5-2010

Traffic Analysis of Anonymity Systems

Ryan Craven
Clemson University, crm810@gmail.com

Traffic Analysis of Anonymity Systems

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Ryan Michael Craven
May 2010

Accepted by:
Dr. Richard R. Brooks, Committee Chair
Dr. Timothy Burg
Dr. Christopher Griffin

Abstract

This research applies statistical methods in pattern recognition to test the privacy capabilities of a very popular anonymity tool used on the Internet known as Tor.

Using a recently developed algorithm known as Causal State Splitting and Reconstruction (CSSR), we can create hidden Markov models of network processes proxied through Tor. In contrast to other techniques, our CSSR extensions create a minimum entropy model without any prior knowledge of the underlying state structure. The inter-packet time delays of the network process, preserved by Tor, can be symbolized into ranges and used to construct the models.

After the construction of training models, detection is performed using Confidence Intervals. New test data can be fed through a model to determine the intervals and estimate how well the data matches the model. If a match is found, the state sequence, or path, can be used to uniquely describe the data with respect to the model. It is by comparing these paths that Tor users can be identified.

Packet data from any two computers using the Tor network can be matched to a model and their state sequences can be compared to give a statistical likelihood that the two systems are actually communicating together over Tor. We perform experiments on a private Tor network to validate this. Results showed that communicating systems could be identified with a 95% accuracy in our test scenario.

This attack differs from previous maximum likelihood-based approaches in that it can be performed between just two computers using Tor. The adversary does not need to be a global observer. The attack can also be performed in real-time provided that a matching model had already been constructed.

Dedication

This thesis is dedicated to my family, especially my wife Heather. I am forever grateful for the unwavering love and support they have given me throughout my life.

Acknowledgments

Most importantly, I would like to thank my advisor, Dr. Richard R. Brooks. The creation of this document and the underlying research would not have been possible without your capable guidance and direction. Our frequent discussions this past year and a half have not just helped me complete this work, but have imparted a deeper understanding of the challenging issues that are faced in security and privacy.

I would like to thank Dr. Timothy Burg and Dr. Christopher Griffin for serving on my committee. I would also like to thank Dr. Burg for helping me on my path to graduate school. It was during my time working on the EE senior project and our creative inquiry undergraduate research that I made the decision to apply for entry into the Master's program.

During my work on this thesis, I have also had the pleasure of working with some very intelligent and helpful students in our research group. My interactions with them have made me a better researcher and their contributions have been invaluable. In particular, a large degree of appreciation is owed to Jason Schwier, Hari Bhanu, and Chen Lu.

I would also like to acknowledge my employer, the Space and Naval Warfare Systems Center in Charleston, SC. Their flexibility and support for my continued education have set them apart from other organizations.

My time in graduate school would not have been possible without the monetary assistance of the Holcombe Department of Electrical and Computer Engineering. I received a teaching assistantship and the Mr. & Mrs. Alan Griffith Stanford fellowship for all four semesters of my Master's program. I was humbled by and will forever appreciate their support.

Finally, I would like to acknowledge that this material is based upon work supported by, or in part by, the Air Force Office of Scientific Research contract/grant number FA9550-09-1-0173.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Signals intelligence, commonly referred to as SIGINT, is a widely encompassing intelligence gathering field involving the analysis of transmitted signals for access to private information [1]. Traditionally only a military discipline, the importance of the field is well understood within the defense community [2]. More recently however, with the proliferation of large communication networks used by the public (such as the Internet, cellular networks, or wireless for other mobile devices), SIGINT and SIGINT countermeasures have become a more general concern.

With examples visible throughout military history [3, 4], the importance of SIGINT has only been fully recognized since World War II. During the war, mathematicians and other cryptography analysts collaborated to break enemy codes, providing valuable information to field commanders [5]. Cryptanalysis has since become a very important component of the SIGINT field.

But what happened when codes could not be decrypted? Out of necessity, analysts quickly found that other information could be extracted from the electronic signals. In an early example, they found that radio operators could be uniquely identified by the way they typed Morse code [4]. And during World War II, even though the Americans were unable to decipher Japanese naval code during the Guadalcanal campaign, they were able to successfully rely on other information derived from their signals. Volume, source, and patterns in the signal traffic provided valuable intelligence to naval commanders [5].

The process of extracting high level information from communications even when the actual message data cannot be read is known as *traffic analysis*. The importance of traffic analysis has grown and expanded along with the SIGINT field. Increasing reliance on more open communication networks has only accelerated the growth of the field's importance. In particular, the Internet provides a variety of opportunities for traffic analysis.

1.1 Traffic Analysis

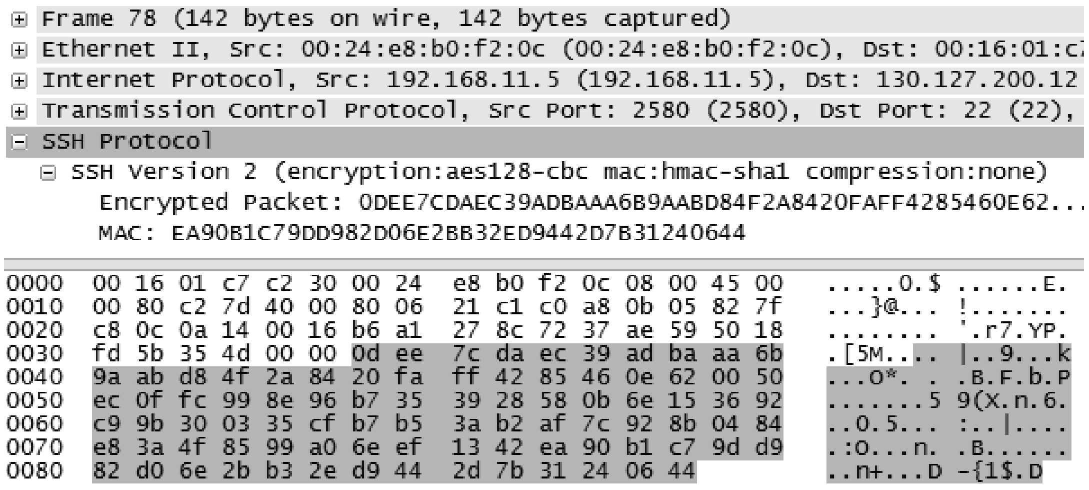
When sensitive information is transmitted through the Internet, it is typically encrypted in order to prevent others from being able to view it. This process obscures the sensitive data residing within the packet, leaving only ciphertext visible to a potential adversary. Assuming that the encryption is strong, meaning that the keys cannot be guessed within a very long amount of time, no one should be able to view the protected data as it traverses the Internet.

But even if the data is successfully hidden, will the sender be able to maintain a full expectation of privacy? Unfortunately, it is not that easy. For successful communication, important routing information must be left unencrypted. The packet headers, timing, and size leave a great deal of high-level information available to possible attackers, even for encrypted protocols like Secure Shell (SSH).

Figure 1.1 shows the contents of a single packet captured during an encrypted SSH session using the popular capturing tool Wireshark. Even though the sensitive data has been encrypted, a variety of information is still readily available to a potential adversary. Information that can be deduced from this example includes:

- Who is talking (the source)
- To whom they are talking (the destination)
- What application-layer protocols are in use (SSH, HTTPS, etc.)
- The amount of data contained within each packet
- Packet timings

Each of these pieces of information, especially when combined with similar information from other packets in a flow or stream, can be very revealing about the data within an encrypted message.



```
⊞ Frame 78 (142 bytes on wire, 142 bytes captured)
⊞ Ethernet II, Src: 00:24:e8:b0:f2:0c (00:24:e8:b0:f2:0c), Dst: 00:16:01:c
⊞ Internet Protocol, Src: 192.168.11.5 (192.168.11.5), Dst: 130.127.200.12
⊞ Transmission Control Protocol, Src Port: 2580 (2580), Dst Port: 22 (22),
⊟ SSH Protocol
  ⊟ SSH Version 2 (encryption:aes128-cbc mac:hmac-sha1 compression:none)
      Encrypted Packet: 0DEE7CDAEC39ADBAAA6B9AABD84F2A8420FAFF4285460E62...
      MAC: EA90B1C79DD982D06E2BB32ED9442D7B31240644

0000  00 16 01 c7 c2 30 00 24  e8 b0 f2 0c 08 00 45 00   .....0.$ ......E.
0010  00 80 c2 7d 40 00 80 06  21 c1 c0 a8 0b 05 82 7f   ...}@... !.......
0020  c8 0c 0a 14 00 16 b6 a1  27 8c 72 37 ae 59 50 18   ........ '.r7.YP.
0030  fd 5b 35 4d 00 00 0d ee  7c da ec 39 ad ba aa 6b   .[5M.... |..9...k
0040  9a ab d8 4f 2a 84 20 fa  ff 42 85 46 0e 62 00 50   ...O*. . .B.F.b.P
0050  ec 0f fc 99 8e 96 b7 35  39 28 58 0b 6e 15 36 92   .......5 9(X.n.6.
0060  c9 9b 30 03 35 cf b7 b5  3a b2 af 7c 92 8b 04 84   ..0.5... :..|....
0070  e8 3a 4f 85 99 a0 6e ef  13 42 ea 90 b1 c7 9d d9   .:O...n. .B......
0080  82 d0 6e 2b b3 2e d9 44  2d 7b 31 24 06 44         ..n+...D -{1$.D
```

Figure 1.1: Packet captured from an encrypted SSH session

1.1.1 Important Traffic Analysis Attacks

What follows are some examples of traffic analysis attacks on encrypted protocols. Researchers used the attacks to infer information that would jeopardize the security of the communications.

Song et al. created profiles of users' typing characteristics and used them to guess which keys were being typed in an interactive SSH session [6]. Since SSH immediately transmits each key press in its own separate packet across the network, they were able to measure the delays between packets and match that to pre-determined delays for certain key pair combinations. Prediction of the key sequences was done using hidden Markov models. The attack was used to drastically enhance the probability of breaking a password that was entered over SSH, while simultaneously reducing the time required to do so.

Hintz found that unique signatures could be created for various HTTPS websites based on the resources they load (e.g. HTML, images, CSS, and JavaScript content). If a catalog of SSL-enabled websites could be accessed and profiled beforehand, the timing signatures could be used to detect which websites were being accessed through an encrypting web proxy [7]. A more robust variant of this attack was described by Bissias et al. [8] and was further refined by Liberatore and Levine [9]. Most recently, Herrmann et al. integrated new fingerprinting techniques and generalized the attack to a broader variety of systems [10].

An attack by Wright et al. exploited an encoding scheme used in Voice over IP (VoIP) to identify the language being spoken in encrypted conversations [11]. The attack was shown to be successful when a variable bit-rate encoder was used and was tested on conversations between over two thousand native speakers in 21 different languages. They continued to explore how the privacy of VoIP users could be further compromised by using hidden Markov models to predict when certain phrases were used during the call [12].

On a more closely related note, Wright et al. also explored the topic of protocol identification in encrypted tunnels. They devised a new traffic analysis technique to classify encrypted network traffic based on which protocol, or set of protocols, were in use within a connection stream [13]. We use a similar approach, which also uses hidden Markov models, to label network traffic by its underlying protocol.

For further reading on traffic analysis attacks and anonymity on the Internet, see [4, 14].

1.2 Anonymity Systems

In response to such successful traffic analysis attacks, many tools and systems have been developed with the goal of enhancing security and privacy over the Internet. These tools go far beyond simply encrypting the transmitted data. Chaum first proposed the idea of creating a system that would mix traffic with other connections in order to provide a level of anonymity [15]. Today's systems employ a wide number of techniques, such as:

- Proxying connections through any number of systems around the world
- Mixing and reordering packet flows
- Controlling packet flow rates with timers and other batching strategies
- Padding packet sizes to a fixed length
- Sending fake cover traffic

Proxying the connections is done by most if not all anonymity systems in order to hide the source from the destination. Different systems employ various combinations of the other techniques in order to make it difficult, or in the best case impossible, to trace and analyze the traffic. In particular, techniques that modify the timing of packet flows have the greatest effect on a system.

Based on the use of these timing techniques, there are two classes of systems: high-latency and low-latency.

High-latency systems like Mixmaster [16] and Mixminion [17] are much better at hindering attacks based on timings. The reordering, mixing, and batching strategies they perform limit what can be gained by performing traffic analysis on the packet delays [18]. While very effective from an anonymity perspective, these systems are not as widely used due to the impracticality of time constraints. When accessing web sites, or typing keys into an SSH session, users do not want to deal with a very large time delay. Responses from the other system are desired as quickly as possible, which the timing strategies prohibit.

Low-latency systems like Tor [19], Java Anon Proxy (also known JAP or JonDo) [20], and Invisible Internet Protocol (I2P) [21] do not disrupt the timing of the packets as they propagate the network. This method is better suited for commonly used protocols like HTTP for web browsing and interactive protocols like SSH. We use Tor for our experiments since it is one of the more popular low-latency anonymity systems in use today. Chapter 2 will describe how Tor works in further detail.

1.3 Research Questions

Recent advancements with pattern recognition tools [22, 23, 24] provide the basis for this research. We ask: can we apply these tools to a data sequence generated from a network capture and achieve a model of the underlying process? In particular, can this be done when the process is streamed over Tor? Since Tor does not make any specific efforts to reorder the packets in a flow or introduce extra latencies, we hypothesize that this should be possible.

These tools [24] will allow us to test the confidence of the models we generate. We can also match new sequences of data to those models (or reject them all if none are a statistically significant match) [22, 24]. The application of these processes should give us an ability similar to that of Wright et al. [13] where we can classify a sequence of traffic, about which we know nothing, based on its underlying protocol. We propose that our methods will be an improvement because we can use our tools to show when we have collected enough training data and also to determine how well a new set of data matches up against a library of models. The latter can be achieved through the use of models with confidence intervals rather than the maximum likelihood approach.

We would like to take the detection a step further and see if it is possible to use our models to follow the path taken through the Tor network. Previous work done in this area uses temporal or frequency domain-based correlation to match input flows to output flows [25, 26]. We ask: if tracing the model paths is successful, how will it compare to these previous techniques?

Finally, can we develop and apply a well-defined process to construct models, match protocols, and perform path detection? If so, what degree of traffic analysis capabilities will this give us on Tor connections?

1.4 Organization

Previous sections of Chapter 1 outlined the motivation behind this work. The rapidly growing field of traffic analysis was surveyed and we demonstrated its importance as a field of study. The effectiveness of traffic analysis furthered the need for additional privacy, which led to the creation of a variety of anonymity systems. Also, we identified and described the primary questions we seek to answer through this research.

Chapter 2 explains the relevant background behind the tools used in this research. Since we use Tor as the anonymity system in our experiments, we take a deeper look at how it works and how it is able to provide that additional level of privacy. The limitations of Tor, some inherent and some by design, are discussed with the goal of creating a clearer understanding of the capabilities of Tor. Tor's popularity and widespread appeal has attracted a great deal of attention from researchers interested in security and privacy. We explore their work and discuss what attacks have been previously performed against Tor. We also provide some background on the pattern recognition tools described in Section 1.3.

Chapter 3 describes the process we use to construct models from the data we capture and how we test our confidence in them. We begin by looking at how to convert network traffic into data sequences, keeping what information we feel provides the best balance of reliability and potential for exploitation, the inter-packet time delays. The numerical data values must be symbolized for use by the model construction algorithm, and tools are described for how to approach the problem. Once the data has been symbolized, a model of the underlying process can be constructed. A proof-of-concept is demonstrated for this process. Finally, the constructed models are tested for how well they statistically represent the original process. We check to see if enough data was used in construction, and describe a method for pruning transitions out of the model that do not have enough data to make them statistically significant. An example of this is provided as well.

Chapter 4 focuses on how we can perform protocol and path detection using the models we generate. We show how to compute confidence intervals on the transitions of our models by running data through them. An example of this process is shown, where confidence intervals are generated for a model and are used for detection. We also show how the models can be used to perform path detection through Tor and compare the approach to that of Zhu et al. [25, 26].

Chapter 5 describes our results from using these pattern recognition tools to perform traffic analysis on a live Tor network. Our experiment utilizes the full range of processes from data capture to model construction to path matching. This chapter describes from start to finish what is needed to carry out our traffic analysis attack.

Chapter 6 summarizes the work, highlighting the conclusions we were able to draw from our experiments. We also propose some interesting questions that would be good candidates for future exploration.

Chapter 2

Background

Tor [19] is one of the most popular anonymity systems in use today. As of April 2010, there were almost 1800 relay servers on the public Tor network [27] and that number continues to grow. Tor's stability and deployment breadth can be attributed to its practical design, a product of the large amount of privacy and usability research done by its designers [28, 29, 30, 31, 18, 19, 32, 33, 34]. Until more scalable peer-to-peer solutions like MorphMix [35], Tarzan [36], and Torsk [37] develop from their fledgling experimental states into more stable, well-understood platforms, Tor is the anonymity system with perhaps the best long-term viability. It is for this reason that we have selected it for our traffic analysis research.

2.1 Tor

Tor is a low-latency anonymity system that operates as an overlay network on the Internet. An overlay network is a smaller sub-network that has been built over a larger, and usually pre-existing, network. Individual systems in the overlaid network communicate through virtual links that are physically transmitted by the underlying network, but are encapsulated so that they stay logically separated from regular traffic. Examples of this type of network would include a Virtual Private Network (VPN) or a peer-to-peer network, with the Internet being the underlying network.

Tor primarily consists of computers running any number of three types of services: relay, directory server, or client. Relays, also sometimes called nodes, routers, or onion routers, are the backbone of the network. Relays transfer data from clients to other relays, between two relays, or retrieve external resources for a client. By default, relays listen on TCP port 9001 for incoming requests. While they are active, they publish their status to a list of predefined directory servers. Directory servers catalog the information they have about various relays and vote amongst each other on which ones to list as running, valid, stable, etc. After the servers vote and all agree on a list, they come to a consensus. The consensus is published on a TCP port (9030 by default) where it can be downloaded by clients.

The client service sets up a listener for TCP streams that the user wishes to route through the network. The listener is a SOCKS [38, 39] proxy that listens on port 9050 by default. Before packets arrive to be proxied, the client initializes some circuits with the relays listed in the downloaded consensus. To create a circuit, the client chooses a relay to be the first node in the new circuit and sends it a creation request. Once created, the client uses the same circuit to extend out to a second and then third node. This way, the client chooses and knows each node in the path its traffic will take. Also, this gives the client the opportunity to set up symmetric key pairs with each node for performing encryption. Building the circuit is analogous to extending a spyglass from the source to the destination, with the end of each ring being a node of the circuit.

The flowchart in Figure 2.1 summarizes the process a Tor client follows to prepare for a data transfer. Once the first piece of data for a new TCP stream arrives at the proxy, one of the previously created circuits is selected for the transfer. If there are no circuits available, a new one is created. Before any data is transferred, the destination of the packet is sent through the circuit to the last node so that it can start the connection. This node is called the exit node since it is where all of the packets will exit the Tor overlay and continue on to their final destination. The exit node sends confirmation to the client once it has connected to the destination, allowing the data transfer to begin.

As each data packet is ready to be sent, it is split or padded into cells of 512 bytes, which are then iteratively encrypted using the key of each successive relay of the circuit. In other words, the original packet is encrypted with the key for the last node, which is wrapped in another packet and encrypted with the key for the second node, which is encapsulated again and encrypted with the key for the first node. The result is similar in nature to an onion. As each relay in the circuit receives the onion packet, it will decrypt and peel away a layer, forwarding on what remains.

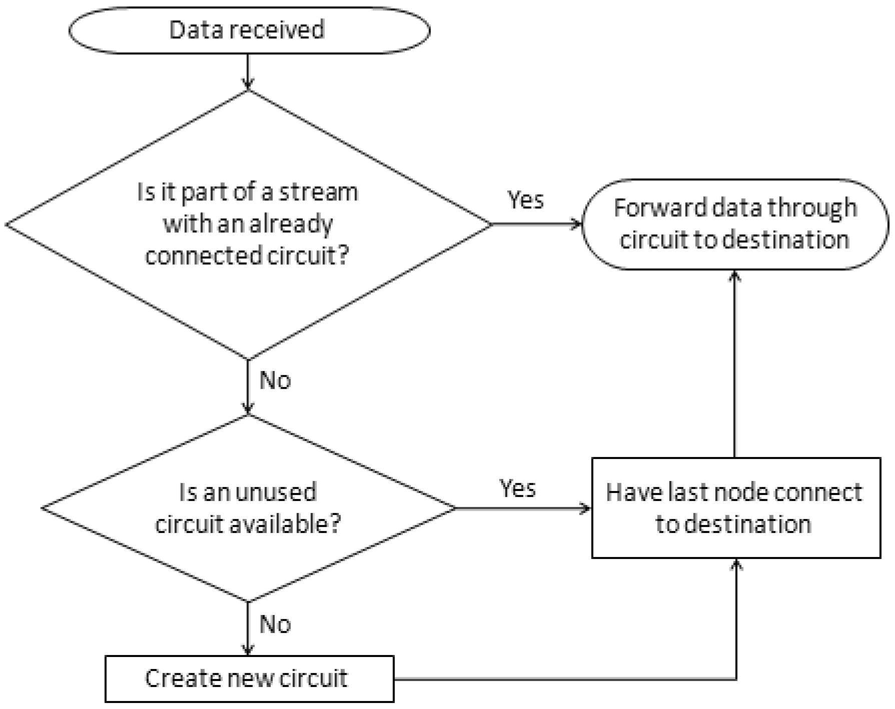

Data received
Is it part of a stream with an already connected circuit?
Yes
Forward data through circuit to destination
No
Is an unused circuit available?
Yes
Have last node connect to destination
No
Create new circuit

Figure 2.1: Preparation for data transfer

Figure 2.2 shows the onion packet being sent from the source, which we will call Alice, to the first node in the circuit. The onion packet contains four layers at this point, and only the headers of the outermost layer are visible. This is represented in the figure by the hash symbols where the headers of the inner layers would be. This is to imply that they are encrypted within the payload of this specific packet and cannot be read by anyone sniffing the line.

Once the packet is received by the first node, the old headers (or in the terms of our onion analogy, the outermost layer) are stripped away. The payload is decrypted using the key that was previously established with Alice, revealing the next hop of the circuit. This process is illustrated by Figure 2.3. The relay then forwards the remainder of the onion packet to the next hop, which is shown in Figure 2.4. Notice that this relay cannot determine that the packet originated from Alice. It can only see the packet coming from the first node. Even after it goes through the same unwrapping and decrypting process, it will also not be able to see the destination, Bob. It will only see another onion packet that is to be relayed to the third node in the circuit.

After the third node unwraps the last layer of the onion packet, it has the original packet from Alice. The packet is sent “as is” to Bob through the previously established connection (shown

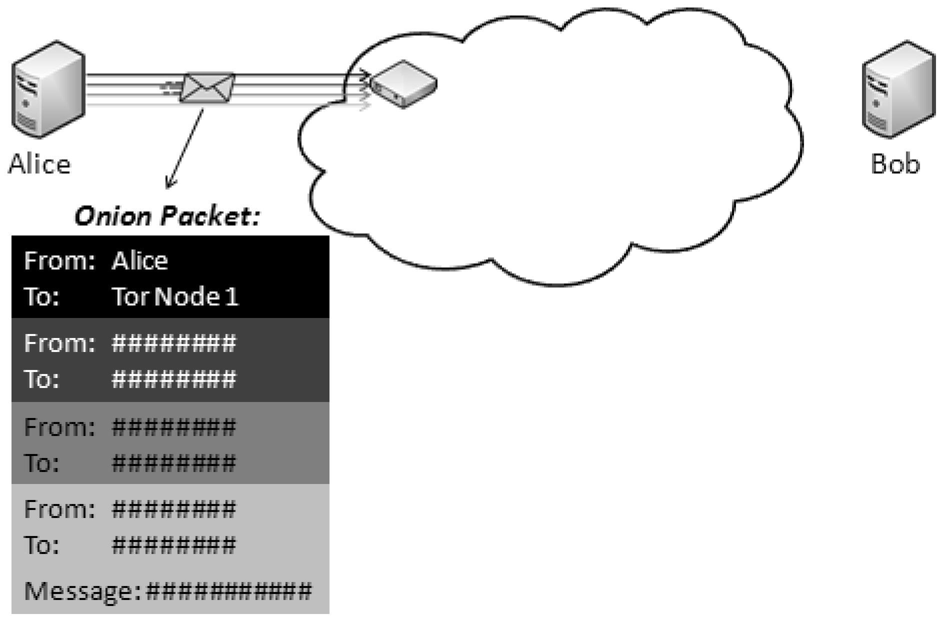


Figure 2.2: Onion packet entering Tor circuit

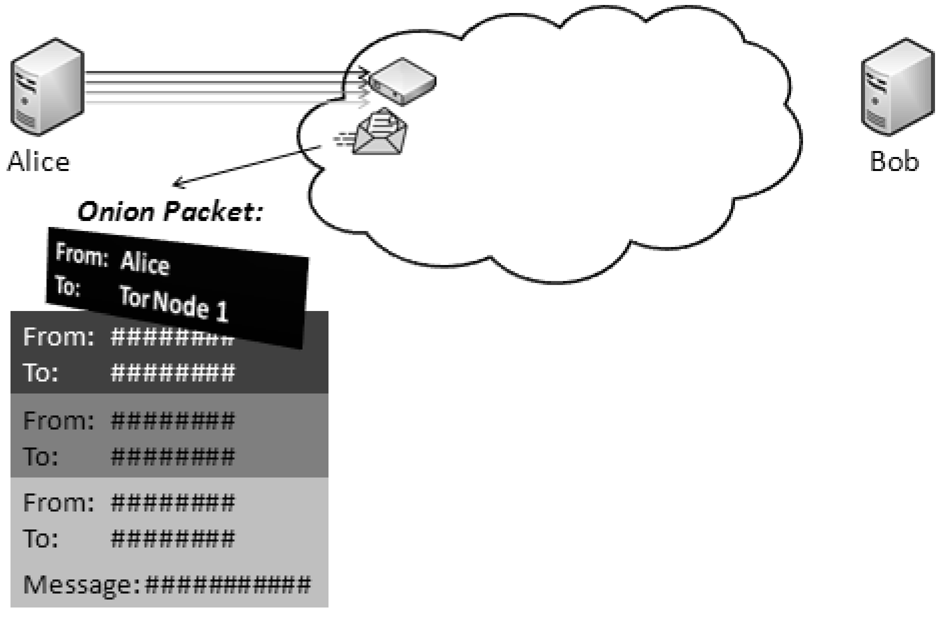


Figure 2.3: Outer layer of onion packet is peeled away and next layer is decrypted

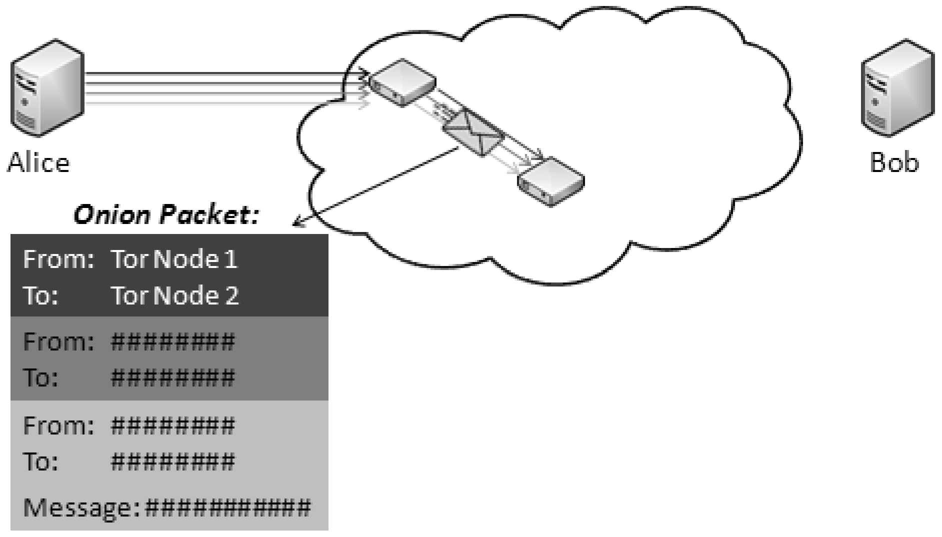


Figure 2.4: Remaining onion packet going to second Tor relay in circuit

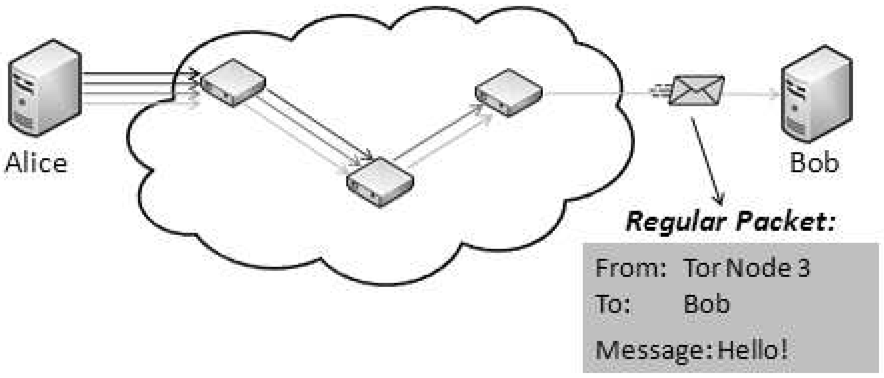


Figure 2.5: Exit relay forwards original packet to destination

in Figure 2.5). That is, if the packet was originally in cleartext, it is sent in cleartext. To achieve encryption for this step of the circuit, Alice must have used a secure protocol like SSH or HTTPS. A common misconception is that Tor always provides end-to-end encryption, but this is not the case. This misconception has allowed people to use Tor for capturing usernames and passwords [40].

When the destination, Bob, replies to Alice's request, the same process is followed in reverse order. The response packets are successively wrapped in relay cells at each node in the circuit. Once again, the cells are iteratively encrypted by each relay using the key set up for that circuit. When Alice gets the response packet, her Tor client unwraps and decrypts each layer to return the original packet from Bob up through the SOCKS proxy and back to the application she is using.

Obviously, there are many more details to the process, such as encryption schemes, integrity checking, congestion handling, and path selection. For more in-depth information on those topics, see the Tor specification [19]. Also, Murdoch and Watson provide another good overview and more updated information about Tor's path selection techniques. An alternative technique is suggested, but it is found that Tor's current method of weighting nodes by bandwidth is much better for performance, while at the same time provides better security against botnets running Tor relays [41].

2.2 Attacks on Tor

We are not the first ones to focus our research on Tor. For the same reasons we outlined earlier, many other researchers have examined the abilities of Tor to resist various forms of traffic analysis. Some of these works led to successful attacks on anonymity and stimulated further discussion on how to handle those situations.

Returning to the path selection analysis by Murdoch and Watson [41], a large amount of compromised computers, "bots", that were controlled by an adversary, a "bot herder", could be used to run a very extensive and diverse set of relays. With such a diverse set of relays, they could expect to capture a broad sampling of the traffic going through Tor. Other work by Murdoch suggests an adversary may only need 1 out of every 2000 packets to perform a successful timing attack [42]. This could become a serious threat to privacy, and the best path selection strategy to prevent this was found to be the one currently in place. Because bots are typically on low-bandwidth connections, consistently choosing paths with higher available bandwidth would help users avoid botnet relays, and reduce the incentive for bot herders to run them.

At one time however, the path selection strategy was found to be worse for anonymity. Bauer et al. found that the Tor directory servers did not do anything to verify the bandwidth claims reported by a relay [43]. By modifying the relay code to falsely report a large amount of bandwidth, an attacker could attract a large amount of connections to themselves and gain a higher probability of seeing each circuit. The path selection technique was kept, but mitigations were made to keep low-bandwidth relays from abusing it, such as integrating reputation into the directory server consensus.

Murdoch and Danezis performed traffic analysis on the Tor network using a timing-based congestion attack [44]. The attack required an adversary to create a single hop circuit with a relay in the global Tor network and flood it with data. Then whenever the timing slowed, assuming it was due to the relay and not the link, one could guess that the relay was processing data in a circuit. If enough relays were observed using this technique, slow downs in the flows would line up and show the attacker which three relays were being used in a circuit. This attack was devised and orchestrated very early in Tor's history when there were only 13 nodes on the entire network. Newer research suggests however, that with a much larger Tor network and heavier traffic that the attack is no longer possible [45].

Hopper et al. compiled catalogs of round trip times and used them with a malicious web server to limit the degree of Tor's anonymity [46]. To collect round trip times for comparison, the attacker must connect to servers on various subnets around the Internet and compile a latency map to those subnets. Then, if a user could be directed to the malicious web server, they could measure how long the packets took to go round trip to the client. The client's round trip time could be compared to the round trip times on the latency map and a reasonable estimate could be made of 2 to 4 subnets in which the client likely resides.

Zhu et al. used temporal and frequency domain-based correlation of a network's entrance and exit times to attack the anonymity of mix networks like Tor [25, 26]. Using just the captured packet times, streams from the source flowing into the mix or network of mixes are matched to streams flowing out by a maximum likelihood method. This attack requires the global observation of all entrances and exits from the network.

2.3 Pattern Recognition Tools

For our attack on Tor, we use various techniques described by Schwier [24]. In his work, Schwier further develops an algorithm to generate hidden Markov models, or HMMs, from a time series and provides new tools for effective pattern detection using the HMMs and a test sequence. We now provide some background on those tools. Our notation should be consistent with that of Schwier's.

2.3.1 Zero-Knowledge CSSR

The Causal State Splitting and Reconstruction, or CSSR, algorithm can be used to generate HMMs from discrete sequences of data without having to make any prior assumptions about the model structure. The algorithm was first proposed by Shalizi and Crutchfield [47, 48] and was further refined in their later works [49, 50]. The algorithm determines the underlying state structure, called the causal states, and the transition probabilities of the process representing the input data with the least amount of uncertainty. See Figure 2.6 for an example result model.

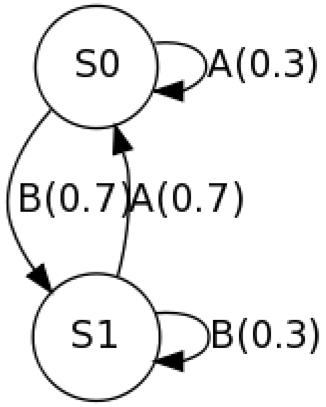


Figure 2.6: Example of a hidden Markov model created using CSSR

The original algorithm by Shalizi and Crutchfield required only the data sequence from which to construct the model and a maximum string length parameter, L. The first step of the algorithm involves splitting the data sequence into smaller subsequences of incrementally increasing lengths, and this parameter tells it when to stop.

Their implementation made two important assumptions: that one knew the best value of L to use and that there was more than enough data in the input sequence. However, these assumptions do not always hold when dealing with realistic situations like data captured from an unknown computer network process. This is where the work of Schwier et al. [24, 23] applies.

To resolve the first assumption, an algorithm was developed to find the best value of L to use. CSSR is run with incrementally increasing values of L, starting at $L = 2$. After CSSR generates the model for that value of L, the original data is traced through the model and the sequence of states it takes is recorded. The state sequences are compared for each L_n and L_{n-1}. When two are found that match, the state structure is assumed to have stabilized and the previous value of L is said to be the best value to use.

Using CSSR and the process for determining L, an HMM of a process can be created from a sequence of observations without any knowledge of parameters *a priori*.

2.3.2 Model Confidence

The second assumption Shalizi made was that there was always more than enough data in the input sequence. This is how he reasoned that the generated model would be the model that most optimally represented the underlying process. By always using enough data, CSSR sees all possible transition events of the underlying process and properly represents them all in the model it creates.

However, we will only ever have finite amounts of data, which may or may not include all possible events the process will take. This is especially true in our application. We need a way to be statistically confident that certain unexpected events will not occur within a given probability. Schwier uses the z-test to calculate a metric of statistical confidence. The z-test reports how much data is needed to have confidently captured enough low-probability events. It also finds how much confidence there is in the generated model in cases where not enough data was used.

In those cases when the test finds that an insufficient amount of data was used, it is because low-probability events were visible but there are not enough of them to be confident in conditions under which they occur. In these cases, CSSR includes states and transitions in the generated models without being confident in the underlying conditional probabilities for those events. Schwier also provides a method to prune these unsubstantiated states and transitions from the model by thresholding the asymptotic state probabilities. More information about this process is provided in Section 3.6.1.

2.3.3 Confidence Intervals

Traditionally, new incoming data is matched to a model using the Forward-Backward procedure [51]. The forward part of the procedure uses maximum likelihood to find the best match out of a set of models, and the backward part tunes the model parameters to the data.

The use of confidence intervals provides an alternative method for matching data to a model [22]. Confidence intervals are calculated for each transition probability in an HMM. This is done by running the test data through the model and counting the frequency that a transition is taken out of the total number of times the model had the opportunity to take it.

The confidence interval method has a few advantages over the maximum likelihood approach for our application. Most notably, we are not forced to choose a winner out of a set of models when using confidence intervals. While maximum likelihood simply selects the model with the least amount of difference from the data, confidence intervals provide a metric for describing how similar each model is to the data. If none of them are within a specified threshold, called the rejection rate, we can reject them all and say that the data does not match any of the protocols we have modeled.

Another more intuitively appealing aspect of the confidence interval approach is how increasing amounts of data are handled. When the first few samples have been used, the intervals are very wide, having a high degree of variance. As more data samples are used for detection, the intervals tighten up around the transition probability due to lower variance.

Also, the number of multiplications is constant for any amount of data samples. This differs from maximum likelihood, where the number of multiplications is linear with respect to the amount of data samples. This presents a problem because each multiplication is always by a probability—a floating point number between 0 and 1. This makes the result get smaller and smaller as it asymptotically approaches zero. One way to mitigate this is to normalize all the values after each step by scaling them. However, this is a poor solution for computer implementations because the additional multiplication operations cause more floating point error to accumulate in the result [52].

The results of Schwier et al. found that the desired false negative rate could be calculated when using confidence intervals, but the false positive rate could not. Receiver operating characteristic (ROC) curves were used to find the best parameters in each situation. In general, both the true positive and false positive rates would be higher for detection with confidence intervals, but there would be a lower false negative rate. Maximum likelihood would have a lower false positive rate [22, 24]. In this sense, confidence intervals are better suited for identification, while maximum likelihood is better for classification.

2.3.4 Window Sizes

The final problem examined by Schwier was how to determine the best window size to use when matching data. Typically, our data sequences will be split into much smaller subsequences, or windows, when being used for detection. This is done to allow us to tell which model best matches the data at various points in time during the sequence. For instance, protocol A may be running for the first few minutes of a test capture and then the system might switch to protocol B for the next few minutes, and then on to protocol C. If we run the entire test set through the detection routine, we will just see a match to one of them (or possibly none of them if confidence intervals are used). By windowing the data, we get a better feel of which process is running at different times.

Optimally windowing a data set is a very complicated task requiring the user to make trade-offs. If the window size is too small, there may not be enough data to make a statistically confident decision about which process is the best match. The confidence intervals will be too large with too much variance, enabling multiple models to have high acceptance rates.

If the window size is too large, it will take longer to recognize a change in the underlying process. Once the change occurs, newer data will have to fill up more than half of the window to statistically overpower the old data in the window. It is only after this point that the new underlying process could be recognized.

The methods by Schwier [24] determine a good window size to use for comparing data to a pair of models. The resulting size should be no greater than the minimum necessary to tell the difference between sequences from the two models. Any larger and we would waste time in spotting changes between the two behaviors. The minimum necessary value is found by analyzing binomial distributions of two transitions to determine the likelihood of making a decision error.

2.4 Summary

In this chapter we described Tor and presented our reasoning for using it in our experiments. The net effect of using Tor was that websites or other things you use on the Internet will not know who you are. The level of privacy provided extends so that no one single point in a circuit knows both the source and the destination. Our reason for using it was its popularity and large amount of active research. This research has yielded viable attacks on the anonymity guarantee that Tor tries to provide, and the results of those attacks were described.

We also introduced the pattern recognition tools used in this research and highlighted some details that are relevant to this work. These tools provide the capability to generate models of a process by observing a sequence of its outputs. The CSSR algorithm has been proven to find the set of states for a model that best represent the underlying process when given enough data. We described a tool that can be used to test whether we have used enough data, and another one to prune the models we generate of any transitions that are not statistically significant. We also examined a way to detect whether a new sequence of data was a statistical match to a model through the use of confidence intervals.

Chapter 3

Model Construction

In order to run our attack on Tor, we need to create training models to use in our matching process. For any connections that we try to match up, we must first have a valid hidden Markov model that represents the protocol in use. To do this, we must collect training data from a Tor connection where we know the protocol being used. Then, a specific process is followed to create a good model from the data. This chapter describes that process in detail and discusses our results from using it.

3.1 Process Overview

We follow the process described by the flowchart in Figure 3.1 to create the models required by our attack. First, data is collected from a connection where we know the protocol that is in use. This will be our training data. The model that is created from it can be used to detect that same protocol in future connections we test.

After symbolizing the data we captured, the CSSR algorithm introduced in Section 2.3.1 is used to create a model. We start with $L = 2$ and increase it as needed. To determine if an increase is needed, we first apply the z-test described in Section 2.3.2 to make sure we used enough data. Once it is established that enough data was used, the value for L is incremented if the state structure of the model changed. Sometimes enough data (according to the z-test) cannot be captured. We also discuss the reason for this and what can be done in order to still make use of the model.

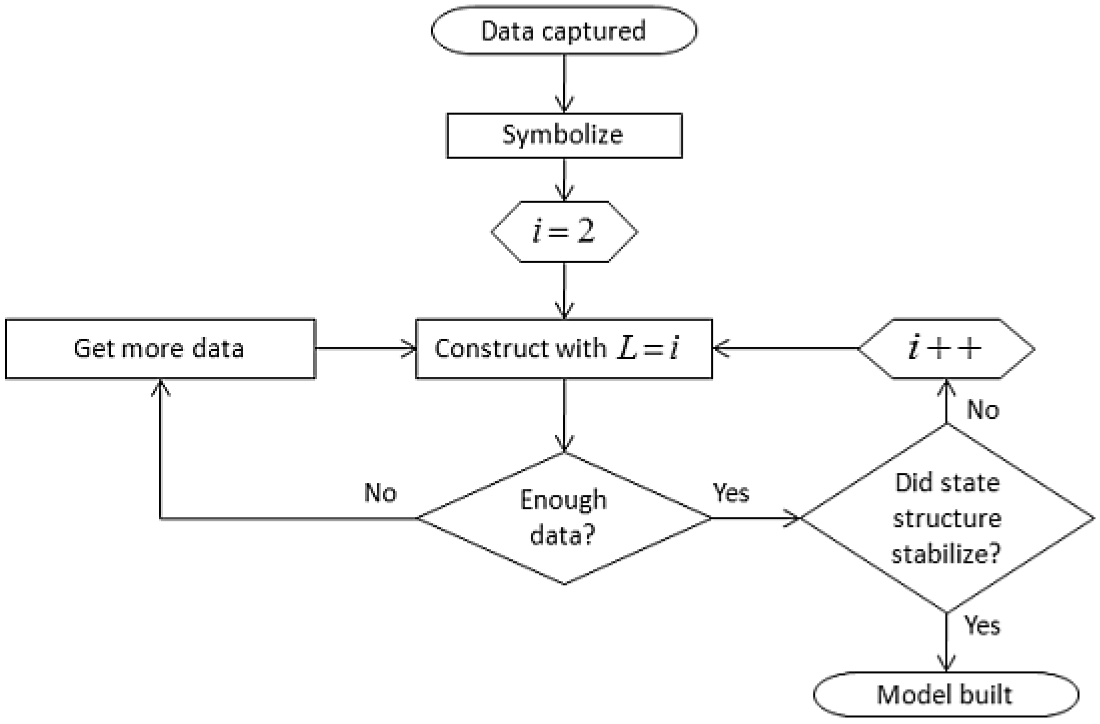


Figure 3.1: Flowchart summarizing the model construction process

3.2 Data Collection

For our experiments, all data collection is done on processes sent through Tor. Our experiments use data captured in our private Tor network testbed. The network is made up of a collection of systems all running Tor and our test protocols. For more information on how the network and Tor were configured, see Appendix A.

The tshark [53] program is used to capture packets within the network. It is a command line version of Wireshark and is usually included in the Wireshark package. tshark runs with less overhead than its graphical counterpart. This allows more of the system's memory to be used to capture larger data sets. tshark was chosen over other command line capture tools, like tcpdump [54], because it still provides access to the very versatile filtering rules in the full Wireshark.

For filtering, tshark is configured to only capture packets with a data length greater than zero. This will throw out plain ACK packets, as we are only interested in the ones carrying data from the protocol. These packets will make up our observations of the network process. In addition, we may filter for a certain port or IP address depending on what type of traffic we want to capture.

For outputting, tshark is configured to only record the time that each packet was received, separated by newlines. The output value is a Unix timestamp (the number of seconds since midnight on 1 January 1970) [55]. All other data about the packets (e.g. sequence numbers or lengths) is ignored.

Below is an example of a tshark command to record packet times for all non-empty packets going to a Tor SOCKS proxy:

```
tshark -i eth0 -n -l -o column.format:'"timestamps","%t"'
-R "tcp.srcport==9050 && tcp.len > 0"
```

3.3 Symbolization

Before any collected data can be used by our pattern recognition tools, it must be symbolized. To do this, the difference between each successive packet time is computed and that value is used by a lookup table to determine the appropriate symbol it should be given.

The time difference, Δt, is calculated by subtracting the receive time of the previous packet from the time of the current packet. In other words, $\Delta t_i = t_i - t_{i-1}$ where t is the receive time of packet i. Obviously, the first packet in our data sequence will not have a value, so we just start with $i = 2$. This step is done so that network latencies between the source and the destination are filtered out. The data sequence we capture will look the same at either end of the connection.

Time each packet is sent, t_i	**Delta at sender,** Δt_i	**Time each packet is received,** t_i	**Delta at receiver,** Δt_i
1269407285.323450	**N/A**	1269407285.490769	**N/A**
1269407285.789346	**0.466**	1269407285.956495	**0.466**
1269407285.916296	**0.127**	1269407286.083583	**0.127**
1269407286.483149	**0.567**	1269407286.650283	**0.567**
1269407286.909037	**0.426**	1269407287.076253	**0.426**
1269407287.034984	**0.126**	1269407287.202075	**0.126**

Table 3.1: Deltas are used to ignore constant latencies

Using deltas will allow us to ignore any constant latencies in the network connection. Notice in Table 3.1 that there is a 167ms delay between the send time and receive times, but this does not affect the delta values; they are the same for each packet. This technique works well for constant latencies, but if the latency suddenly increases, it could cause a time sample to be incorrectly symbolized. The extra delay could cause the sample to be grouped into a higher interval. Table 3.2 demonstrates this point. Extra latency after the fourth packet was sent causes it to be received later and all of that extra latency carries over to the delta value.

Time each packet is sent, t_i	**Delta at sender, Δt_i**	**Time each packet is received, t_i**	**Delta at receiver, Δt_i**
1269407285.323450	**N/A**	1269407285.490769	**N/A**
1269407285.789346	**0.466**	1269407285.956495	**0.466**
1269407285.916296	**0.127**	1269407286.083583	**0.127**
1269407286.483149	***0.567***	1269407287.241283	***1.158***
1269407286.909037	***0.426***	1269407287.427253	***0.186***
1269407287.034984	**0.126**	1269407287.553253	**0.126**

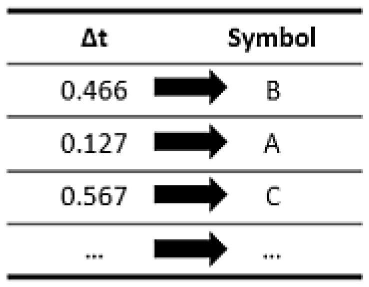
Table 3.2: Deltas do not handle variable latencies

When working offline, a Perl script is used to parse the output from tshark and calculate time deltas to 10^{-4} precision. If an incoming data sequence is being used for real-time detection, we let that detection program calculate the time delta to the same precision.

Using the calculated values of Δt, we symbolize the data by grouping it into ranges and assigning anything in that range a unique symbol such as A or B. To do this, we must have a lookup table such as Table 3.3. Figure 3.2 demonstrates the symbolization process on some sample data.

Symbol	**Range**
A	{0.000,0.300}
B	{0.301,0.500}
C	{0.501,1.000}

Table 3.3: Example lookup table



Δt		Symbol
0.466	➡	B
0.127	➡	A
0.567	➡	C
...	➡	...

Figure 3.2: The symbolization process

If we are generating a model from new training data, we will have to create a lookup table for our new data. We derived our intervals by splitting the packets into distinct timing clusters. We did this by plotting a histogram of the full data sequence and looking for local minima in the discrete curve. In our process, we create the symbolization ranges by hand using this plot as a guide. This step could be automated if a completely autonomous model construction process is required.

Figure 3.3 shows a histogram of SSH file transfer data captured over Tor. We split the packet times into groups at the dotted lines. Again, these ranges were selected by hand. The x-axis values of each of these lines make up the interval boundaries for the new lookup table.

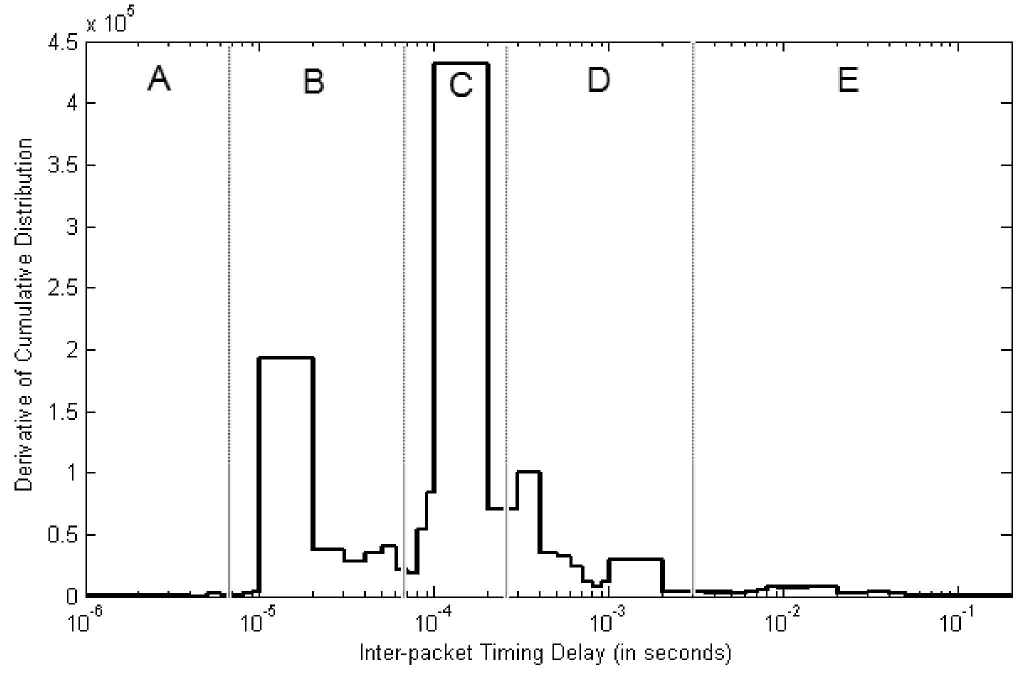


Figure 3.3: Probability distribution of inter-packet time delays during an SSH file transfer

3.4 CSSR

The input data is a discrete sequence of observations χ that are observed from some unknown process, where $\chi = [\chi_1, \chi_2, \ldots, \chi_n]$. In this case, n is the total number of observations in the data sequence χ. The individual observations can be any symbol out of a finite alphabet $\mathcal{A}$. To illustrate this, assume an unknown process is generating a sequence of outputs. If we observe ten of these outputs, our data sequence for CSSR might be

$$\chi = [\ A\ B\ A\ B\ A\ A\ B\ B\ A\ B\ ]$$

and the alphabet would be $\mathcal{A} = [A, B]$. In practice the sequences will be much longer, containing thousands or even millions of symbols.

CSSR works by splitting the sequence up into smaller subsequences and looking at the probability of seeing a certain symbol after each different type of subsequence. The splitting is done for incrementally increasing subsequence lengths, from zero up to a preset limit of L. That is, we perform the split process for $\{i : 0 \leq i \leq L\}$. In our unknown process example, if we let $L = 2$, we will split for $i = 0$, $i = 1$, and then $i = 2$. The sets of split subsequences would be

$$W_1 = [\ A\ B\ A\ B\ A\ A\ B\ B\ A\ B\ ] \qquad \text{for } i = 1$$

$$W_2 = [\ AB\ BA\ AB\ BA\ AA\ AB\ BB\ BA\ AB\ ] \qquad \text{for } i = 2$$

We do not show the case of $i = 0$ in the figure, because it is just a null set. Also note that there will be $n + 1 - i$ subsequences in each subsequence set W_i.

After the first step of $i = 0$, the model will have just one state. Additional states may be added during the rest of the split steps $1 \leq i \leq L$. A new state is added when the conditional probability of a subsequence and its following symbol does not match that of any other state. These conditional probabilities are calculated by counting the number of times a certain symbol follows each subsequence in W_i.

For example, at $i = 1$, the conditional probabilities for the set W_1 would look like

$$Pr(A \mid A) = \frac{1}{5} \qquad Pr(B \mid A) = \frac{4}{5}$$
$$Pr(A \mid B) = \frac{3}{4} \qquad Pr(B \mid B) = \frac{1}{4}$$

At $i = 2$, the probabilities for set W_2 will be

$$Pr(A \mid AA) = \frac{0}{1} \quad Pr(A \mid BA) = \frac{1}{3} \quad Pr(B \mid AA) = \frac{1}{1} \quad Pr(B \mid BA) = \frac{2}{3}$$
$$Pr(A \mid AB) = \frac{2}{3} \quad Pr(A \mid BB) = \frac{1}{1} \quad Pr(B \mid AB) = \frac{1}{3} \quad Pr(B \mid BB) = \frac{0}{1}$$

After determining the conditional probabilities for a step, the χ^2 statistical test is used to compare each probability to the ones from the previous step. If the test determines that it does not match the probabilities for a state, a new one is created and the subsequence is added to the new state's history. For the full pseudocode, see [24].

Models created by our implementation of CSSR have two important properties that will allow us to use them for detection purposes. Most notably, all models will:

- Have transient states removed
- Be deterministic

During construction, the algorithm removes any transient states that may arise. A transient state is a state that cannot be reached again once the model has transitioned out of it. For example, in the model in Figure 3.4, state S0 is a transient state and will be removed. We perform this step because a model cannot be positive recurrent if it contains any transient states. Positive recurrence is an important property that will allow us to compute asymptotic state probabilities for a model. Schwier explains the conditions for this property and how to calculate the asymptotic, or steady state, probabilities [24].

Deterministic models do not have states where the same symbol can cause more than one possible outbound transition. That is, in any state of a deterministic model, there will be a unique transition for each possible symbol. Figure 3.5 demonstrates this property. The model in 3.5a is non-deterministic because at state S0, an A can cause either a transition to S1 or back to S0. The model in 3.5b is deterministic. This property is important because it means that any sequence of

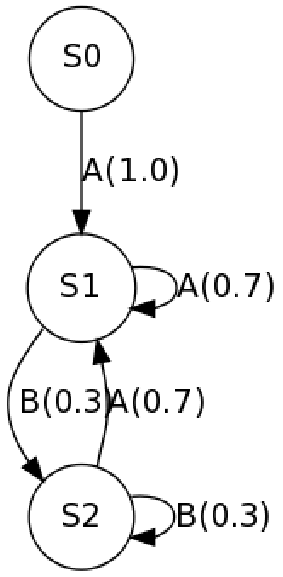
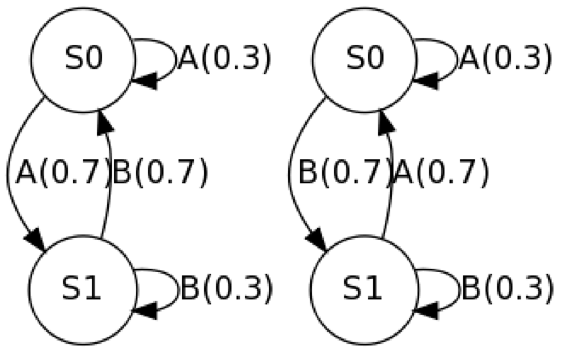


Figure 3.4: A model that contains a transient state



(a) Non-deterministic (b) Deterministic

Figure 3.5: Example of determinism property

transitions has only one possible sequence of states. In this work, the sequence of transitions is called the data and the sequence of states is known as the path taken by that data through the model.

3.5 Proof-of-Concept

Our first experiment was to test the data collection, symbolization, and CSSR steps. This proof-of-concept test was designed to be very simple, but still require each of the steps to work properly to achieve the desired result.

Using the private Tor network, two systems were configured with simple client and server processes. One computer was designated as the server and the other was the client. The client would connect to the server and just listen. The server, once the client had connected, would send data packets to the client based on a preloaded model. The model used by the server in this experiment is the one depicted in Figure 3.6.

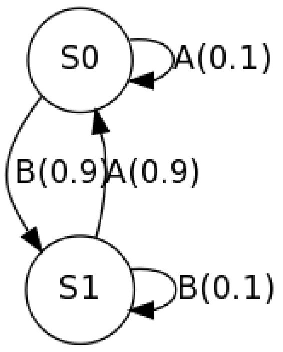


Figure 3.6: Original model used by the server

The server starts the process by randomly selecting a state in its model to be the start state. To send each packet, a transition is taken from the current state and the corresponding symbol is sent in a packet to the client. If there is more than one possible transition out of a state, the transition is chosen randomly, weighted on the probability of each transition. Most importantly, the time between sending each packet depends on the symbol of the transition. Each symbol is assigned a specified time delay in milliseconds and the server waits that amount of time before sending the packet to the client.

For example, if the server generates an A from its model, a packet is sent after a delay of 0.1 seconds. If the next symbol generated is also an A, then another packet is sent after the same time delay. If the next symbol is a B, however, then a delay of 0.9 seconds is waited before sending the next packet.

This technique relates the inter-packet delays to transitions of the HMM. In other words, the transitions of the model will be our observations of the underlying process. This is the behavior we expect in real protocols, that the packet times will be related to processing required by a specific task in the process.

The client and server processes are set to generate 10,000 symbols of data. All of the packet times are captured and symbolized using the processes described in previous sections of this chapter. CSSR is used to construct a model from the symbolized data. With $L = 2$, CSSR yields the model shown in Figure 3.7.

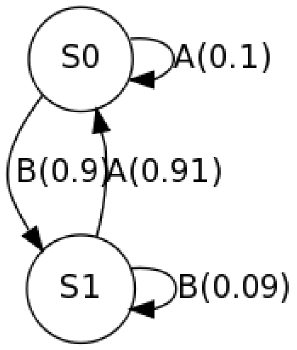


Figure 3.7: Model reconstructed from captured data

We can see that the reconstructed model has the same state structure and almost the same transition probabilities as the original model. If the process is re-run for 100,000 symbols, we find that the probabilities do match the original model.

This illustrates a point made earlier during the discussion of CSSR. If an insufficient amount of data samples are used, the algorithm only creates a model of the data, not of the underlying process. To get a model of the underlying process, there must be enough data samples available to fully describe that process. This is what makes the difference in the transition probabilities, in seeing 0.91 instead of 0.90.

3.6 Model Confidence

The results of our proof-of-concept experiment demonstrate that we need to have a way of calculating if we used enough data. And in cases when we did not use enough, we must find out how much more data we would need to use to construct a statistically confident model.

It makes sense that we can never be completely sure that the model we get perfectly represents the underlying process. Every model we build is constructed from a finite amount of data. We would have to use an infinite amount of data to get a model which perfectly represents the underlying process. This is because if the first time at which a given event occurs approaches infinity, the probability of seeing it approaches zero. If we know that there is an extremely small probability that our captured data contains all possible events, we need to capture more data until we achieve an acceptable probability that we saw everything.

Fortunately, Schwier provides a method to calculate the amount of data samples necessary to achieve a specific level of confidence in a model [24]. It is calculated from the smallest asymptotic state probability in a model, or in other words, the least likely event to occur. The result can be tuned by the parameter κ, which is usually set to 0.05 times the smallest asymptotic state probability. κ is the probability of seeing a joint event. Schwier defines a joint event as an occurrence where the system takes a transition that has not yet been observed. The higher κ is, the lesser the amount of data will be needed. The offset is that there is more risk in choosing a higher value of κ because it assumes that the system is less likely to take an unknown transition.

As an example, imagine that a process is represented by the model in Figure 3.8. The transition of the dotted line occurs very rarely, say once a millennium. If we observe this process for any period of time less than a millennium, there is a chance that we may not ever see this transition. In fact, the probability of not seeing it increases as our observation time decreases.

If we happen to observe the process at the right time and see the dotted transition, CSSR will add it to our model. But should we have added it? The z-test may tell us that we need data for fifty millennia before we confidently understand the conditions in which the dotted transition occurs. For all practical purposes, we do not have that kind of time. What we are left with is an event about which we are unsure. It may happen under the same conditions every millennium, may happen under different conditions, or it may have just been noise and never happen again.

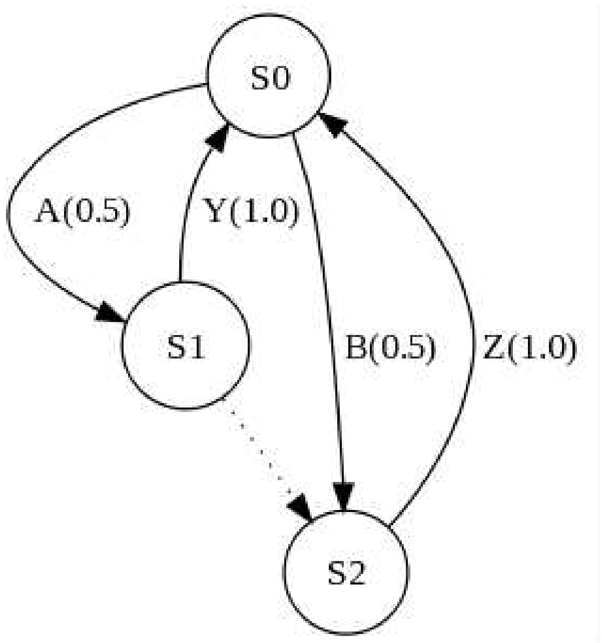


Figure 3.8: Model of a process with a very rare event

In practice, we find that the best solution is to ignore the event and note that rarely occurring events have not been included in the model. We record the probability at which we start ignoring these rare events. This provides us with a model that works most of the time and a measurement that tells us how often we should expect it to not be able to fully represent the process.

3.6.1 Pruning Rare Events

To handle rare events and maintain model confidence, we use a threshold on asymptotic state probabilities to prune them out. After a model has been constructed with CSSR, it may have transitions that are taken very rarely and states that are visited very rarely. By setting a threshold on the asymptotic state probabilities, we accomplish the goal from the end of the previous section; rare events are trimmed from the model. The value of the probability threshold is how often we should expect the process to deviate from the model.

A state's asymptotic state probability is a measure of how often that state is visited as the data length goes to infinity. More specifically, it is the probability that the model will be in that state during an infinite period of time. In his work, Schwier explains how to calculate this value for each state in a model [24].

Any state with an asymptotic probability below the threshold is removed from the model. Any transitions going to or leaving from that state are also removed. Finally, any state or set of states that cannot be reached due to a removal are also deleted. This leaves the model with only the states and transitions for which we have enough data. When we are unable to collect enough data to be confident in the full model, we leave out the parts where we would need more data to achieve confidence.

3.7 Pruning Experiment

We then test this process over Tor. In a Tor connection, there are times when a circuit fails or changes, or a relay gets too busy and delays a packet. There is some extra variable latency that affects roughly one out of every 200 packets. These glitches cause the packet to arrive later than it should have and because of that, it is incorrectly symbolized.

Typically, noise in the data can be mitigated by collecting more and more data until the underlying process overwhelms the noise. This works well for the simple model in Section 3.5. But for more complicated systems, glitches at different times can cause many more errors in the model because there is a larger state space.

For this experiment, we use the same client and server method described in Section 3.5. All we change is the server's model and the symbol-to-delay translation table. The model now used by the server is the model in Figure 3.9. The translation table is Table 3.4.

Symbol	**Delay** (in seconds)
A	0.2
B	0.4
C	0.1
Y	0.3
Z	0.5

Table 3.4: Symbol-to-delay translation table for pruning experiment

The process is configured to generate 200,000 data packets. After completion of the process, deltas of the captured packet times are calculated and symbolized. Visual inspection of the data

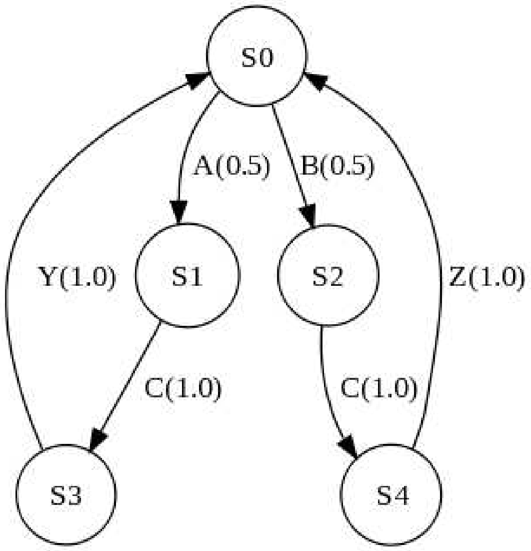


Figure 3.9: Original five state model for the pruning experiment

reveals that added delays cause various samples to be symbolized incorrectly. For example, the following subsequence is found in the data:

$$W = [\ A\ C\ Y\ B\ A\ Z\ B\ C\ Z\ Z\ C\ Y\ ]$$

However, since we know the model, we know that the subsequence should really be:

$$W = [\ A\ C\ Y\ B\ C\ Z\ B\ C\ Z\ A\ C\ Y\ ]$$

But CSSR does not know this. CSSR can only make a model from the data it has been given. The model it constructs will have extra states and transitions to handle these glitches. This means having a much bigger model than expected, which is much different from the original model. Figure 3.10 shows the model that CSSR constructs for these first 200,000 packets.

When the confidence test is run on the model, we find that it requires 20,624,750 data samples. This means we need to capture more data and rebuild. Because the amount of required data is so large, it has to be generated in lengths of 200,000 packets at a time. After each set of 200,000, we rebuild the model and run the confidence test again.

Oddly enough, the required amount of data keeps increasing with each set. This is because different incorrect symbolizations at different points in the data sequence cause even more states

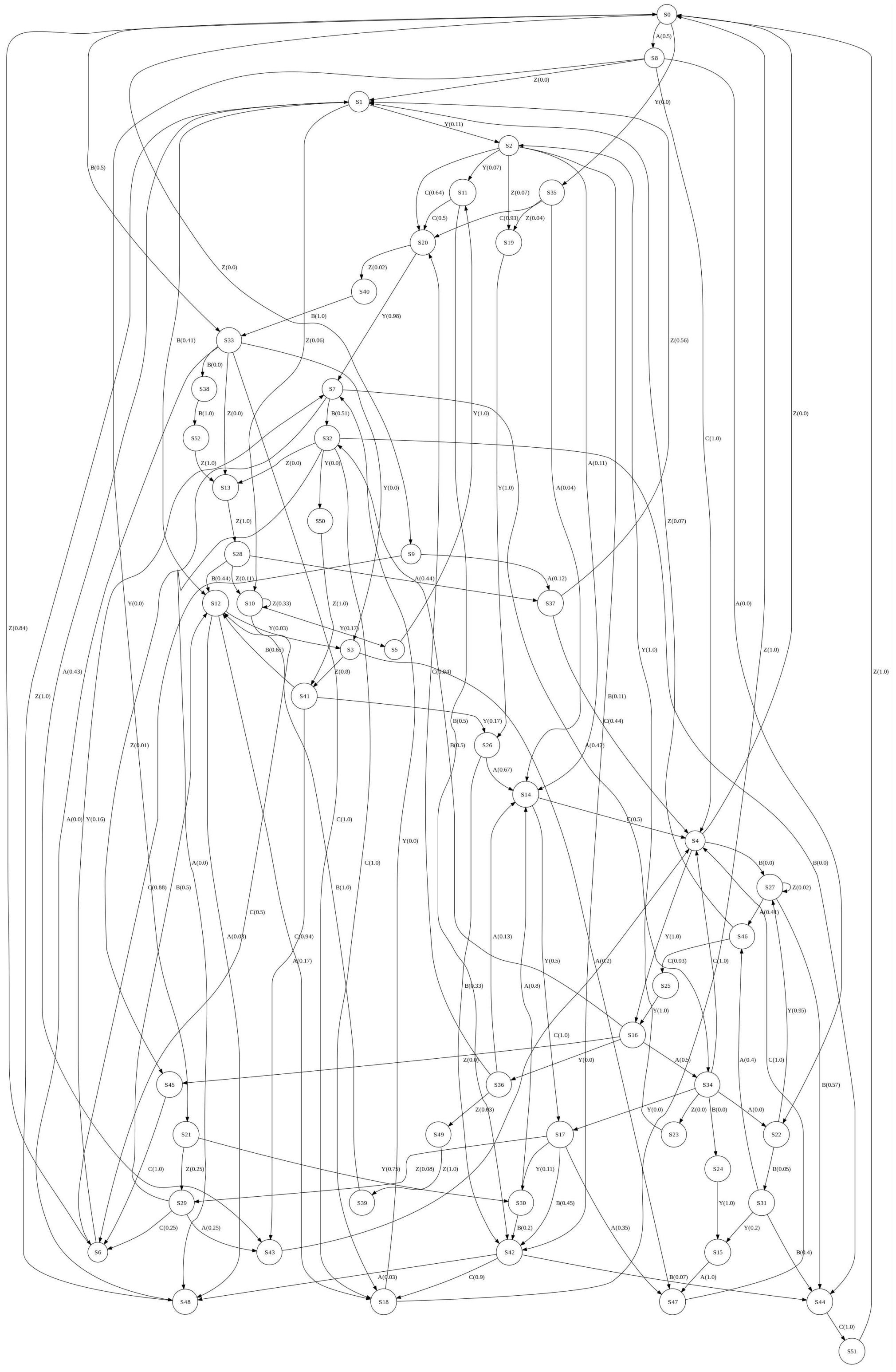


Figure 3.10: Model that is reconstructed from first 200,000 packets of captured data

and transitions to be added to the model. All of these new events are very low probability, which results in a lower minimum asymptotic state probability for each new set. This lower probability causes the confidence test to increase the amount of data required.

Even after capturing a million packets, the required amount from the confidence test continues to increase. The smallest asymptotic state probability and corresponding result from the confidence test are plotted against the number of packets captured in Figure 3.11. The steady increase suggests we will not easily capture enough data to rebuild the model confidently.

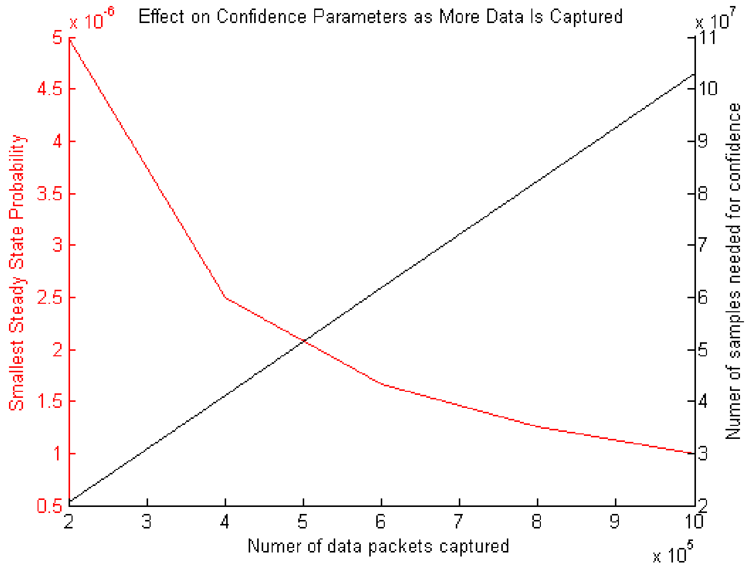


Figure 3.11: Plot of model confidence results as more data is captured

As discussed in Section 3.6.1, the alternative is to prune the low-probability states and transitions from the model. The generated model using all million packets is shown in Figure 3.12. The model contains 79 states and 274 transitions. Analysis of the asymptotic state probabilities shows a large gap between 71 of the states and the other eight. The 71 have probabilities below 0.06%, while the other eight have probabilities above 8.2%. That is a break of over two orders of magnitude.

The difference between the states generated by the actual underlying process and the states generated by glitches and noise is obvious. This separation makes a good pruning threshold. Following the pruning process, the model in Figure 3.13 results with a threshold of 0.01 (or 1%). By manually tracing the model, we find that it is essentially the same as the original model of Figure 3.9.

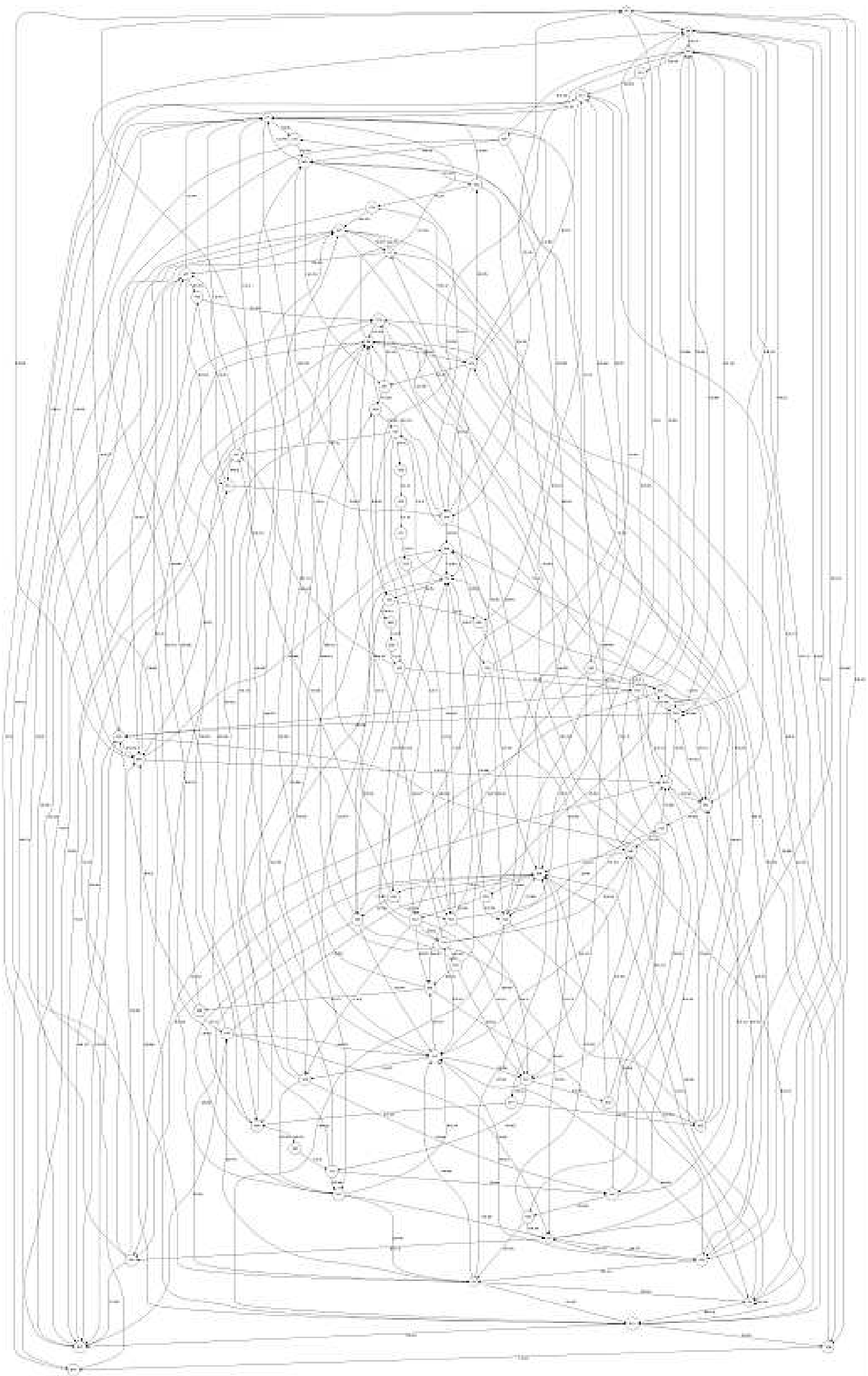


Figure 3.12: Model that is reconstructed from all million packets of captured data

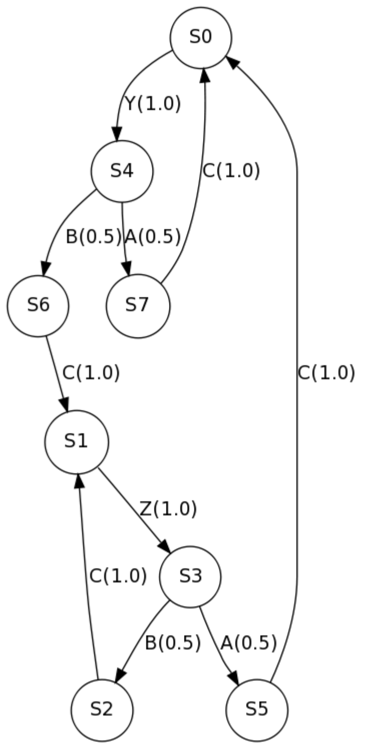


Figure 3.13: Result after pruning low-probability states and transitions

3.8 Summary

In this chapter we described the model construction part of our attack and specified a process to build HMMs from data samples of a network protocol. These models are to be built beforehand from labeled training data in order to provide a library of models for use in protocol detection. A proof-of-concept experiment on our private Tor network showed that a model could successfully be reconstructed from inter-packet timings. Additionally, we discussed how to handle problems with variable latency that would disrupt model construction. Pruning the reconstructed model successfully showed that underlying original model was still captured, but had just been obscured by noise.

Chapter 4

Detection

Once a model has been constructed from a set of training data, it can immediately be used for detection. Determining confidence interval bounds on each transition in the model will allow us to know how likely it is that the data can be explained by a model. This method will detect the protocol in a Tor connection. To detect the path, we go one step further once we have two captures that are matching to the same model with confidence intervals. We trace the state sequence for each of the two captures and say they are the same connection if the state sequences match.

4.1 Confidence Intervals

For the reasons described in Section 2.3.3, confidence intervals provide a better alternative to maximum likelihood when determining whether data matches a model. To compute confidence intervals, the data to be matched is traced through the model and the interval bounds are estimated using frequency count for each transition.

We track two counters for each transition, $c_{i,j}$ and c_i. c_i is the number of times that we have the opportunity to take the transition, and $c_{i,j}$ is the number of times we actually take it. In terms of the model in Figure 4.1, c_0 is the number of times that we are in state S0 while tracing our data. $c_{0,1}$ is the number of times we take the A transition to S1 and $c_{0,2}$ is the number of times we take the B transition to S2.

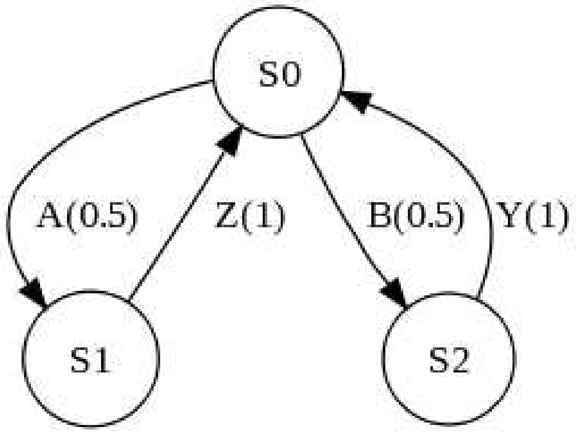


Figure 4.1: A model used demonstrate the calculation of confidence intervals

These counters are used to calculate three values for each transition: an estimate of the transition probability, and the upper and lower bounds of the confidence interval. The estimated transition probability is simply Equation 4.1.

$$\hat{p}_{i,j} = \frac{c_{i,j}}{c_i} \tag{4.1}$$

For Figure 4.1, we would expect to take A about half of the time and B about half of the time. This means that each $c_{0,1}$ and $c_{0,2}$ would be half of c_i.

The confidence interval bounds are calculated by Equation 4.2.

$$p_{i,j} \pm Z_{\frac{\alpha}{2}} \sqrt{\frac{p_{i,j}(1 - p_{i,j})}{c_{i,j}}} \tag{4.2}$$

Notice that p is used here and not $\hat{p}$. This means that the intervals are always calculated using the transition probability in the original model, not the estimated one. Also, Z is taken from the Normal distribution. The α is typically 0.05, meaning that we desire 95% confidence intervals. This is another way of saying that the probability of making an error by our decision boundaries is 5%.

After all of the data in the test set has been traced through, if the estimated transition probability is within the confidence interval, we have found a match. This means that the data is consistent with the model since the value estimated from the data fit within the bound on the model's parameter.

4.2 Protocol Detection

The flowchart in Figure 4.2 describes the process used for protocol detection. As input, the process requires a library of models and a test sequence of data. Three parameters are also provided for input that can be used to tune the results: window size, confidence interval percentage, and rejection rate.

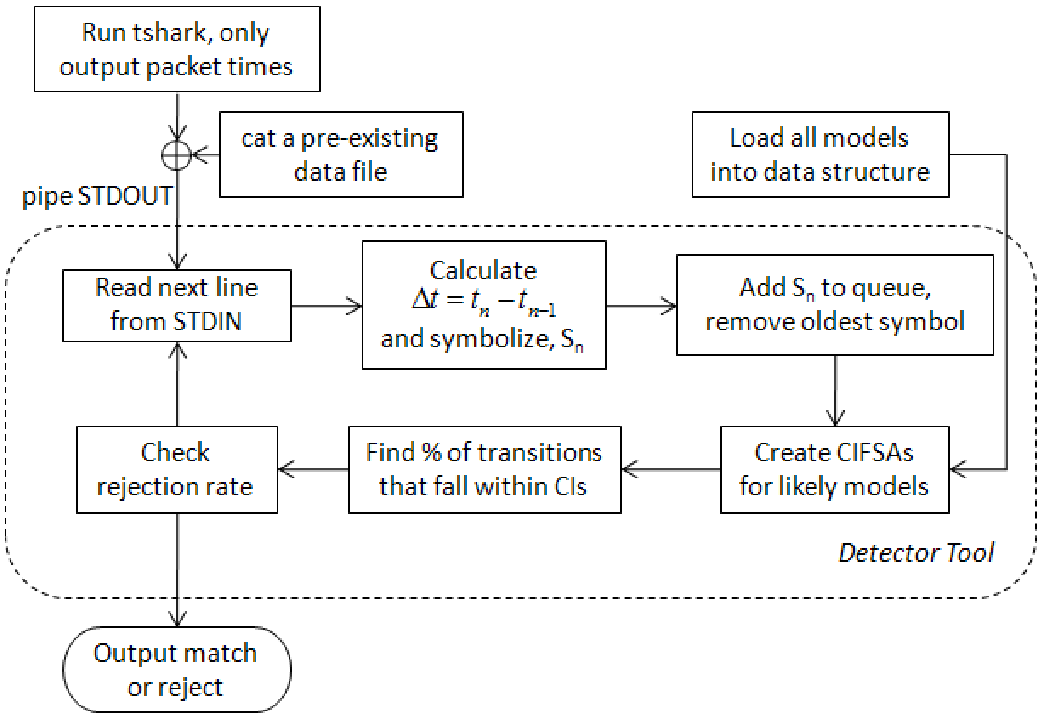

Run tshark, only output packet times
cat a pre-existing data file
pipe STDOUT
Load all models into data structure
Read next line from STDIN
Calculate $\Delta t = t_n - t_{n-1}$ and symbolize, S_n
Add S_n to queue, remove oldest symbol
Check rejection rate
Find % of transitions that fall within CIs
Create CIFSAs for likely models
Detector Tool
Output match or reject

Figure 4.2: Flowchart describing the protocol detection process

We will sometimes refer to the library of models as a bootstrap tree. It is essentially just a data structure that has functions for efficiently determining which models contain the symbols in our test data. This structure and the models within it must be compiled prior to performing the detection process.

The test data sequence, however, can all be captured and the detection performed offline at a later time, or it can be done online in real-time as data samples are received. The flowchart works for both modes. If in offline mode, the data file is read line by line as input to the process. And in online mode, a packet capture program like tshark can pipe the latest packet time to the STDIN of the detector program. It is even possible to SSH to another system and have it run the tshark capture program on that system, returning all of the STDOUT data back to your terminal, which can then be piped to the detector program. Many options exist for getting the test sequence into the detector, regardless of the mode in which it is used.

The first step that is taken after reading in a new data sample is to compute the delta and symbolize it. A separate variable holds the value of the last data sample (the time of the last packet). This is subtracted from the incoming packet's time, leaving the delta. If it was the first packet seen, the previous time variable is populated and the rest of the detection steps are skipped; we must have at least two packets before we actually get our first symbol of data. To get the symbol, the delta value can be translated by looking up the appropriate range for it in a table. We only work with the symbols, so the packet time and the delta can be discarded after symbolization.

The next step is window management. The window is a sliding window with the latest w data samples. The window size parameter w is one of the three user-specified tuning parameters mentioned earlier. When a new packet comes in, it is pushed on to the end of the data structure for the window. If the size of the window data is greater than the specified window size w, then the oldest element is popped off the front of the queue.

Once it is full, the window is used to calculate the confidence intervals. The bootstrap tree returns all of its models that will accept the symbols in the window, and a model with confidence intervals on the transitions is created for each of them.

Out of all the transitions in each model, a certain percent of them will have an estimated probability that falls within the confidence interval. This percentage tells us how likely it is that that model represents the test data. If this percentage is above the user-defined rejection rate, we can say that it is a match. Note that there may be more than one matching model at each window. At this point, a tie-breaker can be used to select just one, whichever has the highest percent of matching transitions.

The user receives the name of the model(s) that matched the data and their match percentages as output. Note that this whole process is just for one window. As each sample arrives, the window slides forward and the whole process is repeated. The process ends when there are no more incoming data samples or when it is stopped by the user.

4.3 Viterbi Path

While performing protocol detection, the window of data is traced through the model to calculate confidence interval bounds. As an additional step, we can store the list of states that are visited as the data is traced. This sequence of states is known as the Viterbi path.

More officially, the Viterbi path is the most likely path that a sequence of data takes through a model [51, 24]. In our models, even though the definition is the same, calculating the Viterbi path is very trivial. This is because there is no probability involved in tracing the path. There will always only be a single path through the model for any given data sequence and start state. As mentioned before, this is due to the fact that all of the models created by CSSR will be deterministic.

Given any sequence of data and a model, the path can be determined by Algorithm 1. The algorithm does not need to know the starting state of the model. It first searches the model for all possible start states by looking for an outbound transition matching the first symbol of data. From there, the data is traced from each possible start state until a point arises where there is no possible transition for a symbol, or until the data has been fully traced through the model.

Algorithm 1 Get the Viterbi path for a given a data sequence

```
for each state in the model do
  if state has transition == $data[0]$ then
    push state onto vector $V$
  end if
end for
if $V$.length == 0 then
  return false
end if
for each state $S$ in $V$ do
  set state of model to $S$
  for each symbol $p$ in $data$ do
    if state does not have transition == $p$ then
      break
    end if
    advance the model
    push state id onto vector $path$
  end for
  if $path$.length == $data$.length then
    return $path$
  end if
end for
return false
```

Algorithm 2 adds window management of two data sets to the functionality of Algorithm 1 to generate detection rates. The algorithm takes two data sets and a window size as input, and returns the number of windows that either matched, got rejected, or were accepted by the model, but did not match up. A "match" should occur if the two data sets are from the same instance of the same process. An "accepted, but no match" should occur if the two data sets are from the same process, but not the same instance of it. A "rejection" should occur if the data sets are from different processes.

Algorithm 2 Perform matching using Viterbi path

```
Require: $data1$.length == $data2$.length
  initialize two queues, $w1$ and $w2$
  add $data1$[ 0:window_size–2 ] to $w1$ and $data2$[ 0:window_size–2 ] to $w2$
  for $i$ = window_size–1 to $data1$.length do
    append $data1[i]$ to $w1$ and $data2[i]$ to $w2$
    if queue size > window_size then
      pop off the front (oldest element)
    end if
    using Algorithm 1, $path1$ = viterbi($w1$) and $path2$ = viterbi($w2$)
    if ($path1$ or $path2$) == false then
      Output: rejected
    else if $path1$ == $path2$ then
      Output: matched
    else
      Output: accepted, but no match
    end if
  end for
```

4.3.1 Model Path vs. Tor Path

Sometimes, the use of the word "path" can be somewhat confusing. This is because there is a path through the model and a path through the Tor network. The path through the model, the Viterbi path, is the unique sequence of states visited as a result of following the transition symbols. This only applies to one of our hidden Markov models and is completely separate from Tor.

The path through the Tor network is the sequence of onion routers, or relays, that proxy a stream through Tor between the sender and the receiver. There are by default three relays in a circuit which define the path taken through the network.

The connection between the two is that it is possible to determine the Tor path by matching up the Viterbi paths taken by data captured at various nodes in the network. If we take on the role of the global observer over a Tor network, we can capture traffic from all of the relays. At this point, it will be possible to determine which three relays are involved in a user's Tor circuit.

4.4 Path Matching Experiment

To test the path matching process, we devise another experiment using the client and server test programs. The client connects to the server, which sends packets based on the five state model in Figure 4.3. The process runs among two systems that are connected to the Tor network, as is shown in Figure 4.4.

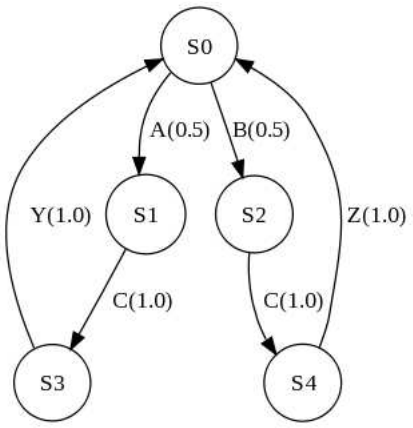


Figure 4.3: Five state model used for path matching experiment

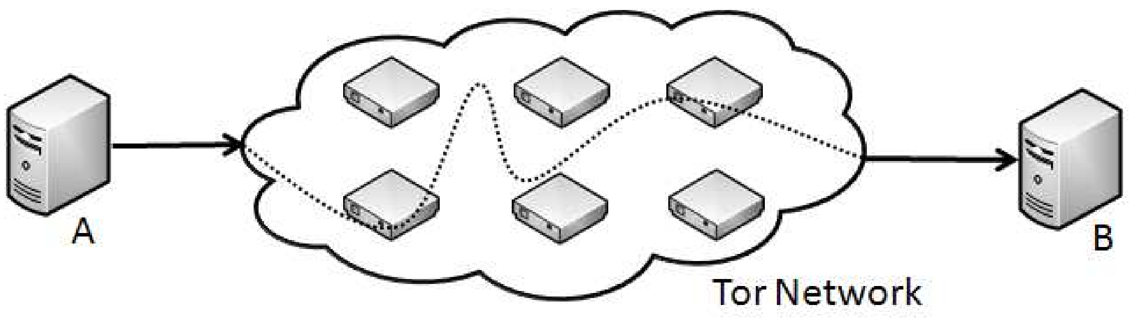


Figure 4.4: Diagram of path matching experiment showing Tor network

Instead of capturing training data and building a model from that, we will just use the original model for matching with new test data. If we had captured new training data, our model construction process would be exactly the same as that in Section 3.7; the same process is used to generate that model.

To get the test sets, packet captures are run on both systems, A and B. This yields two data sets of the same instance of the same process, but from different prospectives. The paths for these two sets should match up since they are the same instance of the same process. Results of matching are shown in Table 4.1.

	Detection Rate	**Rejection Rate**
$w = 10$	97.23%	2.77%
$w = 20$	94.66%	5.34%
$w = 30$	92.12%	7.88%
$w = 40$	89.79%	10.21%
$w = 50$	87.51%	12.49%

Table 4.1: Results for path matching experiment

4.5 Flow Correlation

An alternative approach to detection is to use the flow correlation method [25, 26]. Basically, a distance metric is calculated between a raw time series and all of the other data sets. The one with the smallest distance value is declared the most likely to be the match. Note that this method requires a comparison of all system on the network, not just any two. The necessary steps for computation are described in Algorithm 3.

There are two correlation measures [26] which can be used: a time-spectrum approach and a frequency-spectrum approach based on a matched filter. We select the frequency-spectrum approach. After converting the time data into pattern vectors, we calculate the discrete fast Fourier transform [56] of the pattern vectors. Then, using the following equation, we compute the distance

Algorithm 3 Perform matching using flow correlation with windowing

```
Require: all other data sets are the same length as $data1$
  for $i = 0$ to ($data1$.length - window_size) do
    for each data set $j$ other than $data1$ do
      partition $data1$ and $j$ into subsequences of time length $T$
      for each subsequence, $k$ do
        store $k$.length into pattern vector
      end for
      perform the fast Fourier transform on each pattern vector, $X_f = \mathrm{fft}(X)$ and $Y_f = \mathrm{fft}(Y)$
      calculate the distance measure using Equation 4.3
    end for
    select match for this window as the argmin of the distance measures
  end for
```

measure, which records how much difference there is between the data sets.

$$d_{x,y} = \frac{\sqrt{\langle Y_f, Y_f \rangle}}{\langle X_f, Y_f \rangle} \tag{4.3}$$

4.5.1 Advantages of Flow Correlation

There are distinct differences in the two approaches of Viterbi path matching and flow correlation. The flow correlation method is a direct comparison of the time series, whereas our path matching method first symbolizes the data and matches it to a training model. The initial requirements of our method, a training model and window size, are not required by flow correlation.

It is an advantage for flow correlation that it does not require a model to be constructed from training data prior to matching. It directly compares the time sequences without first translating the problem to a model representation of an underlying protocol. This could be an advantage in the sense that it is easier to implement and there are less factors with which to contend.

One such factor is the window size. With flow correlation, the entire data set can be directly compared. The implementation in Algorithm 3 only involves windowing so that it will be easy to compare the results with values from path matching. Without windowing, all the data available is used to do the correlation, which increases the accuracy of the measure. In our matching, we cannot use all of the data because it becomes more and more likely that a glitch will disrupt the path.

4.5.2 Disadvantages of Flow Correlation

Flow correlation also has its disadvantages. The most notable is that it requires the adversary to be a global observer. He must be capturing data on every Tor node in the network and correlating them all together to find the other end with the lowest distance value. It is very unlikely that this would happen given the size of the Tor network today.

Even if the scope of the correlation is reduced, it is more difficult to run the attack online in real-time. Each node in the correlation set needs to have an active packet capture feeding the timing data back to a central server which can perform the correlations amongst all of the data sets. Whereas, with the Viterbi path matching, an adversary need only watch two nodes using the network to get an idea of the likelihood that they are communicating.

Windowing can also be a disadvantage depending on the usage conditions. If only short time periods are available for test data, the correlation will be less accurate at the smaller data amounts. With windowing, path matching is already set to handle these small amounts of test data.

4.6 Summary

In this chapter, we described the protocol and path detection processes that make use of the models generated by the work in the previous chapter. Our implementation of these processes was detailed and an experiment was conducted with them to make sure they worked correctly. Our detection procedures are unique from more traditional approaches because they do not rely on maximum likelihood to select the best match out of a set. We only need test data and a single model to determine a statistical measure of how well that model represents the data. For path matching in the Tor attack, we only need two systems both using Tor to tell if they are communicating together or with another party.

Chapter 5

An Illustrative Example

This chapter presents an illustrative example covering all of the techniques described in Chapters 3 and 4. We start with a network process and go through all of the necessary steps to be able to perform detection. We show results for the detection and compare them with results from the flow correlation technique.

5.1 Experimental Setup

5.1.1 Layout

This experiment uses the same private Tor network used by previous experiments, except this time, four systems will be involved. We designate two pairs of systems to communicate on the Tor network at the same time. Each pair of systems is communicating between themselves and are running the exact same process.

The diagram in Figure 5.1 shows the layout of the systems used in the experiment. The system 192.168.69.10 will communicate with 192.168.69.13, and 192.168.69.11 will communicate with 192.168.69.12. Each of their connections goes through Tor and we do not record which relays were being used. We will need to use path detection in order to determine which two computers are talking.

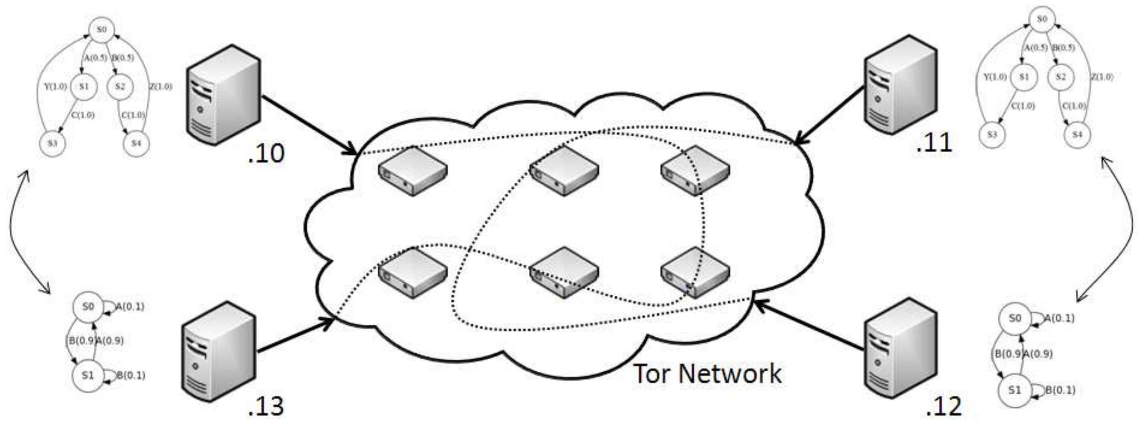


Figure 5.1: Layout of the ping-pong experiment

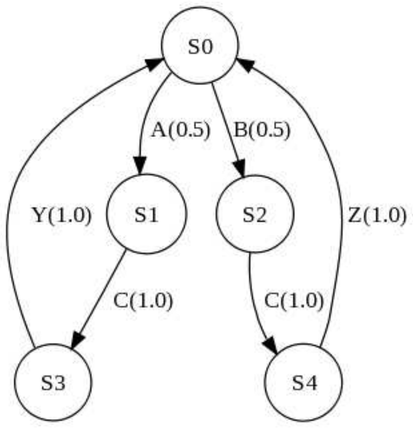
5.1.2 The Ping-Pong Process

The collective process that each pair is running is referred to as the “ping-pong” process. In this process, the client end is no longer just a passive listener. Each party runs both a client and a server thread that can communicate together inside the computer. One of the two systems is designated as the master, which starts the communications, while the other, the slave, waits for the master to start the process.

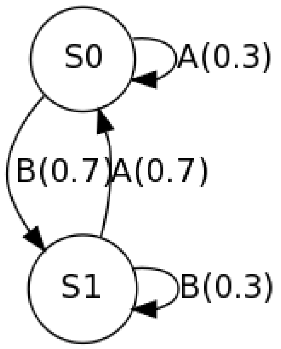
The master begins at state zero and takes a transition in its model. The model used by the master is shown in Figure 5.2a. It then looks up the associated delay, in milliseconds, for the symbol it generated and sends the packet after that amount of time has passed. The symbol-to-delay translation table used by the master is Table 5.1a.

Once the slave receives the packet from the master, it also starts at state zero in its model and takes a random transition. The model used by the slave in this experiment is shown in Figure 5.2b. The slave has its own symbol-to-delay translation table to convert symbols from its model into time delays. The table used by the slave is Table 5.1b.

The process continues back and forth until the required amount of packets have been generated. The delays of the packets leaving the master have two components: the delay added by the master and the delay added by the slave. When we symbolize, we have to take this into account and break the range out into ten symbols. There is a symbol for each of the five symbols in the master’s model combined with either of the two symbols from the slave’s model.



(a) 5-state model used by the master



(b) 2-state model used by the slave

Figure 5.2: Models used in the ping-pong experiment

Symbol	**Delay** (in milliseconds)
A	200
B	400
C	100
Y	300
Z	500

(a) Delays used by master

Symbol	**Delay** (in milliseconds)
A	10
B	50

(b) Delays used by slave

Table 5.1: Symbol-to-delay translation tables for ping-pong experiment

5.2 Data Capture

Once again, data is captured using tshark. In order to see all of the data passing through the network stack at each end, the exit node functionality has to be disabled for any systems in the experiment. This is because Tor is smart enough to know when a destination is running a Tor relay. It will deliberately choose that relay as the last node in the circuit (the exit node), so that the connection to the final destination port occurs locally and does not need to go across the network.

This is good for privacy, because it reduces the chance of an adversary spying on the last hop of the connection, which is usually unencrypted. But it is bad for us because tshark would have to watch all traffic entering the Tor relay vice to a separate port in order to see the traffic.

On the other end of the connection, we use a small program called 3proxy [57] to create a local TCP port forward. Any data sent to the port is automatically forwarded along to a Tor SOCKS proxy for entrance to the Tor network. This is a very useful tool for programs that do not have SOCKS support, such as with our small client / server test programs.

5.3 Symbolization

To symbolize, the numerical derivative of the CDF (the PDF) is plotted to determine the ranges. We know that we are looking for ten symbols. Figure 5.3 shows the plot with ranges overlaid by the dotted lines.

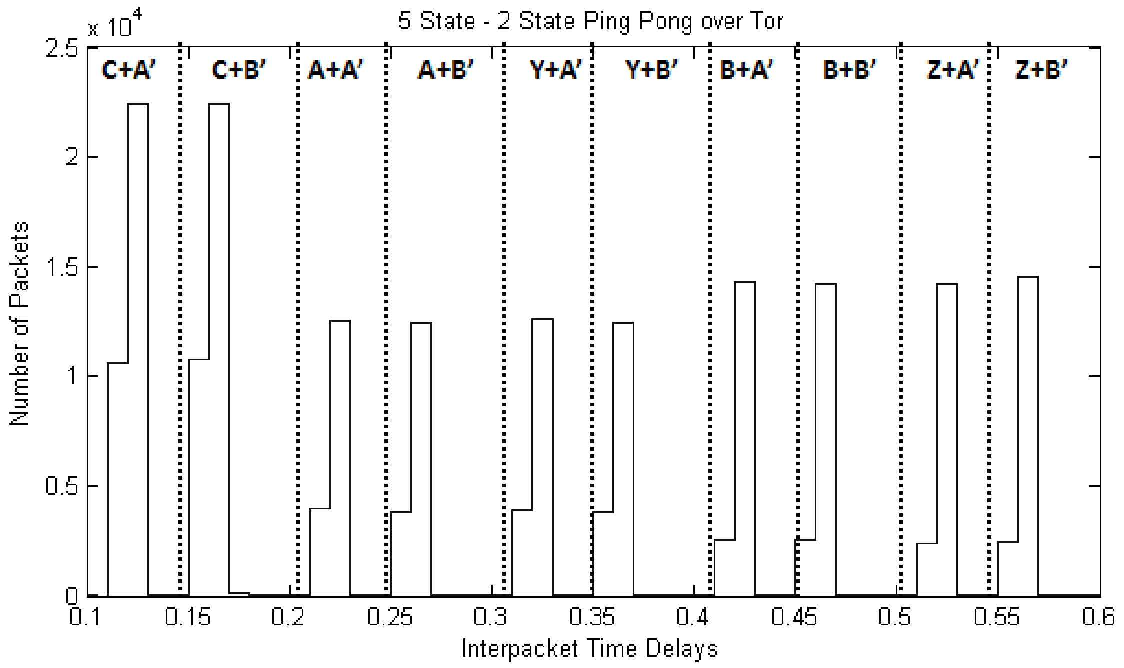


Figure 5.3: Plot of PDF of data with selected ranges and symbols overlaid

The combined symbolization table contains ranges to convert delays into one of ten symbols. There is a symbol to represent each of the ten possible combinations of symbols from the five state and two state models. Table 5.2 shows the symbolization table we derive from the plot analysis.

Symbol	**Range** (in seconds)
C+A’	{0.0000,0.1500}
C+B’	{0.1501,0.2100}
A+A’	{0.2101,0.2500}
A+B’	{0.2501,0.3100}
Y+A’	{0.3101,0.3500}
Y+B’	{0.3501,0.4100}
B+A’	{0.4101,0.4500}
B+B’	{0.4501,0.5100}
Z+A’	{0.5101,0.5500}
Z+B’	{0.5501,10.0000}

Table 5.2: Ranges for all ten symbols of ping pong experiment

5.4 Training Model Construction

As data is captured in increments of 200,000 packets, CSSR is used to construct a model at each step. By running the confidence test at each successive interval, we find that we have the same problem described in Section 3.7. As the data used for model construction increases, the smallest asymptotic state probability decreases and the amount of data required by the model confidence test increases. At a million packets captured, the confidence test tells us we need over 288 million packets. We cannot capture enough data because of the glitches.

Using the same approach from Section 3.7, we stop capturing after a million packets and just keep the model constructed with that amount of data. We can later prune out states and transitions that have a very low probability of occurring. The model reconstructed from a million packets is too large to include as a figure. It has 425 states and 1,523 transitions—over five times as large as the model shown in Figure 3.12.

For comparison purposes, we also use artificial data to generate the combined model. Artificial data is generated from the 5 state and 2 state machines and is concatenated together to get data for the combined model. In other words, an A from the 5 state model and a B from the 2 state model yield an A+B’ symbol for that sample in our concatenated data set.

CSSR with $L = 3$ is used to construct the model shown in Figure 5.4. By manually comparing it to the models from Figure 5.2, we find that it is the correctly combined model.

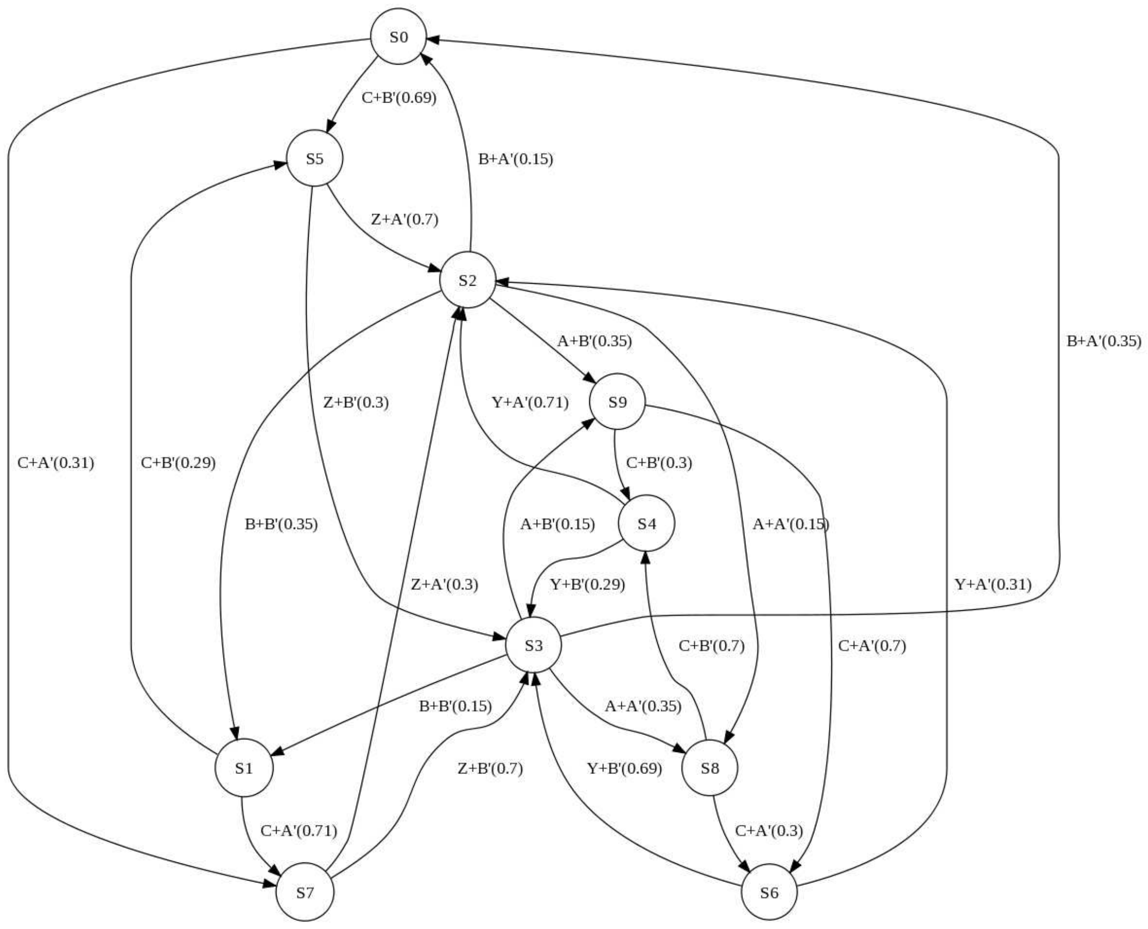


Figure 5.4: A combination of the 5 state and 2 state models, generated from artificial training data

5.5 Pruning

To select a good value for pruning, a histogram is made of the asymptotic state probabilities of the reconstructed model. The histogram is shown in Figure 5.5. From looking at the histogram, it is obvious that a large majority of the probabilities are very small. In fact, 378 of the 425 states have an asymptotic state probability below 0.02% while the other 47 states have a probability above 0.3%. We will threshold in this gap.

We choose $\beta = 0.0005$ as the threshold value and prune the model of any states with asymptotic state probabilities below this value. The resulting model is shown in Figure 5.6. The pruned model is considerably reduced in size from the original model, but still contains 99.3% of the information from that model. This is because the original model would use the majority of its

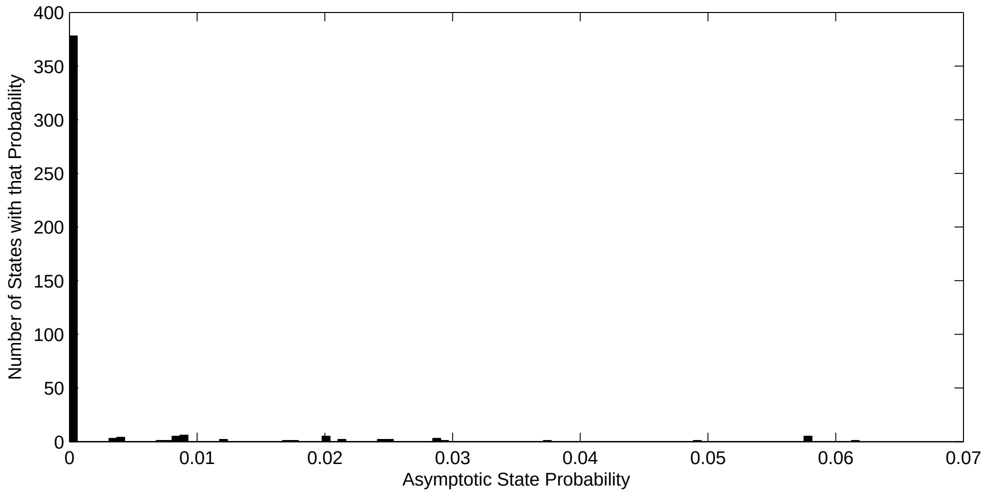


Figure 5.5: A histogram of the asymptotic state probabilities

states less than 0.7% of the time.

If we run the model confidence test on the pruned model, we find that only 27,990 symbols are needed to make a confident model. Since we have a million symbols, there is more than enough data to be confident in the remaining states and transitions after pruning.

It is possible to prune the model too aggressively. If this is done, it eliminates some of the important states and takes away paths for valid data. For instance, at $\beta = 0.013$ the model, shown in Figure 5.7, now only contains the most common events in the original process. It is missing transitions for all of the symbol patterns that can occur in the original model. This will cause an excessively high rate of rejections when performing detection, which is confirmed in the next section.

5.6 Detection Results

After constructing the models, confidence interval and Viterbi path detection was performed against the systems using Tor. Recall from Section 5.1.1 that .10 and .13 are communicating together and .11 and .12 are communicating. The first three octets of the IP addresses (192.168.69) are omitted to save space in the tables.

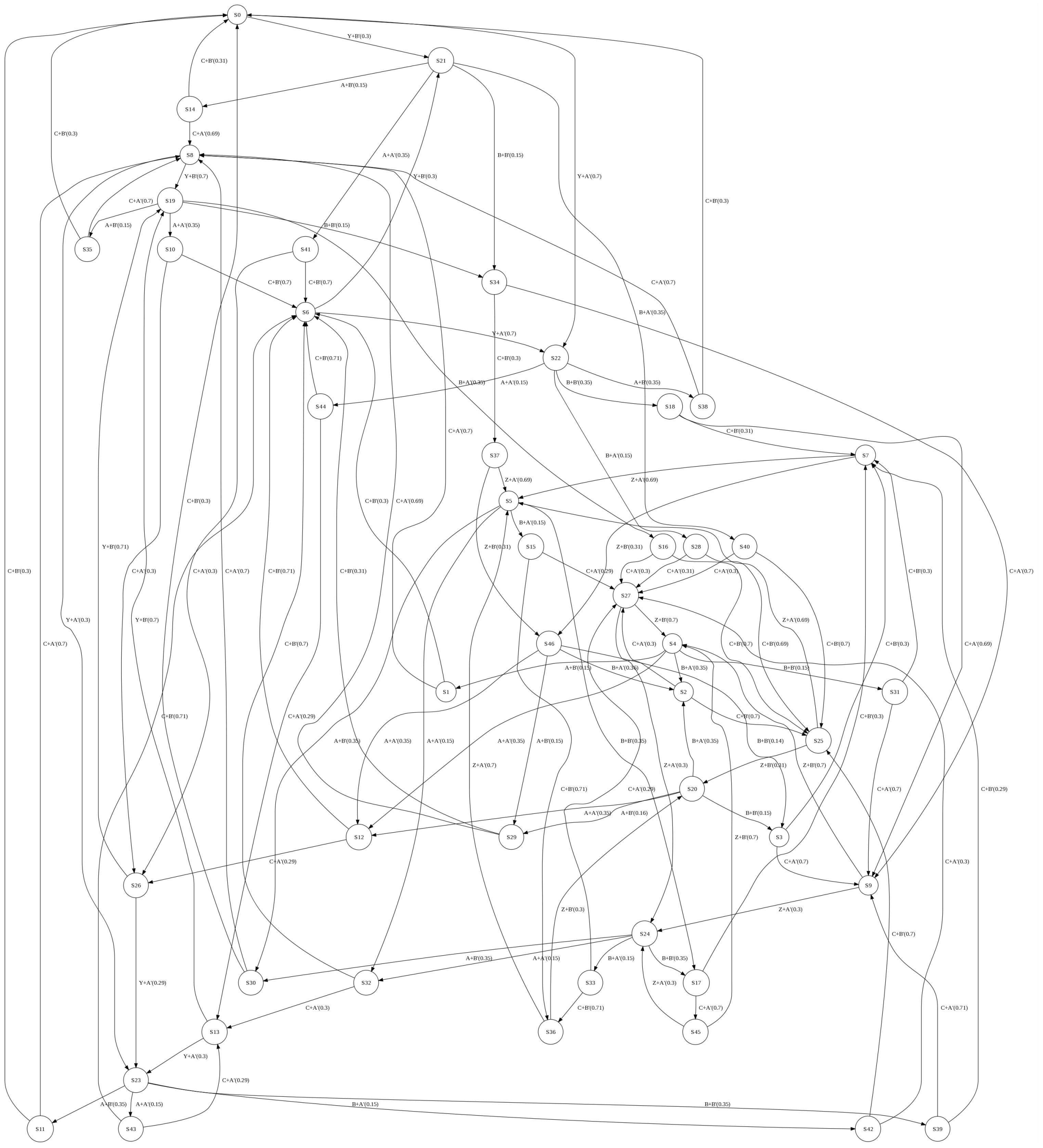


Figure 5.6: The training model pruned with a threshold of $\beta = 0.0005$

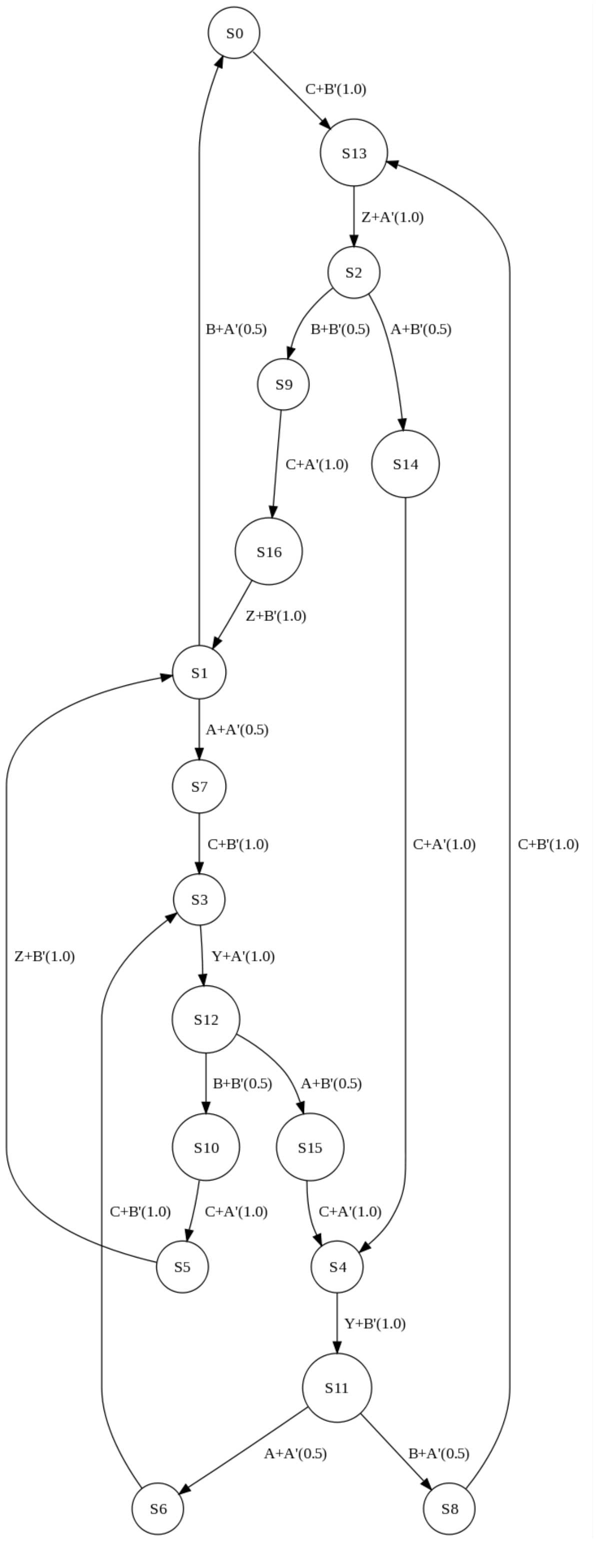


Figure 5.7: The training model pruned with a threshold of $\beta = 0.013$

5.6.1 Detection Rates

Detection rates are compiled for path matching using various models. Specifically, we present results for the following models:

- **Table 5.3**: Original, combined model from Figure 5.4
- **Table 5.4**: Reconstructed model, no pruning
- **Table 5.5**: Reconstructed model pruned at $\beta = 0.0005$ (shown in Figure 5.6)
- **Table 5.6**: Reconstructed model pruned at $\beta = 0.013$ (shown in Figure 5.7)

	.10	**.11**	**.12**
.11	0.02%		
.12	0.02%	93.78%	
.13	95.24%	0.02%	0.02%

(a) Window size = 10

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	86.98%	
.13	90.62%	0.00%	0.00%

(b) Window size = 25

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	72.51%	
.13	78.44%	0.00%	0.00%

(c) Window size = 50

Table 5.3: Detection rates for system pairs using *original model*

The rates in each table are the percentage of the windows that have matching paths when both windows are accepted by the model. In other words, they represent the ability of an adversary to correlate Tor connections. The higher the rate, the more often it is that the process works as expected. This means that each window is accepted by the model (that it does not contain any events which the model cannot handle), and that the Viterbi paths of the two windows match exactly.

In the results for each model, larger window sizes have lower true positive rates. This is because a larger window allows a glitch to disrupt the path matching for a longer period of time. New data must come in to slide the window along and eventually flush out the glitch. This takes longer when the window size is large and so the detection rates are lower.

	.10	**.11**	**.12**
.11	0.02%		
.12	0.02%	95.88%	
.13	96.66%	0.02%	0.02%

(a) Window size = 10

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	90.36%	
.13	92.20%	0.00%	0.00%

(b) Window size = 25

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	82.08%	
.13	85.46%	0.00%	0.00%

(c) Window size = 50

Table 5.4: Detection rates for system pairs using *reconstructed model*

	.10	**.11**	**.12**
.11	0.02%		
.12	0.02%	94.40%	
.13	95.99%	0.02%	0.02%

(a) Window size = 10

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	86.98%	
.13	90.62%	0.00%	0.00%

(b) Window size = 25

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	75.99%	
.13	82.62%	0.00%	0.00%

(c) Window size = 50

Table 5.5: Detection rates for system pairs using *reconstructed model pruned at* $\beta = 0.0005$

	.10	**.11**	**.12**
.11	0.01%		
.12	0.01%	3.79%	
.13	3.85%	0.00%	0.00%

(a) Window size = 10

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	0.02%	
.13	0.02%	0.00%	0.00%

(b) Window size = 25

	.10	**.11**	**.12**
.11	0.00%		
.12	0.00%	0.00%	
.13	0.00%	0.00%	0.00%

(c) Window size = 50

Table 5.6: Detection rates for system pairs using *reconstructed model pruned at* $\beta = 0.013$

Out of the window sizes shown, $w = 10$ provides the most optimum result, i.e. is closest to the point (1,0) in a receiver operating characteristic curve. This value gives the largest amount of true positives while limiting false positives.

We can see from the values in Table 5.4 that the reconstructed model which is not pruned is the best for matching the data. Using this model in practice, however, is extremely difficult as the numerous states and transitions cause a large increase in the processing time for the detector.

The model that is pruned at $\beta = 0.0005$ provides a good compromise between higher detection rates and lower processing time. Its results are slightly worse than the reconstructed model, but are better than using the original model.

The over-pruned model does not fare well. Its detection rates approach 4% in the best of the test cases. This is reasonable because paths that are taken reasonably often are stripped from the model. As confirmed by the results, the model is rejecting the vast majority of the windows.

5.6.2 Rejection Rates

There are three possible decisions the detector can make for each pair of windows:

1. That they are both accepted by the model and their Viterbi paths exactly match
2. That they are both accepted by the model, but their Viterbi paths do not match
3. That one or more of the windows are rejected by the model

A rejection means that there is an event in the window that cannot be explained by the state structure and transitions of the model. It also means that confidence intervals cannot be calculated from the window.

Ideally, a rejection only happens if the two processes from which the data windows came are different. If the processes are the same, like in this experiment, a rejection should not occur. However, in practice extra latency results in glitches that cause a symbol to be misclassified. Because of this, it is important that we keep rejection rates in mind when selecting our model and window size.

Tables 5.7 and 5.8 show the rejection rates for the various combinations of window sizes and available models. Table 5.7 shows the case of matching the .10 data to data from .13 and Table 5.8 shows the case of .11 and .12.

	Original Model	**Recon. Model**	**Pruned at** $\beta = 0.0005$	**Pruned at** $\beta = 0.013$
$w = 10$	3.83%	1.48%	3.06%	96.15%
$w = 25$	7.23%	3.63%	7.23%	99.98%
$w = 50$	17.87%	7.07%	13.48%	100.00%

Table 5.7: Rejection rates for .10 – .13

	Original Model	**Recon. Model**	**Pruned at** $\beta = 0.0005$	**Pruned at** $\beta = 0.013$
$w = 10$	5.00%	1.94%	4.31%	96.21%
$w = 25$	10.14%	4.85%	10.14%	99.98%
$w = 50$	22.89%	9.48%	18.96%	100.00%

Table 5.8: Rejection rates for .11 – .12

Again, the best results are obtained with the reconstructed model and a window size of 10. The reconstructed model still rejects some windows because new glitches are seen during collection of the test data that are not seen while collecting the training data used to build the model. Since they are not seen in the training data, the model cannot account for them and rejects those windows.

These figures allow us to see that for the model pruned at $\beta = 0.013$, all of the false negatives are caused by rejections, rather than mismatching Viterbi paths. This is interesting, because there may be a way to still use this model if the rejection rate could be improved. We discuss an idea for this in Section 6.2.

5.7 Comparison to Flow Correlation

We now compare the results of the flow correlation method from Section 4.5 with our Viterbi path matching method for this experiment.

There are some fundamental differences in the two methods. First, flow correlation is a maximum likelihood approach. Because of this, it chooses the combination of data sets that is most likely and classifies them together. Even if none of the sets are anything like the others, it will still choose the best relationship. This means that we get a result for every window and that there are no rejections like with the confidence interval method.

Second, flow correlation does not detect protocols, it merely matches up the timings after they have been converted into pattern vectors. Third, there is an additional parameter—a period T. T controls the bin size for the packet counter when the time vectors are converted to pattern vectors. If $T = 2$, then the elements of the pattern vector contain the number of packets with times $t < 2$, $2 \leq t < 4$, $4 \leq t < 6$, etc. The selection of T can have a large impact on the results.

These differences require that we present the results in a different manner. We still window the data to maintain consistency across the two methods, but there are no detection or rejection rates for each combination. Instead, we take data from one node and compare it to all of the other nodes. The value recorded is the true positive rate, which indicates that the flow correlation algorithm correctly chose the other node.

Table 5.9 shows the true positive rates for .10 and .13, while Table 5.10 shows the rates for .11 and .12. Any other match or tied value is considered an incorrect classification. The tables show how the rate varies with respect to window size, w, and period size (bin size), T.

The best rate in the tables comes when the window size is highest and the bin size is smallest. The larger window size means using more data for the classification. This result is consistent with those of Zhu et al. They found that detection rates improved asymptotically to 100% as more and more data was used for classification [25, 26].

	$T = 0.5$ s	$T = 1$ s	$T = 2$ s
$w = 10$	84.33%	58.62%	67.44%
$w = 25$	99.51%	86.37%	75.86%
$w = 50$	99.94%	96.97%	78.97%

Table 5.9: True positive rates for .10 − .13

	$T = 0.5$ s	$T = 1$ s	$T = 2$ s
$w = 10$	86.78%	67.20%	67.53%
$w = 25$	99.64%	90.32%	84.66%
$w = 50$	99.98%	98.08%	86.84%

Table 5.10: True positive rates for .11 − .12

The higher window size actually works better for flow correlation because it does not rely on matching the data to a model first. This is both a good thing and a bad thing. The negative aspect is that we lose the ability to detect protocols, but the positive aspect is that we handle glitches better. In the Viterbi path analysis any incorrect symbol can disrupt the path match, making it better to keep the window sizes small in order to see the bad symbols as little as possible. With flow correlation, however, it is better to see as much data at once as possible so that other values can overwhelm the problematic ones in the distance calculations.

Given the choice, we actually prefer the smaller window sizes. This is because we get more granularity within the data sets and can detect a change in behavior more quickly. For example, if the data sets from computer A and B are matching, but all of a sudden computer A starts talking to computer D instead of B, we will notice the change more quickly with a smaller window size. This is because data from the first behavior clears out from the window more quickly since the window is smaller and has less data in it. Once a majority of the data in the window is from the new behavior, we will begin making the correct classification again.

The bin size T must be selected to provide the best balance of latency flexibility. If there is a great deal of variable latency in the connection, higher values of T would work better because those extra latencies would not move packets to a new bin. Lower values of T work best in this scenario because they give the most resolution. The local network environment does not cause a great deal of glitches relative to how many there would be on a connection on the public Tor network.

The latency adapting behavior of T is more noticeable in the smallest window size. In the smallest window size, any variable delay is emphasized more since there are less other symbols to regulate the averages. This is why the rate is actually slightly better for $T = 2$ than for $T = 1$ in both scenarios with a window size of $w = 10$.

The final tables, Table 5.11 and 5.12, show a side-by-side comparison of the detection rates for both methods. For the Viterbi path column, the results are when the pruned model ($\beta = 0.0005$) was used. And for the flow correlation column, the results are when $T = 1$ second. These parameter choices represent the middle ground for each method to provide a better basis of comparison.

	Viterbi Path	**Flow Correlation**
$w = 10$	95.99%	58.62%
$w = 25$	90.62%	86.37%
$w = 50$	82.62%	96.97%

Table 5.11: Method comparison of detection rates for .10 – .13

	Viterbi Path	**Flow Correlation**
$w = 10$	94.40%	67.20%
$w = 25$	86.98%	90.32%
$w = 50$	75.99%	98.08%

Table 5.12: Method comparison of detection rates for .11 – .12

5.8 Summary

The experiment showed that the processes outlined in Chapters 3 and 4 can be used to successfully model a protocol and use it to match Tor users. Model pruning, as long as the value of β was not too aggressive, worked very well for the detection process. Its smaller size allowed less computation time while maintaining some extra paths for the more common glitches. This resulted in a smaller rejection rate than the original model and more accurate detection results.

Chapter 6

Conclusions

6.1 Concluding Summary

In this work, we showed how recently developed pattern recognition tools could be applied to traffic analysis of anonymity systems. We examined their effectiveness on Tor specifically, due to its popularity and low-latency design. We showed how to use the tools to attack the anonymity and match the network streams of two or more systems.

A private Tor network was implemented on standalone network for data collection. Custom client and server programs using this network provided much of the experimental basis for this work.

It was found that the hidden Markov model of an underlying network process could be reconstructed from observations of the inter-packet timings of the packets. Measures were used to validate the models and check to make sure enough data was used for their construction. In cases when not enough data was available, the models could be used after a technique was applied to prune them of any low-probability states and transitions.

The pruning technique worked well and was validated by the results from detection. Using a pruned model in the detection step achieved results and runtime between the original model and reconstructed model without pruning. This was a good compromise when better results were desired over using the original model, but at a lower runtime than the reconstructed model without pruning.

By calculating confidence intervals and comparing the paths taken by test data through one of these models, we gained a statistical measure of how likely it was that two systems were talking through Tor. If two systems were both using Tor, but tunneling different protocols, the confidence intervals showed a rejection. If the two systems were both using Tor and the same protocol, the confidence intervals accepted the data. As an additional step, we found that if the Viterbi paths match, the two systems were using the same instance of the same protocol.

This type of analysis could allow an adversary to defeat the anonymity of Tor without being a global observer. If an adversary is able to observe two systems, these tools can be used to determine if that pair of systems are communicating through the anonymity network.

6.2 Recommendations for Further Research

This work inspired many interesting questions that could not be answered due to time constraints. We will list and quickly discuss them here as topics for future research.

Most interesting, and probably one of the most important follow-up questions is how well does the attack work on the real Tor network? Will the increased amount of variable delays overwhelm the pruning process and reduce detection rates? To answer these questions, the five to two state ping-pong experiment should be re-run over a wide-area Tor network to see if similar results can be achieved. Possible options for a larger Tor network could be the public, production network or maybe a private one on a distributed test environment like PlanetLab [58].

Also, most of the experiments done for this work involved artificially created protocols that followed the state machines we had given it. More research could be done with real-world protocols to see if they can be modeled and detected using the same process. Good candidates would include multiplayer games or Voice over IP (VoIP). It is important to note though that the protocols must be able to be sent over TCP because Tor will only transport TCP streams. Work done by Wang et al. seems to suggest that it should be possible with VoIP [59].

In the arms race that is the field of traffic analysis, every new attack brings up the question "what can be done to defend against it?" We propose that same question here. Using techniques with Tor that disrupt the timing and order should work very well against this attack because it will make building the models difficult. Another alternative that has been proposed is to use a distributed VPN through a half-open Tor circuit. These half open Tor circuits, called hidden services, may be able to be combined with Bittorrent and OnionCat [60] to produce a system robust at deterring this attack.

Another interesting idea would be to replace obviously mistranslated symbols with a wild card symbol to improve the handling of glitches. Say that Tor introduces a delay of one second into a packet and that packet gets incorrectly symbolized into some other symbol. If we knew that it was wrong, we could instead swap it out for our wild card symbol. The wild card symbol would not stop a path from matching, it would just take on whatever value is needed and allow the path matching to continue. The problem could become very complex and computationally expensive if there are many transitions the wild card symbol can take. This would be a challenge to efficiently enumerate and handle all of the possible path combinations until the right one is found.

Appendices

Appendix A How to Configure a Private Tor Network

A small, standalone test network was created to conduct experiments on Tor. This was done so as to not disturb any nodes that were functioning as part of the live global Tor network operating on the Internet. This appendix describes the configuration and layout of our Tor testbed network.

A.1 Network Layout

The network consists of sixteen total systems. All systems are running Tor version 0.2.1.19, which was the latest stable version as of August 2009 [61]. There are two Dell Optiplex GX260 desktops running CentOS 5, two Dell Precision 370 mini-towers running Fedora Core 6, and twelve WebDT 166 thin clients running the lightweight Debian-based Damn Small Linux [62]. The network is a combination of various types of systems on different platforms, much like on the Internet.

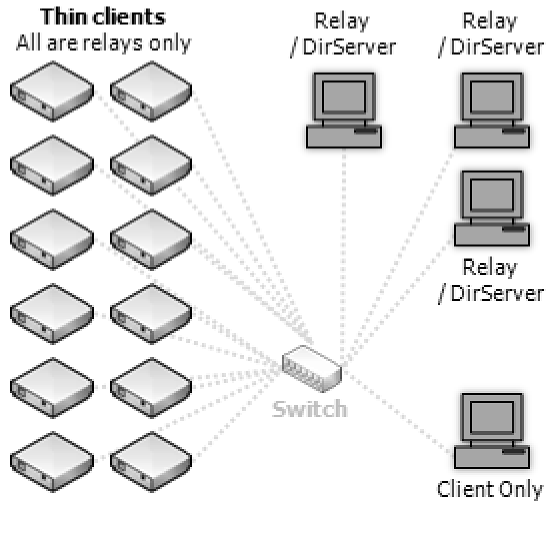


Figure 1: Network diagram of the private Tor network testbed

Figure 1 shows the layout of the network. The smaller icons on the left symbolize the thin clients. Each thin client runs Tor with the client and relay services enabled. Three of the larger Dell systems run Tor with the client, relay, and directory server services enabled, and the last system runs Tor with just the client service enabled. Finally, all systems are connected via a network switch.

A.2 Tor Configuration

Tor determines all of its configuration settings from a file called the `torrc` file. Settings listed in this file tell Tor what name to use, what services to run, what types of logging to perform, what policies to enforce, and, for our standalone environment, which directory servers to use. The specification of these is important because the relays must know where to upload their descriptors and the clients must know where they can find a list of relays to use. By default, Tor uses the directory servers that have been hard-coded into it. By providing a list of our own directory servers, we are telling Tor to use one of ours and to not use any of the preloaded ones.

```
## TOR Configuration File


## Section 1: Basic Settings ##########################################

## Send all messages of level 'notice' or higher to /var/log/tor/notices.log
#Log notice file /var/log/tor/notices.log
## Send every possible message to /var/log/tor/debug.log
Log debug file /var/log/tor/debug.log

## If set to 1, Tor will scrub any personally identifying information
## from the log files
SafeLogging 0

## Start the process in the background
RunAsDaemon 1

## The directory for keeping all the keys/etc.
DataDirectory /var/lib/tor


## Section 2: Client Settings #########################################

## Define on which port the SOCKS client will listen
#SocksPort 0     # don't function as a client
SocksPort 9050   # what port to open for local application connections

## Define on which IP address the client will listen
#SocksListenAddress 127.0.0.1  # accept connections only from localhost
SocksListenAddress 0.0.0.0     # accept connections from anyone


## Section 3: Relay Settings ##########################################

## A unique name for the server
Nickname tigertor10

## The IP or FQDN of the server
Address 192.168.69.10

## Contact info to be published in the directory
ContactInfo Nobody <nobody AT example dot com>

## Port to advertise for Tor connections
ORPort 9001

```

```
47 ## Publish server descriptors of certain versions
48 PublishServerDescriptor v2,v3
49
50 ## A comma-separated list of exit policies
51 #ExitPolicy accept *:*  # any exiting ports/protocols allowed
52 ExitPolicy reject *:*   # no exiting ports/protocols allowed
53
54
55
56 ## Section 4: Authoritative Directory Settings #######################
57
58 ## What port to advertise for incoming directory connections
59 DirPort 9030
60
61 ## Become an authoritative directory
62 AuthoritativeDirectory 1
63
64 ## Specify which versions to publish
65 V2AuthoritativeDirectory 1
66 V3AuthoritativeDirectory 1
67
68 ## To make routers show up as "named" in the directory
69 NamingAuthoritativeDirectory 1
70
71 # Display this HTML page at the root of the directory server's port
72 DirPortFrontPage /var/lib/tor/webpage.html
73
74 ## Entrance policy for this directory server
75 DirPolicy accept *:*    # any ports/protocols allowed
76
77
78 ## Section 5: Settings for running Tor on a private network ##########
79
80 ## List of authoritative routers
81 DirServer tigertor10 v3ident=V3IDENTKEY 192.168.69.10:9030 FINGERPRINT
82 DirServer tigertor11 v3ident=V3IDENTKEY 192.168.69.11:9030 FINGERPRINT
83 DirServer tigertor12 v3ident=V3IDENTKEY 192.168.69.12:9030 FINGERPRINT
84
85 ## Makes running on a private Tor network possible by speeding up the
86 ## voting process and disabling some security restrictions
87 TestingTorNetwork 1
```

Listing 1: Sample torrc File

Listing 1 is the `torrc` configuration file for one of our systems running a directory server. The settings in the file are grouped into sections by the service to which they apply (client, relay, dirserver, etc.). This listing shows an example of the configurations that must be made to Tor for it to run in a standalone environment. The most important of which are:

- Enabling and configuring the V3 authoritative directory server (lines 56–75)
- Configuring Tor to use the private directory servers (lines 81–83)
- Enabling the `TestingTorNetwork` setting (line 87)

The `TestingTorNetwork` setting is actually an umbrella setting that configures a multitude of individual settings. Some of the changes include: shortening the timing intervals for V3 directory voting processes, allowing Tor to accept non-publicly routable IP addresses, and making reachability information available more quickly from the directories. Tor will not be able to operate on our private network without these settings configured. The `TestingTorNetwork` option is available in Tor versions 0.2.1.2-alpha and later [63].

A.3 Directory Server Keys

In Listing 1, when the private directory servers are enumerated, there are two fields that must be filled with information from the user's Tor instance.

The field for "FINGERPRINT" can be determined by inspecting the output of the command `tor --list-fingerprint` for the fingerprint. Each relay on the network must have a unique fingerprint, so it is important not to copy the contents of Tor's data directory from one system to another when installing the systems.

The field for "V3IDENTKEY" can be found once the identity and signing keys have been generated for the authority. To generate them, change to Tor's keys directory and run the generate new certificate command:

```
cd /var/lib/tor/keys
tor-gencert --create-identity-key
```

The authority_certificate file that is generated by the command will contain the value for the field near the top of the file in a line beginning with "fingerprint". The character string after the word "fingerprint" is what should replace "V3IDENTKEY" in the `torrc` file. Care should be taken not to confuse the two different fingerprints.

The keys generated by the command expire once a year. They can be renewed by simply running the `tor-gencert` command from the keys directory again (but this time without the create flag). The identity fingerprints should remain the same.

It is important to note that one of the key files generated by the command should be kept secret, authority_identity_key. If it is leaked or stolen, an adversary could create a mimic of the directory server and our relays would trust it. This is less of a risk on a private experimental network, but it is still recommended that the file be moved to a more secure location. If this is done, it needs to be moved back when the keys are renewed, otherwise the fingerprint will be regenerated.

A.4 System Clocks

Another thing that must be done when constructing a private Tor network is to maintain the clocks on each system in the network. We use the network time protocol [64] to keep all test systems' clocks at close to the same time. This is needed because a Tor client will not build a circuit with a relay if the system clocks are off by “more than a few hours” [65].

A.5 Summary

Detailed information has been provided about the configuration of our private Tor network used in our experiments. Once Tor has been installed on a system, the configuration file listed in Section A.2 can be used to replicate one of our directory servers. Additional directory servers can be created from the same file by simply modifying lines 36 and 39. The file can also be adjusted to create Tor nodes that are just relays by removing the authoritative directory section.

Bibliography

[1] Joint Doctrine Division, J-7, Joint Staff, *Joint Publication 1-02, DoD Dictionary of Military and Associated Terms.* United States Department of Defense, October 2009. [Online]. Available: http://www.dtic.mil/doctrine/new_pubs/jp1_02.pdf

[2] T. L. Burns, "The origins of the national security agency 1940–1952," in *United States Cryptologic History*, ser. V. Fort Meade, MD: Center for Cryptologic History, NSA, 1990, vol. 1.

[3] C. H. Sterling, Ed., *Military Communications: From Ancient Times to the 21st Century.* Santa Barbara, CA: ABC-CLIO, Inc., 2008.

[4] G. Danezis and R. Clayton, *Digital Privacy: Theory, Technologies, and Practices.* Boca Raton, FL: Auerbach Publications, 2008, ch. 5, pp. 95–116.

[5] R. Lewin, "A signal-intelligence war," *Journal of Contemporary History*, vol. 16, no. 3, pp. 501–512, July 1981.

[6] D. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, August 2001, pp. 337–352.

[7] A. Hintz, "Fingerprinting websites using traffic analysis," in *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*, R. Dingledine and P. Syverson, Eds. San Francisco, CA: Springer-Verlag, April 2002.

[8] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," in *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, May 2005, pp. 1–11.

[9] M. Liberatore and B. N. Levine, "Inferring the source of encrypted HTTP connections," in *CCS '06: Proceedings of the 13th ACM conference on Computer and Communications Security.* New York, NY: ACM, October 2006, pp. 255–263.

[10] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security.* New York, NY: ACM, November 2009, pp. 31–42.

[11] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?" in *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, August 2007, pp. 43–54.

[12] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations," in *2008 IEEE Symposium on Security and Privacy*, May 2008, pp. 35–49.

[13] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006.

[14] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven anonymity bibliography," March 2010. [Online]. Available: http://www.freehaven.net/anonbib/full/date.html

[15] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, February 1981.

[16] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster protocol — version 2," IETF Internet Draft, July 2003. [Online]. Available: http://www.abditum.com/mixmaster-spec.txt

[17] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy.* IEEE CS, May 2003, pp. 2–15.

[18] N. Mathewson and R. Dingledine, "Practical traffic analysis: Extending and resisting statistical disclosure," in *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, D. Martin and A. Serjantov, Eds., vol. 3424, May 2004, pp. 17–34.

[19] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, August 2004, pp. 303–320.

[20] O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: A system for anonymous and unobservable internet access," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, July 2000, pp. 115–129.

[21] jrandom, "I2P: A scalable framework for anonymous communication," 2010. [Online]. Available: http://www.i2p2.de/techintro.html

[22] R. R. Brooks, J. M. Schwier, and C. Griffin, "Behavior detection using confidence intervals of hidden markov models," *IEEE Trans. Syst., Man, Cybern. B*, vol. 39, no. 6, pp. 1484–1492, December 2009.

[23] J. M. Schwier, R. R. Brooks, C. Griffin, and S. Bukkapatnam, "Zero knowledge hidden Markov model inference," *Pattern Recognition Letters*, vol. 30, no. 14, pp. 1273–1280, 2009.

[24] J. M. Schwier, "Pattern recognition for command and control data systems," Ph.D. dissertation, Clemson University, Clemson, SC, August 2009. [Online]. Available: http://etd.lib.clemson.edu/documents/1252424695/

[25] Y. Zhu, X. Fu, R. Bettati, and W. Zhao, "Anonymity analysis of mix networks against flow-correlation attacks," in *Proceedings of the 2005 IEEE Global Telecommunications Conference (GLOBECOM '05)*, vol. 3, St. Louis, MO, 2005, pp. 1801–1805.

[26] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "Correlation-based traffic analysis attacks on anonymity networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. PP, no. 99, August 2009.

[27] "TorStatus – Tor network status," March 2010. [Online]. Available: http://torstatus.cyberphunk.org

[28] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 4, pp. 482–494, May 1998.

[29] R. Dingledine and P. Syverson, "Reliable MIX cascade networks through reputation," in *Proceedings of Financial Cryptography (FC '02)*, M. Blaze, Ed. Springer-Verlag, March 2002.

[30] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood: Active attacks on several mix types," in *Proceedings of Information Hiding Workshop (IH 2002)*, F. Petitcolas, Ed. Springer-Verlag, October 2002.

[31] A. Acquisti, R. Dingledine, and P. Syverson, "On the economics of anonymity," in *Proceedings of Financial Cryptography (FC '03)*, R. N. Wright, Ed. Springer-Verlag, January 2003.

[32] R. Dingledine and N. Mathewson, "Anonymity loves company: Usability and the network effect," in *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, R. Anderson, Ed., Cambridge, UK, June 2006.

[33] R. Dingledine, N. Mathewson, and P. Syverson, "Deploying low-latency anonymity: Design challenges and social factors," *IEEE Security Privacy*, vol. 5, no. 5, pp. 83–87, October 2007.

[34] R. Dingledine, "Current events in Tor development," presented at the 24^{th} Chaos Communication Congress (24C3), Berlin, Germany, December 2007.

[35] M. Rennhard and B. Plattner, "Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, November 2002, pp. 91–102.

[36] M. J. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9^{th} ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[37] J. McLachlan, A. Tran, N. Hopper, and Y. Kim, "Scalable onion routing with Torsk," in *Proceedings of CCS 2009*, November 2009.

[38] D. Koblas and M. R. Koblas, "SOCKS," in *Proceedings of the 3^{rd} USENIX Security Symposium*, 1992, pp. 77–83. [Online]. Available: http://www.usenix.org/events/sec92/full_papers/koblas.pdf

[39] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "Socks protocol version 5," RFC number 1928, March 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1928.txt

[40] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the Tor network," in *Proceedings of the 8^{th} International Symposium on Privacy Enhancing Technologies (PETS 2008)*, N. Borisov and I. Goldberg, Eds. Leuven, Belgium: Springer, July 2008, pp. 63–76.

[41] S. J. Murdoch and R. N. M. Watson, "Metrics for security and performance in low-latency anonymity networks," in *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, N. Borisov and I. Goldberg, Eds. Leuven, Belgium: Springer, July 2008, pp. 115–132.

[42] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *Proceedings of the 7^{th} Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borisov and P. Golle, Eds. Ottawa, Canada: Springer, June 2007, pp. 167–183.

[43] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against anonymous systems," University of Colorado, Boulder, CO, Tech. Rep., 2007. [Online]. Available: http://www.cs.colorado.edu/department/publications/reports/docs/CU-CS-1025-07.pdf

[44] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy.* Oakland, CA: IEEE CS, May 2005, pp. 183–195.

[45] N. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on Tor using long paths," in *Proceedings of the 18th USENIX Security Symposium*, Montreal, Canada, August 2009, pp. 33–50.

[46] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM Trans. Inf. Syst. Secur. (TISSEC)*, vol. 13, no. 2, pp. 1–28, 2010.

[47] C. R. Shalizi, "Causal architecture, complexity, and self-organization in time series and cellular automata," Ph.D. dissertation, University of Wisconsin-Madison, May 2001.

[48] C. R. Shalizi and J. P. Crutchfield, "Computational mechanics: Pattern and prediction, structure and simplicity," *Journal of Statistical Physics*, vol. 104, no. 3–4, pp. 817–879, 2001.

[49] C. R. Shalizi, K. L. Shalizi, and J. P. Crutchfield, "An algorithm for pattern discovery in time series," *arXiv:cs.LG/0210025*, November 2002. [Online]. Available: http://arxiv.org/abs/cs.LG/0210025

[50] C. R. Shalizi and K. L. Shalizi, "Blind construction of optimal nonlinear recursive predictors for discrete sequences," in *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI 2004)*, M. Chickering and J. Y. Halpern, Eds. Arlington, VA: AUAI Press, 2004, pp. 504–511.

[51] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, February 1989.

[52] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Survey*, vol. 23, no. 1, pp. 5–48, 1991.

[53] "TShark manual," March 2010. [Online]. Available: http://www.wireshark.org/docs/man-pages/tshark.html

[54] "TCPDUMP/LIBPCAP public repository," March 2010. [Online]. Available: http://www.tcpdump.org

[55] "IEEE standard for information technology - portable operating system interface (POSIX)," *IEEE Std 1003.1, 2004 Edition. The Open Group Technical Standard. Base Specifications, Issue 6.*, 2004. [Online]. Available: http://www.unix.org/single_unix_specification/

[56] "Discrete Fourier transform," March 2010. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/fft.shtml

[57] "3proxy tiny free proxy server," March 2010. [Online]. Available: http://www.3proxy.ru

[58] "PlanetLab," March 2010. [Online]. Available: http://www.planet-lab.org/

[59] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the internet," in *Proceedings of the ACM Conference on Computer and Communications Security*, November 2005, pp. 81–91.

[60] "OnionCat," March 2010. [Online]. Available: http://www.cypherpunk.at/onioncat/

[61] "Tor release notes," March 2010. [Online]. Available: http://gitweb.torproject.org/tor.git?a=blob_plain;hb=maint-0.2.1;f=ReleaseNotes

[62] "DSL information," March 2010. [Online]. Available: http://www.damnsmalllinux.org

[63] K. Loesing, "Simplify configuration of private Tor networks," Feature request, March 2010. [Online]. Available: https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/135-private-tor-networks.txt

[64] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, pp. 1482–1493, October 1991.

[65] "Torfaq," March 2010. [Online]. Available: http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ