those sent from the client to the server, and those sent from server to client. We then run the k-means clustering algorithm separately on each set of vectors to find a representative set of vectors, or codewords, for the packets in the given set. For a quantizer with a codebook of N codewords, for each of the two sets of packets, we begin by randomly selecting $k = N/2$ vectors as cluster centroids. Then, in each iteration, for each ⟨time, size⟩ vector, we find its nearest centroid and assign the vector to the corresponding cluster. We recalculate each centroid at the end of each iteration as the vector mean of all the vectors currently assigned to the cluster. We stop iterating when the fraction of vectors which move from one cluster to another drops below some threshold (currently set to 1 percent).

After clustering both sets of packet vectors, we take the list of centroid vectors as the codebook for our quantizer. To quantize the vector representation of a packet, we simply find the codeword nearest the vector, and encode the packet as the given codeword's index in the codebook. After vector quantization of the training sequences, we can then build discrete HMMs as before, using codeword numbers as the HMM's output alphabet. Before classifying test sequences, we use the codebook built on the training sequences to quantize them in the same manner.

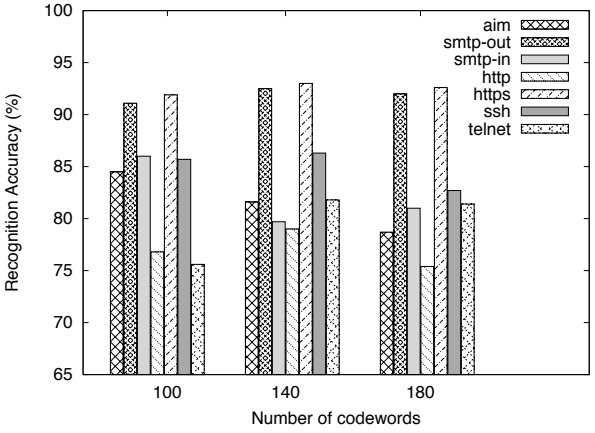

Figure 4: Viterbi classifier with profile topology for codebook sizes of 100, 140, and 180

Table 3 depicts the results for the Viterbi classifier on profile topology using a codebook of size 140. We find that, by including both size and timing information in the same profile model, we are able to recognize SSH traffic more accurately — its recognition rate is now almost 85 percent, which reflects a 26 percent improvement over the size-only Maximum Likelihood classifier in Table 1. Moreover, the vector quantized profile models are not very sensitive to codebook size (see Figure 4) greater than about 140. While less accurate than the mixture models for HTTP and SMTP, at the protocol level, the VQ models' misidentification still tends to be somewhat correct — when HTTP is misclassified, it is usually mistaken for HTTPS, and the

Classification Probability									
Protocol	AIM	SMTP-out	SMTP-in	HTTP	HTTPS	FTP	SSH	Telnet	none
AIM	80.3	0.6	0.0	0.4	5.4	0.5	11.7	1.0	0.0
SMTP-out	1.8	87.9	0.6	0.1	0.1	4.2	2.6	2.5	0.1
SMTP-in	1.0	8.6	79.5	1.2	0.2	5.8	2.7	0.9	0.0
HTTP	0.5	0.5	0.0	82.6	10.8	0.0	4.8	0.5	0.3
HTTPS	0.3	0.6	0.1	3.2	88.4	1.6	4.6	0.7	0.6
FTP	3.0	4.0	1.0	25.8	2.2	42.8	5.6	15.7	0.0
SSH	4.6	1.1	0.0	1.6	2.2	1.9	84.9	1.6	2.2
Telnet	3.3	2.6	0.3	4.2	1.3	5.2	8.8	73.9	0.3

Table 3: Viterbi classifier with profile topology, 140-codeword VQ

Algorithm 1 HMM-k-MEANS-CLUSTER(S, k)

INPUT: Training set S, number of clusters k
OUTPUT: set of models λ
randomly assign sequences $s_i \in S$ to clusters c_i
while no convergence **do**
  for each cluster c_i and its corresponding model λ_i **do**
    $\lambda_i \leftarrow$ Build-Profile-HMM(c_i) {see Figure 2}
  end for
  for each sequence s_i in the training set S **do**
    $b_i \leftarrow$ Best-Match(s_i, λ) {e.g., using Equation (1)}
    if $b_i \neq$ the previous match p_i **then**
      $c_{p_i} \leftarrow c_{p_i} \setminus s_i$ {remove s_i from its current cluster}
      $c_{b_i} \leftarrow c_{b_i} \cup s_i$ {add s_i to the cluster which provided the best match}
      $p_i \leftarrow b_i$
    end if
  end for
end while
return set of models λ_1, ..., λ_k

two directions of SMTP are much more frequently confused with each other than with other protocols. Furthermore, the confusions of Telnet and SSH as HTTP and HTTPS, respectively, noted in §4.1, are now less than half their previous value.

Like our other profile model classifiers, the vector quantized versions do not recognize FTP well; indeed, its 42.8% recognition rate is the lowest of any of our current classifiers. We address this decrease in accuracy in the following section.

4.3 Mixture Modeling with HMMs

One hypothesis for the poor recognition performance in Tables 1 and 2 for FTP, SSH and Telnet (particularly under the first classifier) is that these protocols exhibit more than one characteristic behavior pattern, and thus are not well represented by a single profile model. We attempt to address this shortcoming by modeling these protocols as mixtures of multiple distinct behavior patterns.

Finite mixture models are used for data which is believed to "arise from two or more underlying groups with common distributional form but with different parameters" [1]. A mixture model with k components consists of a set of statistical models $\lambda = \lambda_1$, ..., λ_k and mixing proportions π_1, ..., π_k where $\sum_{i=1}^{k} \pi_i = 1$. The i^{th} component model λ_i describes the data in the i^{th} underlying group, and the mixing proportion π_i gives the fraction of all data which belong to the i^{th} group.

In this work, we use HMMs both for the component models λ_i and for the mixture model. Because HMMs do not lend themselves easily to the standard methods of building mixture models, we employ a version of k-means model-based

Classification Probability									
Protocol	AIM	SMTP-out	SMTP-in	HTTP	HTTPS	FTP	SSH	Telnet	none
AIM	83.2	0.4	1.1	3.4	5.5	3.5	1.0	1.8	0.1
SMTP-out	1.3	88.7	1.2	0.0	0.1	3.0	0.3	5.0	0.4
SMTP-in	1.5	2.3	89.9	2.4	0.1	1.6	1.2	1.0	0.1
HTTP	0.3	0.1	0.1	91.6	6.5	0.1	0.5	0.7	0.2
HTTPS	0.7	0.1	0.1	4.8	92.0	0.4	0.9	0.3	0.7
FTP	2.2	1.0	14.0	7.2	0.3	72.7	1.1	1.4	0.0
SSH	3.2	0.6	0.1	1.3	10.4	2.4	77.0	1.4	3.6
Telnet	3.3	0.0	1.7	8.8	3.1	6.3	2.6	74.1	0.1

Table 4: Viterbi classifier with Mixture topology for FTP with 3 mixture components, profile topology for others

clustering to simultaneously find the component behavior patterns in a dataset and model them each with a profile HMM, then assemble the component HMMs into a final mixture HMM.

The k-means clustering algorithm assumes that the data belong to k distinct components, or clusters, and attempts to iteratively assign data points to the clusters until convergence — stopping when the clusters remain relatively stable across two consecutive iterations. Whereas the standard k-means algorithm assigns datapoints to clusters using a simple distance metric by picking the cluster whose center is "closest" to the given datapoint, the model-based version of the algorithm [12] describes each cluster with a mathematical model and assigns datapoints to the cluster whose model assigns them the highest probability. The model most commonly used for finite-dimensional data is the Gaussian family, wherein each cluster is represented by a mean vector and covariance matrix. However, for time course data, other types of models, such as HMMs, may be more appropriate. Schliep *et al.* [18] present an application of model-based k-means clustering to biological sequences, using profile Hidden Markov Models for the cluster centers. Here, we apply a similar technique to cluster packet sequences and to build the component profiles of our mixture HMMs.

As depicted in Algorithm 1, we begin by assigning each sequence randomly to one of the k clusters. Then, in each iteration, for each cluster, we build a profile HMM to represent the cluster's "center", using as training data the sequences currently assigned to the given cluster. We then assign each sequence to the cluster whose model matches the sequence most closely, using our Viterbi classifier and the newly-constructed cluster center HMMs. The process stops when, in some iteration, very few sequences are reassigned from one cluster to another. In practice, we find it unreasonable to expect the number of reassignments to converge to zero, and that stopping when less than 5 percent of the sequences are reassigned in a single iteration works well.

The output of the clustering algorithm is a set of k distinct profile models λ_1, ..., λ_k, each of which represents a behavior pattern exhibited by a subset of the traces for the given protocol. To use these models with our existing classifiers, we must first assemble them into a single unified mixture HMM λ_{mix} for the protocol as a whole. To do so, we insert a new start state, and, for each component profile HMM λ_i, we add a silent transition with probability π_i from the new start state to the start state of model λ_i, where π_i is calculated as the fraction of training sequences belonging to the i^{th} cluster.

Discussion: We find that recognition rates, especially those for interactive protocols, are often improved by the addition of a second and third profile to the protocols' HMMs. However, each additional component in the mixtures adds to the computational costs of both building the models and of classifying new sessions,

and we find that mixture models with more than three components do not offer a substantial increase in recognition rate over the smaller mixtures. Therefore we can construct a classifier with a balance of accuracy and computational complexity by building mixture models only for those protocols that involve terminal input from a user.

We find that, from our empirical evaluation, the only interactive protocol that benefits substantially from mixture models is `FTP`, so we present in Table 4 the classification results for the Viterbi classifier on packet sizes, using for `FTP` a mixture HMM with 3 components, and using basic profile HMMs for the other protocols. With the additional two behavior profiles in the model for `FTP`, this classifier is able to recognize `FTP` about as well as the standard Viterbi classifier is able to recognize the more interactive protocols like `Telnet` and `SSH`.

One explanation for the difference in our classifiers' ability to recognize `FTP` and `SSH` may be that while `SSH` is better represented by its timing characteristics, the differences in behaviors of `FTP` in this data seem to have less to do with inter-packet timing.

5 Performance Analysis

The classifiers presented here operate in an offline manner; that is, they require that sufficient traffic be captured before the traces can be classified. Our envisioned use of these classifiers is in a scenario where a network administrator attempts to analyze traces, at some regular interval, in order to gain a better view of the types of traffic on his network. The length of the analysis interval could be for example, an hour, a period of several hours, or a full day. In any case, we consider performance to be sufficient for offline analysis as long as the classifier is capable of analyzing all traffic for a given interval during the next interval.

To estimate the runtime performance of our classifiers, we evaluate the time required to perform our classification experiment on an hour of traffic from a peak period (11am-12pm) on a randomly-selected day in our test dataset. For our evaluation, we use models built *a priori* for our accuracy tests (see for example Tables 1 - 4) to classify all TCP connections observed in the hour-long trace on the well-known ports for the 8 protocols in this study. While a real usage scenario would involve classifying all TCP connections on all ports, we believe that the protocols we analyze here represent a wide-enough range of behaviors to give us a reasonable estimate of the system's performance on real traffic.

Classifier	Evaluate (*min*)
Viterbi (sizes)	27.8
140-codeword VQ	25.2
Mixture Models	46.2

Table 5: Runtime to build models on $\approx$ 400 training sequences for each protocol, and to classify 1 hour of peak traffic.

Table 5 depicts the results. We see that the Viterbi classifier using a 140-word codebook—which is fairly accurate in identifying most protocols—can classify the test data in about 25 minutes[2]. Using mixture models for `FTP`, `SSH` and `Telnet` alongside profile models for the other protocols, classification time increases significantly to just over 46 minutes. While this offline cost is relatively high, we argue that since the traffic loads at most networks exhibit strong patterns of heavy traffic during weekdays and light traffic at night and during weekends, then down times could provide an ideal time for an offline classifier to analyze data captured during the busiest parts of the day. Additionally, our current implementation is fairly inefficient and can be substantially optimized.

6 Related Work

The application of (un)supervised learning techniques to computer security domains is by no means new, and has been addressed in a number of recent work. The most closely related is

[2] A Linux machine equipped with a 2.4GHz Xeon processor and 2 GB RAM served as our experimental platform.

that of Early *et al.* [7] where a decision tree classifier that used n-grams of packets was proposed for distinguishing among flows from `HTTP`, `SMTP`, `FTP`, `SSH` and `Telnet` server. Features included in the decision tree include TCP flags, average inter-arrival time, and average packet size. Our approach is similar to that of [7], but differs in a few important ways. First, we build models based on each packet's characteristics instead of using average values over n-grams of several packets. Second, their classifier uses data from TCP headers, while ours uses only the information that remains intact and observable after encryption. Additionally, while their decision tree algorithm uses information gain estimation to automatically select the best features, we use only one or two predetermined features in each of our models. Surprisingly, this latter simplification still yields comparable accuracy [3].

Similar to the idea present here, Coull *et al.* [5] recently used sequence alignment techniques to address a very different task — namely masquerade detection. To detect anomalies in a user's shell behavior they apply a pairwise, semi-global alignment algorithm to sequences of shell commands, and demonstrate results on par with the best performers from an earlier comparison of detection algorithms [19]. We believe our results in this paper validate their application of sequence alignment methods for the purpose of masquerade detection. However, unlike [5], our profiling technique does not require pairwise alignments of all sequences, and is therefore better suited for studying network protocols (where the training data requirements may be fairly large).

Sun *et al.* [20] demonstrate remarkable success in identifying web pages accessed in `SSL`–encrypted connections, using only the count and sizes of HTML objects returned in the `HTTP` response. By representing web pages as multisets of object sizes, similarity is expressed as the number of sizes in common across two web pages, and pages with a similarity score above some threshold are considered to be the same page. While their sample size was small, this rather simplistic technique was found to be quite reliable, identifying around 75 percent of target pages, with a false-positive rate of under 1.5 percent. Furthermore, countermeasures such as padding the object sizes appear to have less effect on detection accuracy than expected, requiring a substantial increase in the total size of most web pages to thwart identification. We believe our work shows that this result holds at lower layers of the protocol stack as well, when the objects under consideration are not HTML objects but individual packets in generic TCP connections.

More distantly related work is that on backdoor and stepping stone detection. Zhang and Paxson [24] present one of the earliest studies of techniques for network protocol recognition without using port numbers. By correlating the timing of on/off periods in inbound and outbound interactive connection, the authors [25] demonstrate how to detect "stepping stone" connections whereby an adversary tries to conceal the true source of an attack by hopping from one host to another. Wang *et al.* [22] and Yoda and Etoh [23] subsequently used methods similar to sequence alignment to detect stepping stones by identifying TCP connections with similar packet streams — the general idea being to find good alignments of the streams by identifying locations where the two subsequences of inter-arrival times are most similar. Lastly, Donoho *et al.* [6] extend this idea to exploit the property of maximum tolerable delay and develop a stepping stone detection method that is robust against an active adversary who can jitter her connections and inject spurious packets. Essentially, the approach in [6] uses wavelets and multiscale analysis to identify related packet streams of sufficient length which do not deviate from each other by more than the maximum delay an interactive human user is willing to tolerate.

[3] While the approach achieved accuracy between 85 and 100 percent, it is unclear from [7] whether these empirical results are based on LAN or WAN traffic.

7 Conclusions and Future Work

We build multiple classifiers capable of recognizing several of the most common application protocols in real Internet TCP connections, using only the size and timing of packets. With a new classifier based on Viterbi paths, a HMM topology for mixture models, and a vector quantization technique for combining packet timing and size information in a single model, we show significant improvements in accuracy over prior work. Most notably, we observe an increase in recognition of over 30% for the interactive protocols.

Our short-term plans include a more rigorous evaluation of our classifiers' accuracy on a larger selection of traces, as well as testing on tunneled traffic generated using `SSH`'s port forwarding facility. We are also exploring ways to improve the overhead induced by the use of mixture models, and working towards a design that is more appropriate for network intrusion detection systems.

Acknowledgments

We are very grateful to the Statistics group at GMU for sharing their packet traces and computational resources.

References

[1] M. Aitkin and D. B. Rubin. Estimation and hypothesis testing in finite mixture models. *Journal of the Royal Statistical Society*, 47(1):67–75, 1985.

[2] Anonymized. HMM profiles for network traffic classification (extended abstract). In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pages 9–15, October 2004.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, February 1970.

[4] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[5] S. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *Proceedings of the* 19^{th} *Annual Computer Security Applications Conference*, pages 24–33, December 2003.

[6] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.

[7] J. Early, C. Brodley, and C. Rosenberg. Behavioral authentication of server flows. In *Proceedings of the* 19^{th} *Annual Computer Security Applications Conference*, pages 46–55, December 2003.

[8] S. Eddy. Multiple alignment using hidden markov models. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 114–120. AAAI Press, July 1995.

[9] S. R. Eliason. *Maximum Likelihood Estimation: Logic and Practice, Vol.96*. SAGE Publications, August 1993.

[10] D. Faxon, R. D. King, J. T. Rigsby, S. Bernard, and E. J. Wegman. Data cleansing and preparation at the gates: A data-streaming perspective. In *2004 Proceedings of the American Statistical Association*, August 2004.

[11] A. Feldmann. *Characteristics of TCP connection arrivals*. Park and Willinger (Ed). Wiley-Interscience, 2000.

[12] C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97(458):611–631, June 2002.

[13] N. W. Group. RFC 2401: Security architecture for the Internet Protocol, November 1998.

[14] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov Models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, February 1994.

[15] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissmann. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*. DISCEX 2000, January 2000.

[16] V. Paxson. Emprically-derived analytic models of wide-area tcp connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[17] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2000.

[18] A. Schliep, A. Schönhuth, and C. Steinhoff. Using hidden markov models to analyze gene expression time course data. *Bioinformatics*, 19(supplement 1):i255–i263, July 2003.

[19] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 56(1):58–74, February 2001.

[20] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2002.

[21] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, 1967.

[22] X. Wang, D. S. Reeves, and S. F. Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In 7^{th} *European Symposium on Research in Computer Security (ESORICS)*, October 2002.

[23] K. Yoda and H. Etoh. Finding a connection chain for tracing intruders. In 6^{th} *European Symposium on Research in Computer Security (ESORICS)*, October 2000.

[24] Y. Zhang and V. Paxson. Detecting back doors. In *Proceedings of the* 9^{th} *USENIX Security Symposium*, August 2000.

[25] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the* 9^{th} *USENIX Security Symposium*, August 2000.

A

Table 6 provides a summary of the data in our 10 hour snapshot used for training. As is the case with other studies of network traffic (for example [16]) we express averages μ_{geom} and standard deviations σ_{geom} calculated using the geometric mean instead of the arithmetic mean; when dealing with data such as session lengths or interarrival times, which can take on a wide range of positive but unbounded values, the geometric mean is less skewed by the largest elements of the sample and thus presents a more accurate description of the data.

GMU Traffic (10 hours)							
Protocol	Connections		Length (packets)		Bytes	Duration (s)	IA (s)
	#	%	μ_{geom}	σ_{geom}		μ_{geom}	μ_{geom}
aim	22043	(23.8%)	26.21	2.56	99.1 MB	382.13	1.48e+00
smtp-out	19374	(21.0%)	25.13	2.09	199.1 MB	6.87	2.48e-02
smtp-in	15973	(17.2%)	25.88	1.99	237.9 MB	4.04	1.22e-02
http	11437	(12.4%)	16.64	2.46	374.9 MB	3.53	8.25e-03
https	21739	(23.5%)	29.30	2.43	513.6 MB	9.80	2.01e-02
ftp	679	(0.73%)	16.99	2.61	1.6 MB	11.53	1.07e-01
ssh	701	(0.76%)	55.39	8.63	72.7 MB	55.79	2.51e-02
telnet	531	(0.57%)	178.37	7.62	28.8 MB	91.97	8.49e-02

Table 6: Summary of a random 10 hour period of traffic in the GMU traces.