Analyzing the Effectiveness of Passive Correlation Attacks on the Tor Anonymity Network

by

Sam DeFabbia-Kane
Class of 2011

A thesis submitted to the
faculty of Wesleyan University
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
with Departmental Honors in Computer Science

Acknowledgements

I first want to thank Norman Danner and Danny Krizanc, who have allowed me to work with them on Tor-related projects since my sophomore year, and who have been my advisors for this thesis. I am exceedingly grateful for their time, advice, and patience. I have been able to have spend quite a lot of my time at Wesleyan working on a topic I find extremely interesting, and I count myself very lucky to have had that opportunity.

I also would like to thank my friends, who have supported me during this process and throughout my time at Wesleyan. I have learned just as much from all of them than I have from the classes I've taken here. I would like especially to thank my housemates. Andrew, Dan, Dave, Ryan, Jess, and Lindsey have have remained patient with me over the past few weeks despite me being tired, stressed, and irritable, and they have been amazing friends for the entire time I've known them at Wesleyan.

Finally, I would like to thank my parents. They have always supported and encouraged my curiousity and my interests, and it is that support that has made me into the person I am today.

Abstract

Tor is a widely used low-latency anonymity system. It allows users of web browsers, chat clients, and other common low-latency applications to communicate anonymously online by routing their connection through a circuit of three Tor routers. However, Tor is commonly assumed to be vulnerable to a wide variety of attacks, which might allow Tor operators or outside observers to compromise the anonymity of Tor's users. One of these attacks is an end-to-end correlation attack, whereby an attacker controlling the first and last router in a circuit can use timing and other data to correlate streams observed at those routers and therefore break Tor's anonymity.

Since most prior tests of correlation algorithms have been either in simulation or have only used certain kinds of traffic, our goal was to test how well these algorithms work on the deployed Tor network. In this thesis we tested three correlation algorithms. Two of these algorithms are from prior work, and the third was designed by us. Its design was based on observations and analyses of data we collected during the testing process. We found that while the two previously-existing algorithms we tested both have problems that prevent them being used in certain cases, our algorithm works reliably on all types of data.

Contents

CHAPTER 1

Introduction

Any message sent over the internet contains routing information that can be used to identify the sender and receiver of the message. For many users of the internet, this poses a problem. Activists, whistleblowers, and human rights workers might want to be anonymous to avoid reprisals from oppressive governments or corporations. Military and law enforcement personel might want to be anonymous so that they can gather intelligence or conduct sting operations without identifying themselves online. People living in countries or working at companies with censored internet may use anonymity as a way to circumvent censorship measures. To this end, many *anonymity systems* have been developed with the goal of facilitating anonymous communication online.

These anonymity systems are typically divided into two categories: low-latency systems and high-latency systems. High latency systems—such as Babel, Mixmaster, and Mixminion—implement defense measures such as mixing, padding, batching, and reordering in an attempt to protect against a *global passive adversary* who can observe all network traffic [**4**]. However, such systems can only be used with high-latency communication methods like email, which limits their utility and also limits their user base. Low-latency systems generally do not attempt to protect against a global passive adversary, but are usable with a much wider variety of applications, including web browsers, chat clients, and video streaming.

One popular low-latency anonymity network is Tor [**4**]. Tor works by routing a user's connection through three *onion routers* (ORs), which form a *circuit* and act

as a chain of proxies for the connection. Messages being sent over the connection are layered with encryption (using a technique called *onion encryption* that is detailed in Section 1.1) so that each OR knows only its immediate source and destination. Onion routers are run by volunteers around the world. The routers are coordinated and cataloged by a small set of *directory servers* that provide information about the Tor network and available routers to Tor clients (which are often called *onion proxies* or OPs).

While it does not protect against a global passive adversary, Tor does try to protect against a more limited adversary who can observe some of the traffic going over the network, or who controls some Tor routers. This is important because anyone can run a Tor router, and Tor users have no guarantee that router operators are not malicious. However, despite its design goals, Tor is commonly assumed to be vulnerable to several classes of attacks by non-global adversaries. In this paper, we will examine one of those types of attacks: a passive end-to-end correlation attack whereby an attacker controlling the first and last routers in a circuit can compromise the anonymity of streams going through that circuit. While Tor is assumed to be vulnerable to these kinds of attacks, much prior work in this area has been done in simulation or only in theory. We seek to test the effectiveness of these attacks on the deployed Tor network, and to determine whether we can create a better attack by examining metrics of Tor traffic. This chapter describes how Tor works and what the goals of this thesis are.

1.1. Circuits and Onion Encryption

Users wishing to use Tor proxy their traffic through an *onion proxy* (OP), which transparently handles circuit creation and encryption. Tor's goal is anonymity.

It does not provide end-to-end encryption because it cannot encrypt the step between the exit router and the server the client is connecting to. To do so would require the cooperation of the server, meaning that Tor would not be a transparent proxy. Tor, therefore, is not a replacement for other encryption technologies. However, Tor does use layered encryption interally, which accomplishes two purposes. First, it ensures that each OR knows only about the adjacent nodes in the circuit. Second, it prevents attackers from directly comparing the traffic at any two points in the circuit, because the traffic is differently encrypted (and so looks different) at every point. The OP does this encryption by negotiating a symmetric key with each router in the circuit and encrypting each message with every symmetric key, as described below.

Let $R_1, ..., R_n$ be routers in an n-length circuit and let K_i be a symmetric key negotiated between Alice's OP and R_i. Keys for R_i are negotiated through the previous routers in the circuit, $R_1, ..., R_{i-1}$. When sending a message M, the client first encrypts that message with the key K_n, then K_{n-1}, etc., all the way down to K_1. Consider the case where *Alice* is sending a message to *Bob* over a length-3 circuit $R_1 \rightarrow R_2 \rightarrow R_3$. Let $[M]_{K_i}$ denote the message M encrypted with symmetric key K_i, and let $[M]_{K_{i,j,k}}$ denote the message M encrypted first with K_i, then K_j, then K_k. Alice's OP will first encrypt with key K_3, then K_2, and then K_1, and so the message Alice's OP sends will be $[M]_{K_{3,2,1}}$.

As the message passes through the circuit, each router R_i decrypts the message it receives with its key K_i. It can then pass the message along to the next router in the circuit (or to Bob, if it's the last router in the circuit). So as M goes through the circuit, it looks like this:

$$Alice \xrightarrow{[M]_{K_{3,2,1}}} R_1 \xrightarrow{[M]_{K_{3,2}}} R_2 \xrightarrow{[M]_{K_3}} R_3 \xrightarrow{M} Bob$$

When Bob wants to send a message M' back, he sends M' to R_3, which encrypts it with K_3, and then passes it back along the circuit. Each router R_i in the circuit encrypts it with K_i, and so passage of M' back through the circuit looks very similar to the forward passage of M. Since only Alice knows all three keys K_1, K_2, and K_3, only Alice can decrypt the message and read M'.

$$Alice \xleftarrow{[M']_{K_{3,2,1}}} R_1 \xleftarrow{[M']_{K_{3,2}}} R_2 \xleftarrow{[M']_{K_3}} R_3 \xleftarrow{M'} Bob$$

1.2. Tor Cells

Tor communicates over TCP to ensure in-order delivery. All communication between Tor proxies and routers takes place in an application-level protocol using messages called *Tor Cells*. The protocol is specified in the main Tor specification document, *tor-spec.txt* [**3**]. There are two versions of the protocol. Up-to-date Tor processes will always use version 2 of the specification, and so that is what will be discussed here.

CircId	Command	Payload (0-padded)
2 bytes	1 byte	PAYLOAD_LEN bytes

FIGURE 1.1. Tor Cell Format

Tor cells are 512 bytes long. The format is presented in Figure 1.1. The Command field defines the type and purpose of the cell. Common values for Command include CREATE, CREATED, RELAY, RELAY_EARLY, and DESTROY. CREATE cells are used to initiate a connection between two Tor processes. They are sent by onion proxies to create the first hop in a circuit and also by onion routers to extend a circuit by one hop. CREATED cells are the response to a successful CREATE. RELAY and RELAY_EARLY cells are wrappers which contain any

message sent over an established circuit and will be discussed in more detail below. DESTROY cells are sent to adjacent nodes to tear down a circuit. The Payload field is the part of the cell that gets onion encrypted.

Relay command	'Recognized'	StreamID	Digest	Length	Data
1 byte	2 bytes	2 bytes	4 bytes	2 bytes	498 bytes

FIGURE 1.2. Relay Cell Payload Format

RELAY cells have an additional relay header included in their payload. The format of a RELAY cell payload is shown in Figure 1.2. Relay commands define the purpose of the RELAY cell. BEGIN, END, and CONNECTED relay commands are used for setting up and tearing down TCP streams on a circuit. DATA relay cells are used for sending data across a TCP stream. EXTEND and EXTENDED relay cells are used when constructing a new circuit, and TRUNCATE and TRUNCATED cells are used when tearing a circuit down. Other relay cell types deal with directory server communication, DNS lookup, and congestion control.

The 'Recognized' and Digest fields of the header allow a router to determine whether or not the cell is fully decrypted. A cell is considered fully decrypted if Recognized is set to zero and Digest is the first four bytes of the running digest of all of the bytes destined for or originated from this hop in the circuit. If a cell is not considered fully decrypted, it gets passed on to the next hop in the circuit. The StreamID field is set by the OP and allows the OP and the exit router to distinguish between the multiple streams on a circuit. The Length field is the number of bytes of the Data field which contain actual data. (The remainder of Data is NUL-padded.)

RELAY_EARLY cells are a special type of RELAY cell used for circuit creation. Clients speaking V2 of the link protocol send any EXTEND relay cells as RELAY_EARLY cells instead. An OR receiving more than 8 RELAY_EARLY cells closes the circuit. This limits the maximum length of any circuit, which helps to protect against certain classes of attacks, such as Pappas et al.'s packet spinning attack [**9**].

1.2.1. Example Workflow: Circuit Creation. In Figure 1.3, we present an outline of the workflow for circuit creation. In this diagram, Alice is running an OP and creating the circuit $R1 \rightarrow R2 \rightarrow R3$. (With $K1$, $K2$, and $K3$ being the symmetric keys negotiated during the circuit's creation.)

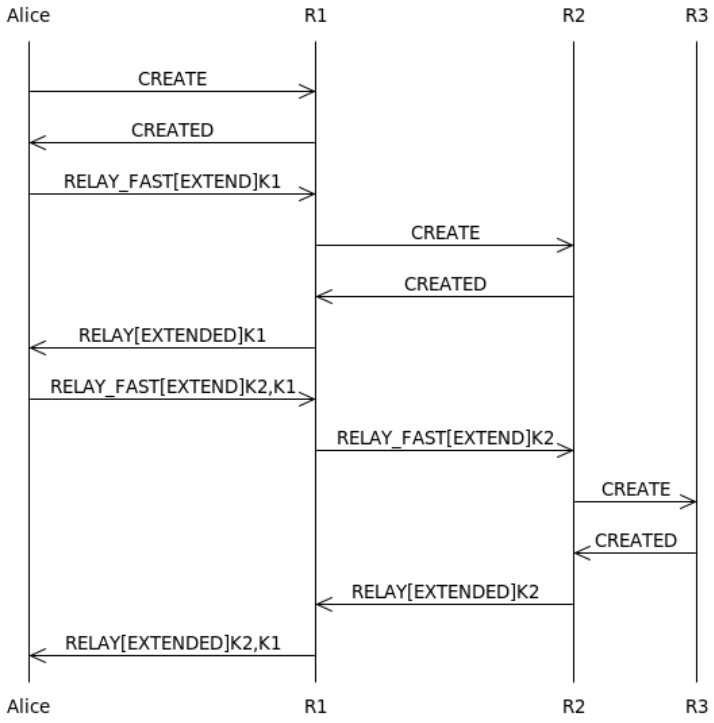


FIGURE 1.3. Circuit Creation Workflow

1.3. Directory Servers

Tor is not a fully-distributed system. A small number of directory servers keep a listing—called a *consensus document*—of all of the routers currently on the network. Every hour the directory servers pool their information and vote to create an updated consensus document. Clients and routers running on the network fetch an updated consensus from a directory server once every hour. The consensus document—along with *router descriptors* published by each router—provide enough information for clients to connect to and verify the identity of the routers on the network.

1.4. Contributions of this Thesis

Tor is commonly assumed to be vulnerable to end-to-end correlation attacks. While the onion encryption performed by Tor prevents direct comparison of packet contents, an attacker controlling the first and last router has access to other information, such as packet timing, and that information is commonly assumed to be enough to break Tor's anonymity. However, prior work on this topic has two problems. First, most of the work has been done only in theory or in simulation, and the simulations do not necessarily take into account all of the factors introduced by Tor that may affect a given correlation algorithm. Second, the existing work that has been done using real data focuses on streams with large numbers of packets sent, which means that a user of Tor might be able to evade an attacker by only sending small amounts of data at once.

This work seeks to answer two questions. First, we seek to determine whether additional factors (such at latency) introduced by Tor, prevent a passive end-to-end correlation attack from working. And second, if correlation is feasible, we

seek to determine whether such attacks can work even when clients transfer only a small amount of data.

Chapter 2 provides an overview of prior work related to timing correlation and other related attacks against Tor. Chapter 3 contains metrics on data collected from Tor. This information will allow us to determine why certain algorithms succeed or fail. Chapter 4 describes our experiment and results for performing correlation over Tor. It includes detailed descriptions of two existing correlation algorithms and a new simple correlation algorithm, the design of which is based on the data we examined in Chapter 3. Finally, Chapter 5 summarizes our work and suggests potential areas for further research.

CHAPTER 2

Attacks on Tor

Many different types of attacks have been proposed to work against low-latency networks in general and Tor in particular. This chapter is a brief survey of some of those attacks. Two of these attacks will be examined in more detail and tested in Chapter 4.

2.1. Stream Correlation

In stream correlation attacks, an attacker who can observe two packet streams attempts to verify that they are the same stream at different points in the anonymity network. Since streams in Tor are onion encrypted, they cannot be compared directly, and the attacker must try to correlate them using other available information.

2.1.1. Packet Counting. Packet counting is one simple form of stream correlation. As proposed by Back et al. [**1**], an attacker who can observe onion routers counts the number of packets entering and leaving the first router to determine what the next step in the circuit is. The procedure is then repeated for later routers in the circuit until the destination is determined. While this form of packet counting is relatively simple to implement, it requires an attacker to be able to observe a very large amount of the network, and assumes that there is never any variation in the number of packets entering and leaving a router on a given stream. As such, packet counting has been largely overshadowed by more sophisticated stream correlation techniques based on packet timing.

2.1.2. Timing Analysis. Packet timing is another piece of data that can be used to correlate network streams. One simple way to use packet timing data is to use some sort of correlation function to attempt to correlate streams based on their *inter-packet delay*—the time between the arrival of packets adjacent on the stream. However, this approach may have problems with dropped packets. Levine et al. [**6**] proposed a correlation algorithm using time series constructed from packing timing information instead. A time series is one way of looking at packet timing data. To create the time series, we set a constant time W, divide the packet streams into windows of size W and count how many packets fall into each window. The correlation function is a normalized dot product. They simulated their correlation algorithm with four types of user traffic (traffic generated from the 1996 Berkeley HomeIP survey, random traffic, constant traffic, and constant traffic with random packets dropped) and showed that they could successfully perform correlations in a majority of situations with minimal false positives.

The weakness of most timing attacks is that they rely on the attackers controlling Tor routers, and require the attackers to control a large portion of the Tor network to be widely effective [**6**]. While there have been improvements proposed (such as Borisov et al.'s denial of service attack whereby attacking routers kill circuits they can't control [**2**]), there are also timing attacks that don't rely on controlling individual Tor routers. Murdoch and Zieliński [**8**] proposed one such attack, where the adversaries control Internet Exchanges and so can observe traffic entering or leaving countries. They showed that they could perform correlation (using an algorithm derived from Bayes' formula) even when they tracked only one packet per two-thousand in a given stream.

2.1.3. Active Timing Correlation. Active correlation attacks are an effort to make time-based correlation easier and more effective. They work by having

an attacking router alter the packet delay signature of a connection by dropping or delaying packets in the stream. They were proposed, but not tested, by Levine et al [**6**].

Wang et al. [**10**] demonstrated that active timing attacks are feasible and effective against highly-interactive protocols like VoIP, even when protected by the *findnot.com* anonymity service. They performed active timing attacks on peer-to-peer Skype calls by creating and injecting a unique watermark into the stream. They found that, if the right parameters were chosen, they could correctly identify 99% of the watermarked streams with a false positive rate of 0%. Increasing the identification rate to 100% came at the cost of only an 0.1% false positive rate.

2.2. Clogging

Murdoch and Danezis [**7**] presented a clogging attack, where they take advantage of the fact that one connection through a router has an effect on other connections through the same router. The attacker must control a Tor router and be able to observe a connection at some point between the Tor exit router and its final destination. Using the compromised Tor router, the attacker can create length-one circuits to all other Tor routers one-by-one to see whether or not this increases the latency of the connection the observer is watching. If it does, then that router is on the circuit. Murdoch and Danezis tested their attack on the nascent Tor network and found that the attack worked against 11 of the 13 routers on the network at the time. However, since there are now almost 2,500 routers running, this attack is not necessarily still viable.

2.3. Round-Trip Travel Time

Hopper et al. [**5**] presented two attacks that revolve around determining the round-trip travel time (RTT) from clients to servers. In the first attack, the attacker is in control of two servers that are receiving connections from the same exit router. The attacker's goal is to determine whether the connections are coming from the same circuit. Through one of several methods (forcing the user's web browser to download thousands of tiny image files sequentially, forcing the user's web browser through a series of HTTP redirects, or the use of an interactive protocol like IRC), both servers obtain a large number of round-trip travel times from the client they're curious about. They then compare the frequency distributions of the RTTs. If the frequency distributions are similar, then the connections are likely to be from the same circuit.

CHAPTER 3

Metrics for Tor Traffic

Our end objective is to evaluate the effectiveness of end-to-end timing correlation attacks on the deployed Tor network. However, most of the correlation algorithms we will discuss rely on multiple distinct factors to perform correlation, and so before testing the correlation algorithms we will first isolate and examine some of those factors individually. This will allow us to understand why certain succeed or fail and will also provide the justification for a new correlation algorithm that we present in Chapter 4. Since we are testing the effectiveness of these algorithms against an attacker who controls Tor routers, the attacker has access to any information the routers have access to, meaning that the attacker can use Tor cell data rather than the raw TCP packet data.

3.1. Traffic Over Tor

First we will examine the effect that Tor has on network traffic. We will look at two factors used by correlation algorithms: latency between Tor cells, and overall stream length. On an ideal network, we would expect that latency would remain constant, and that the stream would take exactly as long to receive as to send. However, Tor routers have varying connection speeds and qualities, and so assuming that Tor is close to an ideal network in these regards may be problematic.

3.1.1. Test Setup. Our goal is to test whether correlation works when the attacker controls both the entry and exit routers, and so we will use private entry

and exit routers running on the same computer for these tests: only the middle routers will change.

For our control group, we will use another private router for the middle router. It will run on the same computer as the entry and exit. Traffic going through this router will not be subject to latency, since the connection will not be over a network. And since there's no other traffic going through the middle router, it won't be under any significant load, so conditions will be as close to ideal as possible.

For our first experimental group, the middle router will be a public router that we control. This router is running on a separate computer, but is on the same local area network as the computer running the private entry and exit routers, and so latency is low and fairly constant. At the time of testing, our router was routing approximately 1Mbit/s of Tor traffic, and so is under load. This test will allow us to determine whether Tor router load affects the metrics.

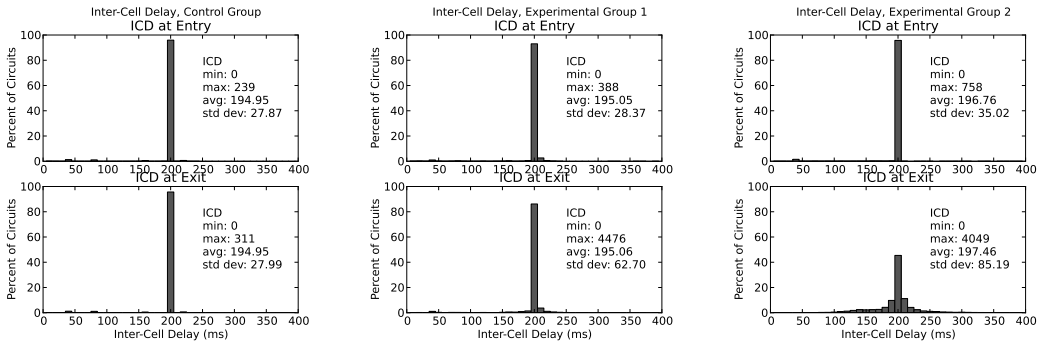
Our second experimental group will use many different middle routers. For each trial, we will choose a router at random from among the routers present on the network to be the middle router. This group will have varying latencies and varying router loads, and will allow us to see their combined effect on the metrics.

For each group, we will run tests with two types of traffic. The first type is a ping client that sends a ping and receives a response every 200ms for 30 seconds. This traffic type will be used to test the effect of Tor on inter-cell latency. The second type of traffic is a 1MiB file download, which will be used to test whether overall stream length varies.

Our data collection consists of collecting the timestamp of each RELAY cell sent or received by the entry and exit routers. We collect this data by modifying

Tor's source code to use Tor's existing logging framework to log Tor cell data to a file.

3.1.2. Results. Since the two types of traffic we're looking at are very different, we'll use different metrics to evaluate the effect that passing through Tor had. For the constant-rate intermittent ("ping") traffic, we'll look at the distributions of delays between consecutive packets. Since the client is sending the pings, we expect the delay between cells at the first router to be almost constant at 200 milliseconds (or very close to it). We hypothesize that the delay will remain constant (or close to it) in our control group, and will vary in both of our experimental groups. We performed rounds of data collection with the control and both of the experimental groups. The inter-cell delay distributions of all three are presented in Figure 3.1.



(a) Control Group (b) Experimental Group 1 (c) Experimental Group 2

FIGURE 3.1. Ping Traffic Inter-Cell Delay Distributions

For the file download, we look at a different metric: the amount of time taken to send the file from the exit back to the middle router, and the amount of time taken to receive the file at the entry router from the middle router. The results for this are displayed in Figure 3.2.

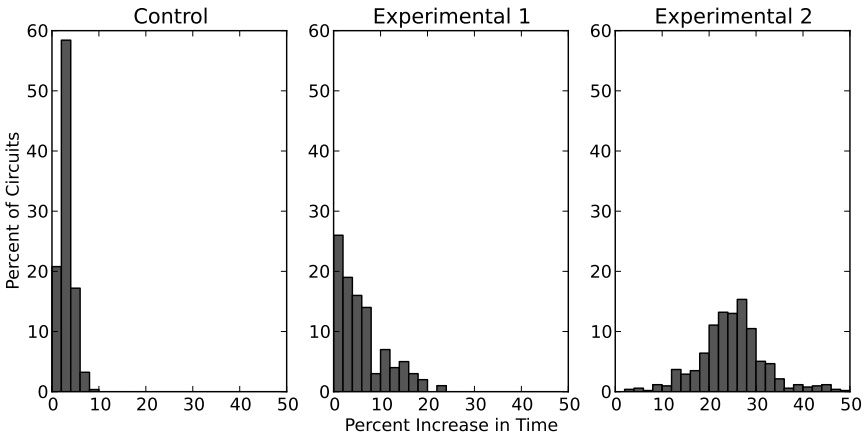


FIGURE 3.2. Increase in Time Taken to Receive a File over Tor

3.1.3. Conclusions. As we can see in Figure 3.1(a), the inter-packet delay of ping traffic going through local, unloaded Tor routers in our control group remains almost exactly the same between the entry and exit routers. The results for the first experimental group—presented in Figure 3.1(b)—vary much more than in the control, with the standard deviation more than doubling. However, the delay still remains within 5ms of the expected 200ms over 80% of the time. The same does not hold true in the results for our experimental group—presented in Figure 3.1(c)—where the standard deviation is even higher, and the variation is within 5ms less than 50% of the time. This means that even small amounts of traffic may be susceptible to varying latencies when traveling over a Tor circuit, which could pose problems for correlation methods that rely on the delay remaining relatively constant.

Figure 3.2 shows us that the time taken to receive data over Tor is significantly higher than the time taken to send it, that the time increase varies and so can not be predicted, and also that at least some of this increase occurs even under ideal network conditions, as was the case in our first experimental group. This suggests

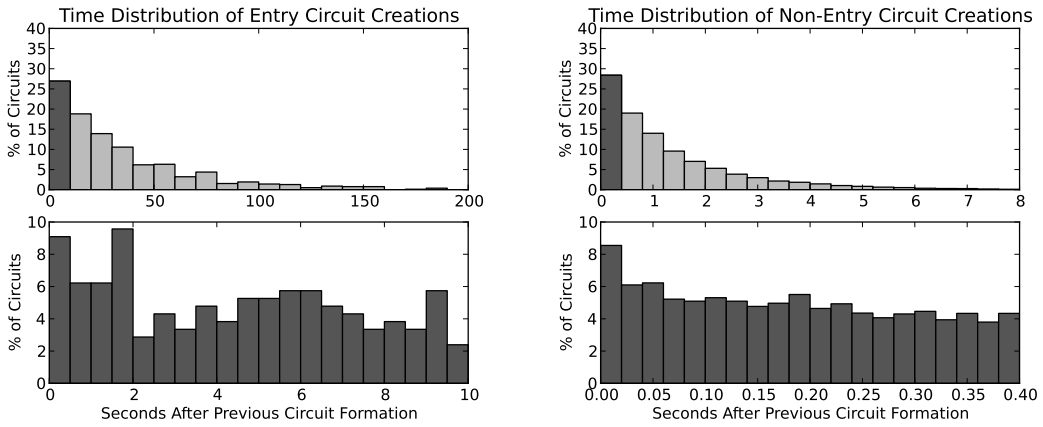
that correlation methods that compare vectors of timing information directly—such as the method presented in Levine et al. [**6**]—may not work as well in practice as they do in theory, as the two streams will have very different lengths.

3.2. Entry Router Traffic

We now have some idea of what happens to traffic when it goes over Tor. Aggregate information about Tor traffic is also useful, as it may allow us to determine factors that are likely to be unique. Since we've already established that traffic over Tor has non-constant latency and overall length (and, therefore, that those may be problematic factors to base a correlation metric on), we will look at two other factors: circuit creation time, and the total number of Tor RELAY cells in each observed stream.

3.2.1. Test Setup. For this test, we will collect data from our public Tor router. We will look at the information about the circuits using our router as an entry or middle router. We guess which we are by checking whether or not the Tor process earlier in the circuit than us is in consensus or not. If it is, we say that we're the middle router. If it's not, we say that we're the entry router. We exclude circuits where fewer than 3 cells are recorded. 2 cells are required for circuit creation when using Tor's standard length-3 circuits. Circuits with fewer than 3 cells were set up but never used to send or receive information, and so trying to correlate them wouldn't give the attacker useful information.

3.2.2. Results. Results are based upon approximately 800 entry circuit creations and 20,000 other circuit creations collected across multiple trials. Results for the time distribution of new circuit creation are presented in Figure 3.3. Results for the distribution of counts of inward-bound cells on circuits at our Tor router are presented in Figure 3.4.



(a) Entry Circuit Creations (b) Non-Entry Circuit Creations

FIGURE 3.3. Distribution of time between consecutive circuit creations at our Tor router.

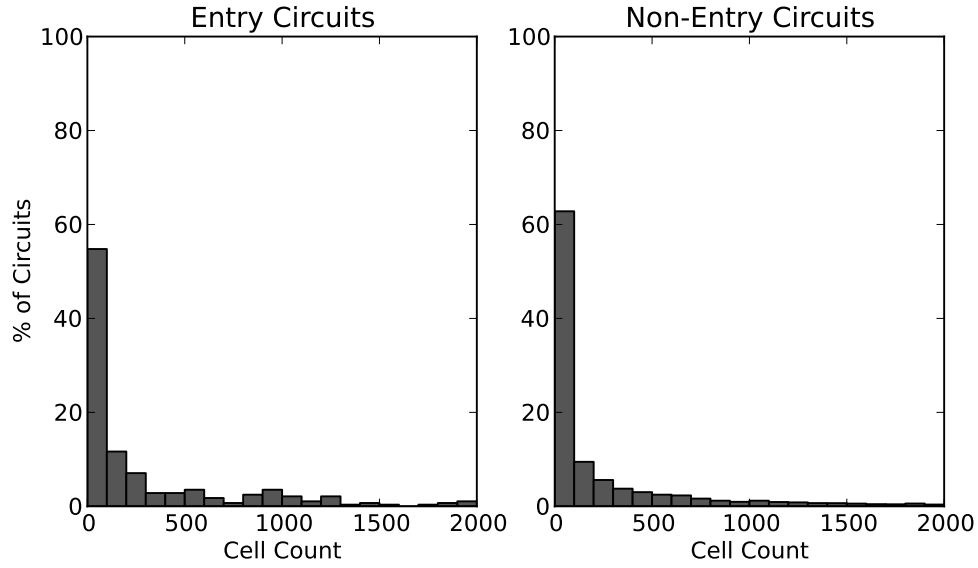


FIGURE 3.4. Distribution of counts of inward-bound cells.

3.2.3. Conclusions. As we can see in Figure 3.3, the shape of the time distributions is very similar for both entry and non-entry circuit creations, but the values are very different. However, our data contained many more non-entry

circuit creations than entry circuit creations, and so this variation in values makes sense. Figure 3(b) has its x-axis scaled so that the values are 4% of the values in Figure 3(a), the ratio of entry to non-entry circuit creations was $4 : 100$ as well, and the distributions look very similar. This means that the reason that the values are significantly different is the rate of circuit creation, not anything fundamentally different about the different types of circuits. This makes sense, since a non-entry circuit creation at our router corresponds to an entry circuit creation on a different Tor router.

The time distribution data also tells us that circuit creation time may be a good metric for differentiating between circuits: even when considering the much greater number of non-entry circuits, under 30% of the circuits had start times within a third of a second of anther circuit.

The count distribution data—presented in Figure 3.4—shows us that most circuits are short: $50-60\%$ of circuits have 100 or fewer cells sent back to the OP. This means that an effective correlation algorithm needs to be able to correlate short circuits as well as longer ones. As with the circuit start time data, the shape of the distributions for the entry and non-entry counts are very similar.

CHAPTER 4

Testing Correlation Algorithms

Having established the effects that network traffic has on Tor, we return to our original goal: testing timing correlation methods on real Tor traffic.

4.1. Attacker Model

For all of the following definitions, let R_1 and R_3 be the first and last routers in a circuit, which are controlled by an attacker whose goal is to break Tor's anonymity for a certain subset of users. Let $x_1 \dots x_n$ be streams of Tor cells observed at R_3, the exit router, and let $y_1 \dots y_m$ be streams of Tor cells observed at R_1, the entry router. For each x_i, the attacker's goal is to determine whether there exists a y_j such that x_i and y_j are the same stream. Since all of the correlation algorithms work on only two recorded streams at once, we will refer to the streams currently being considered as x and y for notational simplicity.

4.2. Correlation Algorithm Definitions

We consider two existing correlation methods—chosen because they are well-defined and work very differently—and one correlation method that we have defined ourselves. First we consider the correlation method of Levine et al. [**6**], which is a normalized dot product that takes into account timing data from all observed Tor cells, and which was originally tested in simulation. Second we consider the correlation method proposed by Murdoch and Zieliński [**8**], which works without considering individual cell timing data, and which was originally tested on real data. Finally, we propose a new simplified correlation algorithm that

also works without considering individual cell data, and which is significantly less computationally intensive than either of the other two methods.

4.2.1. Levine et al. Correlation. The method proposed by Levine et al. uses timing data from all observed Tor cells. While some previously published attacks attempted to use the times between adjacent cells as vectors for correlation, Levine et al. observe that that method is fragile because it is sensitive to dropped packets. Their method is a normalized dot product that operates on a *time series* generated from the observed timing data, rather than directly on the timing data itself.

To construct a time series $\{z_i\}_w$, we pick a window size w, divide the packet stream z into non-overlapping windows of size w (zero-padded so that the time series start and end at the same time), and count the number of packets in each window. The vector of packet counts is the time series.

They define the cross-correlation with delay d of the time series of streams x and y to be

$$r(d) = \frac{\sum_i (y_i - \bar{y})(x_{i+d} - \bar{x})}{\sqrt{\sum_i (y_i - \bar{y})^2}\sqrt{\sum_i (x_{i+d} - \bar{x})^2}}$$

with z_i being the i^{th} element of the time series $\{z_i\}_w$, and $\bar{z}$ being the average of all of the z_i in $\{z_i\}_w$. For their analyses, Levine et al. used a delay of 0 and a window size of 10 seconds.

Here, we consider two vectors $a = y - \bar{y}$ and $b = x - \bar{x}$. The correlation formula is the normalized dot product of those two vectors. The dot product of two vectors is larger the closer the vectors are to being parallel, so a larger result indicates that the two vectors (and, therefore, the two packet streams) are more highly correlated. However, the value of a standard dot product will vary depending on the length of the vectors being compared, and so it is useful to normalize the

values. This can be done by taking advantage of the relation between the value of the dot product and the angle between the two vectors.

$$a \cdot b = |a||b| \cos\theta$$

Here, a and b are vectors, $a \cdot b$ is the dot product of a and b, and $|a|$ is the length of the vector a. To isolate θ and get the angle, we can rewrite the above formula as follows:

$$\theta = \arccos\left(\frac{a \cdot b}{|a||b|}\right)$$

Since the domain of arccos is $[-1, 1]$ (and since we know the argument is valid by the Cauchy–Schwarz inequality), we know that the argument also ranges over $[-1, 1]$. Since arccos ranges and decreases monotonically from $[\pi, 0]$ over that interval, the two vectors are parallel when the argument is 1, and antiparallel when the value is -1. The argument to arccos in the above formula is equivalent to the correlation function, since the top of the fraction in the correlation function is the definition of the dot product, and the bottom is the product of the lengths, calculated using the n-dimensional case of the Pythagorean theorem.

One issue with Levine et al.'s correlation method is that is that the result is undefined in certain cases. Because the algorithm subtracts the average value of a time series from each index before doing correlation[1], a time series with the same number in each index will end up having all zeroes, resulting in a divide-by-zero error. When that happens, we return a correlation of -1, meaning that we effectively do not consider that result. Also, we will only consider correlations with a delay d of 0, since that was what Levine et al. found to be effective.

4.2.2. Murdoch and Zieliński Correlation. The correlation method proposed by Murdoch and Zieliński is derived from Bayes' formula. They model each

[1]The reason for this is not explained in their paper.

flow p as a Poisson process with a start time s, duration l, and rate r (average packets per second). Since the actual flow p is not observable, they consider the two streams x and y instead. Both can be directly observed by the attacker, and are modeled as independent Poisson processes from the parameters of p. They then can derive the probability $P(T_k)$ that x and y_k are from the same flow using Bayes' formula:

$$P(T_k|y_{1..n}, x) = \frac{P(y_{1..n}|T_k, x)P(T_k|x)}{\sum_i P(y_{1..n}|T_i, x)P(T_i|x)}$$

Since they only want relative (rather than absolute) probabilities, they ignore all factors independent of k, which allows them to derive the final probability:

$$P(T_k|x, y_{1..n}) = \frac{P(x, y_k|T_k)}{P(y_k)} \sim \frac{\Gamma(n_{xy_k})}{2^{n_{xy_k}}\Gamma(n_{y_k})} \cdot \frac{n_{y_k}(n_{y_k} - 1)}{n_{xy_k}(n_{xy_k} - 1)} \cdot \frac{l_{y_k}^{n_{y_k}-1}}{l_{xy_k}^{n_{xy_k}-1}}$$

$\Gamma(n) = (n-1)!$, n_y is the total number of packets in y, $n_{xy_k} = n_x + n_{y_k}$, $l_x = x_{max} - x_{min}$ is the observed length of x, and $l_{xy_k} = max(x_{max}, y_{max}) - min(x_{min}, y_{min})$, is the total observed length of x and y.

The formula has three parts, which are multiplied together to find the probability. The first part is based on the rate of packets observed at x and y, the second is a correction factor for the rate, and the third part is length-dependent, which helps to ensure that only streams that occurred at roughly the same time and for roughly the same amount of time are considered.

This correlation method was originally intended to be used by an adversary that controls Internet exchanges, outside the Tor network, and who could observe only small samples (about 1 of every 2000 packets from each stream) of traffic for a given stream. However, this correlation method is also usable within the Tor network by an attacker who controls Tor routers, which is our attacker model. One problem with using this correlation method in our model is that its probabilities are strictly relative and are not normalized, so we can't set a benchmark that

we consider "good enough" for two streams to be considered correlated. This is problematic in cases where we might not necessarily be viewing all of the streams in the network, and so may not be able to correctly correlate a given x with any observable y.

The final result of this correlation calculation often has an extremely small exponent (often in the negative thousands or tens of thousands). Since this formula calculates only relative probabilities, we can take its logarithm and lose no information. We take advantage of this fact and use Stirling's approximation for large factorials to significantly speed up the correlation calculation.

4.2.3. Simplified Correlation. In chapter 3, we quantified the affects of Tor on network traffic, and discovered two things about Tor's affect on network traffic. First, we learned that the latency of individual cells going over Tor is variable, even within a given circuit. Second, we learned that the total time it takes a packet stream to enter Tor at one end is often shorter than the total time it takes for that packet stream to completely exit Tor at the other end. The first of these factors may affect correlation algorithms that rely on individual packet timings (such as the algorithm presented by Levine et al), and the second of these factors may affect correlation algorithms that rely on overall circuit timing, such as the algorithm presented by Murdoch and Zieliński.

In an attempt to counteract both of these issues, we present and test our own correlation algorithm. This algorithm takes into account two factors: the circuit start time, and the total number of Tor cells sent over the circuit. For our algorithm, we define the correlation c to be

$$c = \frac{T - abs(x_{start} - y_{start})}{T} \cdot \frac{|x| + |y| - abs(|x| - |y|)}{|x| + |y|}$$

We saw in chapter 3 that circuit start time is relatively unique, and so we use it as the first part of our correlation. Our correlation looks at the difference in the start times between x and y, and takes into account that difference as a percentage of a fixed time. Since our time measurements are in milliseconds, and since, as we saw in chapter 3, 20 seconds is far longer than any delay likely to occur over a circuit, we use 20 seconds (or 20000 milliseconds) as our fixed time T.[2] When the time difference is small, this first factor will be very close to 1. As the time difference increases, it becomes negative, and so ranges on $[-\inf, 1]$.

We also know that Tor guarantees delivery of Tor cells, and so the number of Tor cells in x and y should be the same after accounting for any cells used to communicate with the middle router. (The process by which that is done is explained in the following section.) Therefore, we use the Tor cell count (the number of cells on stream x is denoted $|x|$). Since this factor is an inverse percentage of the total number of cells sent in both streams, its range is $[0, 1]$.

When multiplied together, the result ranges on $[-\inf, 1]$. However, since negative results mean that the circuits had very different start times, we only really need to take into account results that range on $[0, 1]$. This means that positive correlation values are absolute rather than relative, and can be compared directly across multiple runs of the algorithm, rather than only relatively within a given run.

In addition to being simpler and using less information than either of the two previous methods, our simplified correlation method is also significantly faster to compute, since it relies only on basic arithmetic operations, and deals only with numbers that fit inside a standard *int* class, making it an $O(1)$ operation to perform a single correlation of two streams. In contrast, the Levine et al. correlation

[2] In initial testing, results were relatively constant despite changes to the fixed time.

is $O(n)$ on the number of observed packets to perform a single correlation because it needs to compute a dot product. The Murdoch and Zieliński correlation is $O(1)$ as well when using Stirling's approximation, but it still ends up being significantly slower than our simplified correlation method in practice.

4.3. Test Setup

We control a single public-facing Tor router, which we will use to collect test data. Our router is stable and a guard, meaning that some clients will use it as an entry router. We will use it to collect data when our router is the entry or middle router on a circuit. We will also collect data at a private exit router used only by us. The data collected consists of the timestamps of sent and received RELAY cells, along with the addresses and circuit ids associated with each cell. Since we do not wish to compromise the anonymity of people using our Tor router, we will replace each IP address (which is the only identifying information we log) with a unique random string.

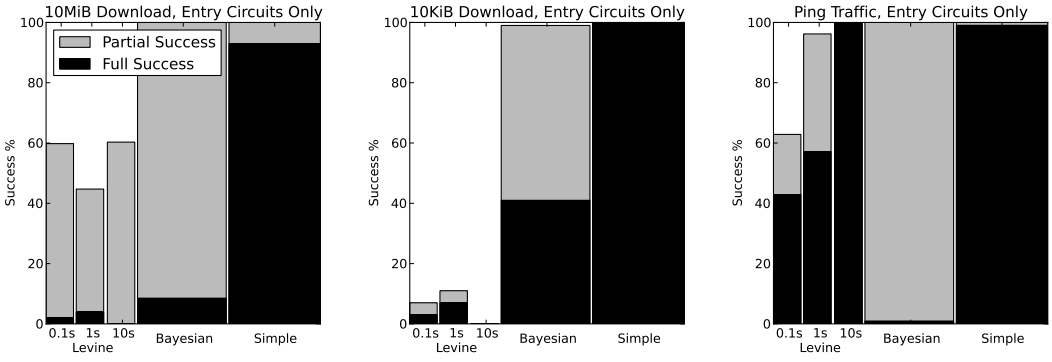
We will create streams with our public router as the entry, a random middle router, and our private exit. We will do many-to-one correlation, correlating all observed data (called $y_1 \dots y_n$ previously) at our public entry router against each single stream (a given x_i) at our private exit router.

Correlation for a given stream recorded at the private exit router (and a given correlation algorithm) will be considered *partially successful* when the correlation algorithm is able to correctly and uniquely identify the corresponding stream at the entry router. However, this information is only useful to an attacker when they know that there exists a given y_j such that x_i and y_j came from the same stream, and so correlation will only be considered *fully successful* when the highest incorrect correlation value is less than all of the other correct correlation values

(for all x_i) in a given run. If a correlation algorithm is consistently fully successful, then the attacker can set a minimum benchmark for correlation, which allows them to determine whether or not there exists a y_j that came from the same stream as x_i, in addition to determining which stream it is.

We will perform tests with three types of traffic. The first two are the 1MiB file download and ping client discussed in chapter 3, and the third is a 10KiB file download, which will allow us to test whether correlation can be done successfully on very short-lived streams.

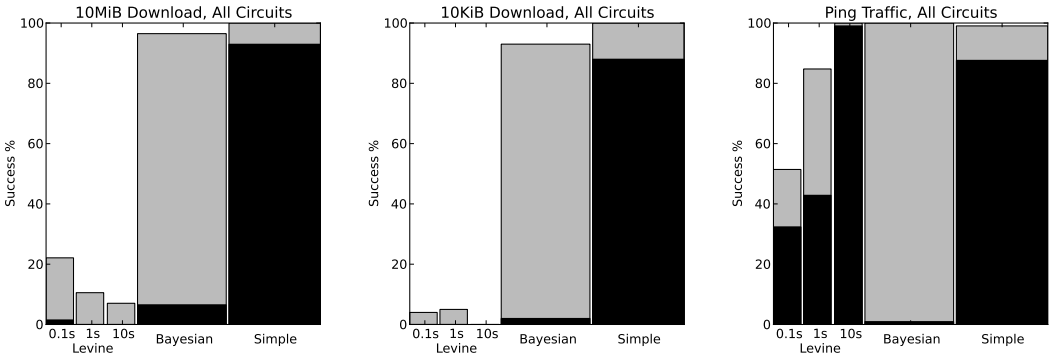
4.4. Results



(a) 10MiB Download (b) 10KiB Download (c) Ping Traffic

FIGURE 4.1. Correlation results with entry circuits only

Results for tests of the correlation algorithms on 3 traffic types are presented in Figure 4.1 and Figure 4.2. While an attacker would realistically only be correlating against the streams where they were the first router in the circuit, we showed in Chapter 3 that the non-entry streams are not characteristically different, and so we present the combined results as well so that we may see how the algorithms perform when correlating against a much larger number of circuits in the same amount of time.



(a) 10MiB Download (b) 10KiB Download (c) Ping Traffic

FIGURE 4.2. Correlation results with all circuits

4.4.1. Levine et al. Correlation Results. As we can see, Levine et al. correlation is effective only for ping traffic, and variations in window size are not enough to make it competitive with either of the other two correlation algorithms for the other traffic types. This is likely because the Levine et al. correlation algorithm relies on individual packet timing data, and, as we showed in chapter 3, the latency of traffic over Tor is highly variable, even within a given packet stream.

4.4.2. Murdoch and Zieliński Correlation Results. The Bayesian algorithm has a high partial success rate, but a low full success rate. This means that an attacker utilizing the Bayesian algorithm would have been able to correlate streams correctly almost all of the time so long as the attacker has information about all (or almost all) possible packet streams. However, since attackers directly control Tor routers in our attacker model, an attacker would need to control most of the Tor network for this to be feasible, and so the Bayesian algorithm is unlikely to be effective for an attacker controlling a smaller number of routers if the attacker cares about preventing false positives. Since the Bayesian algorithm

gives only relative correlations, this result is unsurprising: the algorithm is not designed to deal with potential false positives, because they are much less likely to occur in the original use case. This is because the Bayesian algorithm was originally intended for use by an attacker who controls Internet exchanges which serve as the points of connection between countries. Such an attacker would be able to observe much larger amounts of traffic at once than an attacker controlling a small number of Tor routers.

4.4.3. Simple Correlation Results. Our simple algorithm performs the best overall. It has an almost perfect partial success rate for all traffic types, meaning that if an attacker is able to observe both the input and output stream, they will almost certainly be able to do correlation. It also has a high full success rate, meaning that an attacker using the simplified algorithm will be able to identify most of the cases in which they're attempting to correlate a stream which they did not observe the other end of. This makes sense, since our simplified algorithm produces an absolute correlation value that can be meaningfully compared across streams. This fact makes our simplified algorithm the best choice for the specified attacker model, since it allows attackers that control smaller numbers of routers some degree of protection against false positives. In addition, our simple algorithm's results are consistent across all three of the traffic types we tested, meaning that timing correlation can be done on Tor even in the cases when there is very little information being sent on a circuit. Users cannot evade attackers using the simplified correlation algorithm by sending less traffic.

4.4.4. Applicability of Results. One issue with our results is that they are based upon experiments done with a single public router. Our results do not necessarily generalize to attackers running many routers, because they will be observing

significantly more traffic. Our simplified algorithm relies on two factors—stream start time and the number of cells in the stream—that are relatively unique for the amount of data that we were able to test. Those two factors will necessarily be less unique the more data we have, and so our simplified correlation method will presumably not perform as well.

Consider our test setup with a single Tor router. Based on the number of circuits n_t created through our router in a given time period t, the bandwidth of our router b, and the total bandwidth of the network B, we can roughly estimate the number of circuits created on the whole of Tor in that time period: $\frac{n_t}{b} \cdot \frac{B}{3}$. (We divide by 3 because standard Tor circuits are 3 routers long, meaning that circuit creation had to occur at 3 routers for a single circuit to be created.) In our case, during testing our router had a bandwidth approximately 1MB/s, and the total bandwidth of the network was approximately 900MB/s. This means that an attacker might potentially have to correlate against 300 times as many streams as we did in the same time window.

However, our modeled attackers directly control Tor routers, and so they may collect additional information to take this into account. Most importantly, both the entry and exit routers know the identity of the middle router for any traffic they're sending or receiving. This means that the attacker only needs to try to correlate traffic streams that have the same middle router, which drastically reduces the number of streams that need to be correlated. Even if we exclude exit routers entirely (and Tor's path selection mechanism does not), there are well over 1000 routers that may be used as a middle router. If we could assume that each of the 1000 were chosen equally, that would mean that we'd be able to reduce the number of streams we need to correlate against by a factor of 1000, which more than negates the fact that there's 300 times as many streams overall as are

seen by our router. While some will be used more often than others since Tor's path selection is based on bandwidth (among other factors), there will still be a significant reduction in the number of streams we need to correlate against.

In addition, an attacker may take advantage of the fact that controlling a Tor router allows them to associate streams going in opposite directions on the same circuit. The correlation algorithms above only take into account a stream that moves in a single direction, and the pair (inward cell count, outward cell count) will be more unique than just the cell count going in one direction, which will also help to counteract the increased amount of information.

Therefore, an attacker controlling hundreds or even thousands of routers is not likely to need to correlate a significantly higher number of streams than we have done ourselves with just our one router, meaning that the factors that are unique enough to perform correlation for our data will remain unique enough to perform correlation when an attacker controls more routers.

CHAPTER 5

Conclusion

In this work, our goal was to determine whether it was feasible to perform passive correlation attacks over Tor, and also whether it was possible for users to evade attackers using such attacks by sending only very small amounts of data.

To that end, we tested two existing correlation algorithms and discovered that both have weaknesses. The dot product correlation proposed by Levine et al. is consistently successful only on one of the three traffic types we tested, and the Bayesian correlation proposed by Murdoch and Zieliński has no reliable way of avoiding false positives unless the attacker controls all or almost all of the network. We also designed and tested a new simple correlation algorithm, which can reliably correlate all three traffic types, and which is much more computationally efficient than either of the other two methods. While our results are based only on data collected from a single router, we also explained how our algorithm might scale for attackers controlling many more routers.

With our new simple correlation algorithm, we are able to answer both of our original questions in the affirmative. It is possible and feasible to perform passive correlation attacks on Tor using our algorithm, and our algorithm is capable of performing correlation even when very small amounts of data are being sent over Tor.

Currently, however, our work does require the attacker to directly control Tor routers, because this gives them access to much more precise information and allows them to work with Tor cells rather than TCP packets. This is problematic

because an attacker would need to control a lot of routers in order to compromise a significant amount of the traffic going over Tor. One area for future work would be to see whether this limitation can be lifted: is observing all TCP traffic into and out of a router as powerful as controlling the router directly? If so, this might allow an ISP to effectively compromise any Tor routers they provide internet service to, which, in turn, would possibly allow a government the ability to effectively compromise all routers within its country's borders.

Bibliography

[1] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

[2] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of CCS 2007*, October 2007.

[3] Roger Dingledine and Nick Mathewson. tor-spec.txt. `https://gitweb.torproject.org/torspec.git/blob_plain/3b5b8804f64a4db7ec7fc0185ea1afb7a2713797:/tor-spec.txt`, March 2011.

[4] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, August 2004.

[5] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security*, 13(2), February 2010.

[6] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing attacks in low-latency mix-based systems. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.

[7] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195. IEEE CS, May 2005.

[8] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In Nikita Borisov and Philippe Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, pages 92–102, Ottawa, Canada, June 2007. Springer.

[9] Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P. Markatos. Compromising anonymity using packet spinning. In T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, editors, *Information Security: Proceedings of the 11th International Conference, ISC 2008 (Taipei, Taiwan)*, volume 5222 of *Lecture Notes in Computer Science*, pages 161–174. Springer-Verlag, 2008.

[10] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. Tracking anonymous peer-to-peer voip calls on the internet. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 81–91, November 2005.