

## Early Bird: Catching worms while sysadmins sleep

Andrew Hill, B.Sc., B.Sc. (Math. & Comp. Sci.)

School of Computer Science  
The University of Adelaide  
South Australia

Supervisors: Mr. Kevin Maciunas and Dr. Cheryl Pope

November 2, 2003

Submitted in partial fulfillment of the requirements for the Honours  
Degree of Computer Science

## **Abstract**

This honours thesis demonstrates the need for an automated, anomaly-based Internet worm detection system that is effective at identifying Internet worm packets with a low false-positive rate.

The theory of general Discrete Symbol Hidden Markov Models and the theory of the equivalent on-line models is discussed, and the general structure of Hidden Markov Models is related to the problem of identifying Internet worm packets in a sequence of normal network packets.

The effectiveness of various on-line Hidden Markov Model configurations in detecting Sapphire Internet worm packets in a sequence of normal UDP packets is evaluated, demonstrating that Hidden Markov Models can be successfully used as the basis of an automated, anomaly-based Internet worm detection system.

*In loving memory of Ari. Rest in peace, 20 October 2003.*

# Acknowledgments

I would like to express my sincere appreciation for all the support I have received this year from my family, friends, and fellow students. In particular, I would like to thank:

- Mr. Kevin Maciunas and Dr. Cheryl Pope for their support and encouragement, as well as their invaluable comments and suggestions.
- Professor Langford White for his patience and help with Hidden Markov Models, without which I would never have managed to get anything working.
- My father Robert, brother Chris, and partner Jo for their assistance with proof-reading this thesis.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Internet Worms . . . . .	1
1.2	Intrusion Detection . . . . .	2
1.3	Requirements for an Anomaly-Based Network Intrusion Detection System . . . . .	2
1.4	Previous Work in the Field of Anomaly-Based Intrusion Detection Systems . . . . .	3
1.5	Previous Work in the Field of Anomaly-Based Network Intrusion Detection Systems . . . . .	3
1.6	Would a Hidden Markov Model be Suitable as the Basis of an Anomaly-Based Network Intrusion Detection System? . . . . .	4
1.7	Statement of Purpose and Organisation of Thesis . . . . .	5
<b>2</b>	<b>Background Information</b>	<b>6</b>
2.1	Discrete Symbol Hidden Markov Models . . . . .	6
2.2	Discrete Symbol Hidden Markov Model Algorithms . . . . .	8
2.2.1	Inference Using the Forward-Backward Algorithm . . . . .	8
2.2.2	Training Using the Baum-Welch Method . . . . .	12
2.3	On-line Discrete Symbol Hidden Markov Models . . . . .	15
2.4	On-line Discrete Symbol Hidden Markov Model Algorithms . . . . .	15
2.4.1	Inference Using the On-line Forward-Backward Algorithm . . . . .	15
2.4.2	On-line Training via the Fisher Information Matrix . . . . .	16
2.4.3	Filtering On-line Hidden Markov Models . . . . .	17
2.4.4	Fixed Lag On-line Hidden Markov Models . . . . .	17
2.5	Adding Forgetting to On-line Hidden Markov Models . . . . .	17
2.6	Internet Worms . . . . .	18
2.7	Design of an Anomaly-Based Network Intrusion Detection System Based on Hidden Markov Models . . . . .	19
<b>3</b>	<b>Implementation of an On-line Hidden Markov Model</b>	<b>21</b>
3.1	Implementation . . . . .	21
3.2	Verification of Implementation . . . . .	22

<b>4</b>	<b>On-line Hidden Markov Model Configuration Experiments Toward the Design of a Model for UDP Internet Worm De- tection</b>	<b>25</b>
4.1	The Relationship Between Hidden Markov Models and Inter- net Packets . . . . .	25
4.2	Collection of an Internet Datagram “Observation” Sequence .	26
4.3	Observation Sequence Prediction Experiments . . . . .	26
4.3.1	Effect of the Number of Hidden States . . . . .	27
4.3.2	Effect of the Value of $\Delta$ . . . . .	27
4.3.3	Effect of the Value of $\rho$ . . . . .	27
4.3.4	Effect of Different Observations . . . . .	28
4.4	Implementation Speed Experiments . . . . .	29
4.4.1	Effect of the Observation Sequence Length . . . . .	30
4.4.2	Effect of the Number of Hidden States . . . . .	30
4.5	Model Design . . . . .	30
<b>5</b>	<b>Internet Worm Packet Detection with an On-line Hidden Markov Model</b>	<b>41</b>
5.1	The Sapphire Internet Worm . . . . .	41
5.2	Sapphire Worm Packet Detection Experiments With Set “At- tack” Periods . . . . .	41
5.2.1	Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of the Observation Sequence . . . . .	42
5.2.2	Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of a Limited Time Period of the Observation Sequence . . . . .	43
5.2.3	Failure to Predict the Observation Sequence . . . . .	44
5.3	Sapphire Worm Packet Detection Experiments Without Set “Attack” Periods and Results . . . . .	44
5.3.1	Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of the Observation Sequence . . . . .	45
5.3.2	Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of a Limited Time Period of the Observation Sequence . . . . .	45
5.3.3	Failure to Predict the Observation Sequence . . . . .	46
5.4	Summary of the Results . . . . .	46
<b>6</b>	<b>Conclusions and Suggestions for Future Work</b>	<b>60</b>
6.1	Conclusions . . . . .	60
6.2	Suggestions for Future Work . . . . .	60
6.2.1	Improving the Internet Worm Detection Capabilities .	60
6.2.2	Testing with Other Internet Worms . . . . .	61

6.2.3 Implementation Speed . . . . .	61
<b>Bibliography</b>	<b>63</b>
<b>A Informal Order Analysis of the On-line Hidden Markov Model Algorithms</b>	<b>66</b>

## List of Tables

2.1	The Internet Protocol version 4 datagram format . . . . .	18
2.2	The Transmission Control Protocol datagram format . . . . .	19
2.3	The User Datagram Protocol datagram format . . . . .	19

## List of Figures

2.1	Graphical representation of a Hidden Markov Model . . . . .	7
2.2	The induction step of the forward pass of the Forward-Backward algorithm . . . . .	10
2.3	The induction step of the backward pass of the Forward-Backward algorithm . . . . .	11
2.4	Graphical representation of the derivation of $\xi_t(i, j)$ in the Baum-Welch method . . . . .	13
3.1	Convergence of the state transition probability matrix, $\Delta = 0$ .	23
3.2	Convergence of the observation probability matrix, $\Delta = 0$ .	23
3.3	Convergence of the state transition probability matrix, $\Delta = 5$ .	24
3.4	Convergence of the observation probability matrix, $\Delta = 5$ .	24
4.1	Example model training times for HMMs with different numbers of states and for different observation sequence lengths. $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. Example demonstrated is a “near-worst case” observation sequence. . . . .	29
4.2	Example 2 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	31
4.3	Example 8 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	31
4.4	Example 14 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	32
4.5	Example 20 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	32
4.6	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	33
4.7	Example 2 state HMM with $\Delta = 20$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	33
4.8	Example 2 state HMM with $\Delta = 50$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	34
4.9	Example 2 state HMM with $\Delta = 100$ , $\rho = 0.99$ , trained with UDP datagram source and destination port observations. . .	34

4.10	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.95$ , trained with UDP datagram source and destination port observations. . .	35
4.11	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.80$ , trained with UDP datagram source and destination port observations. . .	35
4.12	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.50$ , trained with UDP datagram source and destination port observations. . .	36
4.13	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.10$ , trained with UDP datagram source and destination port observations. . .	36
4.14	Example 2 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations. . . . .	37
4.15	Example 20 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations. . . . .	37
4.16	Example 50 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations. . . . .	38
4.17	Example 100 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations. . . . .	38
4.18	Example 2 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length. . . . .	39
4.19	Example 20 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length. . . . .	39
4.20	Example 50 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length. . . . .	40
4.21	Example 100 state HMM with $\Delta = 1$ , $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length. . . . .	40
5.1	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. . . . .	47
5.2	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	47
5.3	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $20 \times \Delta$ observations in red. . .	48

5.4	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $20 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	48
5.5	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. . . . .	49
5.6	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	49
5.7	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $100 \times \Delta$ observations in red. . . . .	50
5.8	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $100 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	50
5.9	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $10 \times \Delta$ observations in red. . . . .	51
5.10	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $10 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams (with an increased probability of 50%) are marked with circles. . . . .	51
5.11	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. . . . .	52
5.12	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams (with an increased probability of 50%) are marked with circles. . . . .	52
5.13	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. . . . .	53

5.14	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	53
5.15	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. . . . .	54
5.16	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	54
5.17	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. . . . .	55
5.18	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	55
5.19	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $20 \times \Delta$ observations in red. . . . .	56
5.20	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $20 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	56
5.21	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. . . . .	57
5.22	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $50 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	57
5.23	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $100 \times \Delta$ observations in red. . . . .	58
5.24	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last $100 \times \Delta$ observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	58

5.25	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. . . . .	59
5.26	Example 2 state HMM with $\Delta = 5$ , $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles. . . . .	59

# Chapter 1

## Introduction

### 1.1 Internet Worms

Worms<sup>1</sup> are programs that propagate over computer networks, replicating as they go. Experiments with worms were first performed in the early 1980s [27] and the first worm to cause wide-spread “damage” was released onto the Internet in 1988 [30].

Internet worms are capable of causing damage in three main ways – firstly, through the economic cost of loss of services due to their consumption of bandwidth, and through the cost associated with identifying and cleaning infected systems. The global economic impact of just one such Internet worm – Code Red – was estimated to be U.S.\$2.62 billion in 2001 [33].

Secondly, Internet worms may allow infected systems to be accessed remotely, circumventing authentication systems and resulting in the compromise of sensitive or proprietary data [31].

Thirdly, there is the potential for Internet worms to cause loss of life through the interruption of critical computer systems – for example, by interrupting the operation of nuclear power plant safety monitoring systems [24].

It is therefore clear that the threat of Internet worms must be taken seriously.

Simulations of Internet-wide cooperative responses to Code Red-style worm attacks have shown that if techniques requiring knowledge of how the worm works – in this case, packet-based content filtering – are used to try to prevent infections, the response must happen within around 2 hours of the worm being released [20].

However, the recent Sapphire worm infected more than 90% of vulnerable hosts on the Internet in less than 10 minutes [19], and there are techniques that the Sapphire worm did not use – such as partitioning the IP address

---

<sup>1</sup>From John Brunner’s 1975 science fiction novel “The Shockwave Rider.”

space so that scanning and infection of hosts is performed in a controlled fashion – that could increase the speed of infection even further [31].

## 1.2 Intrusion Detection

It seems reasonable to assume that the likelihood of an Internet-wide response system that is capable of reacting to a new worm in less than 10 minutes being deployed at any time in the near future is low. Thus, it would seem, individuals and organisations must assume the responsibility of protecting themselves from the potential damage of Internet worms, and/or minimizing the spread of worms to and from their computers – but how should they do this?

Knowing the potential speed with which hosts can become infected, it seems clear that some kind of “Early Bird” automated *intrusion detection* system is required to catch the worm.

Intrusion detection is the process of “detecting inappropriate, incorrect, or anomalous activity” [17]. There are two main approaches to intrusion detection. The first, called *misuse detection*, is to construct an expert system that knows about certain types of attacks (in the case of Internet worms, it may know about worm packet signatures, for example). The expert system then tries to match observed behaviour to these rules. Unfortunately, as outlined above, the potential speed with which worms can infect hosts on the Internet means that it may not be possible to create and implement such expert system rules before all vulnerable hosts have been infected.

The second approach to intrusion detection is called *anomaly detection*. This approach attempts to classify what is “normal” behaviour for a system, and to flag deviations from this behaviour as anomalous.

In the case of an individual or organisation trying to detect Internet worms as anomalous behaviour, it seems logical to look at the local network traffic going to/from a computer or passing through a corporate firewall – in essence, building an anomaly-based network intrusion detection system.

## 1.3 Requirements for an Anomaly-Based Network Intrusion Detection System

The requirements for an anomaly-based network intrusion detection system for Internet worms are:

- The system must be *on-line*: that is, it needs to be able to monitor the network traffic in real time, and react as anomalies are found. There is little point in detecting an Internet worm infection from logged data days after the infection has occurred.

- The system should ideally recognise that network packets are not discrete events, but are interrelated. Such recognition should allow better detection of anomalies [32].
- Minimal training of the intrusion detection system is desired. (System administrators want “plug and work” software – not systems they need to spend weeks setting up.)
- The system should be able to process information quickly, so that the speed at which packets are delivered is not adversely affected.
- The system should be able to accurately and consistently detect anomalous packet traffic associated with Internet worms. A low *false-positive* rate is desirable – many network administrators tend to ignore warnings if there are too many false-positives [1]. (False-positives in this domain would be network traffic classified as Internet worm data when in fact it is valid, non-worm traffic.)
- The system should be able to protect either an individual machine, or, by running on a corporate network firewall, an entire network of machines.

## 1.4 Previous Work in the Field of Anomaly-Based Intrusion Detection Systems

Obviously, anomaly-based intrusion detection systems do not need to be limited to the analysis of network traffic. For example, so-called *misuse detection* systems are anomaly-based systems that look at the various commands that a user issues to a computer to detect either unauthorized use by that user, or to detect deviations from that user’s expected behaviour, perhaps indicating that the user’s password has been compromised, and that another person is using their account.

Various techniques have been used in building successful misuse detection systems. These include Artificial Neural Networks [26] (ANNs) and Hidden Markov Models [15, 16] (HMMs).

It would seem reasonable to conclude from this that these two techniques might also have application in the field of anomaly-based network intrusion detection systems.

## 1.5 Previous Work in the Field of Anomaly-Based Network Intrusion Detection Systems

ANNs have been applied to the task of anomaly-based network intrusion detection [6, 7, 10], as has the technique of signal analysis [3].

Another technique, using the concept of an *activity graph* has been used to build two anomaly-based network intrusion detection systems, built with Internet worms in mind. These systems are GrIDS<sup>2</sup> and the system described in [32].

The GrIDS system attempts to track the “flow” of Internet worm connections from host to host [8]. When the pattern of these connections exceeds set thresholds, the activity is considered anomalous, and so the machines involved in the activity graph can be considered to be infected. The system described in [32] uses a similar system to detect the spread of worms via their *connection history*.

However, both of these systems rely on the ability of individual machines to report connection information to a centralised point for analysis. This precludes the protection of systems that can not run the reporting software required, or systems that have been installed without ensuring the software is installed. They also both rely on the existence of a network of systems that can be monitored – protection for individual machines is not possible. Thus, such a system does not meet the requirements outlined above, that such individual machine protection be provided.

There is no evidence in the literature that HMMs have been used to attempt to detect Internet worms in network traffic, despite their successful application in misuse detection.

## 1.6 Would a Hidden Markov Model be Suitable as the Basis of an Anomaly-Based Network Intrusion Detection System?

It would seem that an HMM might be suitable as the basis of an anomaly-based network intrusion detection system, for the following reasons:

- HMMs have on-line algorithms [14, 28], which would allow a system based on an on-line HMM to not only process packets as they arrive, but also to perform model training while they are run, resulting in minimal training for system administrators.
- The structure of HMMs [25] means that consideration of the interrelated nature of network packets is built into the model.

Of course, the effectiveness of such a system in detecting Internet worms, and the speed with which such a system could process packets would need to be determined.

---

<sup>2</sup><http://seclab.cs.ucdavis.edu/arpa/grids/>

## 1.7 Statement of Purpose and Organisation of Thesis

Given that there is no known literature on the use of HMMs in the field of anomaly-based network intrusion detection, and given their previously successful application to the similar field of misuse detection, there is scope for research in this area.

The purpose of this study is to determine if HMMs can be used as the basis of an anomaly-based network intrusion detection system for Internet worms, and if so, whether or not the system meets the requirements for speed and consistent, accurate detection of worms without false-positives.

The remainder of this thesis is organised in the following way:

- Chapter 2 covers the necessary theoretical background of HMMs, and the various algorithms required to train and use the models. The details of Internet worms and the design of an anomaly-based network intrusion detection system using HMMs as the basis are also outlined.
- Chapter 3 covers the implementation of an HMM system, and details the experiments performed to verify the implementation.
- Chapter 4 outlines HMM configuration experiments performed, allowing a model for detecting UDP-based Internet worms to be designed.
- Chapter 5 covers the results of experiments performed to test the model designed in Chapter 4 for its ability to detect an UDP-based Internet worm.
- Chapter 6 covers the conclusions that can be drawn from the experiments, and outlines suggestions for further study in this field.

## Chapter 2

# Background Information

### 2.1 Discrete Symbol Hidden Markov Models

The Hidden Markov Model (HMM) is one of a group of simplified Bayesian Networks called Dynamic Bayesian Networks [11], which can be applied to the task of modeling time series data. In the following discussion, where appropriate, the notation of [25] has been adopted.

HMMs have the following four assumptions:

1. For a finite sequence of observations  $O_{1:T} = O_1, O_2, \dots, O_T$  there is some process that has a finite sequence of hidden (i.e. non-observable) states  $Q_{1:T} = q_1, q_2, \dots, q_T$  that is responsible for the generation of these observations. Thus,  $O_t$  represents the observation at time  $t$ , while  $q_t$  represents the hidden state of the process at time  $t$ .
2. The hidden states of the process satisfy the *First-Order Markov Property*. That is, the hidden state  $q_t$  is dependent only on the previous hidden state  $q_{t-1}$ . This is equivalent to saying that “the state at some time encapsulates all we need to know about the history of the process in order to predict the future of the process” [11].
3. An observation  $O_t$  is assumed to be dependent only on the process’ current hidden state (i.e.  $q_t$ ).
4. The possible values that each state  $q_t$  can assume are discrete. Here, the set of possible states is  $\{S_1, S_2, \dots, S_N\}$ .

These assumptions allow the joint probability of a sequence of states  $Q_{1:T}$  and a sequence of observations  $O_{1:T}$  to be factored as:

$$P(Q_{1:T}, O_{1:T}) = P(q_1) P(O_1|q_1) \prod_{t=2}^T P(q_t|q_{t-1}) P(O_t|q_t) \quad (2.1)$$

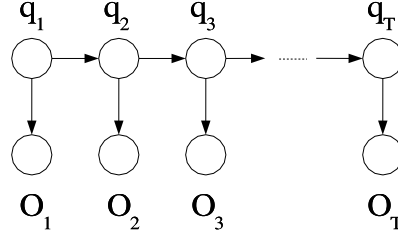


Figure 2.1: Graphical representation of a Hidden Markov Model. Modified from [11].

This joint probability is represented graphically in Figure 2.1.

In order to be able to use an HMM, it is necessary to know some details of the model. Specifically, the following are required:

1. An  $N \times N$  transition matrix that allows  $P(q_t|q_{t-1})$  to be determined. This transition matrix is often given as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}$$

where  $a_{ij}$  represents the probability of transitioning from a state  $q_t = S_i$  to state  $q_{t+1} = S_j$ . Clearly, the sum of each row in the transition matrix  $A$  must be unity, i.e.  $\sum_{j=1}^N a_{ij} = 1 \forall i$ . Generally, HMMs also have an *ergodic* property that  $a_{ij} > 0 \forall i, j$ , which means that for every state it is possible to transition to all other states.

2. A similar  $N \times M$  observation matrix that allows  $P(O_t|q_t)$  to be determined, given discrete observations in the set  $\{1, 2, \dots, M\}$ . That is:

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2M} \\ \vdots & \vdots & & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{NM} \end{bmatrix}$$

where  $b_{ij}$  represents the probability of observing  $O_t = j$  given  $q_t = S_i$ . Again,  $\sum_{j=1}^M b_{ij} = 1 \forall i$ . It is common to replace the matrix  $B$  in various formulae with a *probability mass function* (p.m.f.)  $b(\cdot) = b_i(j)$  of the same properties.

3. An initial state from the set  $\{S_1, S_2, \dots, S_N\}$  for  $q_1$ . Thus, an initial state distribution set  $\pi = \{\pi_i\}$  where  $\pi_i = P(q_1 = S_i), 1 \leq i \leq N$ , is needed.

It is generally assumed for HMMs that the two matrices are time invariant, which vastly simplifies the model.

These three elements,  $A, b(\cdot)$  and  $\pi$  define a discrete symbol HMM, which is often given as  $\lambda = (A, b(\cdot), \pi)$ .

## 2.2 Discrete Symbol Hidden Markov Model Algorithms

HMMs have two main algorithm classes – inference and learning (or training) [11]. Here, an algorithm of each class is examined.

### 2.2.1 Inference Using the Forward-Backward Algorithm

One way that an HMM can be used is to calculate the probability of observing the finite sequence of observations  $O_{1:T}$ . It is possible to calculate the probability by considering *all* possible sequences of states  $Q_{1:T}$ .

For example, consider *one* such sequence  $Q_{1:T}$ . In this case:

$$\begin{aligned} P(O_{1:T}|Q_{1:T}, \lambda) &= \prod_{t=1}^T P(O_t|q_t, \lambda) \\ &= \prod_{t=1}^T b_{q_t}(O_t) \end{aligned} \tag{2.2}$$

Consider also that the probability of the sequence  $Q_{1:T}$  occurring is:

$$\begin{aligned} P(Q_{1:T}|\lambda) &= P(q_1) \prod_{t=2}^T P(q_t|q_{t-1}, \lambda) \\ &= \pi_{q_1} \prod_{t=2}^T a_{q_{t-1}q_t} \end{aligned} \tag{2.3}$$

Thus, given the assumed independence of the states and observations, it is possible to state that the probability of the observation sequence  $O_{1:T}$  occurring simultaneously with the sequence of states  $Q_{1:T}$  is the product of the two probabilities:

$$P(O_{1:T}, Q_{1:T}|\lambda) = P(O_{1:T}|Q_{1:T}, \lambda) P(Q_{1:T}|\lambda) \tag{2.4}$$

(Note that Equation 2.4 is equivalent to Equation 2.1.)

Finally, as mentioned above, the probability of observing the sequence  $O_{1:T}$  can be obtained by considering *all* possible sequences of states. Thus, from Equation 2.4:

$$P(O_{1:T}|\lambda) = \sum_{\text{all } Q_{1:T}} P(O_{1:T}|Q_{1:T}, \lambda) P(Q_{1:T}|\lambda) \quad (2.5)$$

By substituting Equation 2.2 and Equation 2.3 into Equation 2.5 the general result is obtained:

$$P(O_{1:T}|\lambda) = \sum_{\text{all } Q_{1:T}} \left( \pi_{q_1} b_{q_1}(O_1) \prod_{t=2}^T a_{q_{t-1}q_t} b_{q_t}(O_t) \right) \quad (2.6)$$

Unfortunately, Equation 2.6 is of order  $2T \times N^T$  [25], which is intractable for large sequences of observations. Fortunately, the *Forward-Backward* algorithm – a special case of the belief propagation algorithm for Bayesian Networks [11, 29] – provides a better way of calculating the probability of observing the sequence of observations  $O_{1:T}$ .

As the name suggests, the Forward-Backward algorithm has two stages – a forward pass and a backward pass.

In the forward pass, the value of  $\alpha_t(i) = P(O_{1:t}, q_t = S_i|\lambda)$  is solved in two stages:

1. Firstly, the values of  $\alpha_1(i)$  are initialised:

$$\begin{aligned} \alpha_1(i) &= P(O_1, q_1 = S_i|\lambda) \\ &= \pi_i b_i(O_1) \end{aligned}$$

That is, for all possible states  $i = \{S_1, S_2, \dots, S_N\}$  that  $q_1$  could assume,  $\alpha_1(i)$  is set to the probability of being in state  $S_i$  with the required observation  $O_1$ .

2. Secondly, an induction step is used to calculate the probabilities of reaching all successive states with the required observation:

$$\begin{aligned} \alpha_t(i) &= \left( \sum_{j=1}^N P(O_{1:t-1}, q_{t-1} = S_j|\lambda) P(q_t = S_i|q_{t-1}, \lambda) \right) \\ &\quad \times P(O_t|q_t = S_i, \lambda) \\ &= \left( \sum_{j=1}^N \alpha_{t-1}(j) a_{ji} \right) b_i(O_t) \end{aligned} \quad (2.7)$$

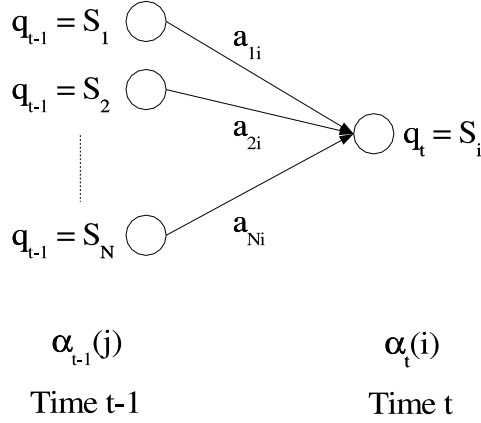


Figure 2.2: The induction step of the forward pass of the Forward-Backward algorithm. Modified from [25].

That is, for all possible states  $i = \{S_1, S_2, \dots, S_N\}$  that  $q_t$  could assume where  $1 < t \leq T$ , the probability of reaching each state with the required observation sequence (so far) of  $O_{1:t}$  is the product of observing  $O_t$  given the state  $q_t$  and the sum of the product of the probabilities of reaching all the preceding hidden states with the required observation sequence and the probability of transitioning from each of these states to the state  $q_t$ . This step is represented in Figure 2.2.

Once the forward pass is completed, it is possible to determine the probability of observing the sequence  $O_{1:T}$  by summing all the terminal  $\alpha_T(i)$  values:

$$P(O_{1:T}|\lambda) = \sum_{i=1}^N \alpha_T(i)$$

The forward pass illustrates the point mentioned earlier that each state contains the history of the process – as can be seen here, at each induction step of the forward pass, the probabilities of the previous states are combined. The forward pass algorithm is of order  $N^2T$  [25].

As may be expected, the Forward-Backward algorithm also has a backward pass. While the backward pass is not required for the solution of the problem of calculating the probability of observing the finite sequence of observations  $O_{1:T}$ , it is required as part of the training algorithm discussed later, and so is considered here.

The backward pass operates in a similar manner to the forward pass, and calculates the value of  $\beta_t(i) = P(O_{t+1:T}|q_t = S_i, \lambda)$  – that is, given the value of a hidden state at time  $t$ , it calculates the probability of observing

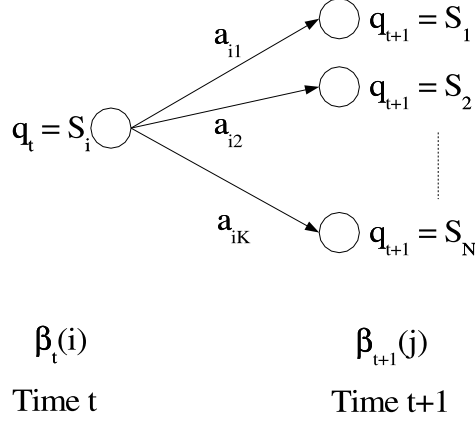


Figure 2.3: The induction step of the backward pass of the Forward-Backward algorithm. Modified from [25].

the (partial) observation sequence  $O_{t+1:T}$ . The two stages of the backward pass are:

1. Firstly, the values of  $\beta_T(i)$  are initialised:

$$\beta_T(i) = 1$$

That is, for all possible states  $i = \{S_1, S_2, \dots, S_N\}$  that  $q_T$  could assume,  $\beta_T(i)$  is arbitrarily set to 1.

2. Secondly, the following induction step is used:

$$\begin{aligned}
 \beta_t(i) &= \sum_{j=1}^N P(O_{t+2:T} | q_{t+1} = S_j, \lambda) \times P(q_{t+1} | q_t = S_i, \lambda) \\
 &\quad \times P(O_{t+1} | q_{t+1}, \lambda) \\
 &= \sum_{j=1}^N \beta_{t+1}(j) a_{ij} b_j(O_{t+1})
 \end{aligned} \tag{2.8}$$

That is, for all possible states  $i = \{S_1, S_2, \dots, S_N\}$  that  $q_t$  could assume where  $1 \leq t < T$ , the probability of observing the (partial) observation sequence  $O_{t+1:T}$  is the product of the probability of transitioning from state  $q_t = S_i$  to state  $q_{t+1}$ , the probability of observing  $O_{t+1}$  given  $q_{t+1}$  and the probability of the remaining partial sequence. This step is represented in Figure 2.3.

The backward pass algorithm is also of order  $N^2T$  [25].

Finally, it is important to note that because  $\alpha_t(i)$  represents the probability of reaching state  $q_t = S_i$  having observed the (partial) sequence  $O_{1:t}$  and that  $\beta_t(i)$  represents the probability of observing the sequence  $O_{t+1:T}$  given that  $q_t = S_i$ , then:

$$\begin{aligned} P(O_{1:T}) &= \sum_{i=1}^N \alpha_T(i) \\ &= \sum_{i=1}^N \alpha_t(i) \beta_t(i) \end{aligned} \tag{2.9}$$

### 2.2.2 Training Using the Baum-Welch Method

One of the hardest parts of building an HMM is determining how to set the two transition matrices and the initial state distribution  $\pi$  so that the probability of the observation sequence  $O_{1:T}$  is maximized – in other words, there has to be some way to train the HMM on valid observation sequences so that the model can be used for inference later on. The Expectation-Maximization algorithm is one such way of allowing the updated parameters of an HMM after training to be estimated [12]. The algorithm follows from the definition:

$$\mathcal{Q}(\bar{\lambda}|\lambda) = E \{ P(Q_{1:T}, O_{1:T}|\bar{\lambda}) \mid \lambda, O_{1:T} \} \tag{2.10}$$

where  $\mathcal{Q}$  is a function to maximize the expected probability  $E$  of the state and observation sequences  $Q_{1:T}, O_{1:T}$  of the trained HMM,  $\bar{\lambda}$ , given then existing model  $\lambda$  and the sequence of observations.

Unfortunately, this function is intractable [12], and so is normally only approximated. Here, the Baum-Welch method of approximating the new model is examined. The Baum-Welch method is equivalent to the Expectation-Maximization method for training Bayesian Networks, except that it allows the states of the model to be hidden [4, 9].

Firstly,  $\gamma_t(i)$  is defined as the probability of the hidden state  $q_t$  being equal to  $S_i$ , given the observation sequence  $O_{1:T}$ :

$$\begin{aligned} \gamma_t(i) &= P(q_t = S_i | O_{1:T}, \lambda) \\ &= \frac{P(q_t = S_i, O_{1:T} | \lambda)}{P(O_{1:T} | \lambda)} \end{aligned}$$

As previously discussed,  $\alpha_t(i)$  represents the probability of reaching state  $q_t = S_i$  having observed the (partial) sequence  $O_{1:t}$  and that  $\beta_t(i)$  represents

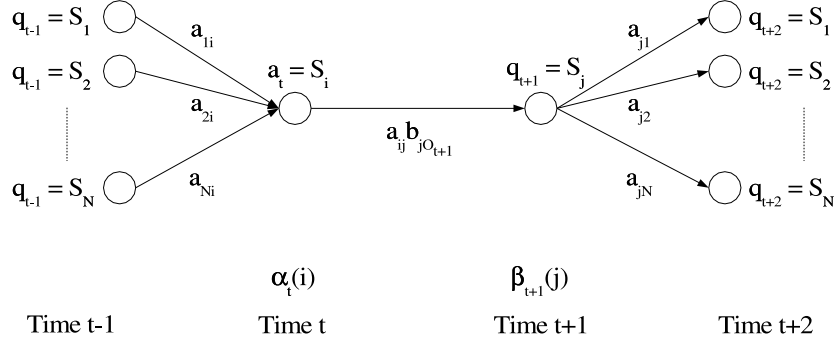


Figure 2.4: Graphical representation of the derivation of  $\xi_t(i, j)$  in the Baum-Welch method. Modified from [25].

the probability of observing the sequence  $O_{t+1:T}$  given that  $q_t = S_i$ . Thus:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{P(O_{1:T}|\lambda)} \quad (2.11)$$

By substituting Equation 2.9 into Equation 2.11:

$$\gamma_t(i) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)} \quad (2.12)$$

Secondly,  $\xi_t(i, j)$  is defined as the probability of the hidden state  $q_t$  being equal to  $S_i$ , and state  $q_{t+1}$  being equal to  $S_j$ , given the observation sequence  $O_{1:T}$ :

$$\begin{aligned} \xi_t(i, j) &= P(q_t = S_i, q_{t+1} = S_j | O_{1:T}, \lambda) \\ &= \frac{P(q_t = S_i, q_{t+1} = S_j, O_{1:T} | \lambda)}{P(O_{1:T} | \lambda)} \end{aligned} \quad (2.13)$$

By substituting the forward and backward pass values from the Forward-Backward algorithm into Equation 2.13, the following is obtained:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \quad (2.14)$$

The derivation of Equation 2.14 can be seen graphically in Figure 2.4.

Note that the two values  $\gamma_t(i)$  and  $\xi_t(i, j)$  can be related by summing over the possible values of  $j$ :

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$$

The sum over time ( $1 \leq t < T$ ) of  $\gamma_t(i)$  represents the number of transitions from a state with value  $S_i$ , while the sum over time ( $1 \leq t < T$ ) of  $\xi_t(i, j)$  represents the number of transitions from a state with value  $S_i$  to a state with value  $S_j$ .

The Baum-Welch *re-estimation formulae* are given by the following:

- The new value for  $a_{ij}$ , i.e.  $\bar{a}_{ij}$ , is the expected number of transitions from a state with value  $S_i$  to a state with value  $S_j$ , divided by the total number of transitions from a state with value  $S_i$ .
- The new value for  $b_i(j)$ , i.e.  $\bar{b}_i(j)$ , is the expected number of times the model is in a state with value  $S_i$  (which is equivalent to the sum over time ( $1 \leq t \leq T$ ) of  $\gamma_t(i)$ ) and the observation  $j$  is observed, divided by the expected number of times the model is in a state with value  $S_i$ .
- The new value for  $\pi_i$ , i.e.  $\bar{\pi}_i$ , is the expected frequency of  $q_1 = S_i$ .

Thus:

$$\begin{aligned} \bar{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T-1} \left( \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \right)}{\sum_{t=1}^{T-1} \left( \frac{\alpha_t(i) \beta_t(i)}{\sum_{i=1}^N \alpha_t(i) \beta_t(i)} \right)} \\ &= \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \end{aligned} \tag{2.15}$$

$$\bar{b}_i(j) = \frac{\sum_{t=1}^T \gamma_t(i) b_i(j)}{\sum_{t=1}^T \gamma_t(i)}$$

$$\bar{\pi}_i = \gamma_1(i)$$

The new HMM  $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$  has been shown to be equivalent to the previous HMM, or more likely to produce the observation sequence  $Q_{1:T}$  [2, 5]. Thus, this *re-estimation process* can be used to train an HMM given one or more sample sequences of observations<sup>1</sup>.

---

<sup>1</sup>It is important to note that as  $\gamma_t(i)$  and  $\xi_t(i, j)$  are calculated using the Forward-Backward algorithm – which leads to local, not global, maxima [25]. Thus, if the initial HMM is created using random variables, it may be necessary to train and test more than one HMM in order to obtain the “best” HMM.

## 2.3 On-line Discrete Symbol Hidden Markov Models

On-line HMMs [14] are a form of HMM that allow the model to be used and trained at the same time, thus alleviating the requirement of obtaining data and training the model before predictions of observation sequences can be obtained.

However, this means that for each observation, while the probability of the observation can be calculated, a new model estimate will also be obtained. Thus, the “model” will actually consist of a series of model estimates,  $\Lambda_T = (\lambda_1, \lambda_2, \dots, \lambda_T)$ , where  $\lambda_t = (A(t), b_i(t, \cdot))$ ,  $A(t) = \{a_{ij}(t)\}$ . (Here, the value of  $\pi$  for each model has been ignored, as it is only required for  $\lambda_0$  in the following algorithms.)

## 2.4 On-line Discrete Symbol Hidden Markov Model Algorithms

Due to the fact that an on-line HMM is actually a series of models, the previous methods for inference and training are no longer suitable, and new methods must be examined.

### 2.4.1 Inference Using the On-line Forward-Backward Algorithm

Given the definition of an on-line HMM above, the definitions of  $\alpha_t(i)$  and  $\beta_t(i)$  for standard HMMs can be updated to  $\alpha_{t|\Lambda_{t-1}}(i)$  and  $\beta_{t|\lambda_{t-1}}(i)$ , given an initial model  $\lambda_0$ :

$$\begin{aligned} \alpha_{1|\Lambda_0}(i) &= \pi_i b_i(0, O_1) \\ \alpha_{t|\Lambda_{t-1}}(i) &= \left( \sum_{j=1}^N \alpha_{t-1|\Lambda_{t-2}}(j) a_{ji}(t-1) \right) b_i(t-1, O_t) \\ \beta_T(i) &= 1 \\ \beta_{t|\lambda_{t-1}}(i) &= \sum_{j=1}^N \beta_{t|\lambda_{t-1}}(j) a_{ij}(t-1) b_j(t-1, O_{t+1}) \end{aligned} \tag{2.16}$$

It is worth noting that for on-line HMMs, it is standard practice to have a p.m.f.  $b(t, \cdot)$  that only accounts for those observations that have actually been observed up to time  $t$ . This is in contrast with standard HMMs, where the  $B$  array is fully populated before use and training of the HMM commences.

While it is clear that the values of  $\alpha_{t|\Lambda_{t-1}}(i)$  for a sequence of HMMs  $\Lambda_T$  can be calculated as the models are “developed” (trained), it should also be clear that the values of  $\beta_{t|\lambda_{t-1}}(i)$  can not be obtained as the models are developed, as these values rely on the complete sequence of observations  $O_{t+1:T}$  (and therefore, the complete sequence of HMMs  $\Lambda_{t-1}$ ) to be known. Thus, the method of training an on-line HMM must differ from that of the standard HMM.

### 2.4.2 On-line Training via the Fisher Information Matrix

Just as for the normal HMM, training an on-line HMM requires a technique for the approximation of the Expectation-Maximization algorithm (Equation 2.10).

Firstly, by replacing the  $\alpha_{t|\Lambda_{t-1}}(i)$  and  $\beta_{t|\lambda_{t-1}}(i)$  values above into Equation 2.12 and Equation 2.14 for  $\alpha_t(i)$  and  $\beta_t(i)$ , the on-line HMM definitions of  $\gamma_{t|\Lambda_{t-1}}(i)$  and  $\xi_{t|\Lambda_{t-1}}(i, j)$  can be obtained [14]:

$$\gamma_{t|\Lambda_{t-1}}(i) = \frac{\alpha_{t|\Lambda_{t-1}}(i) \beta_{t|\lambda_{t-1}}(i)}{\sum_{j=1}^N \alpha_{t|\Lambda_{t-1}}(j) \beta_{t|\lambda_{t-1}}(j)}$$

$$\xi_{t|\Lambda_{t-1}}(i, j) = \frac{\alpha_{t|\Lambda_{t-1}}(i) a_{ij}(t-1) b_j(t-1, O_{t+1}) \beta_{t+1|\lambda_{t-1}}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_{t|\Lambda_{t-1}}(i) a_{ij}(t-1) b_j(t-1, O_{t+1}) \beta_{t+1|\lambda_{t-1}}(j)}$$

Secondly, two new values are defined [14, 28]:

$$\mu_t(i, j) = \frac{\sum_{k=1}^t \xi_{k|\Lambda_{k-1}}(i, j)}{a_{ij}(t)^2} \quad (2.17)$$

$$g_t(i, j) = \frac{\xi_t(i, j)}{a_{ij}(t)}$$

Via the use of the Fisher Information Matrix (FIM), an algorithm for the update of the  $A$  matrix can be defined as [14, 28]:

$$\bar{a}_{ij}(t+1) = \bar{a}_{ij}(t) + \frac{1}{\mu_t(i, j)} \times \left( g_t(i, j) - \frac{\sum_{j'=1}^N g_t(i, j') \mu_t(i, j')}{\sum_{j'=1}^N \frac{1}{\mu_t(i, j')}} \right)$$

Similar use of the FIM results in two update algorithms for the p.m.f.  $b(t, \cdot)$  [28]. Firstly, the p.m.f. for observations  $n \forall n \neq O_t$  are updated:

$$\bar{b}_i(t, n) = \bar{b}_i(t-1, n) - b_i(t-1, O_t) \times \left( \frac{\gamma_{t|\Lambda_{t-1}}(i)}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)} \right)$$

$$\times \left( \frac{\frac{b_i(t, n)^2}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)}}{\sum_{p=1}^M \left( \frac{b_i(t, p)^2}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)} \right)} \right) \quad (2.18)$$

Secondly, the p.m.f. for the observation  $n = O_t$  is updated:

$$\bar{b}_i(t, n) = 1 - \sum_{p \neq n}^M \bar{b}_i(t, p)$$

Note the use of  $\beta_{t|\lambda_{t-1}}(i)$  values in these definitions means that, as noted above, the complete observation sequence  $O_{1:T}$  is required to calculate these values. In order to overcome this problem, two types of on-line HMM are considered – filtering and fixed lag.

### 2.4.3 Filtering On-line Hidden Markov Models

A filtering on-line HMM simply assumes that the sequence of observations  $O_{1:T}$  is a sequence on *one* observation. Thus, at each time interval, the model is updated using a single observation in isolation. As a result of Equation 2.16, this means that  $\beta_1(i) = 1$  is used for all update algorithms.

### 2.4.4 Fixed Lag On-line Hidden Markov Models

A fixed lag on-line HMM at time  $t = k$  assumes a finite sequence  $O_{k:k+\Delta}$ , where  $\Delta > 0$  is the *lag* [14]. By knowing the next  $\Delta$  observations, the  $\beta_{t|\lambda_{t-1}}(i)$  values can be calculated as if  $O_{k:k+\Delta}$  was the complete sequence of observations.

Obviously, if  $\Delta$  was equal to 0, then the model would in fact be a filtering on-line HMM.

## 2.5 Adding Forgetting to On-line Hidden Markov Models

As the FIM sometimes results in a too rapid convergence of the on-line HMM, it is desirable to add forgetting to an on-line HMM, so that the effect of past learning is reduced with respect to newer learning [14].

The *forgetting factor*  $0 < \rho \leq 1$ , where  $\rho = 1$  means no forgetting, is added to Equation 2.17 resulting in:

$$\mu_t(i, j) = \frac{\sum_{k=1}^t \rho^{t-k} \xi_{k|\Lambda_{k-1}}(i, j)}{a_{ij}(t)^2}$$

The forgetting factor is also added to Equation 2.18 by changing the three occurrences of the summation of  $\gamma$  to:

$$\sum_{k=1}^t \rho^{t-k} \gamma_{k|\Lambda_{k-1}}(i)$$

Number of Bits	Field
4	IP Version
4	IP Header Length
8	Type of Service
16	Total IP Datagram Length
16	Identification
3	Control Flags
13	Fragment Offset
8	Time to Live
8	Protocol
16	Header Checksum
32	Source Address
32	Destination Address
Variable	Options
Variable	Data

Table 2.1: The Internet Protocol version 4 datagram format [23].

## 2.6 Internet Worms

For an Internet worm to cause extensive damage, it must be capable of propagating widely over the Internet. This restricts Internet worms to using packets (or *datagrams*) from one of two main protocols – the Transmission Control Protocol (TCP) [21] or the User Datagram Protocol (UDP) [22]. Both of these protocols generally run (at present) over the Internet Protocol (IP) version 4 [23], that is, the data field of an IP datagram can be a TCP or UDP datagram. The details of the datagram structure of these three protocols can be seen in Table 2.1, Table 2.2, and Table 2.3.

The datagram field values (excluding the data fields) are restricted to a finite set of discrete values, and could therefore be considered as observations for a discrete symbol HMM.

Note, however, that not all of this information is likely to be useful in the process of identifying Internet worm packets. For example, the checksum headers can safely be discarded from consideration, as packets with bad checksums should be discarded by the network interface, meaning there is little point in an Internet worm sending packets with bad checksums. The TCP and UDP data field is also unlikely to be useful as an HMM observation symbol, partially due to its potentially very large discrete space, and also due to its variable length.

Number of Bits	Field
16	Source Port
16	Destination Port
32	Sequence Number
32	Acknowledgment Number
4	Data Offset
6	Reserved
8	Control Bits
16	Windows Size
16	TCP Checksum
16	Urgent Pointer
Variable	Options
Variable	Data

Table 2.2: The Transmission Control Protocol datagram format [21].

Number of Bits	Field
16	Source Port
16	Destination Port
16	Total UDP Datagram Length
16	UDP Checksum
Variable	Data

Table 2.3: The User Datagram Protocol datagram format [22].

## 2.7 Design of an Anomaly-Based Network Intrusion Detection System Based on Hidden Markov Models

Given the above information about HMMs and Internet worms, an anomaly-based network intrusion detection system for Internet worms based on the standard, discrete symbol HMM appears to be a less than ideal solution. The HMM's requirement for a  $B$  matrix of dimensions  $N \times M$  could be very large, even with a small number of states, due to the potentially large size of  $M$ .

For example, if the system was designed to attempt to detect Internet worms based only on the source and destination ports of packets, the observations would be represented by a single 32-bit integer. To build the  $B$  matrix in the obvious way (using an actual matrix), this would be the upper limit of possible observation data that could be used, as most programming languages today will only allow the index of such structures to be a 32-bit integer. Thus, trying to detect Internet worms using more information about the packets would not be possible. Indeed, some HMM implementations

(such as UMDHMM [13]) crash when trying to use such a range of symbols.

Also, building a system using a standard HMM would require that the system be fully trained with data, and preferably tested, before it can be used to try to detect Internet worms. Such a system would therefore fail to meet the requirements stated above, that is, that minimal training is desired.

On consideration, an on-line discrete symbol HMM seems a better choice for building an anomaly-based network intrusion detection system for Internet worms. As the  $B$  matrix only needs to deal with observation symbols that have actually been seen, it would seem logical to implement the matrix as some kind of hash table, adding in probability rows for observations as required. Provided the key to the table can exceed 32-bits, then using more information than just, for example, the TCP/UDP datagram port numbers would also be possible.

Finally, as the system is on-line, training of the system would be able to be carried out at the same time as it was in use, freeing system administrators from the time consuming task of training and testing the system.

## Chapter 3

# Implementation of an On-line Hidden Markov Model

### 3.1 Implementation

As no implementation of an on-line Hidden Markov Model (HMM) was freely available, a system was implemented in Java<sup>1</sup>. The system supports both filtering and fixed lag modes, as well as forgetting, as described in Chapter 2.

Important features of the implementation that are generally ignored by the literature are:

- An HMM is initialised with random values for the  $\pi$  and  $A$  matrices, using a supplied pseudo-random number generator (PRNG) seed so that results can be reproduced. Each probability row is normalised to ensure a sum of unity. An initial minimum probability value is also enforced, ensuring that the model is ergodic.
- The  $B$  matrix is initially empty, and columns for observations are only added as the symbols are seen. When a new symbol is seen, the probability for each state in that symbol's column is set to the initial minimum probability value, and the matrix is re-normalised to ensure that each probability row again has a sum of unity. The use of the initial minimum probability value, instead of a random value, allows convergence of the  $B$  matrix values to occur much more rapidly, especially when new observations are encountered often.
- During training, the  $\alpha$  and  $\beta$  are normalised after calculation for each step [18, 25], as are the  $\gamma$  and  $\xi$  values. Once updated, the  $A$  and  $B$  matrices are also normalised.
- The fixed lag mode of operation returns a probability of zero for all observations until at least  $\Delta$  observations have been seen, as this many

---

<sup>1</sup><http://java.sun.com/>

past observations are required to correctly calculate the probability using the forward pass of the Forward-Backward algorithm.

- In order to reduce the possibility of divide-by-zero errors due to near-zero probability values in the  $A$  matrix,  $\mu_t(i, j)$  (Equation 2.17) is actually calculated as  $\frac{1}{\mu_t(i, j)}$ .

The implementation uses a Java `Hashtable` to store the  $B$  matrix, so that observation symbols can be used as index keys. This allows simple observation column lookups when calculating the expected probability of a sequence of symbols using the forward pass of the Forward-Backward algorithm, as well as similarly simple updates of the matrix. However, normalisation of the matrix requires the reconstruction of each row and the storage of the normalised row back into the table again. This aspect of the implementation could be improved upon.

## 3.2 Verification of Implementation

As a verification of the implementation, a two-state, two-observation HMM was constructed with the following matrices:

$$\pi = [ 0.5 \quad 0.5 ] \quad A = \begin{bmatrix} 0.8 & 0.2 \\ 0.45 & 0.55 \end{bmatrix} \quad B = \begin{bmatrix} 0.8 & 0.2 \\ 0.45 & 0.55 \end{bmatrix}$$

This model was used to generate a sequence of observations, which were then used to train two new, randomly initialised two-state, two-symbol models, one with  $\Delta = 0$  and  $\rho = 0.99$ , and the other with  $\Delta = 5$  and  $\rho = 0.99$ . As the training was performed, the values of the  $A$  and  $B$  matrix entries were recorded.

Figure 3.3 shows the four  $A$  matrix values for the HMM with  $\Delta = 5$ , which are clearly converging on the original model's values as the training progresses.

Figure 3.1 shows the four  $A$  matrix values for the HMM with  $\Delta = 0$ , while Figure 3.2 and Figure 3.4 show the four  $B$  matrix values for the two models. While these three graphs show the  $A$  and  $B$  matrix values are converging, the values that they are converging on are close to, but not exactly the same as, the original matrix values. However, it would appear that the results are sufficiently close to the original values for the implementation to be considered valid [14, 28].

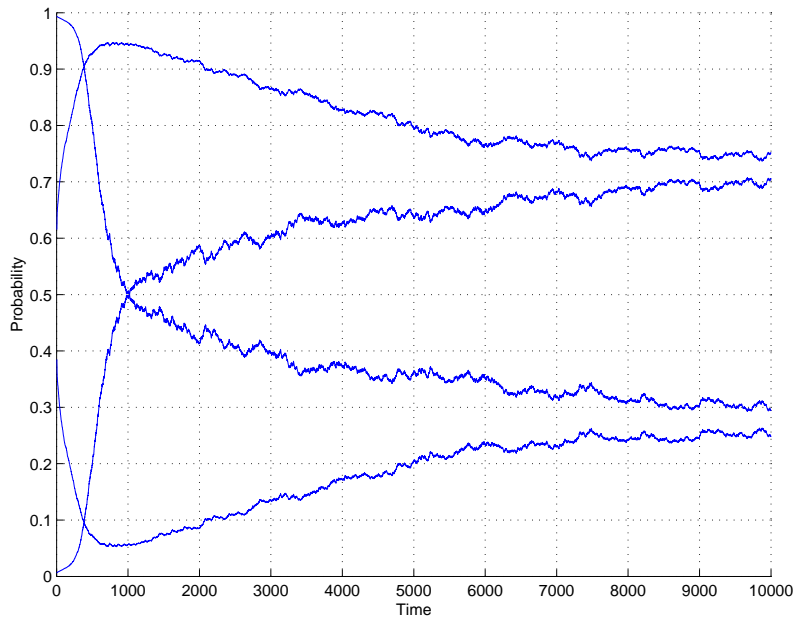


Figure 3.1: Convergence of the state transition probability matrix,  $\Delta = 0$ .

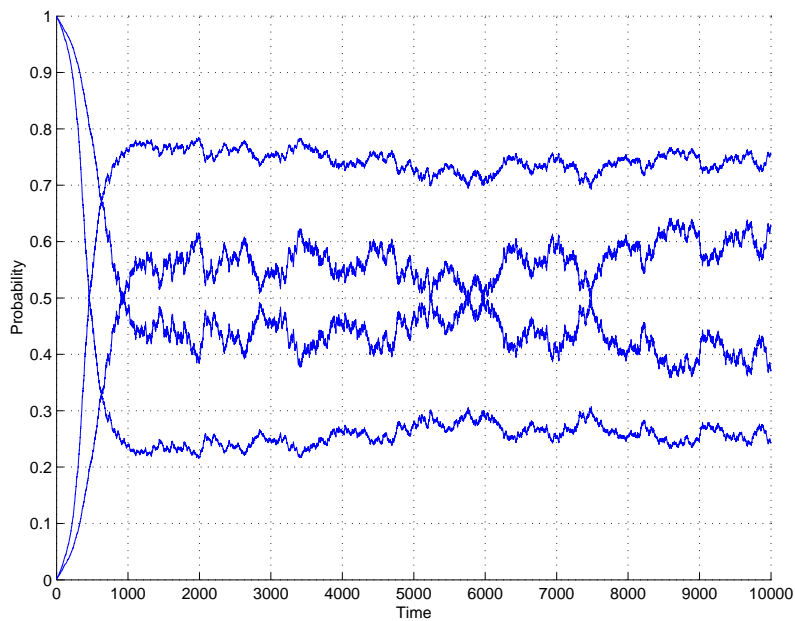


Figure 3.2: Convergence of the observation probability matrix,  $\Delta = 0$ .

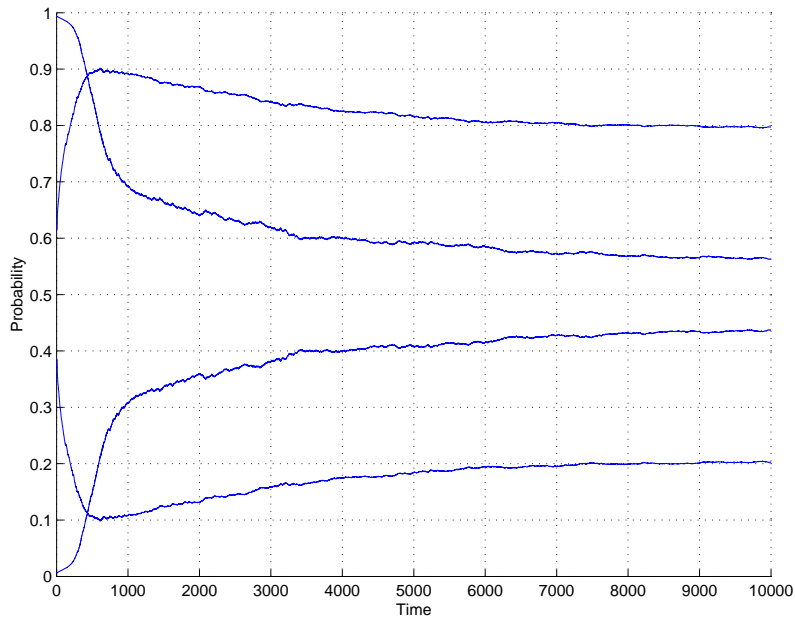


Figure 3.3: Convergence of the state transition probability matrix,  $\Delta = 5$ .

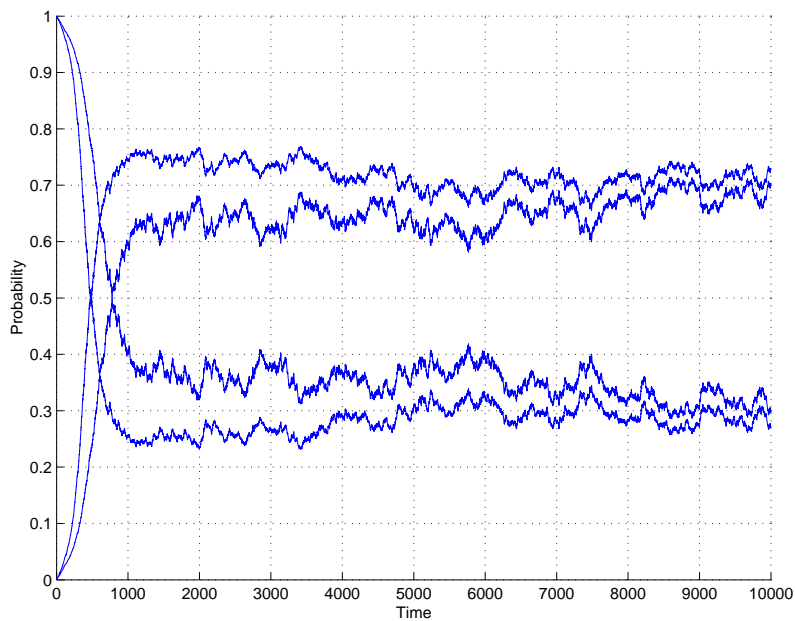


Figure 3.4: Convergence of the observation probability matrix,  $\Delta = 5$ .

## Chapter 4

# On-line Hidden Markov Model Configuration Experiments Toward the Design of a Model for UDP Internet Worm Detection

### 4.1 The Relationship Between Hidden Markov Models and Internet Packets

As discussed in Section 1.6, the structure of a Hidden Markov Model (HMM) is able to take into account the interrelated nature of Internet packets. At this point, it is worth examining this idea further.

Recall that an HMM is a model of a process with a finite sequence of hidden states, generating a finite sequence of observations. Given that a computer connected to the Internet will generate a sequence of TCP/IP or UDP/IP datagrams (both sent and received), some of which will be valid – or “normal” – network packets, and some of which will be undesirable Internet worm packets, it seems obvious to consider the sequence of packets as an HMM’s sequence of observations.

Obviously, such a model would need to have some way of representing the packet observations. As mentioned in Section 2.6, TCP/IP and UDP/IP datagrams consist of a number of different fields – thus, some combination of these fields can be thought of as the observation for packets.

However, if datagram fields are assumed to be the observations, then what is the model’s finite hidden-state sequence process? This process can be considered to be some abstract “Internet-wide” process that is responsible for determining what packets are sent to and from a computer. The number

of states of the HMM represents, in some way, the number of states required to model this abstract, Internet-wide process. There may be many or few states, but provided the model is able to predict normal, non-Internet worm packets in a packet sequence with a high probability, and predict Internet worm datagrams with a low probability, it is not important how many states the model has.

## 4.2 Collection of an Internet Datagram “Observation” Sequence

The TCP/IP or UDP/IP datagram header information of TCP/UDP datagrams being sent both to and from a computer within the University of Adelaide’s School of Computer Science network was collected using the `tcpdump`<sup>1</sup> utility. The packet data were then processed using a Perl<sup>2</sup> script to generate TCP packet and UDP packet SDBM database files. (This database format was selected simply on the basis that it was the only one available on the machines allocated for use in this study.) These two database files were used as the source of datagram observations as described above for the experiments in this chapter and the next.

## 4.3 Observation Sequence Prediction Experiments

In order to determine what kind of HMM configuration might be useful in detecting UDP-based Internet worms, a number of on-line HMMs with different initialisation PRNG seeds, numbers of states,  $\Delta$  values (such that  $\Delta \geq 0$ , so that both filtering and fixed lag models were included), and  $\rho$  values were trained and tested with sequences of datagram field observations from the UDP packet database – the “observation sequences.” The observation sequences were all of length 500 datagrams, and started at random positions in the database of packet data.

These experiments were performed in order to determine what effect these HMM configuration parameters have on the model’s ability to predict UDP observation sequences – or, more accurately, the effect they have on the predicted probability of each packet in the observation sequences.

Note that the decision was made to limit these configuration experiments, and the subsequent Internet worm experiments in Chapter 5 to the UDP/IP protocols. This is because it has been assumed that, as UDP is a connectionless protocol, there is *less* information available to the model than with TCP/IP observations, and so detecting UDP-based Internet worms is *harder* than detecting TCP-based Internet worms.

---

<sup>1</sup><http://www.tcpdump.org/>

<sup>2</sup><http://www.cpan.org/>

### 4.3.1 Effect of the Number of Hidden States

Models with different numbers of hidden states were tested. The initialisation PRNG seed,  $\Delta$  value,  $\rho$  value, random starting packet value, and observation sequence length were kept constant for each model comparison set. The observations used for all of these tests were the datagram source and destination ports of the UDP datagrams.

One example set typical of the results obtained is shown in Figure 4.2 (two states), Figure 4.3 (eight states), Figure 4.4 (fourteen states), and Figure 4.5 (twenty states).

It is clear from these experiments that there is little advantage in using models with a large number of states – at least for the above observation type – as there is little practical difference in the predicted probabilities of the packets in the observation sequence.

### 4.3.2 Effect of the Value of $\Delta$

Models with different  $\Delta$  values were tested. The initialisation PRNG seed, number of hidden states,  $\rho$  value, random starting packet value, and observation sequence length were kept constant for each model comparison set. The observations used for all of these tests were the datagram source and destination ports of the UDP datagrams.

One example set typical of the results obtained is shown in Figure 4.2 ( $\Delta = 1$ ), Figure 4.6 ( $\Delta = 5$ ), Figure 4.7 ( $\Delta = 20$ ), Figure 4.8 ( $\Delta = 50$ ), and Figure 4.9 ( $\Delta = 100$ ).

It is clear from these experiments that increasing the  $\Delta$  value of the model results in smoother curves of the predicted probabilities of the packets in the observation sequence. This is not unexpected, as the probabilities returned are based on the last  $\Delta$  observations, rather than just the single observations. Thus, the effect of sharp probability peaks is reduced by increased  $\Delta$ .

### 4.3.3 Effect of the Value of $\rho$

Models with different  $\rho$  values were tested. The initialisation PRNG seed, number of hidden states,  $\Delta$  value, random starting packet value, and observation sequence length were kept constant for each model comparison set. The observations used for all of these tests were the datagram source and destination ports of the UDP datagrams.

One example set typical of the results obtained is shown in Figure 4.6 ( $\rho = 0.99$ ), Figure 4.10 ( $\rho = 0.95$ ), Figure 4.11 ( $\rho = 0.80$ ), Figure 4.12 ( $\rho = 0.50$ ), and Figure 4.13 ( $\rho = 0.10$ ).

It is clear from these experiments that as the value of  $\rho$  is decreased, the model is less able to predict the observation sequence. This result is also to be expected, because as  $\rho$  decreases, the model forgets previous learning

more quickly, and so the model is increasingly less likely to be able to produce a high probability value for each packet in the observation sequences that it has previously learned about.

#### 4.3.4 Effect of Different Observations

In order to confirm that no advantage in predicting the observation sequence can be obtained by using HMMs with a large number of states, models with different numbers of hidden states were again tested. As before, the initialisation PRNG seed,  $\Delta$  value,  $\rho$  value, random starting packet value, and observation sequence length were kept constant for each model comparison set. However, different observations from before were tested.

The observations tested were:

- UDP datagram source and destination addresses and ports.
- UDP datagram source and destination ports, and data field length.
- UDP datagram source and destination addresses, and data field length.
- UDP datagram source and destination addresses and ports, and data field length.
- UDP datagram source and destination ports, and IP datagram identification field.
- UDP datagram source and destination addresses, and IP datagram identification field.
- UDP datagram source and destination addresses and ports, and IP datagram identification field.
- UDP datagram source and destination addresses and ports, and data field length, and IP datagram identification field.

One example set typical of the results obtained using the UDP datagram source and destination ports, and data field length as the observations can be seen in Figure 4.14, Figure 4.15, Figure 4.16, and Figure 4.17.

Another example set typical of the results obtained using the UDP datagram source and destination addresses and ports, and data field length as the observations can be seen in Figure 4.18, Figure 4.19, Figure 4.20, and Figure 4.21.

These two example sets of results are typical of the results obtained using other observation types (results not shown).

These results confirm that no apparent advantage in predicting the observation sequence can be obtained by using a large number of hidden states.

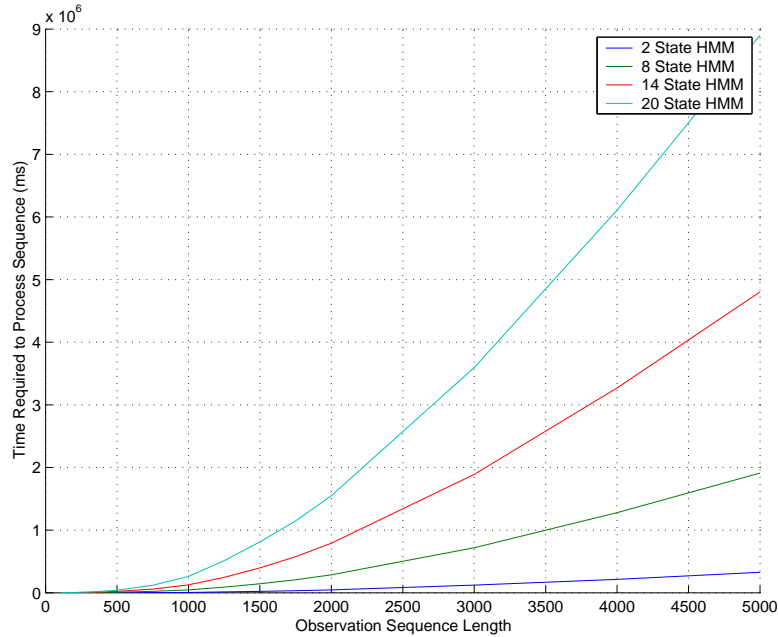


Figure 4.1: Example model training times for HMMs with different numbers of states and for different observation sequence lengths.  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations. Example demonstrated is a “near-worst case” observation sequence.

However, it is important to note that there is little practical difference between these results and the results shown in Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5. This demonstrates that increasing the level of datagram field information used in packet observations does not result in an increase in the predicted probability of the observation sequence.

Thus, very little packet data is likely to be required in an observation for the model to operate effectively.

#### 4.4 Implementation Speed Experiments

Models with two, eight, fourteen and twenty hidden states were trained with observation sequence lengths ranging between 200 and 5000 datagrams, and the time required to train each of the model/observation length combinations recorded. The initialisation PRNG seed,  $\Delta$  value,  $\rho$  value and random starting packet value were kept constant for each comparison set. The observations used for all of these tests were the datagram source and destination ports of the UDP datagrams.

All timing measurements were made on an AMD Athlon XP 2200+ PC, with 256MB RAM.

#### 4.4.1 Effect of the Observation Sequence Length

One example set typical of the results obtained can be seen in Figure 4.1. These results clearly show that increasing the observation sequence length results in a non-linear increase in the time required for the implementation to train the model. (See Appendix A for an order analysis of the training algorithms which supports this result by demonstrating a polynomial increase in training speed with the number of different observations seen by the model – the example results shown in Figure 4.1 have been obtained using an observation sequence with almost no repeated observations, and so is a “near-worst case” example of the increase in training time with the observation sequence length.)

#### 4.4.2 Effect of the Number of Hidden States

The results shown in Figure 4.1 also clearly show that increasing the number of hidden states results in a non-linear increase in the time required for the implementation to train the model. (See Appendix A for an order analysis of the training algorithms supporting this result.)

### 4.5 Model Design

Based on the above results, it would seem that in order to be able to successfully predict a sequence of UDP datagrams, an HMM should have the following configuration:

1. Two hidden states, as no apparent advantage is gained by having more hidden states, while there is a definite speed advantage in having fewer states.
2. A  $\Delta$  value of around 20, to reduce the effect of probability spikes. However, the value of  $\Delta$  is probably the model configuration parameter most likely to affect the ability of an HMM to predict Internet worms, because of its smoothing effect on probability spikes – something that Internet worms will hopefully generate.
3. A  $\rho$  value of 0.99, as increasing the forgetting rate only leads to the decreased ability of the model to predict a sequence of datagrams.

It would also seem that highly detailed information about the UDP datagrams in the form of the model observations used is not required to predict the packet sequence, and that the UDP datagram source and destination ports will be sufficient as the datagram fields for packet observations.

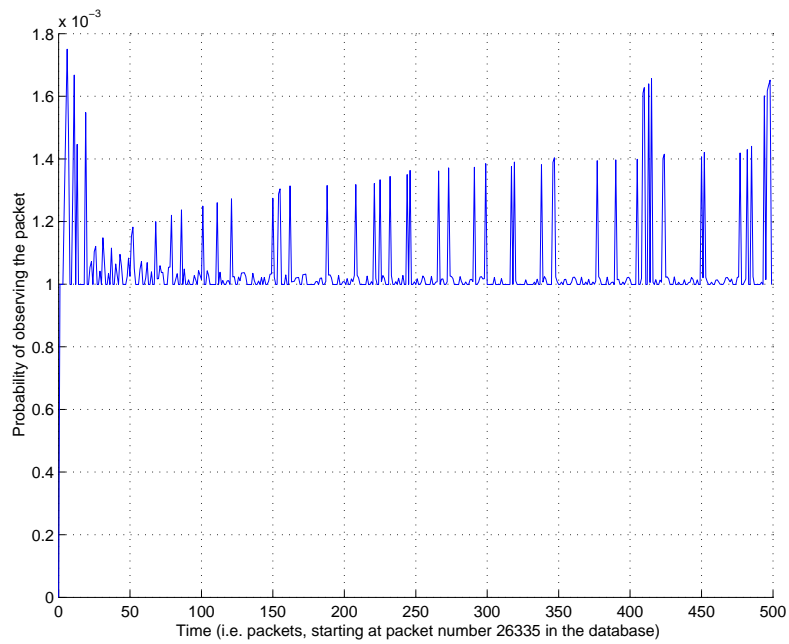


Figure 4.2: Example 2 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

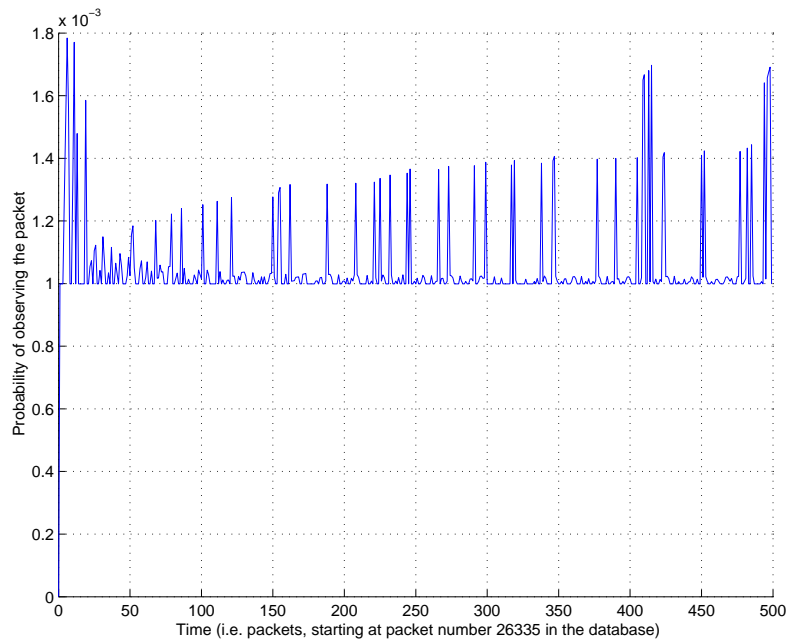


Figure 4.3: Example 8 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

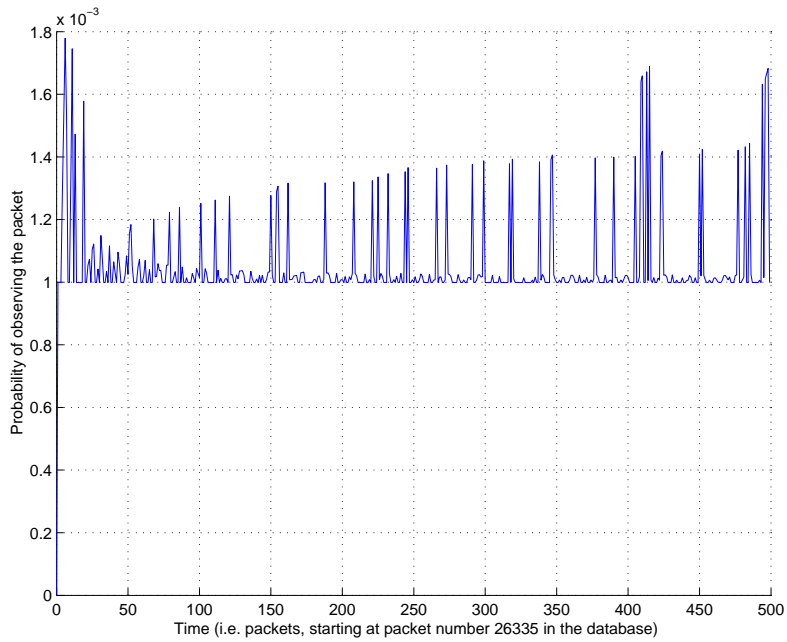


Figure 4.4: Example 14 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

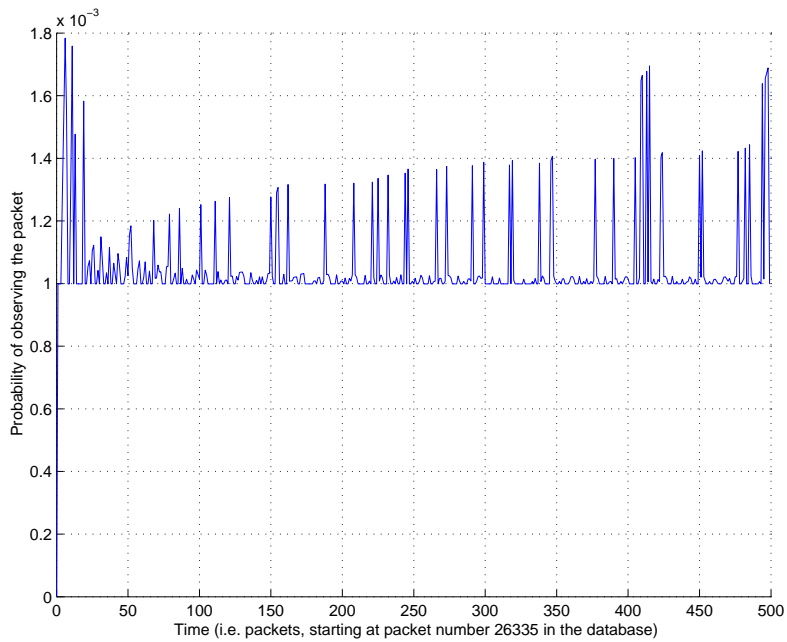


Figure 4.5: Example 20 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

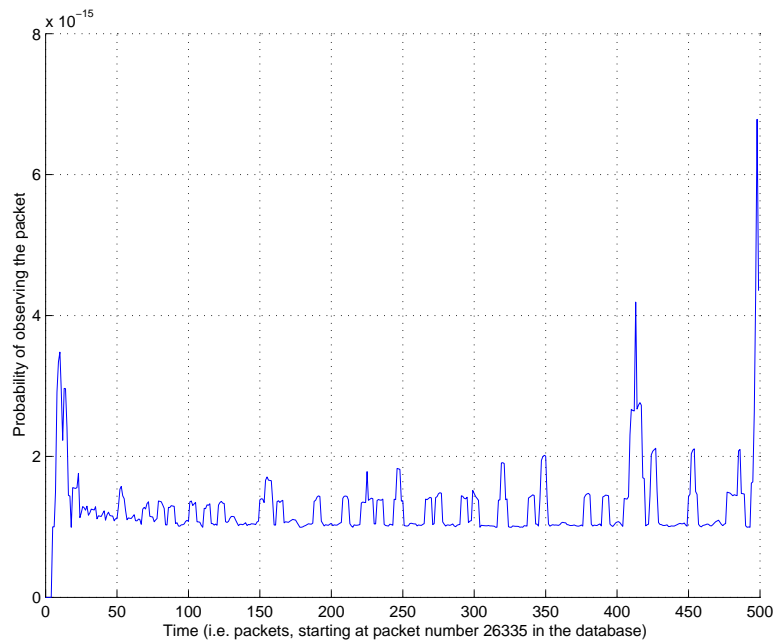


Figure 4.6: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

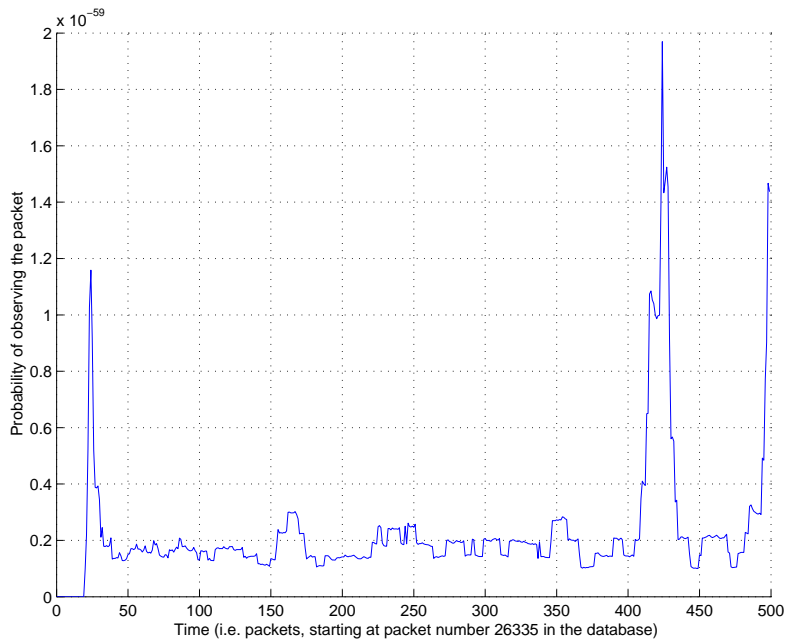


Figure 4.7: Example 2 state HMM with  $\Delta = 20$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

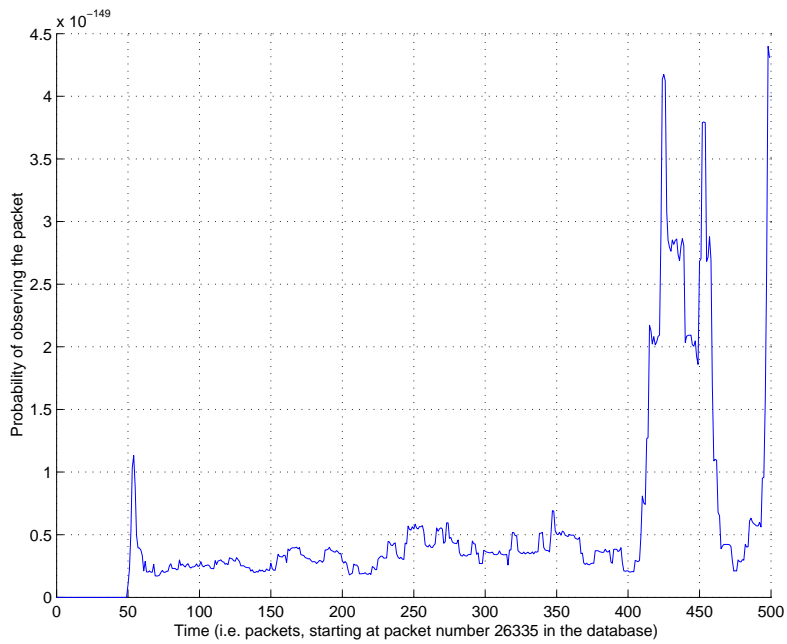


Figure 4.8: Example 2 state HMM with  $\Delta = 50$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

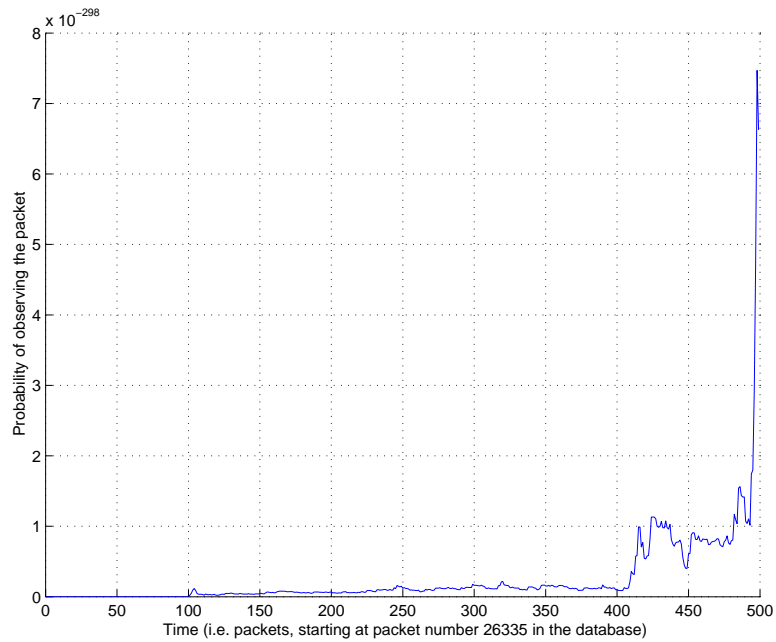


Figure 4.9: Example 2 state HMM with  $\Delta = 100$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination port observations.

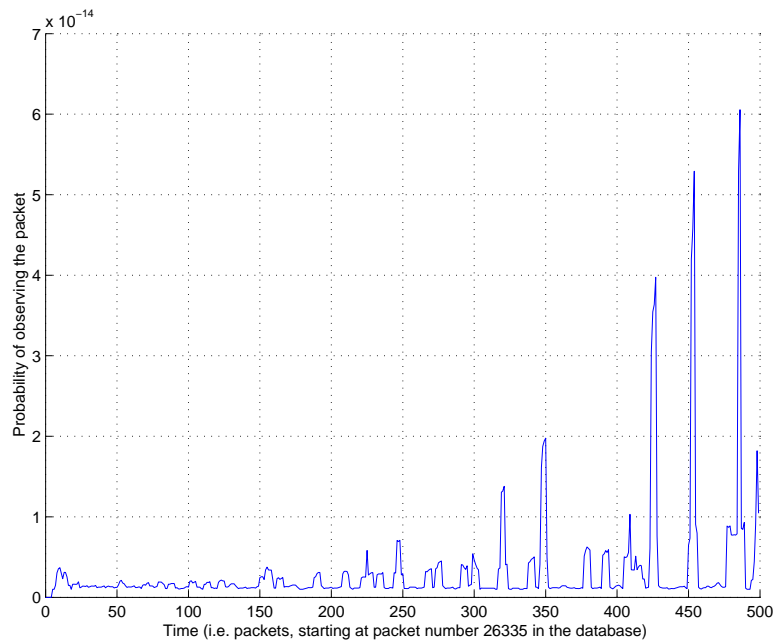


Figure 4.10: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.95$ , trained with UDP datagram source and destination port observations.

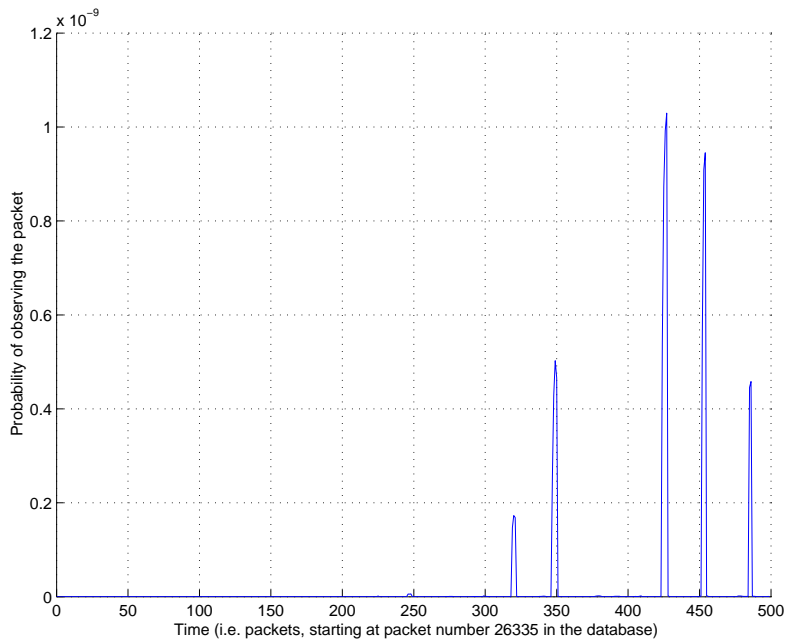


Figure 4.11: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.80$ , trained with UDP datagram source and destination port observations.

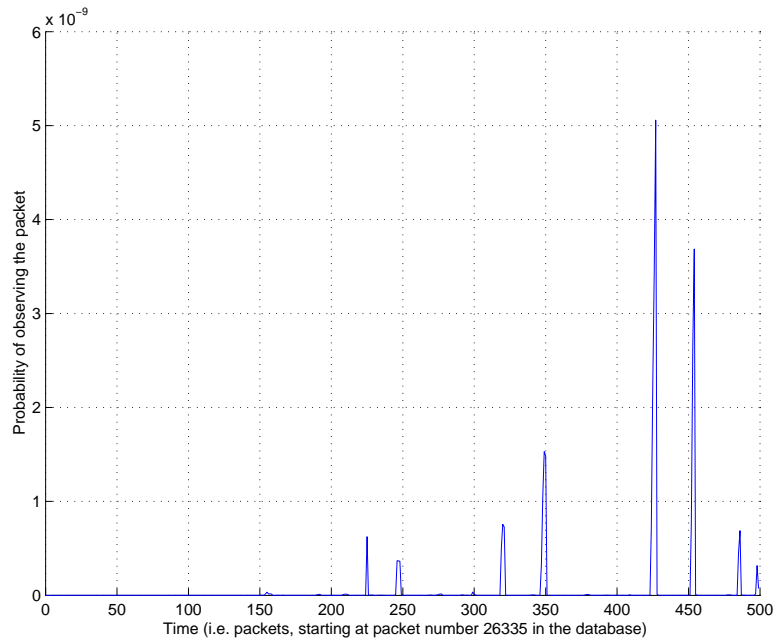


Figure 4.12: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.50$ , trained with UDP datagram source and destination port observations.

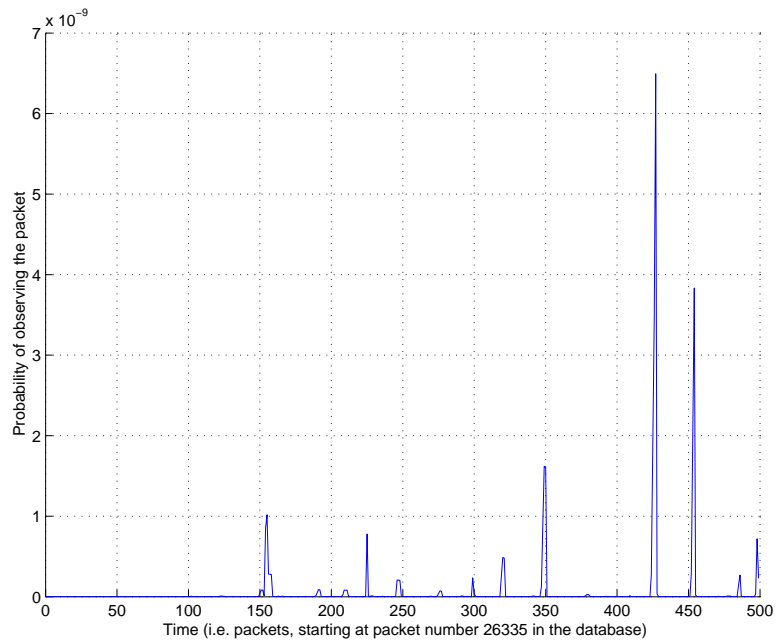


Figure 4.13: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.10$ , trained with UDP datagram source and destination port observations.

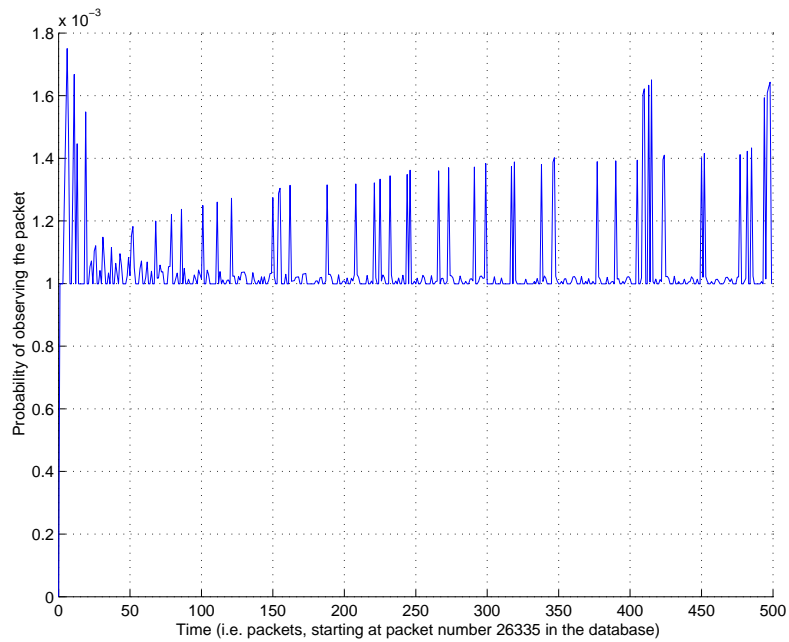


Figure 4.14: Example 2 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations.

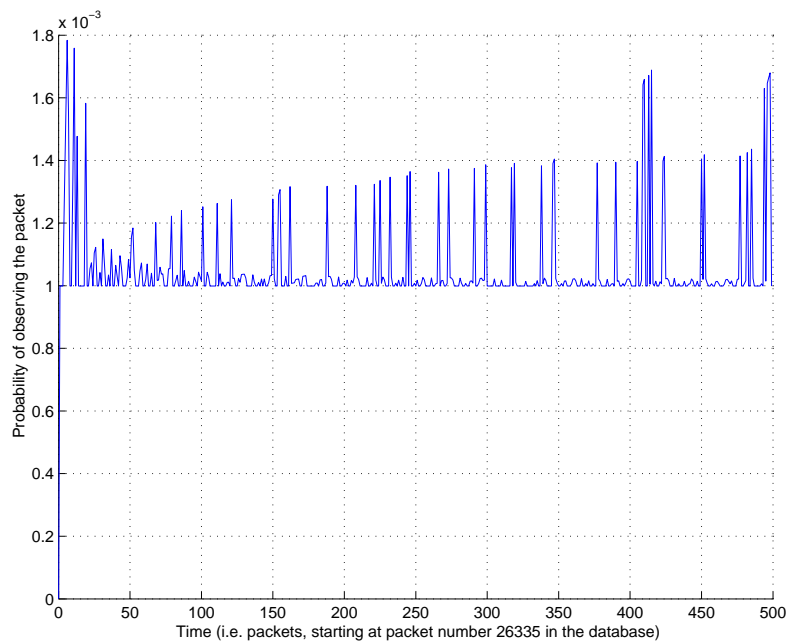


Figure 4.15: Example 20 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations.

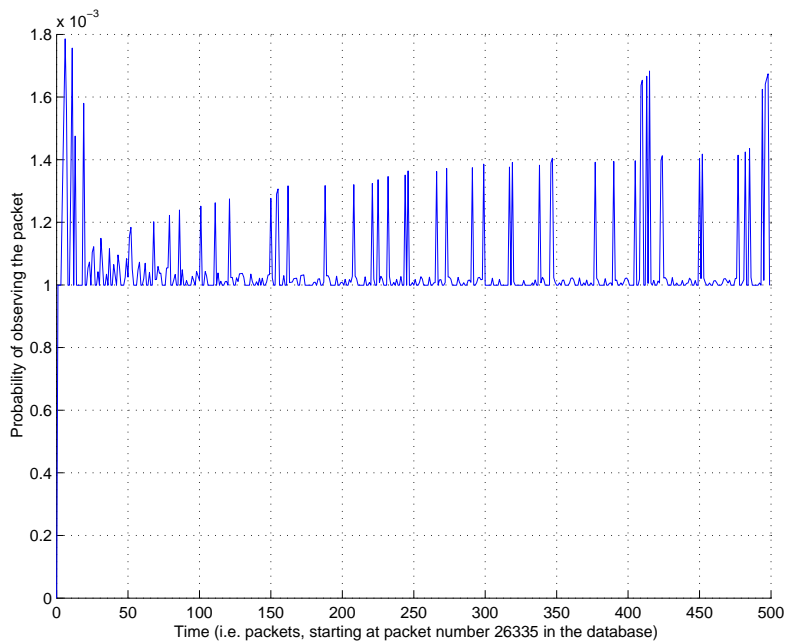


Figure 4.16: Example 50 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations.

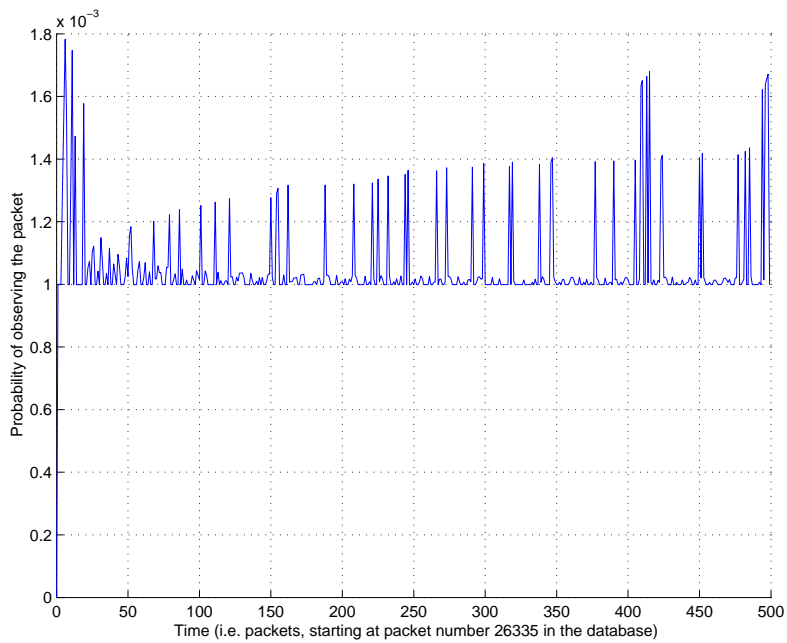


Figure 4.17: Example 100 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports, and data field length observations.

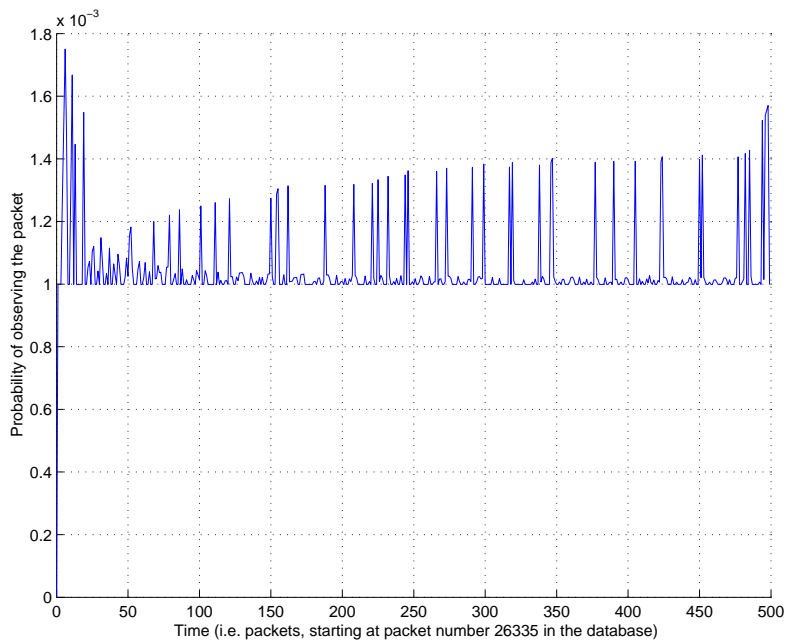


Figure 4.18: Example 2 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length.

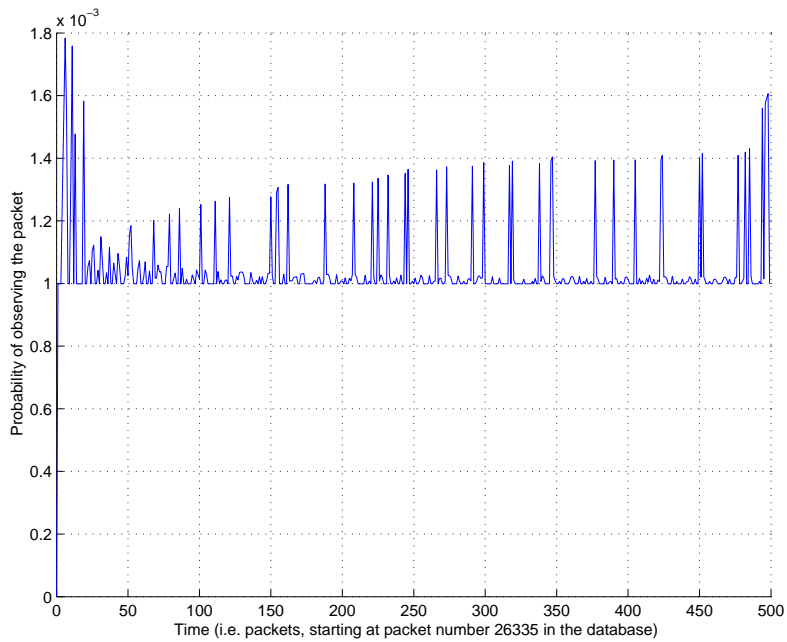


Figure 4.19: Example 20 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length.

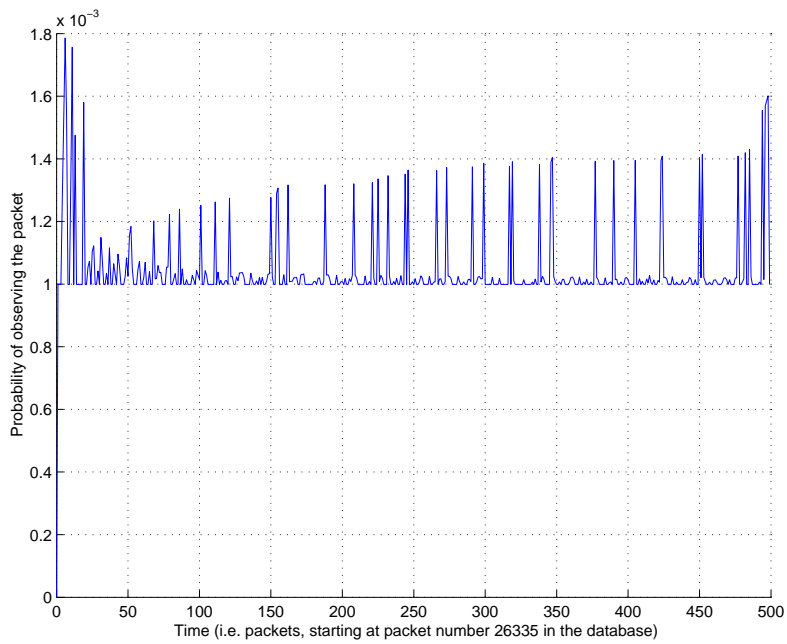


Figure 4.20: Example 50 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length.

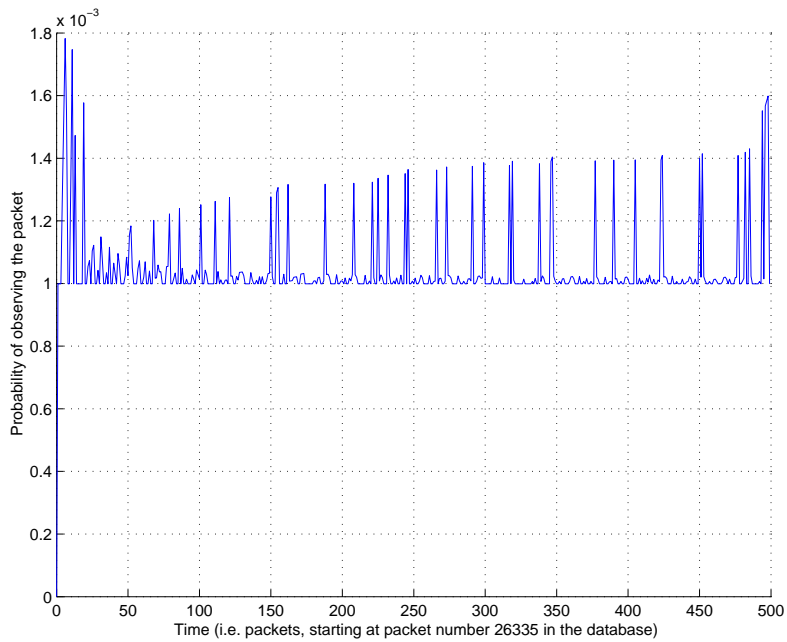


Figure 4.21: Example 100 state HMM with  $\Delta = 1$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination addresses and ports, and data field length.

## Chapter 5

# Internet Worm Packet Detection with an On-line Hidden Markov Model

### 5.1 The Sapphire Internet Worm

The Sapphire (or Slammer) Internet worm exploits a buffer overflow bug in the Microsoft SQL Server and SQL Server Desktop Engine services [19]. It caused wide-spread congestion on the Internet in 2003.

The worm propagates by sending a single UDP packet to port 1434 on randomly selected hosts [19]. The data field of the packet is 376 bytes [19].

### 5.2 Sapphire Worm Packet Detection Experiments With Set “Attack” Periods

Given the above, and the model design of Section 4.5, it was decided that in order to examine the ability of an anomaly-based network intrusion detection system based on HMMs to detect Sapphire worm datagrams, a number of on-line HMMs would be examined with the following properties:

- Various model initialisation PRNG seeds.
- Two hidden states.
- $\Delta$  values between 1 and 50.
- A  $\rho$  value of 0.99.

These models were trained with observation sequences of length 6000 packets – longer observation sequences were not tested due to the polynomial increase in training time required as the number of different observations

seen by the model increases (see Section 4.4 and Appendix A). The UDP source and destination ports were used as the datagram observations, and were taken from the UDP packet database, with various random starting packet values.

Each model was trained twice – once with just the observations from the UDP packet database (the “normal,” non-Internet worm packets), and once with some of the datagrams in the identical observation sequence replaced randomly (with a probability of 5%) with artificially generated Sapphire worm UDP datagrams. Only datagrams in the observation sequence ranges of 2000 to 3000 datagrams, and 4000 to 5000 datagrams were replaced.

The observation sequence values between which Sapphire packets would randomly replace the original datagrams were selected for two reasons. Firstly, they gave the HMMs tested time to be trained with the UDP observations of non-Internet worm traffic before Internet worm packets are seen. Secondly, they allow the behaviour of the system to be observed before, during, between and after periods where Internet worm “attacks” occur.

The Sapphire Internet worm datagrams generated had random values for the UDP/IP datagram fields, except for the UDP destination port (1434), IP don’t fragment flag, which is one of the three control flags (1) and the UDP data length field (376). The random UDP source port was limited to the range 1024 – 65535 (as ports below 1024 are restricted to root processes when sending packets).

### 5.2.1 Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of the Observation Sequence

In order to determine if a datagram is due to an Internet worm or simply part of the normal traffic of packets, it was decided to differentiate packets based on the average predicted probability of all packets in the observation sequence seen so far. Assuming that the HMM is able to predict the normal packets, then the predicted probability of these packets in the observation sequence should be greater than or equal to the average value, while Internet worm packets should have a predicted probability less than the average.

The initialisation PRNG seed used for the HMMs tested had little effect on the results obtained, and examples shown in this chapter are typical of the results obtained using different seeds (results not shown).

Figure 5.1 shows one example UDP datagram observation sequence on an HMM with  $\Delta = 5$ , where no Sapphire packets exist in the observation sequence. This  $\Delta$  value appeared to give the “best” results of the different  $\Delta$  values tested (results not shown), based on the criteria described above for the desired relationship between the predicted probabilities of the normal packets in the observation sequence and the average observation sequence probability. However, some of the normal packets do have a predicted prob-

ability value less than the average, and so would be mis-classified as Internet worm packets – that is, there are false-positives.

Figure 5.2 shows the same packet sequence as in Figure 5.1, but with some of the packets having been replaced with Sapphire datagrams (marked with red circles). As desired, the predicted probabilities of the Sapphire worm packets in the observation sequence are very low.

### 5.2.2 Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of a Limited Time Period of the Observation Sequence

In order to try to improve on the results of the previous section, and remove the false-positive packet classifications, the experiments described above were repeated, except that the average predicted probability of the packets in the observation sequence was calculated over different (limited) ranges of previous observations.

Figure 5.3 shows the results of the same experiment shown in Figure 5.1, with the exception that the average predicted probability value of the observation sequence is calculated over the previous  $20 \times \Delta = 100$  observations only, at each observation. The same model trained with the same observation sequence, but with some packets replaced with Sapphire datagrams, can be seen in Figure 5.4.

Similarly, Figure 5.5 and Figure 5.6 show the same experiments, but with the average calculated over the previous  $50 \times \Delta = 250$  observations, while Figure 5.7 and Figure 5.8 show the same experiments, but with the average calculated over the previous  $100 \times \Delta = 500$  observations.

These results show that, as expected, as the number of observations over which the average predicted probability of the packets in the observation sequence is calculated is increased, the effect of peaks and troughs in the predicted probability of the observation sequence is reduced. Thus, if such a technique were used to differentiate normal packets from Internet worm packets, a compromise between the positions shown in Figure 5.2 and Figure 5.4 must be reached.

To explain the need for this compromise, note that in Figure 5.2, the average is calculated over the entire observation sequence, which results in false-positive results (normal packets being classified as Internet worm datagrams). In Figure 5.4, the average is calculated over only the past 20 observations, and results in Internet worm packets having an immediate effect on the average value. This means that high levels of Internet worm packets in a small period of time could result in the average value dropping to, or below, the predicted probability of some Internet worm packets, resulting in these packets being classified as normal traffic. This situation would result in *false-negatives* (Internet worm packets being missed).

To demonstrate that this situation could occur, the experiments were

repeated, except that the probability of a normal datagram being replaced with a Sapphire Internet worm datagram was increased from 1% to 50%, and the average predicted probability of the packets in the observation sequence was calculated over the previous  $10 \times \Delta = 50$  observations only, at each observation. The results can be seen in Figure 5.9 and Figure 5.10, which clearly show the average value dropping to the predicted probability of Internet worm packets, resulting in false-negatives.

Improved results – obtained by increasing the number of past observations over which the average predicted probability value is calculated to  $50 \times \Delta = 250$  observations – can be seen in Figure 5.11 and Figure 5.12. Nevertheless, there are still some false-negatives, and a further increase would be required for this scenario. This illustrates the compromise that must be reached between having an average calculated over too small a number of past observations, which can result in false-negative results, and having the average calculated over too many, which can result in false-positive results.

### 5.2.3 Failure to Predict the Observation Sequence

Not all normal packet sequences can be predicted as well as demonstrated in the above results. For example, Figure 5.13 shows an observation sequence that results in the predicted probabilities of the packets in the observation sequence approaching zero as training progresses. Figure 5.14 shows that replacing some of the packets in the observation sequence with Sapphire datagrams has little effect on the predicted probabilities at these points. Figure 5.15 and Figure 5.16 are another, even worse example.

The reason for the behaviour observed in these two examples is that almost no observation in the sequence is ever repeated – every single packet has a different source/destination port pair. These example predicted probability results are typical of the behaviour observed when packet observations are not commonly repeated in the observation sequence (results not shown).

It is clear from these results that in order for an anomaly-based network intrusion detection system based on an HMM to be able to successfully predict the observation sequence in the first place – let alone predict Internet worm packets within the sequence – then there must be some form of underlying pattern to the observation sequence. That is, a sequence of non-repeating packet observations will result in the system being unable to predict the observation sequence.

## 5.3 Sapphire Worm Packet Detection Experiments Without Set “Attack” Periods and Results

In order to determine if the models tested above are able to correctly identify Internet worm packets without the benefit of being trained with normal

datagrams before the Internet worm “attacks” commence, the models were again trained with observation sequences 6000 packets in length. However, the sequences with some packets replaced with Sapphire worm datagrams had these packets spread randomly throughout the entire sequence, with a probability of 1%.

### 5.3.1 Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of the Observation Sequence

Figure 5.17 and Figure 5.18 show the same example HMM, trained with the same UDP datagram observation sequences as shown in Figure 5.1 and Figure 5.2. As with the case where there were set Internet worm “attack” periods, the model with a  $\Delta$  value of 5 appeared to give the best results (results not shown).

As with the case where there were set Internet worm “attack” periods, the use of the average predicted probability of all the packets in the observation sequence seen so far to differentiate normal datagrams from Internet worm packets results in some false-positive classifications.

### 5.3.2 Differentiating Normal and Internet Worm Datagrams Based on the Average Probability of a Limited Time Period of the Observation Sequence

As in the case where there were set Internet worm “attack” periods, an attempt was made to improve the results of the previous section by using an average predicted probability of the packets in the observation sequence calculated over different (limited) ranges of previous observations.

The results of calculating the average predicted probability of the observation sequence over the last  $20 \times \Delta = 100$  observations can be seen in Figure 5.19 and Figure 5.20; the results of calculating the average predicted probability of the observation sequence over the last  $50 \times \Delta = 250$  observations can be seen in Figure 5.21 and Figure 5.22; and the results of calculating the average predicted probability of the observation sequence over the last  $100 \times \Delta = 500$  observations can be seen in Figure 5.23 and Figure 5.24.

All of these results show that, just as in the case where there were set Internet worm “attack” periods, increasing the number of past observations over which the average predicted probability of the packets in the observation sequence is calculated reduces the immediate effect that probability peaks and troughs can have on the average. Clearly, then, the same compromise between false-positive results and false-negative results must be made.

### 5.3.3 Failure to Predict the Observation Sequence

Of particular interest are the results shown in Figure 5.25 and Figure 5.26. The HMM and observation sequence is the same as that shown in Figure 5.13 and Figure 5.14, but the Sapphire worm packets in Figure 5.26 have been placed into the observation sequence as described above in Section 5.3.

These results demonstrate the same inability to predict the observation sequence as seen before. However, in this example, the inability to predict the observation sequence combined with Internet worm packets at the start of the sequence results in the false-negative classification of Internet worm packets and normal datagrams.

## 5.4 Summary of the Results

The results described above suggest that an HMM is able to correctly classify normal and Internet worm traffic without false-negative or false-positive results, provided that the model is able to predict the normal sequence of packets, and that an appropriate value to differentiate the two classes of datagrams can be determined.

However, in the case where the model is not able to predict the normal sequence of packets, the results will at best have a large number of false-positive classifications, and at worse have some false-negative classifications.

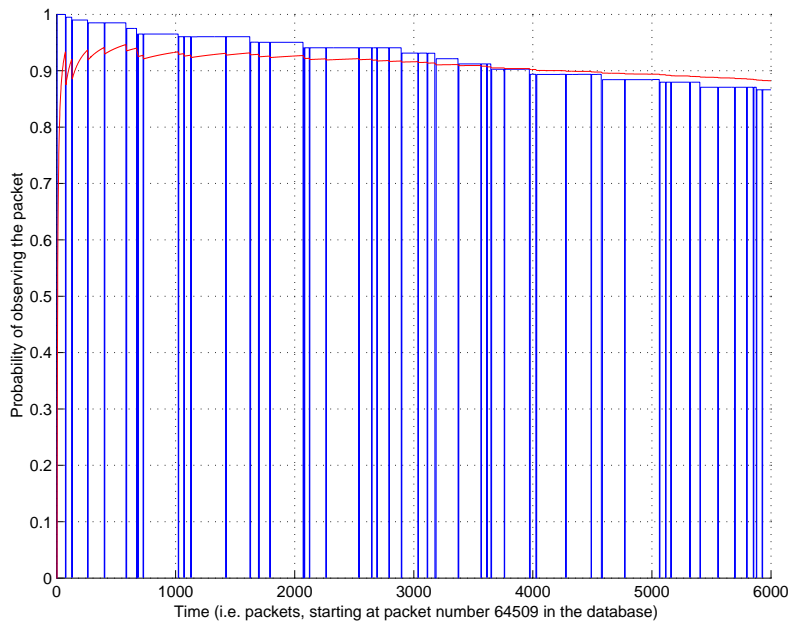


Figure 5.1: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red.

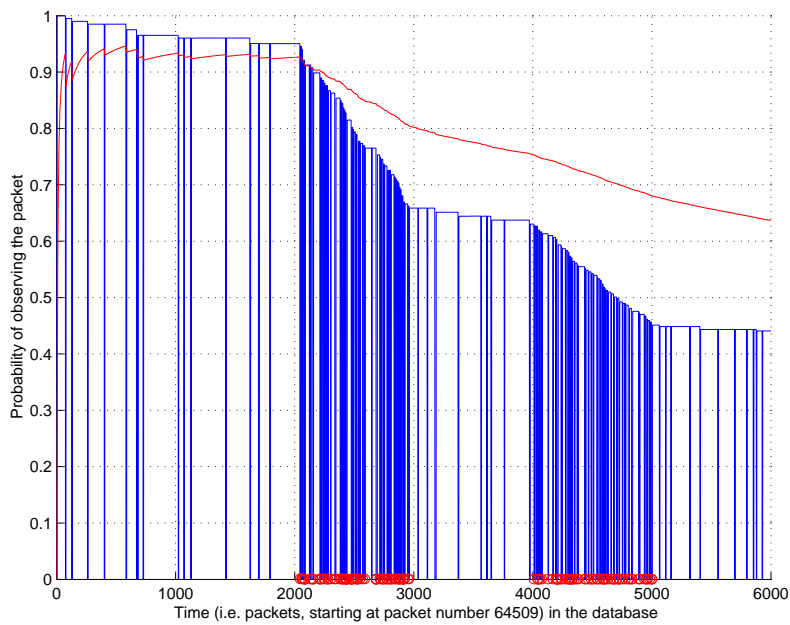


Figure 5.2: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

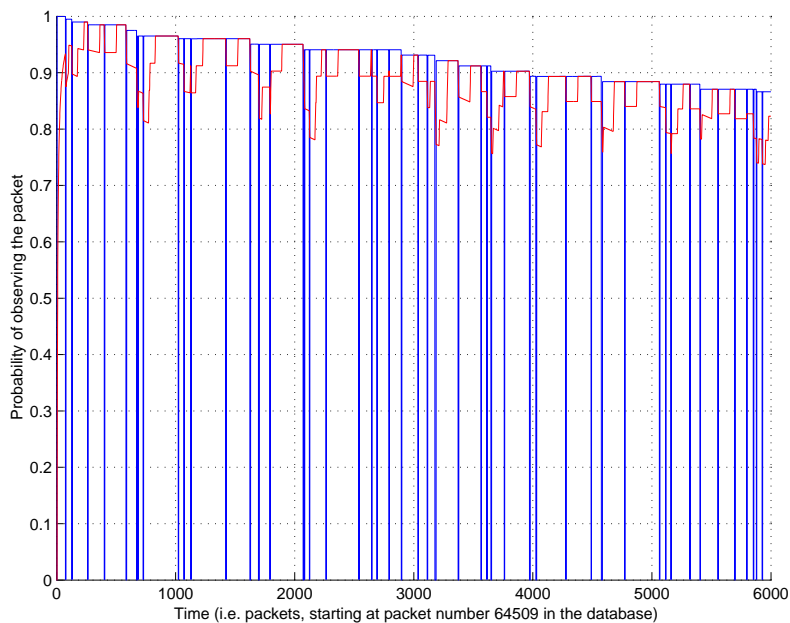


Figure 5.3: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $20 \times \Delta$  observations in red.

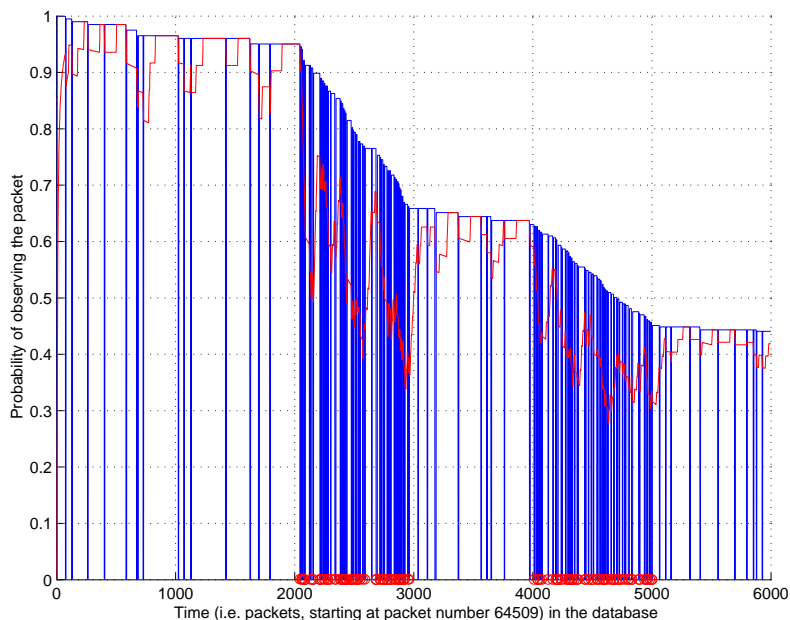


Figure 5.4: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $20 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

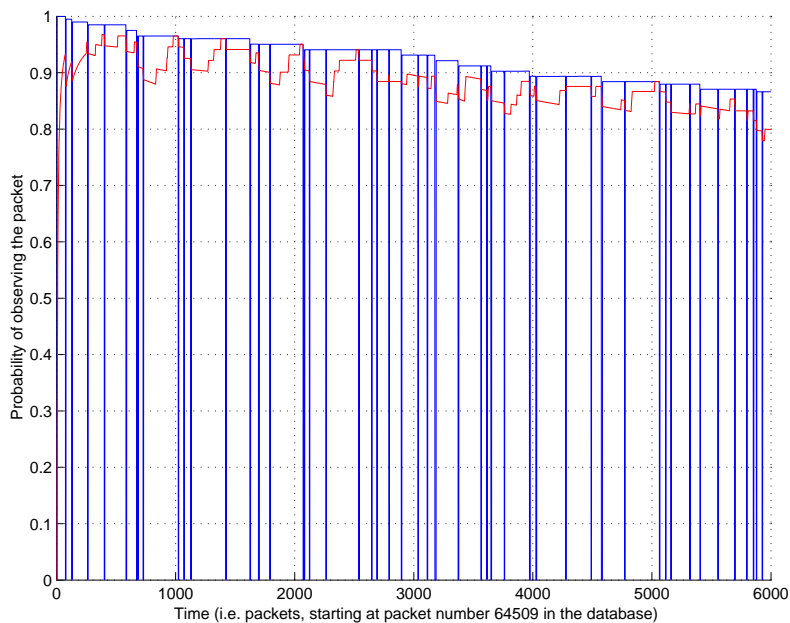


Figure 5.5: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red.

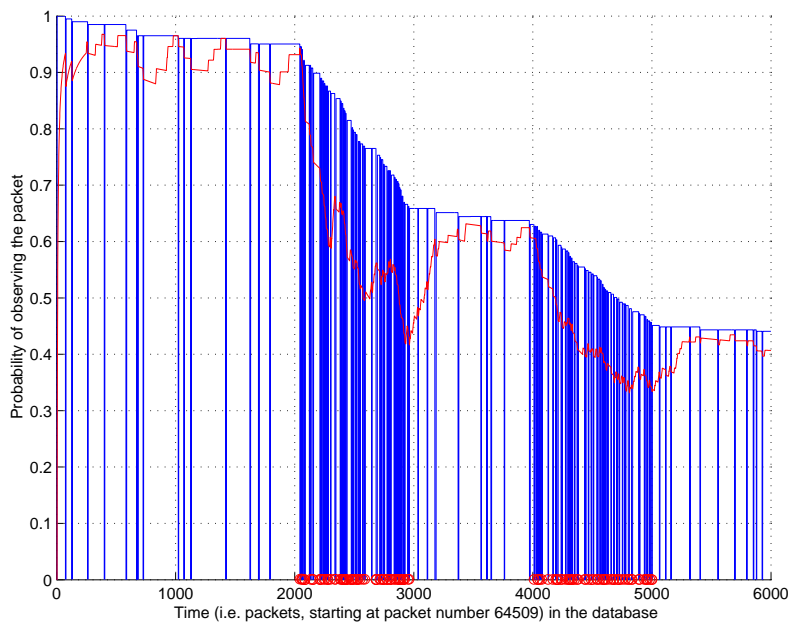


Figure 5.6: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

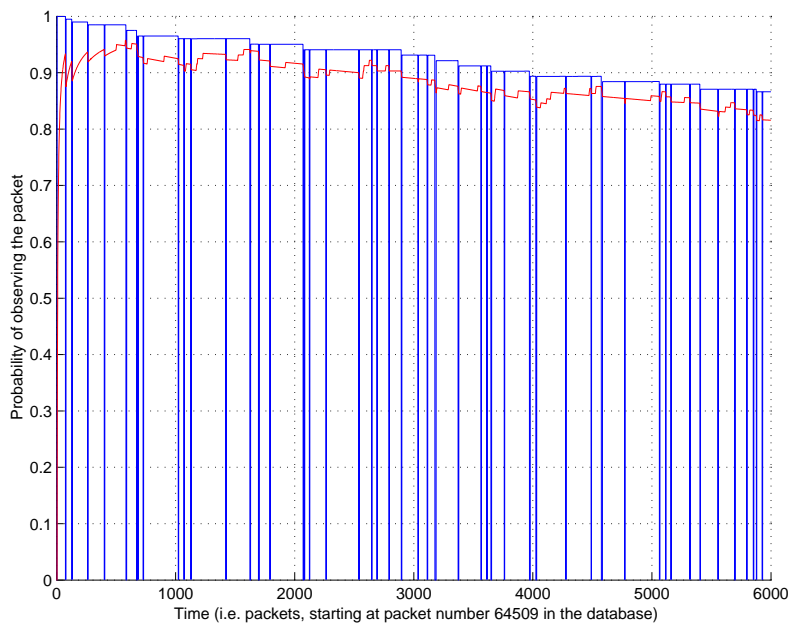


Figure 5.7: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $100 \times \Delta$  observations in red.

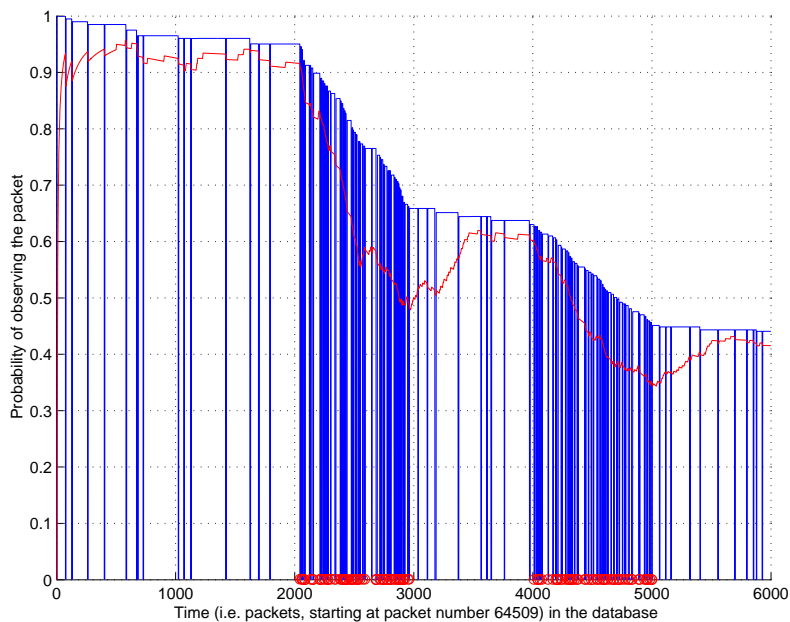


Figure 5.8: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $100 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

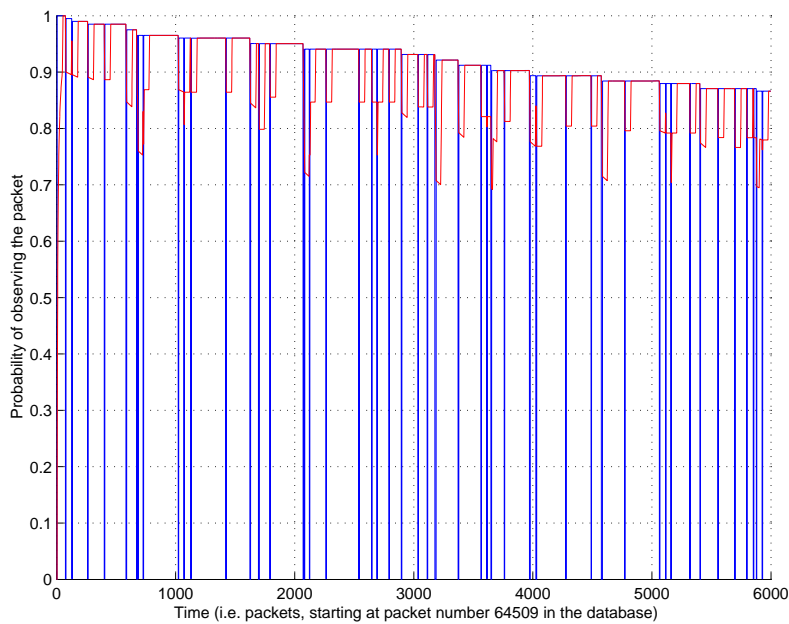


Figure 5.9: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $10 \times \Delta$  observations in red.

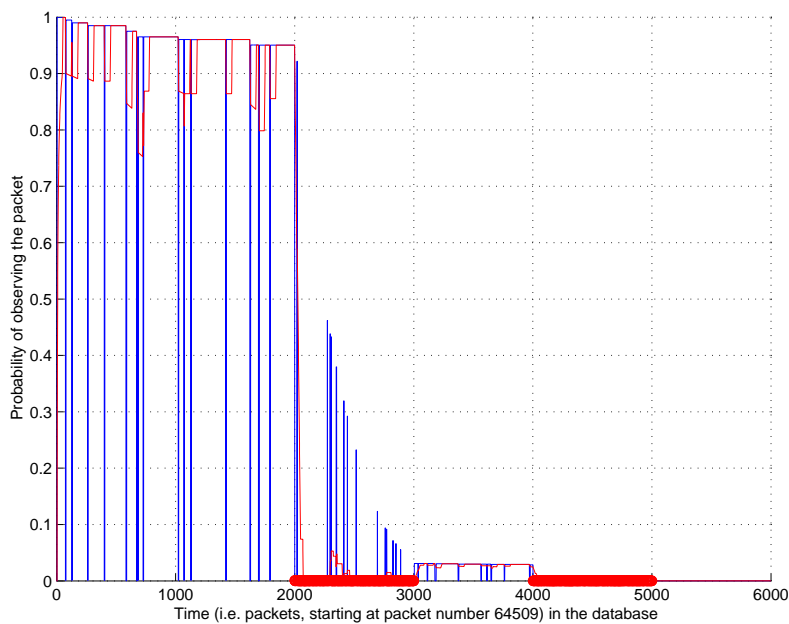


Figure 5.10: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $10 \times \Delta$  observations in red. Packets randomly replaced with generated Saphire datagrams (with an increased probability of 50%) are marked with circles.

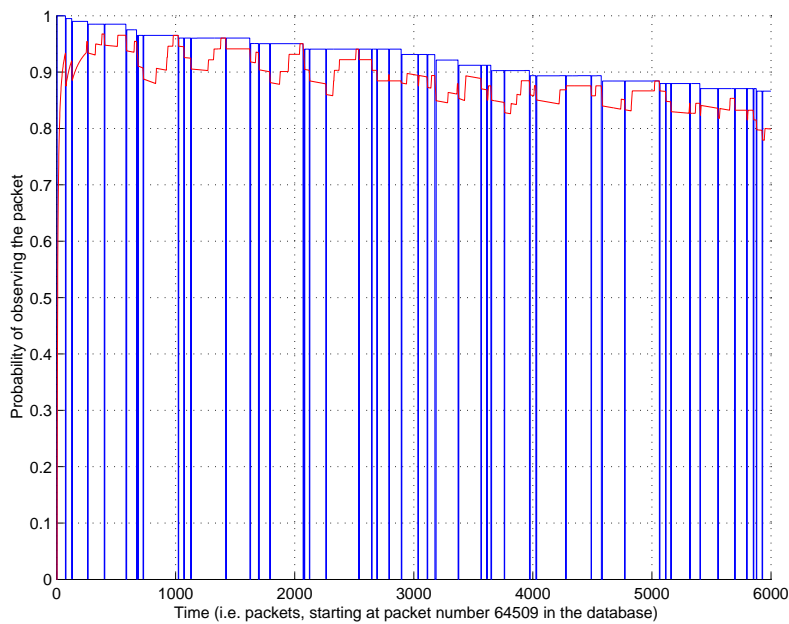


Figure 5.11: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red.

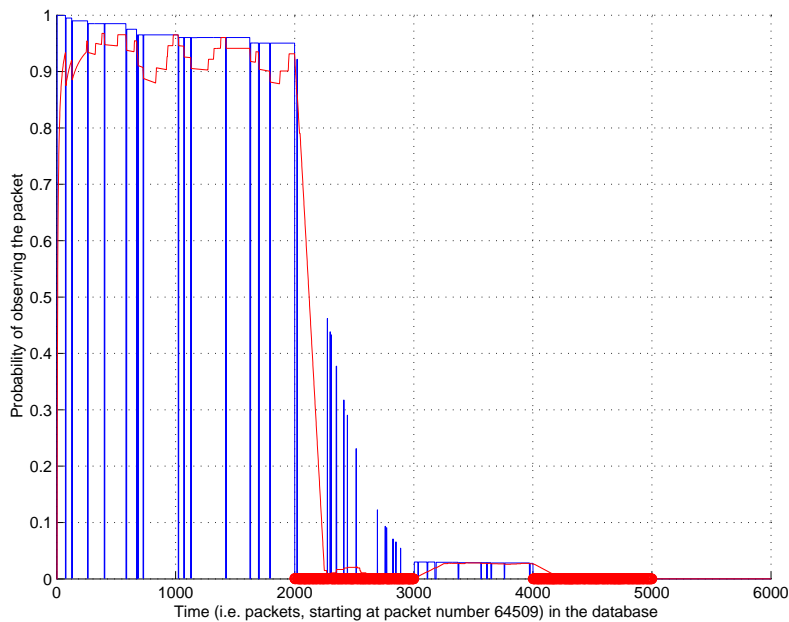


Figure 5.12: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams (with an increased probability of 50%) are marked with circles.

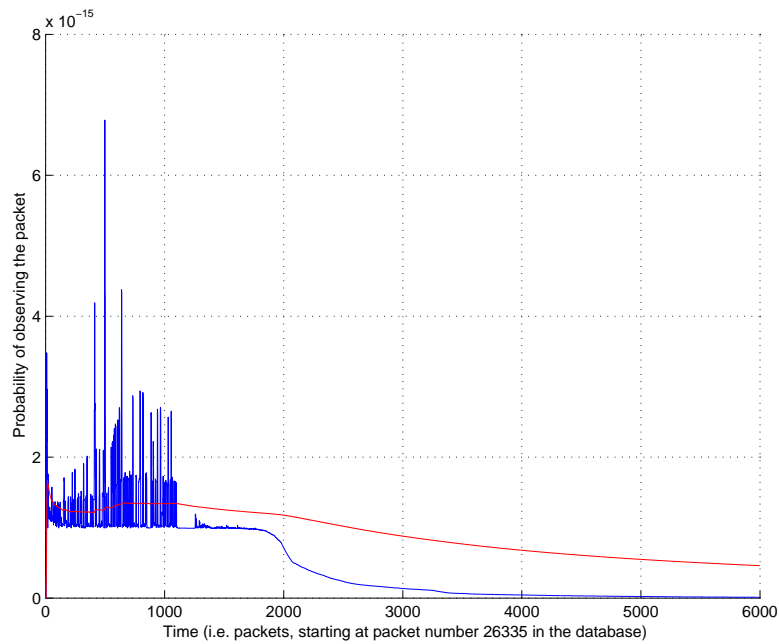


Figure 5.13: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red.

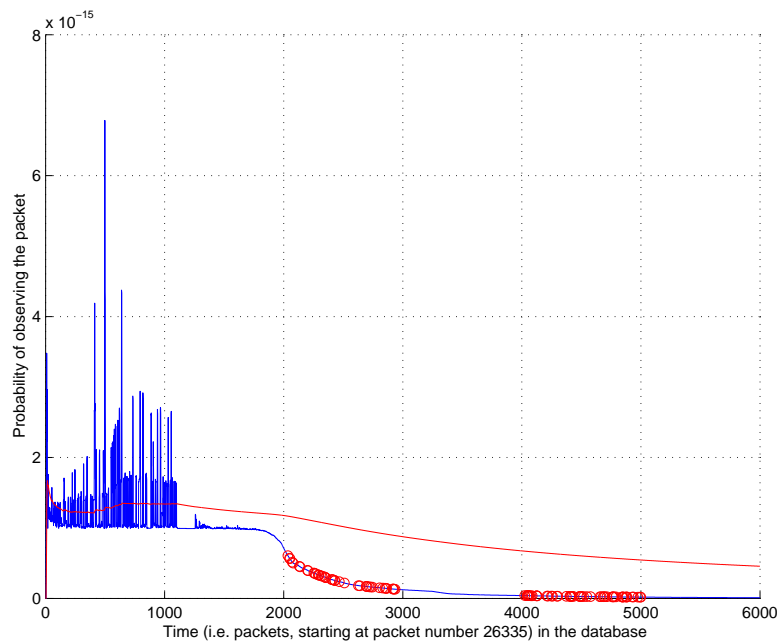


Figure 5.14: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

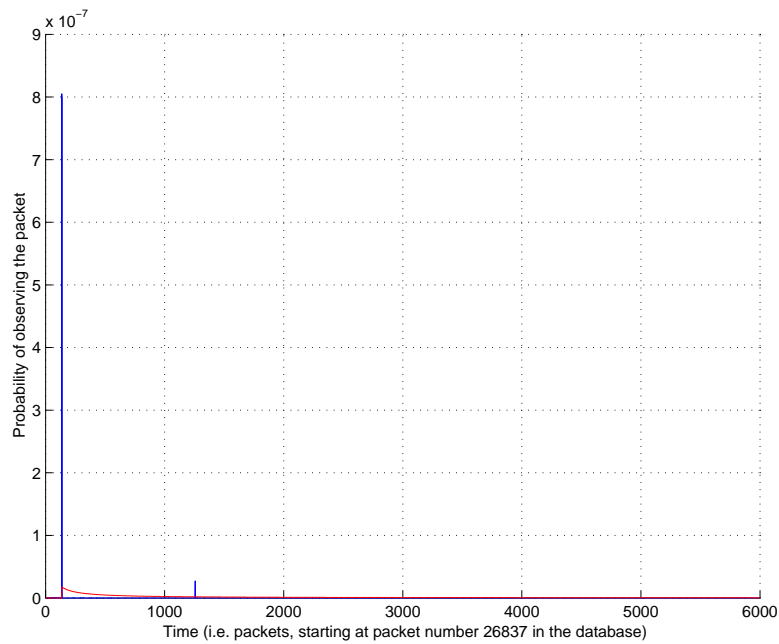


Figure 5.15: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red.

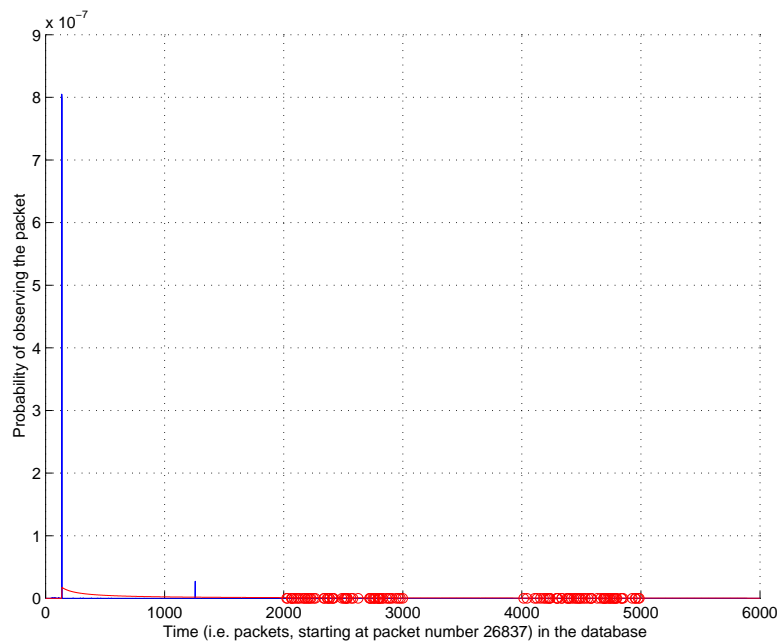


Figure 5.16: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

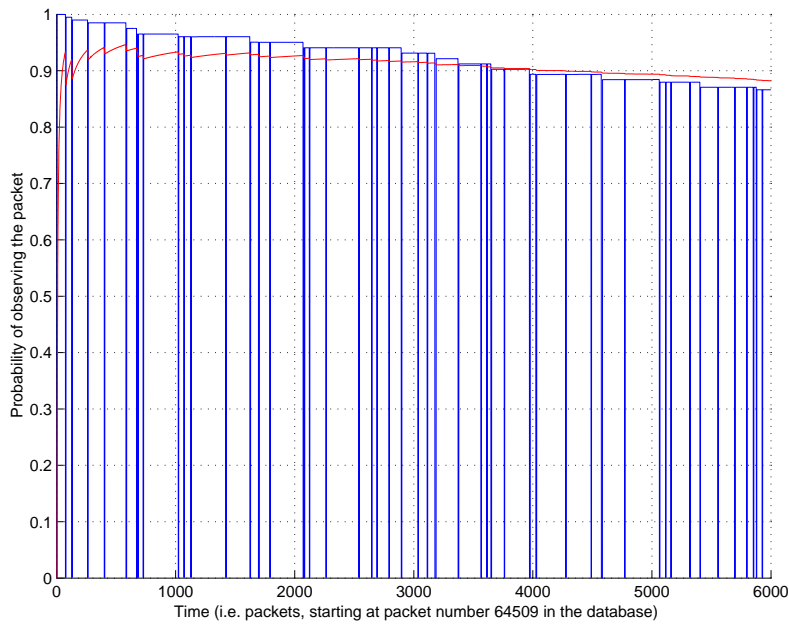


Figure 5.17: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red.

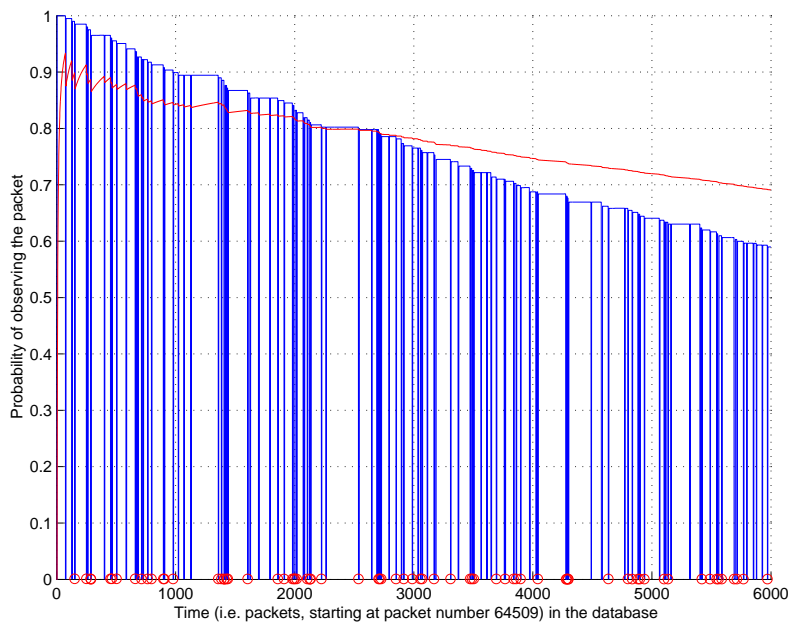


Figure 5.18: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

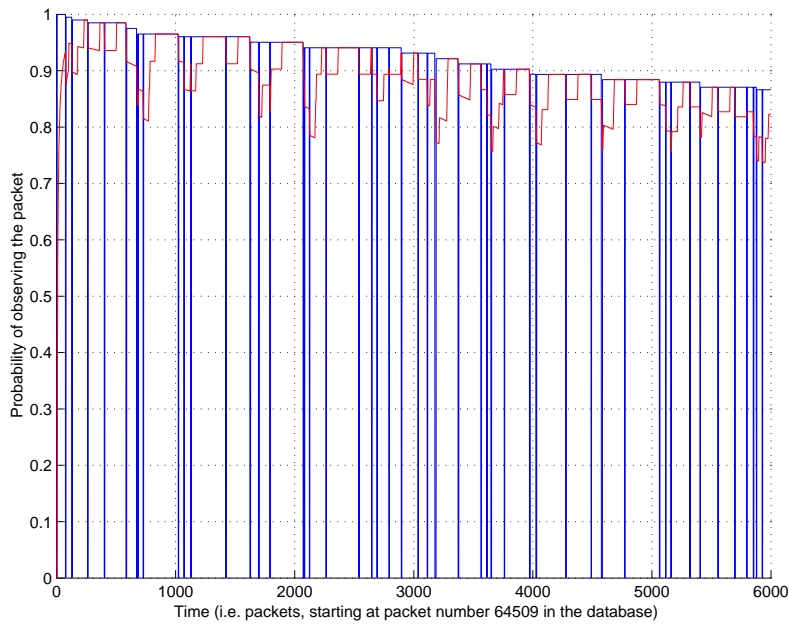


Figure 5.19: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $20 \times \Delta$  observations in red.

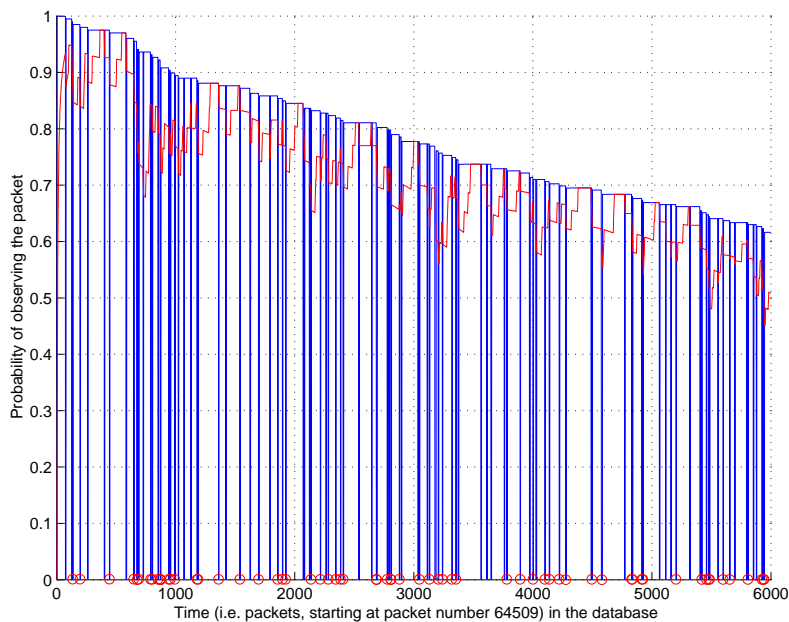


Figure 5.20: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $20 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

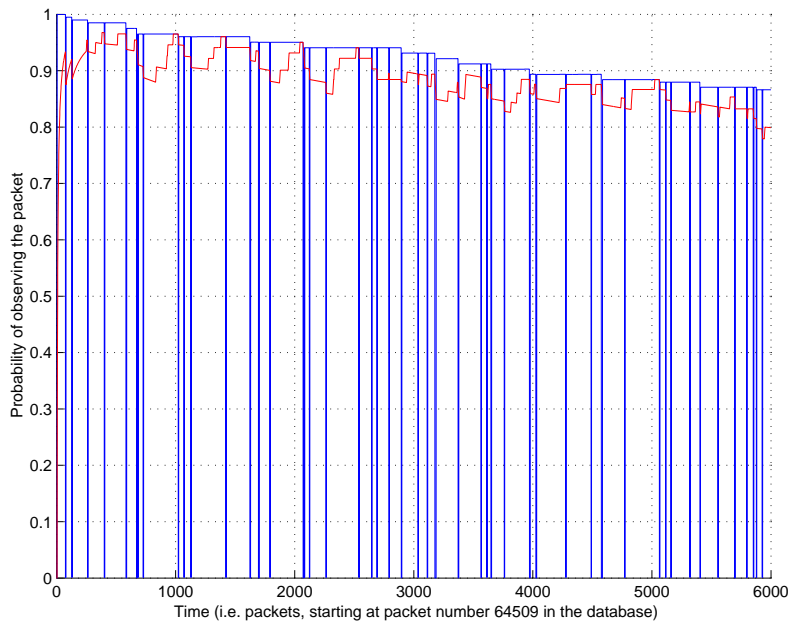


Figure 5.21: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red.

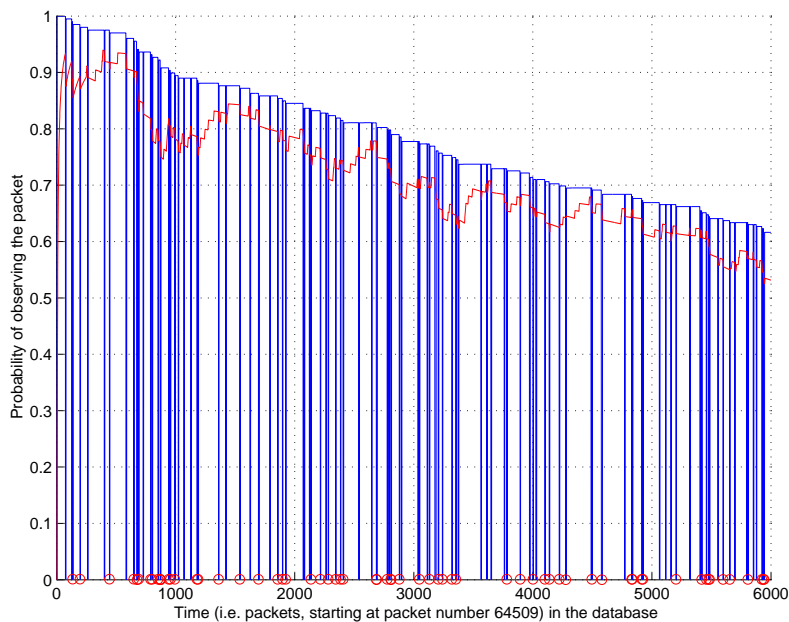


Figure 5.22: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $50 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

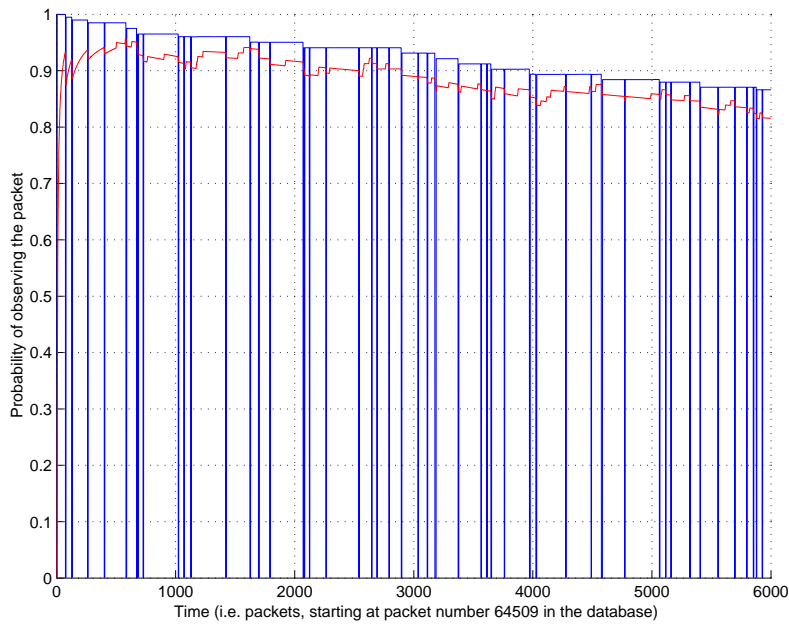


Figure 5.23: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $100 \times \Delta$  observations in red.

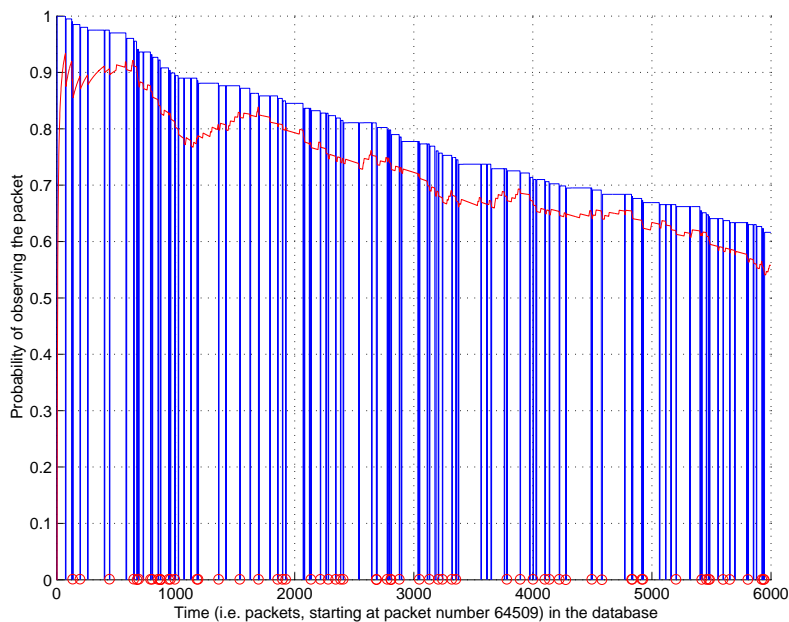


Figure 5.24: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability over the last  $100 \times \Delta$  observations in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

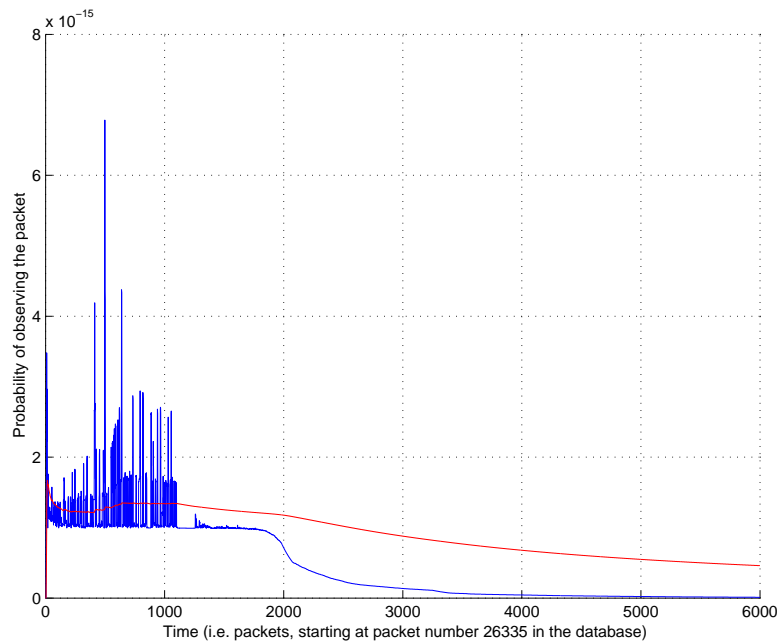


Figure 5.25: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red.

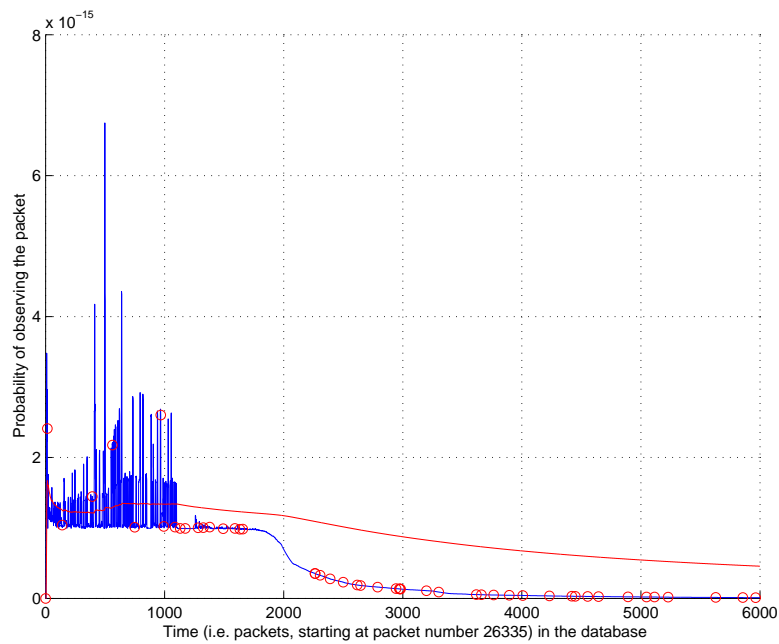


Figure 5.26: Example 2 state HMM with  $\Delta = 5$ ,  $\rho = 0.99$ , trained with UDP datagram source and destination ports. Average sequence probability in red. Packets randomly replaced with generated Sapphire datagrams are marked with circles.

## Chapter 6

# Conclusions and Suggestions for Future Work

### 6.1 Conclusions

This thesis demonstrates that Hidden Markov Models (HMMs) – specifically, on-line HMMs – are able to be successfully used as the basis of an anomaly-based network intrusion detection system for Internet worms.

Such a system is on-line, recognises the fact that network packets are interrelated, requires no initial training, and is able to protect an individual machine – and thus, could be used to protect an entire network of machines by having the system run once for each internal machine on a corporate firewall. This means that such a system meets four of the six requirements outlined in Section 1.3.

The system also meets a fifth requirement, that the system can “accurately and consistently detect anomalous packet traffic associated with Internet worms,” but only under two caveats. Firstly, the sequence of normal Internet packets (in which the Internet worm datagrams exist) must have an underlying pattern that an HMM can predict – that is, the observation sequence can not consist of observations that do not repeat. (There is some evidence that non-repetitive packet sequences are common – see the next section for more details.) Secondly, an appropriate probability value must be selected to differentiate normal Internet datagrams from Internet worm packets.

### 6.2 Suggestions for Future Work

#### 6.2.1 Improving the Internet Worm Detection Capabilities

Obviously, the two caveats described above seriously limit the potential use of an anomaly-based network intrusion detection system for Internet worms

based on an on-line HMM.

As some real-life packet sequences can not be predicted by the HMMs used in Chapter 5, some advantage may be obtained by modifying the system to eliminate the first caveat. The experiments performed in this thesis found that only two of ten different observation sequences tested were able to be predicted by the HMMs used – however, a detailed analysis of the nature of packet sequences should be carried out to determine if non-repetitive packet sequences are sufficiently common in real life that the inability of an HMM to predict these sequences will be a problem.

One possible way to eliminate the first caveat would be to look at how well an HMM is able to predict a sequence of normal packets sent and received from a *single* UDP port. Obviously, the datagram fields used as packet observations would need to be something other than the port number being monitored by the model.

Some form of training of the model before it is used may also prove to be beneficial in cases where the system is less able to predict the sequence of normal packets, as this may help to eliminate the false-negative results seen in Figure 5.26.

There would also clearly be an advantage in eliminating the second caveat, by improving the way the packet differentiation value is selected. One suggestion would be to modify the number of observations over which the average predicted probability of the observation sequence is calculated based on the rate of change of the average – increasing the number of observations to slow rapid change, and decreasing the number of observations when little change is occurring.

Finally, the second caveat could possibly be ignored if the system was to be used as a first line of defense only. If the number of false-positives can be kept relatively small, and there are no false-negatives, then the system may well be useful as a filter for reducing the number of packets that a more effective (and presumably more computationally expensive) Internet worm detection system has to deal with.

## 6.2.2 Testing with Other Internet Worms

This thesis has only looked at one Internet worm out of hundreds. A great deal of testing with different worm types, using both TCP/IP as well as UDP/IP, needs to be performed in order to determine how effective an anomaly-based network intrusion detection system for Internet worms based on HMMs would actually be.

## 6.2.3 Implementation Speed

Of the requirements for an anomaly-based network intrusion detection system for Internet worms set out in Section 1.3, only the ability of the system

“to process information quickly, so that the speed at which packets are delivered is not adversely affected” has not been explicitly investigated in this thesis.

However, it has been shown that as the number of observations seen by an on-line HMM grows, there is a polynomial increase in the training time required (see Section 4.4 and Appendix A). Thus, it would appear that further work will be required to improve the implementation speed before the system described meets all of the requirements outlined. There are several possible approaches that could be considered.

Firstly, it should be possible to alter the implementation described above so that each observation is tagged with the most recent observation time. Once the number of observations seen by the model reaches a point where the training speed is too slow to ensure that packet delivery is not adversely affected, the least recently seen observation could be dropped from the  $B$  matrix. Of course, the effect of such a scheme on the ability of the system to successfully identify Internet worm packets would need to be carefully evaluated.

Secondly, a sawtooth lag on-line HMM exists which is computationally better than the fixed lag model [14], and may prove to be useful in improving the speed of the system. (Note, however, that the literature does not currently have a complete discrete state version of the update algorithms for sawtooth lag models.)

Finally, there may be some advantage in extending the theory of HMMs to create a factorial, on-line HMM. The motivation behind factorial HMMs is to reduce the state space required [12]. However, the exact training algorithm for factorial HMMs is intractable [12], so at least three different training algorithm approximations have been developed [12]. Some of these approximate training algorithms may have speed advantages over the implementation described in this thesis.

## Bibliography

- [1] S. Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.
- [2] J. K. Baker. The DRAGON system - an overview. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23:24–29, 1975.
- [3] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffic Anomalies.
- [4] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markovchains. *Annals of Mathematical Statistics*, 41:164–171, 1970.
- [5] L. E. Baum and G. R. Sell. Growth Transformations for Functions on Manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.
- [6] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, and M. Embrechts. Network-Based Intrusion Detection using Neural Networks. <http://www.cs.rpi.edu/acr/~szymansk/papers/annie02.pdf>, November 2002.
- [7] J. Cannady. Artificial Neural Networks for Misuse Detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98)*, pages 443–456, Arlington, VA, USA, October 1998.
- [8] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Rowe, S. Staniford-Chen, R. Yip, and D. Zerkle. The Design of Grids: A Graph-Based Intrusion Detection System. <http://seclab.cs.ucdavis.edu/arpa/grids/grids.pdf>, 1999. Accessed 14 August 2003.
- [9] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

- [10] J. Frank. Artificial Intelligence and Intrusion Detection: Current and Future Directions. In *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD, USA, 1994.
- [11] Z. Ghahramani. An Introduction to Hidden Markov Models and Bayesian Networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2001.
- [12] Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29:245–275, 1997.
- [13] T. Kanungo. UMDHMM version 1.02. <http://www.cfar.umd.edu/~kanungo/>, 1999.
- [14] V. Krishnamurthy and J. B. Moore. On-Line Estimation of Hidden Markov Model Parameters Based on the Kullback-Leibler Information Measure. *IEEE Transactions on Signal Processing*, 41(8):2557–2573, 1993.
- [15] T. Lane. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, August 2000.
- [16] T. Lane and C. E. Brodley. An Empirical Study of Two Approaches to Sequence Learning for Anomaly Detection. *Machine Learning*, 51:73–107, 2003.
- [17] D. Lehmann. Intrusion Detection FAQ: What is ID? [http://www.sans.org/resources/idfaq/what\\_is\\_id.php](http://www.sans.org/resources/idfaq/what_is_id.php), 2002. Accessed 20 October 2003.
- [18] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. *Bell Systems Technical Journal*, 62(4):1035–1074, April 1983.
- [19] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The Spread of the Sapphire/Slammer Worm. <http://www.cs.berkeley.edu/~nweaver/sapphire/>, 2003. Accessed 19 March 2003.
- [20] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. <http://www.cs.ucsd.edu/users/savage/papers/Infocom03.pdf>, 2003. Accessed 15 March 2003.
- [21] J. Postel. RFC 761: Transmission Control Protocol. <http://www.ietf.org/rfc/rfc0761.txt>, January 1980.

- [22] J. Postel. RFC 768: User Datagram Protocol. <http://www.ietf.org/rfc/rfc0768.txt>, August 1980.
- [23] J. Postel. RFC 791: Internet Protocol. <http://www.ietf.org/rfc/rfc0791.txt>, September 1981.
- [24] K. Poulsen. Slammer worm crashed Ohio nuke plant network. <http://www.securityfocus.com/news/6767>, August 2003. Accessed 22 August 2003.
- [25] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications on Speech Recognition. In *Proceedings of the IEEE*, 1989.
- [26] J. Ryan, M. Lin, and R. Miikkulainen. Intrusion Detection with Neural Networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.
- [27] J. F. Shoch and J. A. Hupp. The “Worm” Programs – Early Experience with a Distributed Computation. *Communications of the ACM*, 25(3):172–180, 1982.
- [28] P. L. Slingsby. Recursive Parameter Estimation for Arbitrary Hidden Markov Models. In *Proceedings of the Fourth IFAC International Symposium on Adaptive Control and Signal Processing, ACASP/92*, pages 105–108, Grenoble, France, July 1992.
- [29] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic Independence Networks for Hidden Markov Probability Models. Technical Report AIM-1565, 1996.
- [30] E. H. Spafford. The Internet worm: Crisis and Aftermath. *Communications of the ACM*, 32(6):678–687, 1989.
- [31] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002.
- [32] T. Toth and C. Kruegel. Connection-history based anomaly detection. In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*, 2002.
- [33] B. Waite. Malicious Code Attacks Had \$13.2 Billion Economic Impact in 2001. <http://www.computereconomics.com/article.cfm?id=133>, January 2002. Accessed 14 March 2003.

## Appendix A

# Informal Order Analysis of the On-line Hidden Markov Model Algorithms

### On-line Forward-Backward Algorithm

It is clear from a comparison of the forward pass of the regular Hidden Markov Model (HMM) Forward-Backward algorithm (Section 2.2.1) and the on-line version of the same (Section 2.4.1) that, knowing the forward pass of the regular Forward-Backward algorithm is of order  $N^2T$  (where  $N$  is the number of states and  $T$  is the observation sequence length), the on-line Forward-Backward algorithm's forward pass will be of order  $N^2$ , as the  $\alpha$  values are stored as “running” values in the series of models, and only need to be updated for a single observation (i.e.  $T = 1$ ).

It is also clear from a comparison of the two different forms of backward pass that, knowing the backward pass of the regular Forward-Backward algorithm is of order  $N^2T$ , the on-line Forward-Backward algorithm's backward pass will be of order  $N^2\Delta$ , as at time  $t = k$ , the observation sequence used in the backward pass will be  $O_{k:k+\Delta}$ , which has length  $\Delta$ .

### On-line Training Algorithms

While the analysis of the training algorithms is generally ignored in the literature due to the fact that most HMMs are trained off-line – and thus the speed of the algorithms is not particularly important – in the case of an on-line HMM, speed of training is an issue due to the fact that the model is trained at the same time it is used. Thus, the on-line training algorithms are considered here.

Firstly, note that the calculation of  $\gamma_{t|\Lambda_{t-1}}(i)$  for each model will obviously be of order  $N$ , while the calculation of  $\xi_{t|\Lambda_{t-1}}(i, j)$  will be of order  $N^2$  (from their definitions, see Section 2.4.2).

Similarly, the calculation of  $\mu_t(i, j)$  will be of order  $N^2$  (as “running” values can be used to store past  $\mu$  values, just as can be done for the  $\alpha$  values), and the calculation of  $g_t(i, j)$  will be of order  $N^2$ .

This means that the update algorithm for the  $A$  matrix values:

$$\bar{a}_{ij}(t+1) = \bar{a}_{ij}(t) + \frac{1}{\mu_t(i, j)} \times \left( g_t(i, j) - \frac{\sum_{j'=1}^N \frac{g_t(i, j')}{\mu_t(i, j')}}{\sum_{j'=1}^N \frac{1}{\mu_t(i, j')}} \right)$$

will also be of order  $N^2$ , provided the summation of the  $\mu_t(i, j)$  and  $g_t(i, j)$  values are calculated during the process of calculating the individual  $\mu$  and  $g$  values.

Similarly, the update algorithm for the  $B$  matrix values for observations  $n \forall n \neq O_t$ :

$$\begin{aligned} \bar{b}_i(t, n) = \bar{b}_i(t-1, n) - b_i(t-1, O_t) \times & \left( \frac{\gamma_{t|\Lambda_{t-1}}(i)}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)} \right) \\ & \times \left( \frac{\frac{b_i(t, n)^2}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)}}{\sum_{p=1}^M \left( \frac{b_i(t, p)^2}{\sum_{k=1}^t \gamma_{k|\Lambda_{k-1}}(i)} \right)} \right) \end{aligned}$$

will be of order  $M^2N$  (where  $M$  is the number of observations seen by the model), while the update algorithm for the  $B$  matrix for the observation  $n = O_t$ :

$$\bar{b}_i(t, n) = 1 - \sum_{p \neq n}^M \bar{b}_i(t, p)$$

will be of order  $NM$ .

This informal order analysis supports the results of Section 4.4, as it shows a polynomial increase in training time as the number of states is increased, as well as showing a polynomial increase in training time as the number of observations is increased.