INVESTIGATING THE TOR NETWORK USING GENI

T H E S I S

for the Degree of

BACHELOR OF SCIENCE

(Electrical Engineering)

AT

NEW YORK UNIVERSITY

TANDON SCHOOL OF ENGINEERING

TAKUYA FUKUI

May 2017

INVESTIGATING THE TOR NETWORK USING GENI

THESIS

Submitted in Partial Fulfillment

of the Requirements for the

Degree of

BACHELOR OF SCIENCE (Electrical Engineering)

at

NEW YORK UNIVERSITY

TANDON SCHOOL OF ENGINEERING

by

Takuya Fukui

May 2017

Copy Number ___

Approval by the Guidance Committee:

Major: Electrical Engineering

Shivendra Panwar

Department Chair,

Electrical and Computer Engineering

Director, CATT

Date

Fraida Fund

Co-Advisor

Electrical and Computer Engineering

Date

VITA

Takuya Fukui was born on October 2, 1995 in Queens, New York. He attended Townsend Harris High School in Queens, New York, a top high school in the country, although mostly known for their excellence in their Humanities programs. After four years of high school at a Humanities school, he knew that studying humanities was not the path for him.

From the beginning, Takuya chose to pursue Electrical Engineering at New York University's Tandon School of Engineering (NYU-Poly at the time), and he stuck with his decision for the next four years there. During his junior year at Tandon, he took a class in Network Communications, from which he felt that he finally found something that really sparked his interest. After having taken that class and being that it was time to start thinking about applying for the BS Thesis course, he conferred with his academic adviser Ellen Daniels, who recommended speaking with Professor Panwar and Fraida Fund of the Center for Advanced Technology in Telecommunications (CATT). Upon discussion of possible topics of research in their department, they became his thesis advisors.

After being given various topics concerning network communications, Takuya chose to study further on Tor, the popular present-day anonymity network. With his advisors' constant help, he was able to complete his research and writing of the thesis over a span of two semesters of senior thesis credit.

DEDICATION

To those who gave me so much support:

My Parents

Yoichi Fukui and Kaori Fukui

My Siblings

Tomihiro, Rei, Michinori, Yukimi, and Daichi

My Thesis Advisors

Shivendra Panwar and Fraida Fund

All of my friends who give me energy each and every day

ABSTRACT

INVESTIGATING THE TOR NETWORK USING GENI

by

Takuya Fukui

Advisor: Shivendra Panwar, Ph.D.

Co-Advisor: Fraida Fund

Submitted in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Science (Electrical Engineering)

May 2017

The Onion Router (Tor) provides anonymity services to over 1.5 million users daily. Meanwhile, attackers are constantly trying to breach the Tor network, while at the same time researchers are trying to find ways to defend from these attacks. It is often helpful when creating defenses for Tor to test on a small private Tor network before applying it to the real world, yet there exists only a few accessible methods of creating a private Tor network. Using GENI, an open infrastructure for distributed systems research, I have shown how researchers can set up a private Tor network for experimenting with Tor, using a website fingerprinting attack as a case study. This thesis provides a step by step process of setting up this private network and fingerprinting attack. Being able to create your own private Tor network and carry out an attack in that controlled environment allows one to see the functionality of Tor's anonymity process and the present weaknesses of Tor. I would like my work to be used as a reference for other researchers and students working with Tor, for them to be able to set up their own private Tor network for identifying attacks and creating defenses for the network.

Table of Contents

List of Figures and Graphs

Chapter 1

Introduction

1.1 Overview

As the Internet continues to become more expansive and accessible, attacks on people using the Internet are also increasing. Tor is an anonymity network that is volunteer-based, where users of Tor can become part of the network to increase its size and randomness. The Tor network is a system of relays that passes around a client's traffic before sending the traffic to the client's actual destination. While a client's traffic traverses through the Tor network, the origin of the traffic becomes obfuscated, and the final destination is only known at the exit point of the Tor network. Also while in the Tor network, client traffic is encrypted. In this way, Tor protects users from attackers who spy on a network and try to steal a user's information.

Onion routing research began in 1995 by David Goldschlag, Michael Reed, and Paul Syverson, with one goal in mind: to separate identification from routing [1]. Authenticating one's identity is done through the data that is passed in the data stream, and so one's location does not need to be known to the network.

Today there are over 6000 Tor relays inside the Tor network, serving over 1.5 million users every day [2]. There are many kinds of Tor users with different goals that they wish to achieve. One group of users is law enforcement and intelligence agency personnel, who must stay anonymous when browsing the Internet to investigate a certain case. Their actions cannot be traced; therefore anonymous routing is the best solution.

Another group of users stems from those who want to voice their opinion on a certain topic without revealing their true identity. Tor provides a means for people to discuss and post on the Internet without a risk of exposure. Then there are other more ordinary people who simply want to make sure that their information is protected. Simply limiting the amount of information that can be seen by a person spying on the network is the main goal that people wish to achieve from using Tor [1].

The Tor network has a wide variety of users, which is essential to the anonymity that is provided by onion routing. For example, using cloaking software exclusively by the US Navy would conversely decloak these users, because an attacker watching the Tor network would know that anything that goes in or out of the network would be by the US Navy. The US Navy required as diverse of a community of people as possible to use Tor in order to hide themselves behind everyone. This was an essential part in fulfilling their primary objective of cloaking the identity of government and intelligence personnel, which was why Tor became an open source consumer product that everyone would be able to use [1].

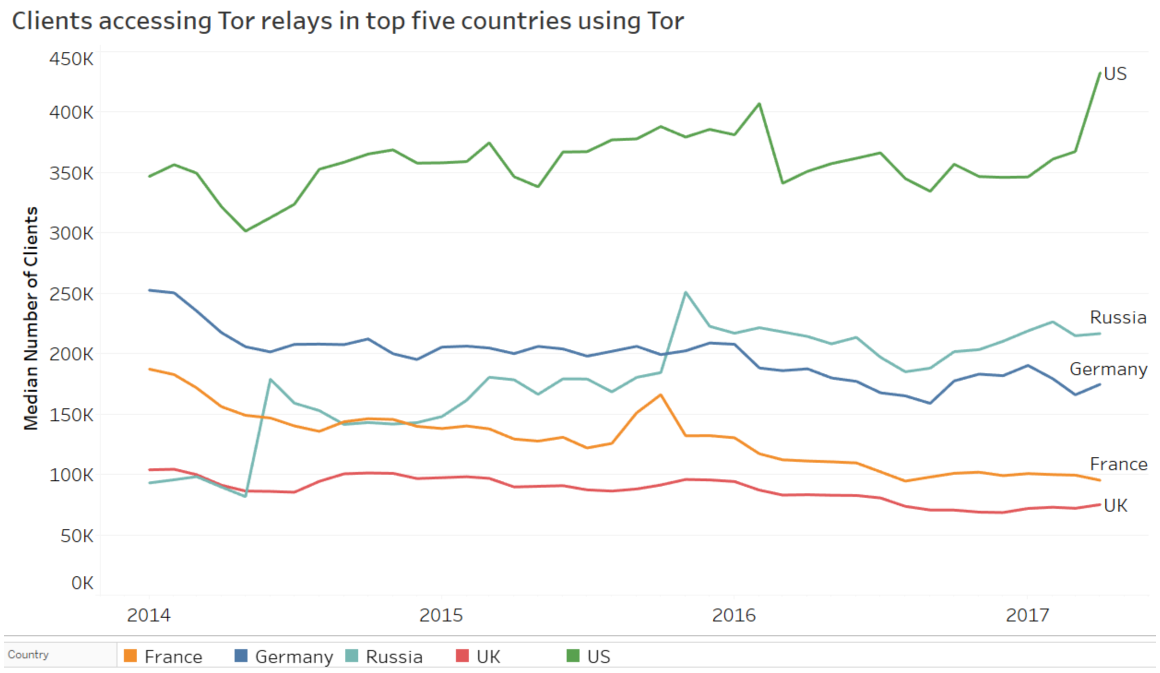


Figure 1: *Graph of number of users vs. date for the top five countries using Tor* [2]

Figure 1 shows the number of Tor clients averaged per month for five countries for the past four years. The countries for whose line chart is shown are the top five countries that are currently using Tor. We can see that all five countries have a couple of hundred thousand users daily. On Russia's chart, in mid-2014 we see a sharp increase in the number of users, from 100K to 200K. Although no single event can be said to be the cause of the sudden increase in users, but the fact that the Russian government had passed censorship legislation during that time plays a large part [3]. In May of 2014, President Putin signed a law requiring registration of bloggers who had more than 3,000 daily views, which further tightened the restriction that the Russian government has over people on the Internet [3].

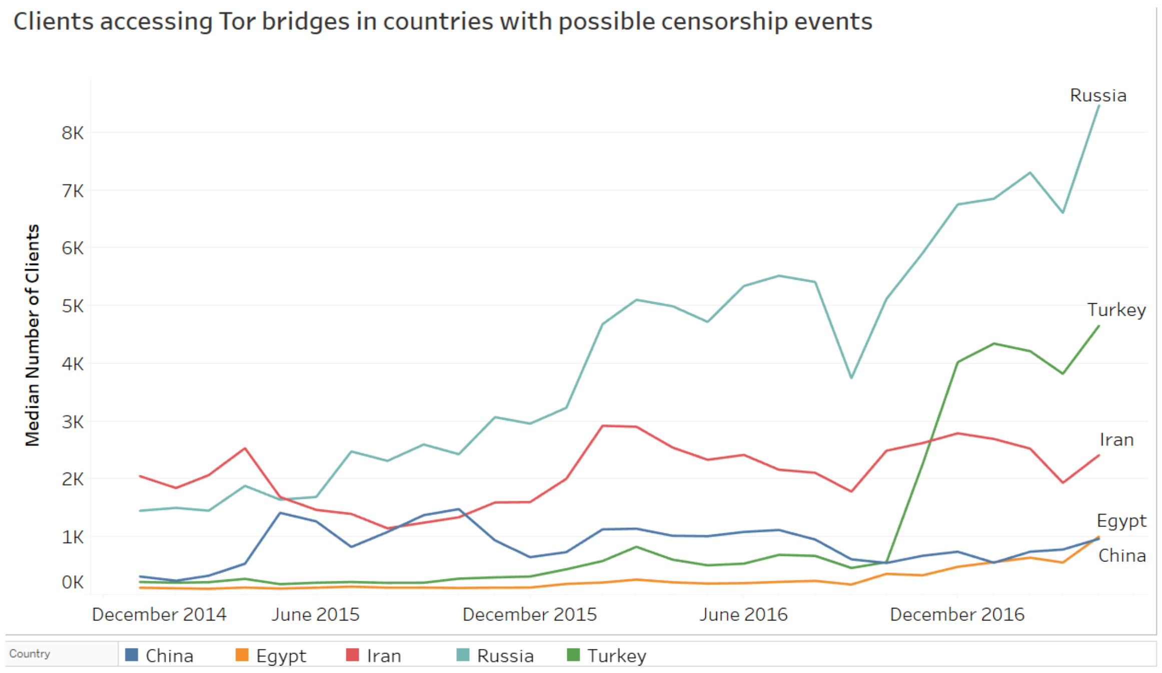


Figure 2: *Graph of clients using Tor bridges in censorship countries* [2]

Figure 2 is a graph of countries with many users who access Tor through bridges. Bridges are Tor relays that are not listed in the main Tor directory [4]. Since bridges are not listed on any public lists, ISPs that that try to block connections to Tor relays will not be able to block bridges [4]. Bridges are ideal for people who live in countries that try to block connections to Tor relays. Jardine's paper on the relationship between political repression and anonymity services [5] shows that anonymity service usage increases when there are low and high levels of political repression, and decreases at medium levels of political repression. Figure 1 shows high Tor usage in liberal countries with low levels of repression (as well as increased use in Russia in response to a possible censorship threat), and Figure 2 shows high Tor bridge usage in countries of high repression with possible censorship events. Tor continues to become a popular tool for people in countries with strict Internet censorship, as well as in liberal countries.

1.2 Attacks and Defenses on Tor

The Tor network's security is never left alone, where attackers are constantly trying to breach the system while the developers are constantly thinking of new ways to strengthen the system. Developers encourage research that involve ways to bring down Tor's anonymity, because it helps them find weaknesses to the system and consequently to fix these holes.

Research is done on two kinds of Tor attacks: active and passive. An active attack is when the adversary modifies the data or the network. An active attack may involve one's own set of colluding relays to carry out the attack. A passive attack involves monitoring a network, and using data-analysis techniques to carry out the attack.

One active attack on Tor is the Low-Resource Routing Attack described by Bauer et al. [6]. In this attack, the adversary controls a large number of low-resource Tor relays that falsely advertise high bandwidth capabilities. Tor relay selection algorithms bias its selection on high bandwidth relays, therefore the user is more likely to choose these adversary relays and thus more prone to being attacked. Other active attacks include the Sniper Attack described by Jansen et al. [7] and the Selective Denial of Service (SDoS) attack described by Borisov et al. [8].

The Website Fingerprinting attack, described by Hermann et al. [9], is an example of a passive attack. This experiment involves capturing packet traces of certain websites that an attacker believes a user often visits, and then creating a fingerprint of these websites. The attacker then snoops on the user's network and listens to and captures the user's traffic. The attacker compares the capture with the fingerprints to see if there is any kind of correlation in order to identify which website the user was visiting. Other passive attacks include the AS-Level Adversary Attack by Edman et al. [10] and the Fingerprinting on Hidden Services Attack by Overlier et al. [11].

1.3 Methods of Researching Tor

Various methods exist of evaluating and measuring Tor networks in order to perform research of the type described in the previous section.

It is often useful to use simulations of a system or a small, private version of that system when conducting research and test experiments. The Tor Path Simulator (TorPS), by Johnson et al. [12], is a path selection module that uses past server directory and consensus files to recreate the conditions in which a user was running Tor. TorPS then runs path selection algorithms under these conditions to see what kinds of circuits would be created based on the user's actions [12]. A major advantage of using TorPS is that it provides the rate of path compromise of a certain scenario of malicious nodes and client behavior. These rates allow us to see how certain client behaviors while using Tor can be more vulnerable to compromise by an attacker. For example, Murdoch et al. [13] used TorPS to run experiments that explore the behavior of various path selection algorithms, and were able to show that anonymity increases when an attacker has a large number of nodes that have low bandwidth [13]. A noticeable disadvantage of using TorPS is that since it is a simulation using past data, it is not an experiment on the dynamic, live Tor network.

Another useful program called Shadow [14] runs the Tor software over a simulated network layer. Shadow provides for a controlled and repeatable method of Tor experimentation as opposed to using the live Tor network. An advantage of using

Shadow is its scalability in the simulated Tor network size, which can go up to the thousands in the number of nodes. Jansen et al. showed that Shadow simulation results were similar to the live Tor network. However, some aspects of the network environment may not be fully captured by a network simulation.

Experiments on the live Tor network have been performed as well, for example in the paper by Hopper et al. [15]. Their experiment allows colluding websites to predict whether two connections from one exit node are using the same circuit or not. Experiments on the live Tor network provide greater insight into the actual behavior of Tor in real time, compared to other methods. However, a major disadvantage of these experiments are the limitations in router modifiability and potential effects on other Tor users [16]. For example, testing a proposed improved router selection algorithm can be easily changed on all routers of a private Tor network, however cannot be changed on routers of a live Tor network other than the ones that you control. Experiments on the live Tor network such as large-scale deanonymization attacks pose actual risk to the anonymity of real Tor users, which becomes a legal and ethical issue [16].

Some researchers use isolated Tor networks on a testbed. In the Low-Resource Routing Attack experiment by Bauer et al. [6], an isolated Tor network is created on the PlanetLab testbed. However, a major shortcoming of PlanetLab is the lack of reproducibility of the experiment, because traffic goes over the commodity Internet [16]. A lack of resources and varying available bandwidth of PlanetLab can make it difficult to expand one's private Tor network to fit one's needs. Alternatively, researchers may set up dedicated Tor testbeds over private networks [16], but this approach does not scale well.

Recently, large-scale networking testbeds such as GENI [17] have become widely available that allow researchers to conduct Internet-sized experiments on a programmable, tightly controlled network. GENI provides the resources to set up a private Tor network. Compared to a PlanetLab test lab, GENI allows the user to have total control over all of the nodes of the network. GENI has a very wide availability of resources, and so the network that is created can be very large, compared to setting up an in-house laboratory network. Although a private Tor network on GENI is not completely live, the user has more control over the nodes than the live Tor network, and is more realistic than a simulation testbed such as Shadow.

1.4 Objectives

My research in Tor involved creating a private Tor network and website fingerprinting attack using GENI. GENI is an open infrastructure for distributed systems research and networking [17]. For research, GENI is well suited for large-scale experiments because of its easy programmability and wide availability of resources such as virtual machines, switches, and links. For educational purposes, GENI becomes a very handy tool due its ease of use in giving students a hands-on learning experience on large-scale network projects.

This thesis describes how to set up a private Tor network on GENI, and how to use it to demonstrate a website fingerprinting attack. This experiment shows how GENI can be used for research on Tor, and also provides a starting point that other researchers can use in their own experiments. The experiments described in this thesis are also published on the “Run My Experiment on GENI” blog [18], [19], a repository of reproducible experiments that run on the GENI testbed.

1.5 Outline

The rest of this thesis is organized as follows. In Chapter 2, I give an overview of Tor’s security, and the basic functions of Tor. Then I go deeper into the specifics of how the anonymity works: examining keys, cells, connection methods, and the very important directory authority. In Chapter 3 I give a step by step explanation on how to set up the private Tor network that I used for my research. I provide a topology of what it should look like on GENI, and provide the code for setting up the experiment. Once the setup is complete I describe the experiment that I created to test the Tor network, first without Tor and then with Tor to see the difference in the security. In Chapter 4, as a preamble to the website fingerprinting experiment I give an overview of the different kinds of ways that Tor’s anonymity has been and still can be breached. In Chapter 5, I provide a step by step explanation on how to set up the website fingerprinting attack on our private Tor network, along with instructions for visualizing the results. Lastly, Chapter 6 is a conclusion to wrap up my findings on the research and to describe future steps for these experiments.

Chapter 2

Overview of Tor's Security

2.1 Tor's Role in Providing Anonymity

Tor provides anonymity to its users by making sure that no router along a path in the Tor network knows both the user's location and the site they are visiting. Let us examine the kinds of protection that Tor provides to its users. Tor can run in conjunction with HTTPS, so we will examine the security benefits provided by Tor and HTTPS, both separately and when both are used in conjunction. Figure 3 is an example of a simple network that we will be observing to further see the capabilities of Tor.

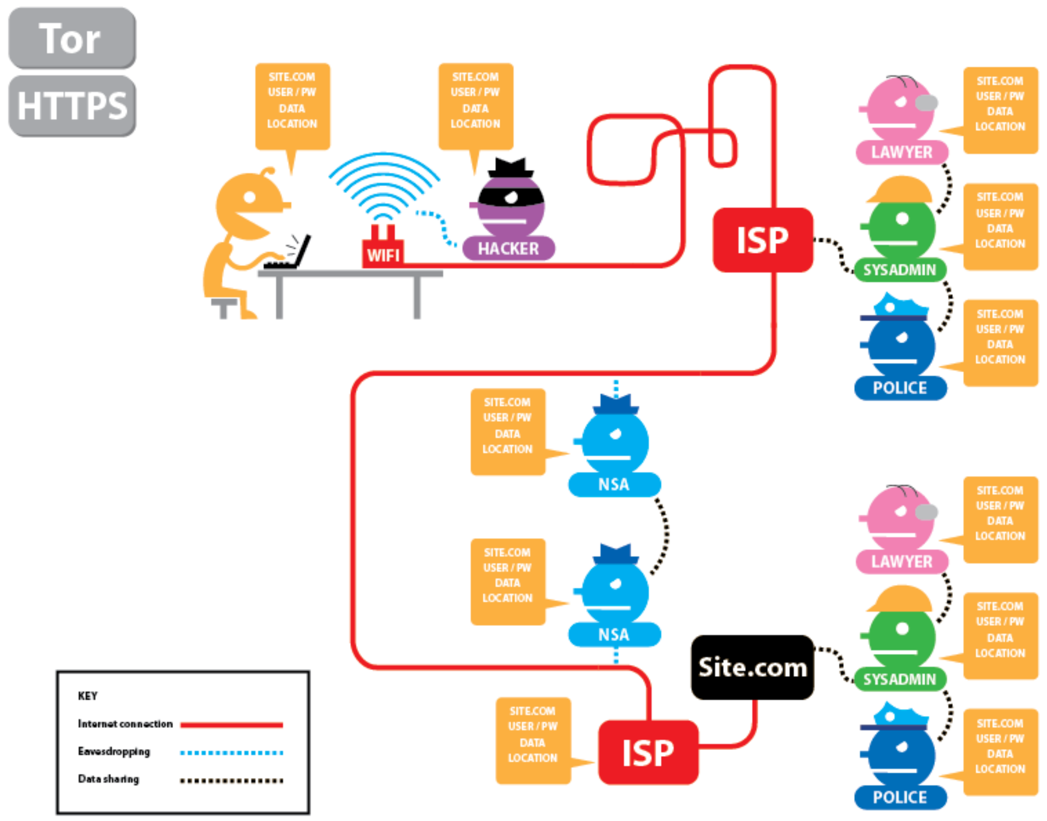


Figure 3: *Simple network with the user using neither Tor nor HTTPS* [20]

Figure 3 shows a number of users/nodes with different roles:

- User: the person using the network
- Hacker: a malicious user who has gained access to the user's network
- NSA: a U.S. government agency
- ISP: the user's Internet service provider
- Site: the website, application, or other online resource that the user is accessing

Those who have data sharing agreements either with the ISP or the site that the user is accessing are also shown. The user whose network is being observed by the hacker is the person on the laptop in Figure 3: *Simple network with the user using neither Tor nor HTTPS*, and the destination web server that the user is trying to access is the black box named Site.com. The yellow box next to each node represents what information about the communication the node can see. We can see that in Figure 3, every node can see all the information that is being exchanged between the user and the website: the site that the user is accessing, the username and password (U/P), the payload/data that is being exchanged, and the location of the user. This scenario is when neither Tor nor HTTPS is being used by the user to access the website. Such a case is ideal for a hacker because they can steal the user's information. In some circumstances, an ISP may share data with other entities, such as law enforcement agencies. In this case where neither Tor nor HTTPS is used, they can also see all of the user's information. Next we have the NSA (National Security Agency) which is able to monitor large parts of the Internet through its partnerships with many different telecom firms and ISPs [21].

Figure 4 is an example of using HTTPS to access a website:

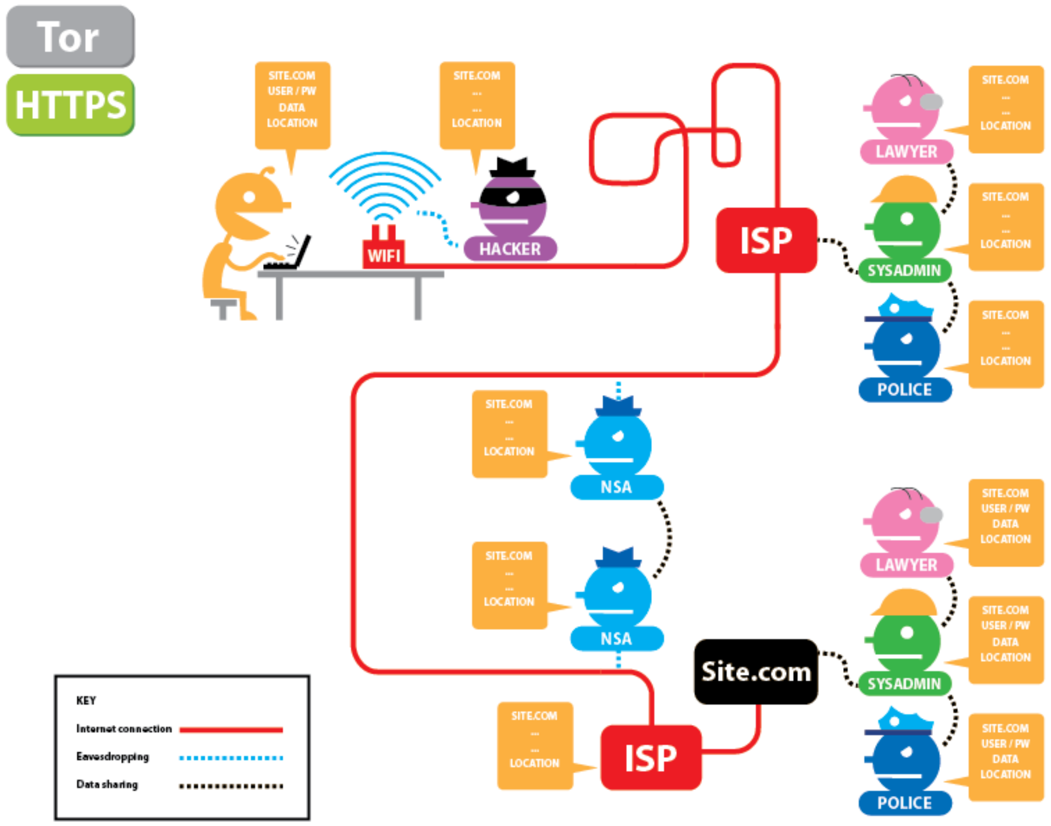


Figure 4: *Simple network with the user using just HTTPS* [20]

The user, website, and those that have data sharing agreements with the website still can see all the information. This is expected because the user and the site are the ends of the network, and HTTPS is responsible for encrypting the information that travels between both ends. Due to this encryption, the rest of the network can no longer see the U/P, or even the traffic. Even with data sharing agreements on the ISP, these nodes cannot decrypt the information. However, because packet headers are visible throughout the network, this information can be used by any of the players listed above to link the user with the website that he/she is visiting.

Now let us examine what happens when we only use Tor to access a website:

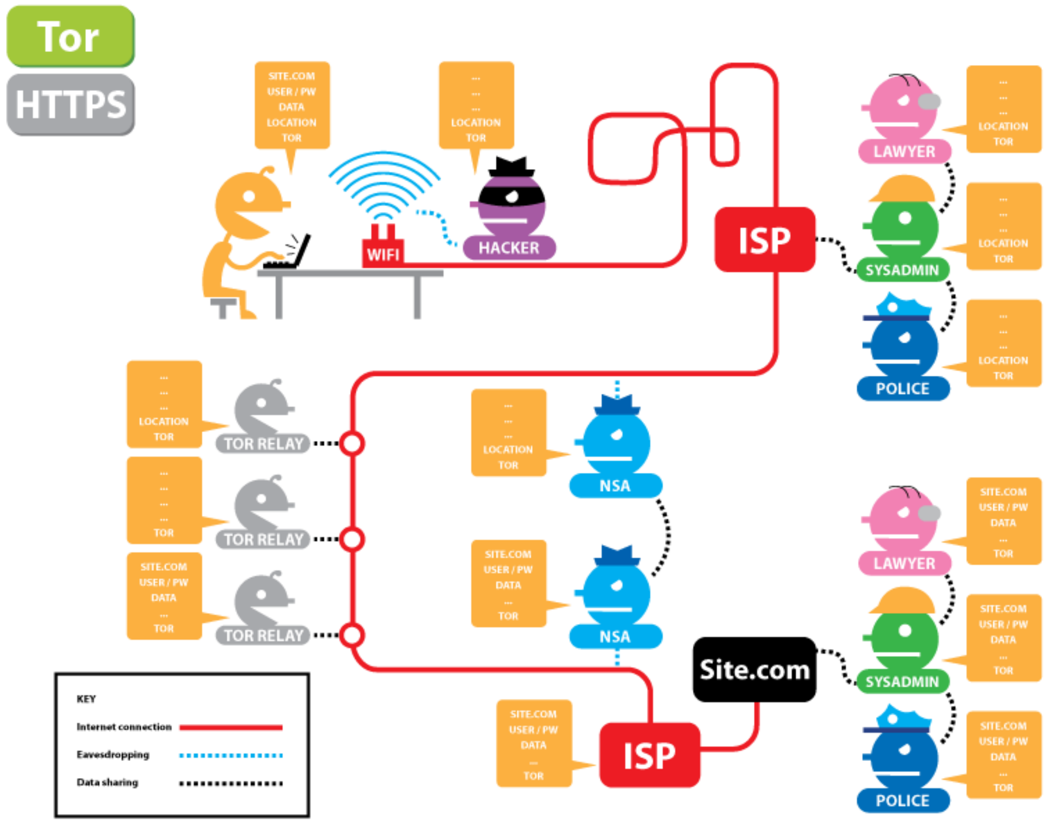


Figure 5*: Simple network with the user using just Tor* [20]

We now observe that in between the two IPSs are three new nodes, which represent three Tor relays. The hacker, the nodes sharing data with the first ISP, and the first NSA node can only see the location of the user sending traffic. They do not know the U/P, the data, nor the website. Why is this the case even when the user is not using HTTPS? Tor encrypts information in a way that only Tor relays can decrypt. Nodes before the first Tor relay do not know which site the user is accessing because the destination is initially the first Tor relay in the circuit. Each relay in the Tor network only knows the certain amount of information that they can decrypt, which is why it becomes harder for hackers to figure out who is communicating with whom [22]. The last Tor relay knows the U/P, the data, and the site to access because it is the exit relay's job to send the data to the site. At this point the traffic exits the Tor network and is no longer encrypted. All information can be seen by the nodes that are eavesdropping on the network, except for the user location. After having been bounced around the Tor network, the traffic no longer contains the original user's location, which makes it difficult to match which user is accessing the site. However since the U/P and data can still be seen, this information can be stolen which is a weakness of using Tor alone, without end-to-end encryption.

We resolve the problem in which someone that eavesdrops on the second ISP has access to user information, by using Tor in conjunction with HTTPS, shown in Figure 6.

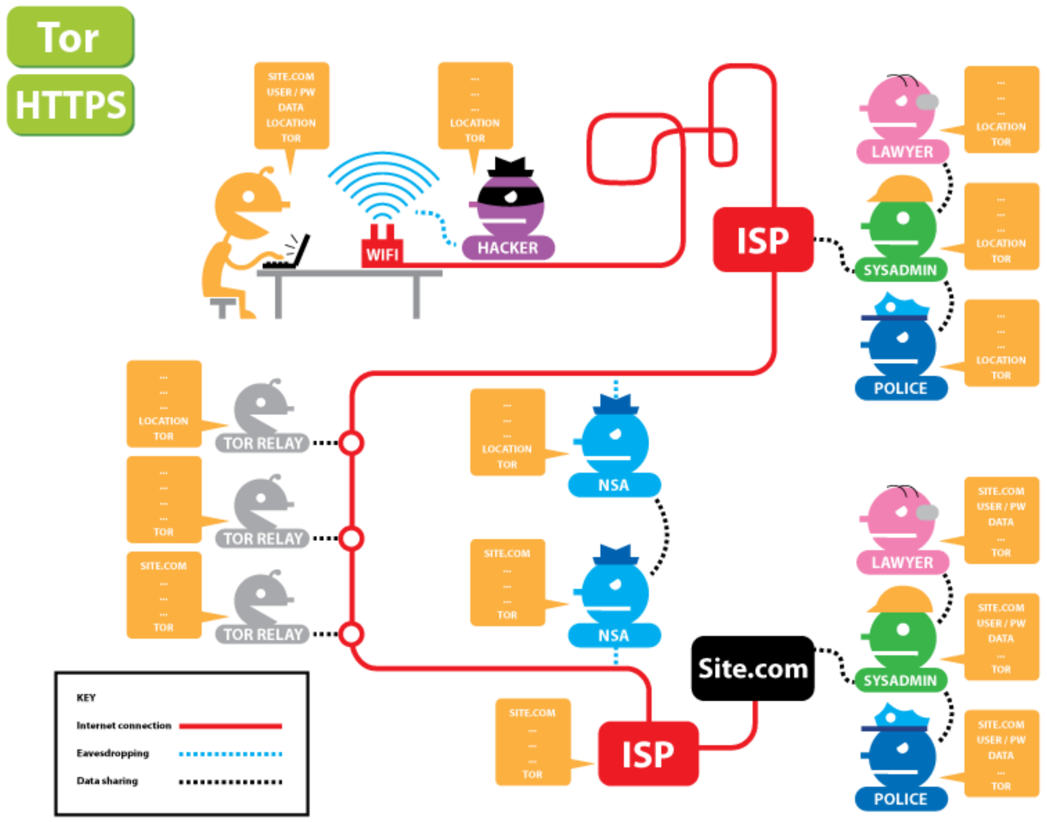


Figure 6: *Simple network with both Tor and HTTPS* [20]

The information that hackers located before the entry Tor node can see is still the same, which is very limited. The biggest change that we can see is that hackers eavesdropping on the second ISP can no longer see the U/P, as well as the payload. This includes the exit relay as well. This is the case because the original traffic being sent from the user is now encrypted with both Tor and HTTPS. When the traffic reaches the exit relay and leaves the Tor network, it is no longer encrypted by Tor, but is still has end-to-end encryption from using HTTPS. The exit relay still knows the site to access because it needs access to the final destination in order to deliver the HTTPS encrypted traffic to the site. The website and the nodes sharing data with the site clearly know the U/P and data because the traffic's HTTPS encryption is decrypted at the destination. However the location of the original user is still unknown to those nodes, maintaining privacy and security. Using Tor on top of HTTPS provides security greater than using neither or using either one alone [20].

The capabilities of Tor in protecting and cloaking a user from hackers and eavesdroppers can be seen from this example. Figure 7 summarizes which nodes can see what information.

	Neither	**HTTPS only**	**Tor only**	**HTTPS + Tor**
Site	Everyone	Everyone	Exit Node ←→Site + Data Sharing	Exit Node ←→Site + Data Sharing
User/Pass	Everyone	User, Site + Data Sharing	Exit Node ←→Site + Data Sharing	User, Site + Data Sharing
Data	Everyone	User, Site + Data Sharing	Exit Node ←→Site + Data Sharing	User, Site + Data Sharing
Location	Everyone	Everyone	User ←→Entry Node	User ←→Entry Node

Figure 7*: Summary of the Tor Network's Capabilities*

2.2 Diving Deeper into Tor

Now that we know what Tor is and the benefits of using Tor to protect our privacy and security, let us examine what goes on behind the scenes of the Tor network. The Tor network consists of onion routers (OR), or nodes that a Tor user's traffic passes through before reaching its destination. There are three kinds of ORs: entry, relay, and exit node. The entry node is where the user's traffic first enters the Tor network. The exit node is conversely where the traffic exits the Tor network. Any nodes in between are relay nodes that simply pass the traffic from one node to the next. Information traveling through the Tor network is encrypted in a way that only Tor relays can decrypt. This encryption begins with the user running the Tor software's proxy called the onion proxy (OP). The OP then accesses a special onion router called the directory server, a certain reliable node that provides directories of known routers, including their location and current state. Using the directory the OP determines which routers are going to be used, negotiates the encryption keys for each router, and encrypts the packet accordingly [22]. In the rest of this section, we will describe in detail how Tor relays set up connections and pass data through the network.

In section 2.2.1 we will discuss Keys, which are vital security measures of authorization in the Tor network, to maintain the anonymous routing protocols of Tor. In section 2.2.2 we examine Cells, which are the set-sized packets in which Tor encrypts packets. We discuss the commands that are necessary for relaying data, as well as establishing and tearing down connections. In section 2.2.3 we take a look at the different kinds of handshakes that are required to establish a connection between two nodes in the Tor network. In section 2.2.4, we examine a special kind of OR called the directory authority (DA), which is a certain more trusted, reliable OR that contains a list of known ORs in the network, including their location and current state.

2.2.1 Keys

Keys used in the Tor network are the vital security measures of establishing connections, sending data, and updating directory authorities. A key is a form of authentication which provides a means to encrypt as well as to decrypt information. Keys allow safe communication of data, making sure that information being sent is only decrypted by the nodes that it was intended for.

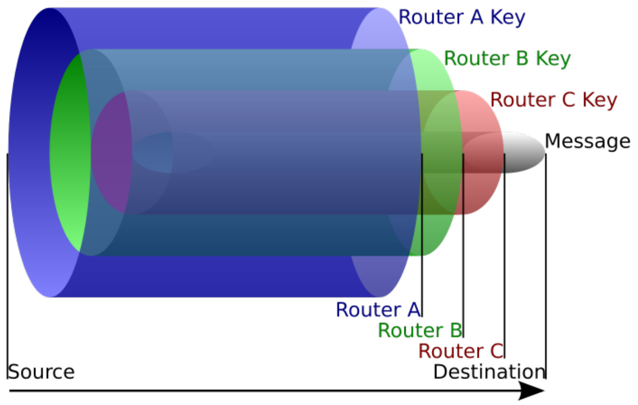


Figure 8: *Onion Routing Diagram with Encryption Keys* [23]

Figure 8 is a diagram of the various layers of Tor's onion protocol. Every packet is wrapped with multiple layers of encryption, each of which can only be decrypted by a specific key, which belongs only to the relay that will transfer the packet at that stage in its path. The information that is decrypted at each node is very limited, where there is no one node that knows the whole mapping of the circuit.

Keys are also used to authenticate nodes in the Tor network. The Tor network is constantly changing, and there needs to be some kind of mechanism to constantly update the DAs. This is where keys come into play. A DA has two kinds of keys: authority identity key and authority signing key [24]. In order to keep information about the DA up to date, it must sign directory information about itself periodically, which is done by the authority signing key. The authority signing key is not a permanent key, and is replaced around every 3-12 months in order to minimize the risk of getting these keys stolen by a hacker [25]. In order to authenticate the signing key, a DA has an authority certificate [24]. This certificate must also be authenticated, which is done by the DA's authority identity key. This identity key is a long term key which identifies that the DA is not a fake DA. In section 3.1.4, we can see the generation of these keys to set up a DA for our private Tor network.

All other ORs follow a similar process. An OR has three kinds of keys: a secret id key, a secret onion key, and a secret onion key ntor [24]. The secret id key is similar to the authority identity key, and is used to sign the router's descriptor, TLS certificates, and to sign directories. A router descriptor contains the specifications of that router, including its keys, location, bandwidth, and exit policy [24]. Directories need to be updated constantly so that the directory authority can provide up-to-date information to the OP and client. The secret onion key is used when establishing a Tor circuit to pass traffic along its network. These keys are not permanent unlike the identity key, and are changed often to prevent compromise. The onion key also creates short-lived keys used to access TLS connections to communicate with one another and the user [24]. The secret onion key ntor is a short-term key used specifically for the opening of a Tor circuit, when a three-way handshake is required. The fingerprint is a hash of the identity key, used so that the identity key's location and security is preserved [24]. In section 3.1.5, we will see how to generate these keys to set up each router for our private Tor network.

2.2.2 Cells

ORs pass traffic along the Tor network in fixed-size cells/packets of 512 bytes. Each cell is encrypted in many levels, where these levels are decrypted by a key at each OR until it reaches the exit node where it is then sent to the website. There are two main types of cells: control cells and relay cells [22].

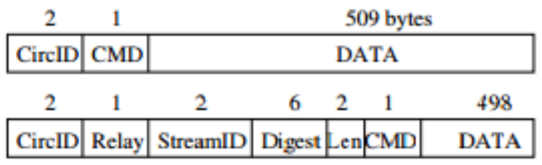


Figure 9*: Structure of a basic Tor cell* [22]

Figure 9 shows the structure of a typical Tor cell, where the top is the control cell structure and the bottom is the relay cell structure. The control cell contains three parts: CircID, CMD, and DATA. CircID is the circuit identifier which specifies the circuit that is being referred to. CMD is the command to be done, and DATA is the payload, which contains specific instructions for the command. There are three main types of control commands for creating a connection: PADDING, CREATE/CREATED, and DESTROY [22]. PADDING is a command used to keep a connection alive. The CREATE, CREATED, and DESTROY commands are used to build and break down a circuit. A CREATE cell's payload contains information on the handshake that is to be used to create a new connection. Relay cells have an additional relay header to identify that they are not command cells. Their header also contains a stream ID, a checksum, and the length of the relay's payload. Unlike command cells, relay cells carry end-to-end stream data. The main commands for a relay cell for managing a connection are the following: RELAY BEGIN/CONNECTED, RELAY END, RELAY TRUNCATE/TRUNCATED, RELAY EXTEND/EXTENDED, and RELAY DATA [22]. RELAY BEGIN is a command to open a stream, and RELAY CONNECTED is to acknowledge this opening. Notice that these commands can only be called once the Tor circuit has been created and there is a connection between the client and the website that is to be accessed. RELAY END is the command to close a stream. RELAY TRUNCATE tears down a part of the circuit and TRUNCATED acknowledges this. RELAY EXTEND does the opposite and tells the end of the circuit to extend by one hop, and EXTENDED acknowledges this. Lastly, RELAY DATA is a command for flowing data along a Tor circuit, which can be done only after a stream is opened [22]. (In section 5.2.3, for our website fingerprinting experiment, the process of creating a fingerprint requires rounding these 512 byte size cells to 600, to create a more useful fingerprint.)

An important process in creating connections is when we need to extend or truncate a circuit. In order to extend a current circuit by one more hop, the OP first sends an RELAY EXTEND cell which is passed along until the last node in the circuit, in which at that point the node is instructed by the OP to send a CREATE cell to extend the circuit to the specified new node [22]. The payload of a RELAY EXTEND cell also contains the handshake instructions of the payload of a CREATE cell. The only additional components of the payload are link specifiers that describe the next node to connect to [22].

2.2.3 Initialization of Connections

Before a connection between two nodes is created and opened, a series of package exchanges must take place to negotiate authenticity of the nodes. This process is called a TLS handshake, where there are three kinds: certificates-up-front, renegotiation and in-protocol [26]. In section 3.1.5, we can see an example of a TLS handshake that takes place when setting up the Tor relays.

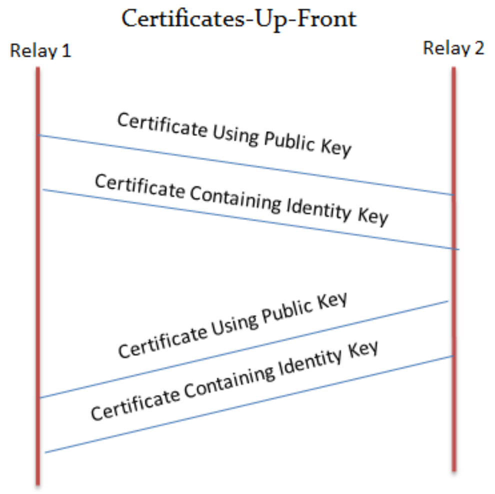

Certificates-Up-Front
Relay 1
Relay 2
Certificate Using Public Key
Certificate Containing Identity Key
Certificate Using Public Key
Certificate Containing Identity Key

Figure 10*: Packet exchange of a Certificates-Up-Front handshake*

The certificates-up-front handshake is the simplest TLS handshake that is supported by all versions of Tor. In this process, both sides of the link must send each other a two-certificate chain [26]. This chain consists of a certificate using a short-term connection public key and a self-signed certificate that contains its identity key. This process is the simplest of the three and securely allows both sides to confirm each other's' availability.

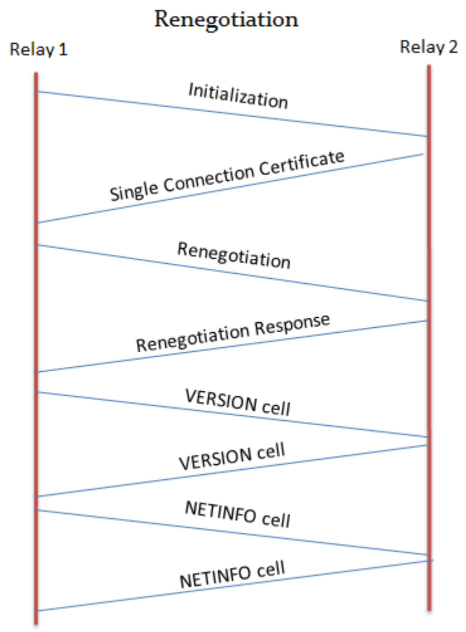


Figure 11*: Packet exchange of a Renegotiation handshake*

For renegotiation and in-protocol handshakes, the following control cell commands are required: VERSIONS, CERTS, AUTH_CHALLENGE, and AUTHENTICATE [26]. After a node initializes the handshake, the responding node sends a single connection certificate as the initial TLS connection. Following this a renegotiation is performed, similar to the certificates-up-front handshake [26]. Both nodes then send a VERSION cell to negotiate which link protocol version will be used. After this is set, both sides send a NETINFO cell to confirm each other's' location and timestamp, which is the local clock of the node [26]. A NETINFO cell also contains address locations of other ORs that are known by that specific OR.

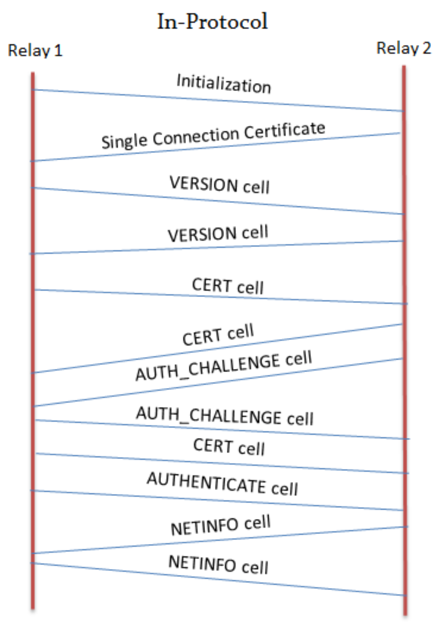


Figure 12*: Packet exchange of an In-Protocol handshake*

An in-protocol handshake requires a few different steps from the renegotiation handshake. After the initial TLS connection, renegotiation is skipped, and instead both sides send a VERSION cell to negotiate the link protocol version [26]. Then the initiator sends a CERT cell, which contains a list of keys and certificates that a node claims to have in possession. The responder sends back its own CERT cell, followed by an AUTH_CHALLENGE which is a request by the responder to authenticate the initiator. The initiator responds with an AUTH_CHALLENGE cell, a CERT cell, and an AUTHENTICATE cell which contains the authentication [26]. Both nodes then send each other a NETINFO cell containing its location and timestamp.

2.2.4 Directory Authority

In an older version of Onion Routing, the state of each router was updated through flooding, where each router sent a status update to its neighbors, which spread to other neighbors [22]. However this design had delays causing different views of the network by each router. From there a new strategy was developed in which a group of reliable and well-known ORs called directory authorities (DA) would be responsible for keeping an updated directory of the states and locations of all Tor routers [22]. Each DA also acts as an HTTPS server for clients to gain the most up-to-date states of the Tor network [22]. The DA does not add ORs who do not have a proper identity key, which prevents attackers from creating fake nodes.

The DA itself can also be vulnerable to attacks by an adversary, which makes the network vulnerable to fake directories of router locations. In order to prevent this, the DAs in the network must be synchronized with one another, agreeing to the same common directory [22]. Clients must make sure that a directory is signed by a number of DAs to consider it trustworthy, a process called consensus. The DAs come to a new consensus every hour and it is frequently checked to make sure that it is not outdated [25].

Chapter 3

Toy Experiment Using Tor on GENI

3.1 Setting up a Private Tor Network

One way to learn about Tor is to create a private Tor network, and to see the functions of Tor on a smaller and more manageable scale. By setting up a private Tor network, we will be in control of our own client, routers, directory server, and web server. The advantages and disadvantages of this approach to studying Tor are described in section 1.3.

Once we are done setting up the private Tor network on GENI, we will be able to see the two ways in which a client's packet can be sent to the web server:

1. Through a direct link between client and web server
2. Through the Tor network

We will be testing both cases to see what kind of information we can see when listening on different parts of the network. We will be looking for the following three pieces of information for these tests:

1. Whether we can see the client address
2. Whether can see the web server address
3. Whether we can see the data packet's contents

For both cases, we will be able to see that not all information can be seen at every part of the network. A version of this experiment with more explicit instructions for each step is published on the Run My Experiment of GENI blog [18]. (The process of setting up the private Tor network is based partly on an experiment by Liu Fengyun [27].)

In the following sections we will first discuss how to get started with GENI, reserving our topology for the private Tor network and installing Tor on each of the nodes in this topology. Then we will go through the steps of setting up each of the nodes of the network, including the directory server, web server, routers, and client.

3.1.1 Reserving our Topology on GENI

Figure 13 illustrates the GENI topology for this experiment:

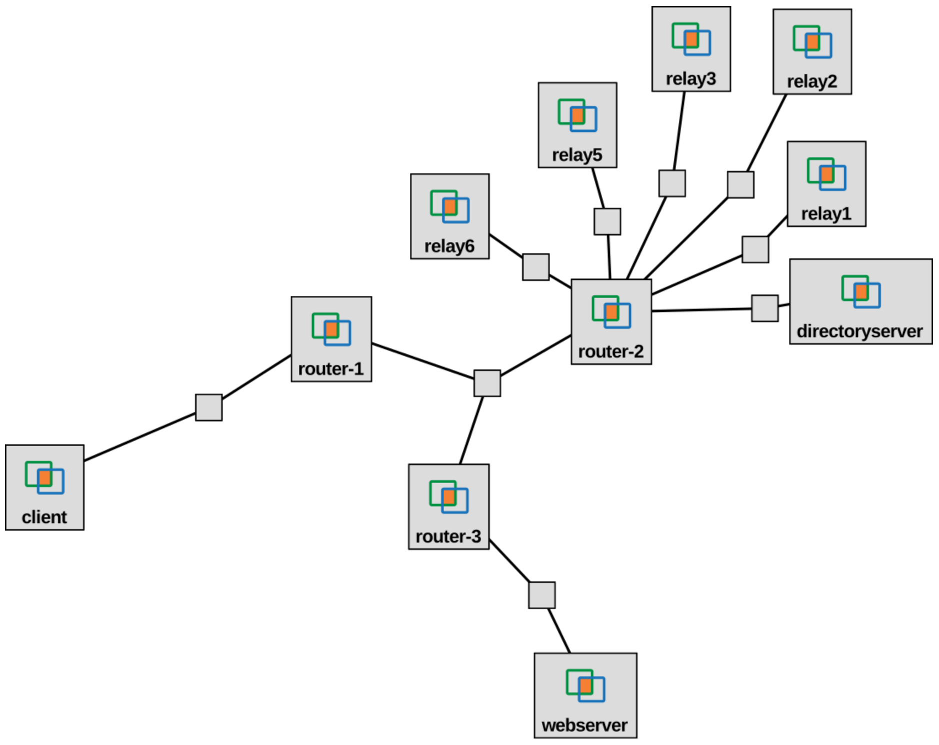


Figure 13: *GENI Topology of Private Tor Network*

This topology is modeled after the Electronic Frontier Foundation [20] example in Figure 6: the client terminal is the user, the webserver is the Site, router-1 is the first ISP on the user side, router-3 is the second ISP on the Site side, and router-2 represents the rest of the Internet. The hacker who has access to the user’s network views the network from the point of view of router-1. The NSA sees the network from the perspective of router-1, router-2, and router-3. The relays and directory server make up the Tor network. By following the EFF example, we will be able to see each player’s view of the network.

3.1.2 Installing the Tor Software

To start, we install Tor on the client, directory server, and relay nodes:

```
sudo sh -c 'echo "deb http://deb.torproject.org/torproject.org trusty main"
>> /etc/apt/sources.list'
sudo sh -c 'echo "deb-src http://deb.torproject.org/torproject.org trusty
main" >> /etc/apt/sources.list'
sudo gpg --keyserver keys.gnupg.net --recv 886DDD89
sudo gpg --export A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89 | sudo apt-key add
-
sudo apt-get update
sudo apt-get -y install tor deb.torproject.org-keyring vim curl tor-arm
```

3.1.3 Setting up the Web Server

On the web server terminal, an application called Apache2 is set up:

```
sudo apt-get update

sudo apt-get -y install apache2 php5 libapache2-mod-php5
```

Then, a simple PHP script is set up which returns the client's IP addresses as the homepage of the web server:

```
sudo rm /var/www/html/index.html
echo '<?php' | sudo tee -a /var/www/html/index.php
echo 'echo "Remote address: " . $_SERVER['REMOTE_ADDR'] . "\n";' | sudo tee -
a /var/www/html/index.php
sudo tee -a /var/www/html/index.php
echo '?>' | sudo tee -a /var/www/html/index.php
```

3.1.4 Setting up the Directory Server

Directory servers (also known as directory authorities) help Tor clients learn the addresses of relays that make up the Tor network. First, we stop any currently running Tor process. Then, the three keys required by a directory server are generated:

```
sudo -u debian-tor mkdir /var/lib/tor/keys
sudo -u debian-tor tor-gencert --create-identity-key -m 12 -a
192.168.1.4:7000 \
            -i /var/lib/tor/keys/authority_identity_key \
            -s /var/lib/tor/keys/authority_signing_key \
            -c /var/lib/tor/keys/authority_certificate
sudo -u debian-tor tor --list-fingerprint --orport 1 \
    --dirserver "x 127.0.0.1:1 ffffffffffffffffffffffffffffffffffffffff" \
    --datadirectory /var/lib/tor/
```

(We run these as the debian-tor user because it owns Tor’s files and directories.)

The output should say something like:

```
Nov 23 12:27:31.540 [notice] Your Tor server's identity key fingerprint is
'Unnamed 84F349212E57E0E33A324849E290331596BB6217' Unnamed 84F3 4921 2E57
E0E3 3A32 4849 E290 3315 96BB 6217
```

Now we'll create a configuration file for the directory authority. First, we create shell variables of two required fingerprints, as well as the hostname, which we will use as the Tor node’s "nickname":

```
finger1=$(sudo cat /var/lib/tor/keys/authority_certificate  | grep
fingerprint | cut -f 2 -d ' ')
finger2=$(sudo cat /var/lib/tor/fingerprint | cut -f 2 -d ' ')
HOSTNAME=$(hostname -s)
```

Torrc is the Tor configuration file which contains information on how the Tor node is to function. An important configuration is the TestingTorNetwork option, which we use to specify that we are operating a private Tor network, rather than joining the public Tor network. The ExitPolicy option defines whether the node can be an exit node or not, as well as the kinds of destinations that it can reach. These configuration options ensure that our Tor network on GENI remains private and dedicated to our experiment.

```
sudo bash -c "cat >/etc/tor/torrc <<EOL
TestingTorNetwork 1
DataDirectory /var/lib/tor
RunAsDaemon 1
ConnLimit 60
Nickname $HOSTNAME
ShutdownWaitLength 0
PidFile /var/lib/tor/pid
Log notice file /var/log/tor/notice.log
Log info file /var/log/tor/info.log
Log debug file /var/log/tor/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0
DirAuthority $HOSTNAME orport=5000 no-v2 hs v3ident=$finger1 192.168.1.4:7000
$finger2
SocksPort 0
OrPort 5000
ControlPort 9051
Address 192.168.1.4
DirPort 7000
# An exit policy that allows exiting to IPv4 LAN
ExitPolicy accept 192.168.1.0/24:*
AuthoritativeDirectory 1
V3AuthoritativeDirectory 1
ContactInfo auth0@test.test
ExitPolicy reject *:*
TestingV3AuthInitialVotingInterval 300
TestingV3AuthInitialVoteDelay 20
TestingV3AuthInitialDistDelay 20
EOL"
```

Since the relay and client configuration files also need the directory server's fingerprints in them, we'll first generate them on the directory server node (where those fingerprints are accessible), then download them to the individual relay and client nodes and customize them further there. We install Apache2 on the directory authority so that we can download the files from there. Next we write the relay configuration file:

```
sudo bash -c "cat >/var/www/html/relay.conf <<EOL
TestingTorNetwork 1
DataDirectory /var/lib/tor
RunAsDaemon 1
ConnLimit 60
ShutdownWaitLength 0
PidFile /var/lib/tor/pid
Log notice file /var/log/tor/notice.log
Log info file /var/log/tor/info.log
Log debug file /var/log/tor/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0
DirAuthority $HOSTNAME orport=5000 no-v2 hs v3ident=$finger1 192.168.1.4:7000
$finger2
SocksPort 0
OrPort 5000
ControlPort 9051
# An exit policy that allows exiting to IPv4 LAN
ExitPolicy accept 192.168.1.0/24:*
ExitPolicy accept 192.168.2.0/24:*
ExitPolicy accept 192.168.3.0/24:*
ExitPolicy accept 192.168.4.0/24:*
EOL"
```

This configuration file created on the directory authority creates a generic configuration file for all relay, which can then be copied over to a relay. The file is saved in the /var/www/html directory. Next, we write the client configuration file:

```
sudo bash -c "cat >/var/www/html/client.conf <<EOL
TestingTorNetwork 1
DataDirectory /var/lib/tor
RunAsDaemon 1
ConnLimit 60
ShutdownWaitLength 0
PidFile /var/lib/tor/pid
Log notice file /var/log/tor/notice.log
Log info file /var/log/tor/info.log
Log debug file /var/log/tor/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0
DirAuthority $HOSTNAME orport=5000 no-v2 hs v3ident=$finger1 192.168.1.4:7000
$finger2
SocksPort 9050
ControlPort 9051
EOL"
```

This configuration file created on the directory authority creates a generic configuration file for a client. Finally, we start Tor.

We can check to see whether our directory server was set up properly by running:

```
sudo cat /var/log/tor/debug.log | grep "Trusted"
```

A notice of the directory server being found should come up:

```
Nov 09 11:03:02.000 [debug] parse_dir_authority_line(): Trusted 100 dirserver
at 192.168.1.4:7000 (CA36BEB3CDA5028BDD7B1E1F743929A81E26A5AA)
```

This indicates that we are properly using our directory server at 192.168.1.4.

3.1.5 Setting up a Tor Relay

First, we stop any currently running Tor process. We then generate fingerprints as the debian-tor user with:

```
sudo -u debian-tor tor --list-fingerprint --orport 1 \
    --dirserver "x 127.0.0.1:1 ffffffffffffffffffffffffffffffffffffffff" \
    --datadirectory /var/lib/tor/
```

Now, we download the generic relay configuration file that we created earlier on the directory server:

```
sudo wget -O /etc/tor/torrc http://directoryserver/relay.conf
```

Now, we add some extra configuration settings that are different on each relay node: the nickname and the address. Add the nickname and address by running:

```
HOSTNAME=$(hostname -s)
echo "Nickname $HOSTNAME" | sudo tee -a /etc/tor/torrc
ADDRESS=$(hostname -I | tr " " "\n" | grep "192.168")
echo "Address $A" | sudo tee -a /etc/tor/torrc
```

Finally, we start the Tor service on the router node.

We can check to see if the routers are beginning to be made aware of each other on the private Tor network by accessing the info.log file which we defined earlier in the torrc configuration file of each node. If each of the routers is to be made aware of each other, one of the handshakes, described above in section 2.2.3, must take place. For example if we want to check whether relay3 has recognized relay2 and has performed a handshake, on the relay3 terminal we run:

```
sudo cat /var/log/tor/info.log | grep "192.168.12.2"
```

We should see some output like:

```
Apr 18 13:25:12.000 [info] channel_tls_process_versions_cell(): Negotiated
version 4 with 192.168.12.2:5000; Sending cells: CERTS
Apr 18 13:25:12.000 [info] channel_tls_process_certs_cell(): Got some good
certificates from 192.168.12.2:5000: Authenticated it.
Apr 18 13:25:12.000 [info] channel_tls_process_auth_challenge_cell(): Got an
AUTH_CHALLENGE cell from 192.168.12.2:5000: Sending authentication
Apr 18 13:25:12.000 [info] channel_tls_process_netinfo_cell(): Got good
NETINFO cell from 192.168.12.2:5000; OR connection is now open, using
protocol version 4. Its ID digest is 4EDC07C69A02236F13A8C6E0261F23B618D5ED19.
Our address is apparently 192.168.13.2.
```

We can see that first the version is negotiated, and then relay3 sends a CERTS cell. After receiving a CERTS cell and AUTH_CHALLENGE from relay2, relay3 authenticates, and then both send a NETINFO cell, confirming their location and timestamp. We can see that the following handshake that they have used to establish their connection is the In-Protocol handshake, described in section 2.2.3 above.

3.1.6 Setting up a Tor Client

First, we stop any currently running Tor process, then generate fingerprints as the debian-tor user in the same way that we did for setting up a Tor relay.

Next we download the client configuration file that we created on the directory server to the default Tor configuration file location with:

```
sudo wget -O /etc/tor/torrc http://directoryserver/client.conf
```

We then add the nickname and address in the same way that we did for the Tor routers. Finally, we start the Tor service on the client node.

3.2 Testing the Private Tor Network

Now that our private Tor network is set up and working, we will be testing this network. In section 3.2.1, we will be testing the network without using Tor. Then in section 3.2.2, we will test the private Tor network. Testing both with and without Tor will provide us results which we can compare with each other to see the functions of Tor.

Lastly in section 3.2.3 we will be using Tcpdump, a packet analyzer software, on the Tor network to see the functions of Tor in providing anonymity.

3.2.1 Testing the Network without Using Tor

In order to access the webserver without going through the Tor network, we simply run `curl http://webserver/` and verify that the server returns the client's IP address. We observe the following: `Remote address: 192.168.3.100`. This tells us that the VM that accessed the webserver is the client, revealing the direct communication between the webserver and client

Next we will be using Tcpdump to watch the traffic on the network. We want Tcpdump to display the output to screen while also saving the output to a file:

```
sudo tcpdump -s 1514 -i any 'port <port_num>' -U -w - | tee <name_file>.pcap
| tcpdump -nnxxXSs 1514 -r -
```

<port_num> is the specific port number to access, and <name_file> is the name that we want to save the file as.

Since we are not using the Tor network, we will be listening on port 80 which is the most common HTTP port. We have two client terminals opened, and on one, we start listening through port 80 by running

```
sudo tcpdump -s 1514 -i any 'port 80' -U -w - | tee clientnotor.pcap |
tcpdump -nnxxXSs 1514 -r -
```

On the web server terminal we also listen through port 80. On the other client terminal, we access the webserver by running the curl function. On both terminals that are listening on the network, we should see something like the following:

```
IP 192.168.3.100.38826 > 192.168.2.200.80
IP 192.168.2.200.80 > 192.168.3.100.38826
```

This shows how we can see that the client and webserver are communicating directly with each other. On the right-hand side of the output we should see something like:

```
MGET./.HTTPS/1.1..User-Agent:.curl/7.35.0..Host:.webserver..Accept:. */*
```

This is a request to access the webserver. We should also see something like:

```
Content-Length:.47..Content-
Type:.text/html....Remote.address:.192.168.3.100.Forwarded.for:...
```

This is the output of the curl command that we ran earlier. We can see exactly what traffic they are passing to each other. Communicating only through HTTP allows a person eavesdropping on this network to see who is communicating with who, and the data contents of the unencrypted packets.

3.2.2 Testing the Network Using Tor

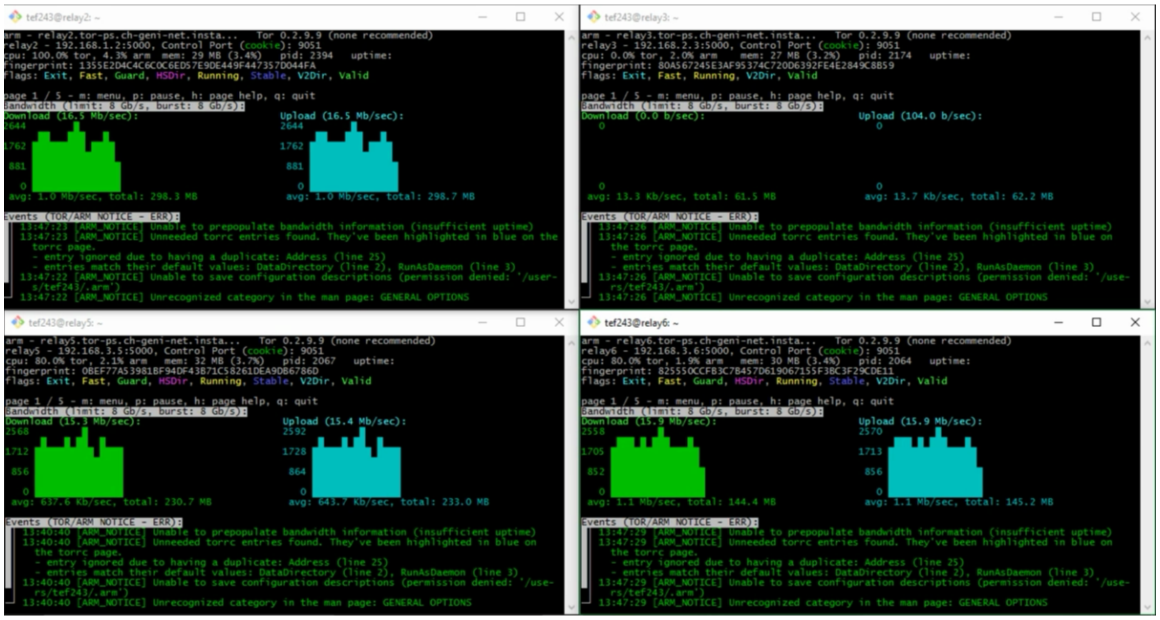
Now we will test the same curl experiment of accessing the web server, through the Tor network. Then we will compare the results with when we do not use Tor. First we need to find the exit relay that is being used in the Tor network. In order to do this we run: `curl -x socks5://127.0.0.1:9050/ http://webserver/.` We can verify that when using the Tor network (through the SOCKS proxy), the server does not know the client's IP address; it returns the IP address of the exit relay. We should see something like: `Remote address: 192.168.11.2.` Clearly the returned address is not the client's IP address, but the IP address of a Tor relay that we set up. This relay is the exit relay of the circuit that is being used (in this example, relay1).

Our next step is to figure out which Tor circuit is being used to access the webserver. This can be done using Tor Arm (anonymizing relay monitor), a program which serves as a terminal status monitor of Tor [28]. Arm provides useful statistics such as bandwidth, CPU, and memory usage, as well as known connections, and the Tor configuration file.

First we run Arm on each of the Tor relays including the client and the directory server. We can see a running display of bandwidth, CPU, and memory usage. On the client's Arm window, if we flip to the second page we can see the Tor circuits that are available to pass traffic. In order to figure out which circuit is being used, we need to pass traffic through the network and observe which Tor relays are passing that same traffic. To do this, we generate a large file on the webserver, to be downloaded by the client and while that is happening observe each relay's Arm window to see the passing traffic. On the webserver we generate a large file. Then, from the client terminal we download the large file using Tor:

```
curl -x socks5://127.0.0.1:9050/ -s http://webserver/large > /dev/null
```

Using the Arm display on the relays, as shown in Figure 14, we look for activity on each relay to identify the circuit that is being used.



```
tef243@relay2: ~
arm - relay2.tor-ps.ch-geni-net.insta...   Tor 0.2.9.9 (none recommended)
relay2 - 192.168.1.2:5000, Control Port (cookie): 9051
cpu: 100.0% tor, 4.3% arm  mem: 29 MB (3.4%)   pid: 2394   uptime:
fingerprint: 1355E2D4C4C6C0C6ED57E9DE449F447357D044FA
flags: Exit, Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 8 Gb/s, burst: 8 Gb/s):
Download (16.5 Mb/sec):                          Upload (16.5 Mb/sec):
2644                                             2644

1762                                             1762

 881                                              881

   0                                                0
 avg: 1.0 Mb/sec, total: 298.3 MB                 avg: 1.0 Mb/sec, total: 298.7 MB

Events (TOR/ARM NOTICE - ERR):
  13:47:23 [ARM_NOTICE] Unable to prepopulate bandwidth information (insufficient uptime)
  13:47:23 [ARM_NOTICE] Unneeded torrc entries found. They've been highlighted in blue on the
    torrc page.
    - entry ignored due to having a duplicate: Address (line 25)
    - entries match their default values: DataDirectory (line 2), RunAsDaemon (line 3)
  13:47:22 [ARM_NOTICE] Unable to save configuration descriptions (permission denied: '/user-
    s/tef243/.arm')
  13:47:22 [ARM_NOTICE] Unrecognized category in the man page: GENERAL OPTIONS
```

```
tef243@relay3: ~
arm - relay3.tor-ps.ch-geni-net.insta...   Tor 0.2.9.9 (none recommended)
relay3 - 192.168.2.3:5000, Control Port (cookie): 9051
cpu: 0.0% tor, 2.0% arm    mem: 27 MB (3.2%)   pid: 2174   uptime:
fingerprint: 80A567245E3AF95374C720D6392FE4E2849C8B59
flags: Exit, Fast, Running, V2Dir, Valid

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 8 Gb/s, burst: 8 Gb/s):
Download (0.0 b/sec):                            Upload (104.0 b/sec):
   0                                                0




   0                                                0
 avg: 13.3 Kb/sec, total: 61.5 MB                 avg: 13.7 Kb/sec, total: 62.2 MB

Events (TOR/ARM NOTICE - ERR):
  13:47:26 [ARM_NOTICE] Unable to prepopulate bandwidth information (insufficient uptime)
  13:47:26 [ARM_NOTICE] Unneeded torrc entries found. They've been highlighted in blue on
    the torrc page.
    - entry ignored due to having a duplicate: Address (line 25)
    - entries match their default values: DataDirectory (line 2), RunAsDaemon (line 3)
  13:47:26 [ARM_NOTICE] Unable to save configuration descriptions (permission denied: '/use-
    rs/tef243/.arm')
  13:47:26 [ARM_NOTICE] Unrecognized category in the man page: GENERAL OPTIONS
```

```
tef243@relay5: ~
arm - relay5.tor-ps.ch-geni-net.insta...   Tor 0.2.9.9 (none recommended)
relay5 - 192.168.3.5:5000, Control Port (cookie): 9051
cpu: 80.0% tor, 2.1% arm   mem: 32 MB (3.7%)   pid: 2067   uptime:
fingerprint: 0BEF77A53981BF94DF43B71C58261DEA9DB6786D
flags: Exit, Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 8 Gb/s, burst: 8 Gb/s):
Download (15.3 Mb/sec):                          Upload (15.4 Mb/sec):
2568                                             2592

1712                                             1728

 856                                              864

   0                                                0
 avg: 637.6 Kb/sec, total: 230.7 MB               avg: 643.7 Kb/sec, total: 233.0 MB

Events (TOR/ARM NOTICE - ERR):
  13:40:40 [ARM_NOTICE] Unable to prepopulate bandwidth information (insufficient uptime)
  13:40:40 [ARM_NOTICE] Unneeded torrc entries found. They've been highlighted in blue on
    the torrc page.
    - entry ignored due to having a duplicate: Address (line 25)
    - entries match their default values: DataDirectory (line 2), RunAsDaemon (line 3)
  13:40:40 [ARM_NOTICE] Unable to save configuration descriptions (permission denied: '/use-
    rs/tef243/.arm')
  13:40:40 [ARM_NOTICE] Unrecognized category in the man page: GENERAL OPTIONS
```

```
tef243@relay6: ~
arm - relay6.tor-ps.ch-geni-net.insta...   Tor 0.2.9.9 (none recommended)
relay6 - 192.168.3.6:5000, Control Port (cookie): 9051
cpu: 80.0% tor, 1.9% arm   mem: 30 MB (3.4%)   pid: 2064   uptime:
fingerprint: 825550CCFB3C7B457D619067155F3BC3F29CDE11
flags: Exit, Fast, Guard, HSDir, Running, Stable, V2Dir, Valid

page 1 / 5 - m: menu, p: pause, h: page help, q: quit
Bandwidth (limit: 8 Gb/s, burst: 8 Gb/s):
Download (15.9 Mb/sec):                          Upload (15.9 Mb/sec):
2558                                             2570

1705                                             1713

 852                                              856

   0                                                0
 avg: 1.1 Mb/sec, total: 144.4 MB                 avg: 1.1 Mb/sec, total: 145.2 MB

Events (TOR/ARM NOTICE - ERR):
  13:47:29 [ARM_NOTICE] Unable to prepopulate bandwidth information (insufficient uptime)
  13:47:29 [ARM_NOTICE] Unneeded torrc entries found. They've been highlighted in blue on
    the torrc page.
    - entry ignored due to having a duplicate: Address (line 25)
    - entries match their default values: DataDirectory (line 2), RunAsDaemon (line 3)
  13:47:29 [ARM_NOTICE] Unable to save configuration descriptions (permission denied: '/use-
    rs/tef243/.arm')
  13:47:29 [ARM_NOTICE] Unrecognized category in the man page: GENERAL OPTIONS
```

Figure 14: *Tor Arm Display of Active Tor Circuit*

3.2.3 Using Tcpdump to Watch Traffic

We again use Tcpdump to listen to the network at different locations. Three terminals are needed for the clients for this part. On one, we run:

```
sudo tcpdump -s 1514 -i any 'port 9050' -U -w - | tee client9050.pcap |
tcpdump -nnxxXSs 1514 -r -
```

This function tells the node to watch traffic that is going through the SOCKS proxy port 9050, and then save what it sees in a pcap file called client9050.pcap. This communication is over the loopback interface. On another client terminal we watch the traffic through port 5000, which is the port on which relays listen on the Tor network. We would be listening for any traffic that is being transferred from the client to the Tor network. Specifically, this communication would be between the client and the entry relay when the traffic enters the Tor network.

Next on each relay that is being used in the circuit, we run:

```
sudo tcpdump -s 1514 -i any 'port 5000' -U -w - | tee $(hostname -s).pcap |
tcpdump -nnxxXSs 1514 -r -
```

$(hostname -s) will be converted to the name of the specific relay that we are working with. This Tcpdump function will watch the traffic that is going through the OR port 5000, and then save what it sees in a pcap file.

On the exit relay, we open another terminal and have the relay listen on port 80, which is the most commonly used port for HTTP. This will capture traffic between the edge of the Tor network and the web server.

Lastly we must also set up the web server to have it listen for traffic as well on port 80. Now at this point we should have seven terminals listening on a network. These terminals are the two client terminals, terminals for the three ORs being used in the circuit, an extra exit relay terminal, and the web server terminal.

Next on the third client terminal, we run the curl command through Tor. Since we are using the Tor network to access the site, the three ORs must have seen some kind of traffic passing through it. Now let us take a look at what the client and each OR saw. When we stop Tcpdump, a file is created containing the traffic that was seen. Otherwise, as we listen on the network, the traffic that we can see will be outputted on the display of each terminal.

3.3 Summary of Toy Experiment

From using Tcpdump when using and not using Tor, there is a clear difference in the information that we can and cannot see at each node. Figure 15 summarizes this on a network that doesn't use Tor:

	Client Address	Site Address	Packet Contents
Client Port 80	Yes	Yes	Yes
Web Server Port 80	Yes	Yes	Yes
Any Middle-Man	Yes	Yes	No

Figure 15: *Table of Information that can be seen when not using Tor*

When not using Tor, all information can be seen. The client and the web server know who it is communicating with, and the packet contents that are being sent. However, an attacker that is listening on the same network can see all the information that both ends of the network can also see. This leads to easy spying, and stealing of information, privacy, and location.

Figure 16 summarizes the experiment while using the Tor network:

	Client Address	Site Address	Packet Contents
Client Port 9050	Yes	No	Yes
Client Port 5000	Yes	No	No
Entry Relay 5 Port 5000	Yes	No	No
Middle Relay 3 Port 5000	No	No	No
Exit Relay 1 Port 5000	No	No	No
Exit Relay 1 Port 80	No	Yes	Yes
Web Server Port 80	No	Yes	Yes

Figure 16: *Table of Information that can be seen when using Tor*

Let us look at each column one at a time. The leftmost column is arranged chronologically in the order which a packet gets transmitted to the web server. First we look at the client address column, where we see that the client address can be seen in the beginning, with the entry relay as the last node that knows the client address. The packet enters the Tor network therefore the client address becomes obfuscated. Inside the Tor network, each relay only knows who sent it the packet and who to send the packet to next. Therefore the entry relay gets the packet from the client, and knows to send it to the middle relay, but once it gets to the middle relay, the middle relay only knows that it got the packet from the entry relay and does not know anything about the client who originally sent the packet.

Taking a look at the next column which is the web server address, we can see that the web server address is not known until it reaches the exit relay and while listening through port 80. The web server address is not known until this point due to the encryption that is done by the onion proxy at the beginning. When the packet reaches the exit relay, the exit relay decrypts the last layer of encryption and at that point, the location of the webserver is first known. Since the exit relay is responsible for sending the packet to the web server, which can't decrypt any kind of Tor encryption, the web server address as well as the packet contents is unencrypted.

The last column of the table is the packet contents column, which tells us who sees encrypted or unencrypted packet information. We can see that the nodes which can see the unencrypted packet data are the nodes at the beginning and the end of the circuit. In order words, the nodes that are inside the Tor network cannot see the packet contents, because the information is encrypted while traveling inside the Tor network and the ends of the circuit cannot decrypt Tor encryption so the information must be unencrypted at the ends.

This experiment validates the main feature of Tor: that when using the Tor network, there is no one node that knows the whole mapping of the circuit. Therefore even if one node is compromised, that node only knows either the location of the client or the location of the site, but not both. As long as two nodes are not compromised by the same attacker, anonymity will stay intact. This is the power of Tor, in being able to cloak client and web server communication as well as their locations and packet contents while the traffic traverses the Tor network.

Chapter 4

Overview of Tor’s Adversaries

4.1 The Imperfections of Tor

Tor developers encourage continued research in finding ways to break the system's anonymity because each new method provides a new perspective on how the Tor network can be strengthened. This section aims to provide readers with an understanding of the various attacks that have been conducted on the Tor network. For each attack we consider:

1. Has this attack been demonstrated, or is it only a theoretical attack?
2. Key concepts related to the attack.
3. Is the attack active or passive?
4. Which part of the network does it attack?
5. Which parts of the network does the attacker need to control or be able to access?
6. What are the specifics of the attack? How is the attack performed?

4.1.1 AS-Level Adversary Attack

1. This attack is described by Edman and Syverson in *AS-awareness in Tor Path Selection* [10]. In this paper, their main goal was to propose possible AS-aware path selection algorithms that would increase anonymity and avoid AS-level adversaries. In

order to show this, they first used past routing information to construct a model of the ASes. Then they used an algorithm to calculate shortest paths between ASes, and from this experiment they found that a single AS could observe both ends of a connection around 10 to 30% of the time.

2. **Autonomous Systems** (ASes) are a series of independent networks that make up the Internet. Whenever a user's data is sent through the Tor network to the requested destination, the data goes through an AS. **Traffic Correlation** is an attack which aims to deanonymize a user by matching incoming and outgoing traffic. A common type of traffic correlation attack known as the end to end confirmation attack is performed by correlating the size of packets incoming and outgoing, as well as the timing of these packets that are transmitted [29]. For example, a user streaming a video would receive packets in a different pattern from someone who is simply browsing through Facebook.

3. This attack is passive because the attacker simply listens to the connection.

4. This attack focuses on the entry and exit points of the Tor network for its attack.

5. The attacker must have control of an AS where the traffic between a user and the entry node, and that between the exit node and destination go through the same AS.

6. When the traffic between the user and entry node, as well as the traffic between the exit node and destination both go through the same AS that is under the control of the attacker, the attacker can see the user's traffic that is incoming and outgoing the Tor network. Using traffic correlation (defined above), the attacker aims to deanonymize the user and figure out who is communicating with whom.

4.1.2 Website Fingerprinting

1. This attack is described by Panchenko et al. in *Website Fingerprinting in Onion Routing Based Anonymization Networks* [30]. The paper describes website fingerprinting as another traffic confirmation attack, but can be performed by a local adversary. For example, for people in oppressive regimes who try to access websites that are located in outside countries, they are still able to deanonymized by a local eavesdropper that is able to listen to the traffic traveling between the user and the entry node of the Tor network.

2. **Fingerprinting** is a kind of traffic analysis that involves observing a user's traffic and identifying certain patterns which can later be used in traffic confirmation to correlate what website a user may be accessing.

3. This attack is passive because the attack is simply gathering fingerprints of the user and website, and is not padding any data into the traffic.

4. This attack is focused mostly on observing the data between the user and the entry node of the Tor network.

5. The attacker must have packet traces of the websites that he is interested in seeing whether a certain user accesses that specific website.

6. First, the attacker collects packet traces from certain websites that he is interested in monitoring. Then the attacker collects packet traces that are sent by the user that he believes may be visiting those sites. He then compares the traces of both by using classification [31]. The local attacker is located on the same network or ISP as the user, or has access to that network.

4.1.3 Selective Denial of Service (SDoS)

1. This attack is a theoretical attack that is described by Borisov et al. in *Denial of Service or Denial of Security?* [8]. This attack is described as having a secondary objective to deanonymizing the user, which is to decrease the reliability and efficiency of the Tor network.

2. **Denial of Service (DoS)** as the name connotes is an attack which aims to disallow a user from using a service. In this case, the DoS refers to disallowing the use of a certain connection in the Tor network to pass traffic.

3. This attack is active due to the action of killing a connection tunnel to disallow the forwarding of a user's traffic.

4. SDoS is a direct attack on the connection tunnel that would be blocked if traffic confirmation is not successful. Otherwise, the attack focuses more on the traffic of a user as it enters and leaves the Tor network.

5. In order to perform this attack successfully, the adversary must be in control of an entry and exit relay that is used on the same circuit of a user to pass traffic.

6. This attack has two possible outcomes, where neither is a failed outcome. One is that if both the entry and exit nodes of a connection are run by the same adversary, then traffic confirmation can be performed to deanonymize the user. If this does not turn out to be the outcome, the adversary will block forwarding of traffic through that connection tunnel, and as a result the user will need to use another circuit. An important point of this active attack is that instead of making users not want to use Tor anymore, the attack merely makes the attack a bit less reliable but nonetheless functional [8]. Tor users that

try to create another circuit will provide adversaries more opportunities of deanonymization, which is an important strength of this attack.

4.1.4 Low-Resource Routing Attack

1. This attack is described by Bauer et al. in *Low-Resource Routing Attacks Against Tor* [6]. An important goal of this experiment was to try to minimize the requirements for an adversary to be able to compromise a user and its destination, especially due to the fact that the adversary relays have minimal resources.

2. **Load Balancing** is a process used in network traffic which aims to balance traffic efficiently across all available resources in the network. In order to maintain a low-latency system, Tor must not only rely on the routers that have high bandwidth, but on all available resources.

3. This attack is an active attack because not only does the adversary install one's own routers, but also logs fake statistics of available bandwidth.

4. This attack focuses on the relays of the network, as the adversary's relays are competing with the other relays to be chosen as a user's relay to direct traffic.

5. This attack requires a number of low-resource routers that can advertise false bandwidth capabilities, as well as a central authority that will link the colluding relays by matching the logs of information that each router collects and to see whether there is any correlation.

6. After an adversary has under his/her possession a number of low-resource relays that can advertise high bandwidth capabilities, and preferably has an unrestricted exit policy to allow all kinds of traffic, the next step is to try to compromise the user and

destination communication before any payload is sent. This means that the attack is focused on the initial circuit-building algorithm stage. The adversary relays are simply falsely advertising bandwidth, therefore will not be able to efficiently forward any actual payload, so the attack must be done in the early stages. Each adversary logs the following information: "(1) its location on the current circuit's path (whether it is an entry, middle, or exit node); (2) local timestamp; (3) previous circuit ID; (4) previous IP address; (5) previous connection's port; (6) next hop's IP address; (7) next hop's port; and (8) next hop's circuit ID." [6]. The colluding central authority then collects these logs from each adversary relay and sees whether there is any correlation to compromise a connection.

4.1.5 Sniper Attack

1. The sniper attack is described by Jansen et al. in their paper *The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network* [7]. The sniper attack is a low-cost yet highly effective denial of service attack. This attack by itself is only capable of performing DoS, and not deanonymization. However, when combined with SDoS, the attacks are very effective.

2. **End-to-End Sliding Window Mechanism** is used by Tor to control congestion in the Tor network. Each node at the end of a circuit (client and exit relay) manages a package window for incoming packets, and a delivery window for outgoing packets.

3. This attack, as it is a DoS attack is certainly active.

4. The main victim that the Sniper Attack aims for is the entry relay that is used in the circuit to pass the user's traffic.

5. There are two parts of the network that an adversary must control in order to conduct this attack: an adversary client and an exit relay. These two colluding parts work together to conduct a DoS on the entry relay.

6. The basic process of this attack is that when the delivery edge of a node stops reading from its TCP port, the receive buffer will fill, which would result in the adjacent node not being able to forward any more traffic, filling up the queue of that node. The malicious exit node ignores any empty package window and continues to send traffic along the circuit. The middle node does not have any congestion control mechanism to stop accepting from the exit relay, so therefore continues to pass the traffic to the entry relay, piling up there.

4.1.6 DNS Correlation Attack

1. This attacked has been developed by Greschbach et al. and is explained in *The Effect of DNS on Tor's Anonymity* [32]. Unlike most traffic confirmation attacks which focus on TCP traffic to correlate a user and its destination, this attack focuses on DNS traffic that is sent alongside TCP traffic. Simply loading a webpage requires generating lots of DNS traffic to many different domains. In this paper a type of attack called DefecTor is introduced, which was produced to test the DNS correlation attack. Compromising a connection using DNS traffic confirmation is still a relatively new idea with this paper serving as one of the forefronts of this research.

2. **Domain Name Server (DNS)** is like a phone book for the Internet, in which they are servers that have a directory of names of domains, and there corresponding IP addresses. Although computers can easily remember a series of numbers, we users have

difficulty doing so, and are able to remember names like Facebook and Youtube more easily. DNS are responsible for being the middleman between us users and our computer so that we can access the specific site that we want.

3. A DNS Correlation Attack is a passive active like an AS-Level Adversary Attack, as it only listens to the network and collects traces of traffic to see if there is any correlation.

4. This attack focuses on the DNS traffic that is generated whenever a Tor user accesses a website.

5. Access to the DNS traffic at both ends of the Tor network is a requirement of this attack.

6. To see the DNS traffic at the entry point of the Tor network, the adversary can operate on the network level, for example to have access to an ISP. Another alternative is to operate on the relay level by running an adversary entry relay. The adversary must also be able to see the outgoing DNS traffic, therefore must either operate on the network level, or run a malicious DNS resolver or server [32]. Operating a malicious exit relay is also an option, but the efficiency would be on par with a normal correlation attack.

4.1.7 Hidden Services Attack

1. This attack is described by Kwon et al. in *Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services* [33]. This attack on users that utilize hidden services of the Tor network is one of the first of its kind. It is described that traffic that is generated to use a hidden service is quite unique from regular Tor traffic, and thus it is easy to separate and focus on the users that use hidden services. The method uses

circuit fingerprinting, a form of attack we observed in the website fingerprinting section. The paper describes how circuit fingerprinting comes in "two settings: open- or closed-world. In the closed-world setting, the attacker assumes that the websites visited are among a list of *k* known websites, and the goal of the attacker is to identify which one. The open-world setting is more realistic in that it assumes that the client will visit a larger set of websites" [33].

2. **Hidden Services** are special services provided by Tor, in which only Tor users can access. Using these services is like an extra layer of anonymity within the Tor network.

3. A Hidden Service Attack is a passive attack because it is a form of circuit fingerprinting, which only observes the traffic in the Tor network and aims to find any correlation in traffic. An extra step that is taken in this attack is that traffic relating to hidden services and traffic of normal Tor usage are separated so a more efficient experiment can be conducted.

4. The focus of attack is on the hidden services of the Tor network, which exhibit different behaviors from regular Tor traffic.

5. An important factor for a successful hidden service attack is that the adversary must be able to listen to the traffic between the user and the hidden service. This communication is not direct, but through the relays that the user chooses to create a circuit to the hidden service.

6. In order for a user to use a hidden service, the hidden service must first choose a random relay to serve as the Introduction Point (IP), in which a message containing the service's public key is sent. The client also must choose a random relay to serve as the

Rendezvous Point (RP) which will serve as a connection point to the hidden service. Both sides must also create a circuit connection to the IP and RP to exchange keys [33]. An attack on the hidden services requires knowing how a connection to a hidden service is established in the first place, and seeing that traffic sent to a hidden service is unique. The actual deanonymization of the circuit is done through a website fingerprinting attack.

4.2 Taxonomy of Tor Attacks

Figure 17 summarizes the attacks described in this section:

	Active/Passive	**Observes Incoming/ Outgoing Traffic**	**Level of Authority**	**Has Been Demonstrated**
AS-Level Adversary	Passive	✓	Network Level	✓
Website Fingerprinting	Passive	✓	Network Level	✓
SDoS	Active	✓	Relay Level	
Low-Resource Routing	Active	✓	Relay Level	✓
Sniper	Active		Relay Level	✓
DNS Correlation	Passive	✓	Network/ Relay Level	✓
Hidden Services	Passive	✓	Network Level	✓

Figure 17: *Taxonomy of the Adversaries of Tor*

The first categorization is whether the attack is active or passive. Active attacks often have padding of data into the traffic or the usage of one's own resources, such as installing one's own adversary relays in order to conduct the attack. Passive attacks on the other hand typically have the one requirement of being on the network level, such as an AS or ISP, to be able to listen to the network for the user's traffic.

The next categorization is whether the attack requires listening to the traffic of a user going into and out of the Tor network, or in other words having to be at the ends of the Tor network to capture user packets. The Sniper Attack is the only attack that does not require any listening on user traffic at the ends of the Tor network because the goal of this attack is not deanonymization but rather a DoS of the entry relay. Making the Tor network less reliable but still functional is the main objective of the Sniper Attack.

The third categorization defines the level of authority or the amount of traffic that the attacker needs to be able to see to be able to perform the attack. For example not every attacker is able to perform a website fingerprinting attack. We define two levels: network and relay. An attack that requires to be on the network level means that the attacker must either be on the same connection as the user, or has control over/has data sharing agreements from the ISP of the user. An attack that requires relay-level authority means that the attacker must install one's own malicious relays, and have those relays be chosen to create a user's circuit. We see a correlation with active attacks needing to be on the relay level, and passive attacks needing to be on the network level.

The last categorization is whether the attack is an actual experiment that has been conducted and proven, or is still just a theoretical attack waiting to be tested. The only attack that has not actually been tested is the SDoS attack. The possible outcomes of such an experiment are given, backed by the theoretical evidence that is presented.

Chapter 5

Website Fingerprinting on GENI

5.1 Understanding Website Fingerprinting

In this section, we will demonstrate the execution of a website fingerprinting attack on a private Tor network running on GENI. However, we must first understand the attack in greater detail. First we define two settings of a dataset that are possible:

- **Closed-world dataset**- the situation where the attacker already knows the set of possible webpages that the victim user uses. This scenario is a bit unrealistic because the probability of compromising the user increases drastically by only having to gain packet traces from a few websites.
- **Open-world dataset**- the more realistic situation where the attacker does not know what kinds of pages the user goes on, which makes the ranges of webpages basically infinite.

For our experiment, we will follow the closed-world dataset because of the difficulty of setting up an open-world dataset experiment, and due to a limit in resources.

Next, let us take a look at the general procedure of website fingerprinting:

1. The attacker first listens to and records packet traces from several websites that the user might be visiting. The attacker uses a traffic analyzer tool for example

Tcpdump or Tshark to capture packet traces of IP layer packets. These packets are known as training instances.

2. Using information captured such as the length of the packet, the time sent and received, etc. the attacker creates a profile of the website, also known as a fingerprint [30].
3. Next the attacker listens on the target user's network and similarly captures packet that is going in and out from the user. However this data is not going to be blatantly similar to the fingerprint that the attacker creates in step 2, due to a difference in user, possible packet fragmentation, and website updates. These packets are known as test instances.
4. The attacker tries to compromise the user by comparing the website fingerprint and user data using statistical methods to probabilistically come to a conclusion. (In our experiment, we will manually inspect visualizations of the packet traces to identify the site that the user visits.)

5.1.1 Characteristics of Packet Traces

Now let us go into more detail the contents of a fingerprint. Many of the specific details that we cover come from an attack that is described by Panchenko, et al., in *Website Fingerprinting in Onion Routing Based Anonymization Networks* [30].

For each website, the attacker first collects a couple instances of traffic. Within this traffic, there are many characteristics that the attacker must look out for to create a detailed fingerprint of the website.

- **Size and Direction**- The size of the packet and whether the packet is incoming or outgoing is recorded. Incoming packets are marked as negative, while those that are outgoing are marked as positive. Keeping track of the order in which the packets arrived is important as well.

- **Filtering ACK Packets**- Acknowledgement packets are sent back and forth almost constantly, and do not provide any useful information to the attacker, so we first filter out these packets.
- **Size Markers**- These are markers to be placed whenever the direction of traffic changes from ingoing to outgoing and vice-versa. These markers must work in conjunction with filtering out the ACK packets. These markers note how many bytes went a certain way before changing direction.
- **HTML Markers**- Whenever a request for a webpage is made, the initial process is to request for the HTML document. Being that every site has an HTML document of a different size, we can use this information to make the packet trace more unique. We place an HTML marker after the HTML document is fully received.
- **Total Transmitted Bytes**- This involves adding up separately the total number of bytes sent and received at the end of the packet trace.
- **Number Markers**- These markers are similar to size markers, but instead mark how many packets came a certain direction before switching to the other direction.
- **Occurring Packet Sizes**- This marker keeps track of the number of unique packet sizes there were in the packet trace, and appended at the end of trace.
- **Percentage of Incoming Packets**- This step involves finding the percentage of packets that were incoming compared to outgoing.
- **Total Number of Packets**- Similar to adding up the total number of bytes sent and received, the total number of packets is also counted.

5.2 Setting up the Website Fingerprinting Experiment

We will now be setting up our own website fingerprinting experiment on GENI. A version of this experiment with more explicit instructions for each step is published on the Run My Experiment on GENI blog [19]. The following steps will be taken for this experiment:

1. Reserve resources and set up new a new private Tor network
2. Get the homepages of a number of sites onto the webserver
3. Create a fingerprint for each site
4. Hit a random site on the client, and see if there is a match with any of the fingerprints.

For each website's fingerprint as well as the client's traffic that we capture, we will be creating plots and a table as visuals to compare the fingerprints.

5.2.1 Setting up a new Private Tor Network

We begin by setting up the same topology on GENI as in Section 3.1.1. Since we have already gone through the steps of setting up each VM, for this experiment we set up the VMs using scripts available on Github [34] that execute the setup procedure described in Chapter 3. Then, we confirm that the private Tor network is up and running by using Tor Arm.

5.2.2 Setting up Websites on the Webserver

Now we will be setting up the webserver for our experiment. We will be saving homepages of 5 different websites onto our webserver, so that we can create fingerprints of them later. For this demo, we used 5 websites:

1. NYU Tandon School of Engineering
2. Facebook
3. Youtube

4. Reddit
5. Official New York Mets Website

For each website, we will be storing the homepage and linked assets in its own directory inside the /var/www/html directory of the webserver. For example, to download the NYU Tandon School of Engineering homepage, we run:

```
cd /var/www/html/
sudo wget -e robots=off --wait 1 -H -p -k http://engineering.nyu.edu/
```

We do the same for the other four websites.

5.2.3 Creating Website Fingerprints

Now we will be creating the fingerprints of each website by accessing the website's homepage that is stored on the webserver from the client node. First we must install proxychains, which we will be using with the wget function to use the Tor network to access the websites. Another function that we need is Tshark, which is a network protocol analyzer similar to Tcpdump. We install proxychains on the client and Tshark on router 1.

When we create the fingerprints of the websites, we will be creating plots and a table to visualize our fingerprints to better compare with the client traffic that we will eventually capture. We will be using seaborn, a python visualization library that is based off the more common matplotlib. On router 1, we run:

```
sudo apt-get -y install python-pip python-dev libfreetype6-dev liblapack-dev
libxft-dev gfortran
sudo pip install stem seaborn pandas
```

Next, in order to make our fingerprints, we must run our website packet captures through a filter, defined by a python script that we have written. This script takes in the packet trace and filters out unneeded packets and places markers of different kinds in the trace, which are all defined in the **Characteristics of Packet Traces** section above. The python script is available on Github [34]. To retrieve this script on router 1, we run:

```
wget https://raw.githubusercontent.com/tfukui95/tor-experiment/master/make-
fingerprint.py
```

Router 1 represents the point of view of someone who can eavesdrop on the client's traffic. After capturing the traffic we place the trace through the filter using the python script in order to create the fingerprint.

Let us first test to make sure that everything is working. On router 1 we turn off tcp segmentation offload so that when capturing packets, we see each packet separately. Then on router 1 we start listening with tshark by running:

```
sudo tshark -i eth1 -n -f "host 192.168.3.100 and tcp and port 5000" -T
fields -e frame.len -e ip.src -e ip.dst -E separator=,
```

The tshark command's flags describe that it is only listening on the eth1 interface (or substitute the name of the interface that is connected to the client as appropriate), and is picking out packets that have the client address in either source or destination IP address field, and either source or destination port 5000. On the client terminal we access the webserver's default index.php page:

```
sudo proxychains wget -p http://192.168.2.200/
```

On router 1, we should see a comma separated list, where the rows contain the packet size, the source of the packet, and the destination of the packet, respectively.

Now we are ready to build the fingerprints of each website. Let us first start with the NYU Engineering site. To run the packet trace through the filter, we must save it as a csv file. On router 1, we run:

```
sudo tshark -i eth1 -n -f "host 192.168.3.100 and tcp and port 5000" -T
fields -e frame.len -e ip.src -e ip.dst -E separator=, >engineering.csv
```

On the other client terminal, we run:

```
proxychains wget -p http://192.168.2.200/engineering.nyu.edu/
```

Now we stop the tshark from listening on the network, which will save the captured traffic into a file called engineering.csv. Now in order to put the file through the filter using the python script, on router 1 we run:

```
python make-fingerprint.py --filename engineering.csv
```

A list of tuples, containing the type of data and the data itself, will be outputted onto the display. An image of three plots is created as a visual of the fingerprint, as well as a table containing all of the other markers that are placed at the ends of a packet instance. The name of each of the images depends on the website, for example, where <sitename> can refer to Facebook. The file <sitename>-fingerprint-grid.png contains three plots. The first one is a size and direction plot showing the size in bytes of each packet transmitted, from the client (positive) and received at the client (negative). The second plot shows the size markers that are placed in the traffic whenever the direction of traffic changes, and note how many bytes went a certain direction before going the other. The third plot shows the number markers which count how many packets went a certain direction before switching to the other direction. The table saved as <sitename>-fingerprint-table.png is a table of the additional markers such as the html, total bytes sent, total packets sent, unique packet sizes, and percentage of incoming/outgoing packets.

Now we must do the same for the other four sites. For Facebook:

```
# On router 1
sudo tshark -i eth1 -n -f "host 192.168.3.100 and tcp and port 5000" -T
fields -e frame.len -e ip.src -e ip.dst -E separator=, >facebook.csv

# On the client
proxychains wget -p http://192.168.2.200/facebook.com/

# After stopping tshark, on router 1
python make-fingerprint.py --filename facebook.csv
```

For the remaining three sites (Youtube, Reddit, NY Mets), we follow the same procedure, replacing the site name and csv file name with the proper name for each site.

After we have made all of the fingerprints, we can use SCP (Secure Copy Protocol) to save all of our images from our remote Linux terminal to our local machine. We go to our GENI slice page, and click Details to get more information of our client VM. We then open up a local terminal and go to any folder to save our plots to. The SCP command for saving the image to our local machine is the following:

```
scp -P <port_number> <user>@<site_address>:~/*.png .
```

Where *.png is a wildcard to indicate all of the png files that we have saved on our client terminal.

5.2.4 Analyzing the Fingerprints

We open up the fingerprint visuals of all five websites, shown in Figure 18:

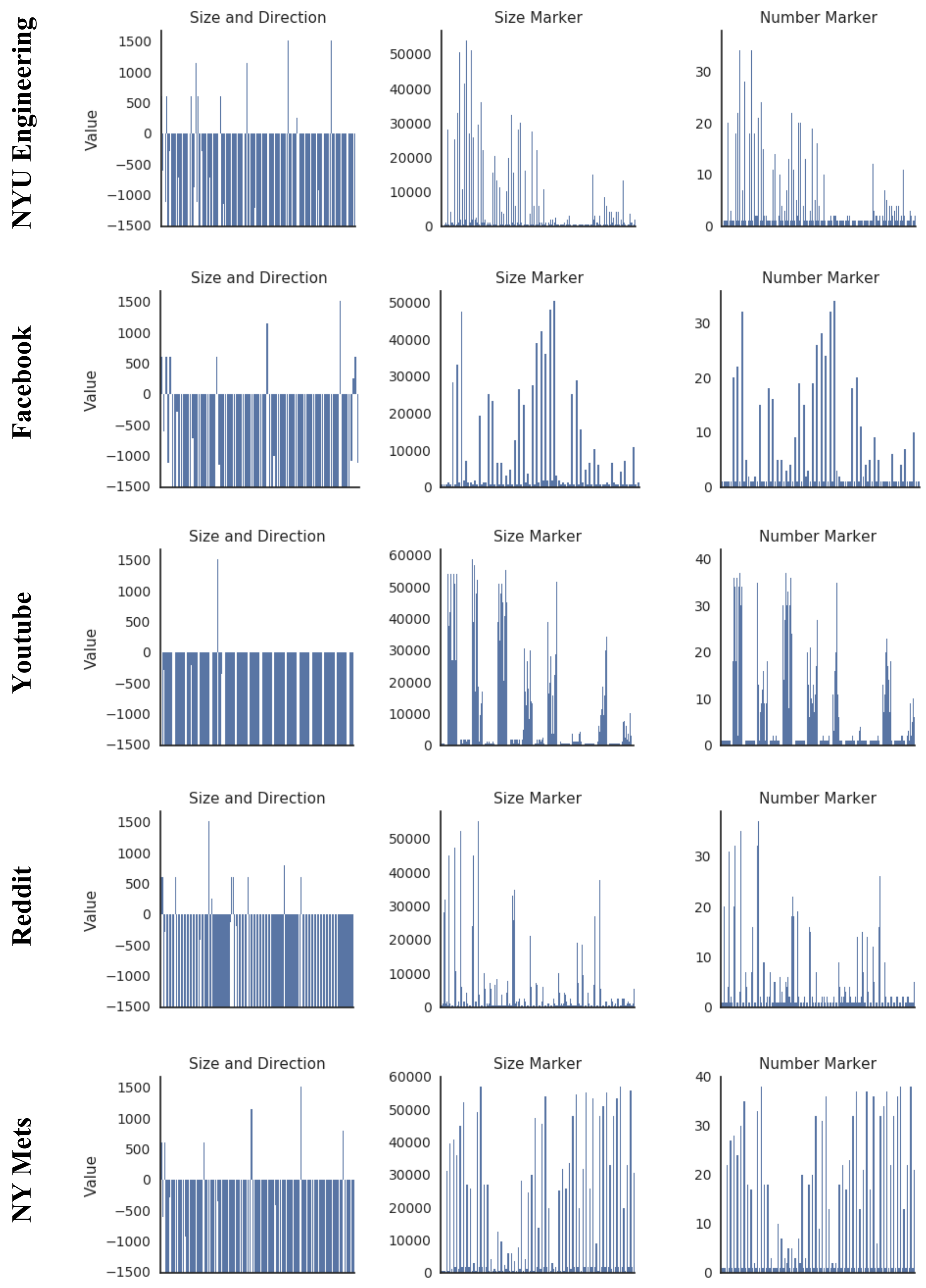


Figure 18: *Fingerprints of the five websites*

For each site we have the three plots: size and direction, size marker, and number marker. We notice that for certain sites it is much easier to distinguish from the rest, for example for the NYU Engineering homepage, we can see that towards the end of the retrieval of the webpage we see a lot of constant back and forth between client and server compared to the first half where there are a lot of packets sent in bunches from the webserver. Other sites are not as easy to distinguish, but we can see that every site's fingerprint has some kind of unique part to it which differentiates it from the rest of them.

NYU Engineering

Marker	Packet Information
HTML	1200
TS+	70000
TS-	860000
OP+	6
OP-	28
PP-	0.15
NP+	105
NP-	630

Facebook

Marker	Packet Information
HTML	1200
TS+	40000
TS-	660000
OP+	8
OP-	22
PP-	0.10
NP+	60
NP-	480

Youtube

Marker	Packet Information
HTML	1200
TS+	190000
TS-	4270000
OP+	8
OP-	56
PP-	0.10
NP+	225
NP-	2925

Reddit

Marker	Packet Information
HTML	1200
TS+	130000
TS-	1760000
OP+	10
OP-	40
PP-	0.15
NP+	195
NP-	1305

NY Mets

Marker	Packet Information
HTML	31200
TS+	70000
TS-	1700000
OP+	10
OP-	26
PP-	0.05
NP+	75
NP-	1155

Figure 19: *Table of relevant markers for each website*

Figure 19 shows the tables of markers that were created as part of the fingerprinting process. HTML represents the HTML marker, TS is the total transmitted bytes marker, OP is the occurring packet sizes marker, PP is the percentage of incoming packets marker, and NP is the total number of packets marker. A detailed explanation of each marker can be found in section 5.1.1.

5.2.5 Testing the Website Fingerprinting Experiment

Now that we have our fingerprints for each site, we will snoop on the client of our private Tor network and see if we can figure out the site the client is visiting. The following is our process:

1. Start listening on the client's network.
2. Have the client randomly choose one of the 5 sites.
3. Capture the traffic that we can see, and create a similar fingerprint plot as the one that we made for the websites.
4. See if we can determine which site the client visited.

Now let us start our experiment. On one client terminal, we run:

```
sudo tshark -i eth1 -n -f "host 192.168.3.100 and tcp and port 5000" -T
fields -e frame.len -e ip.src -e ip.dst -E separator=, >wfpAttack.csv
```

There is a script file available on Github [34] called randomSite.sh. This script makes the client randomly choose one of the five websites that we have set up on the webserver. The process to create the fingerprint is the same as before.

We open up the fingerprint visual for the client traffic that we captured, shown in Figure 20:

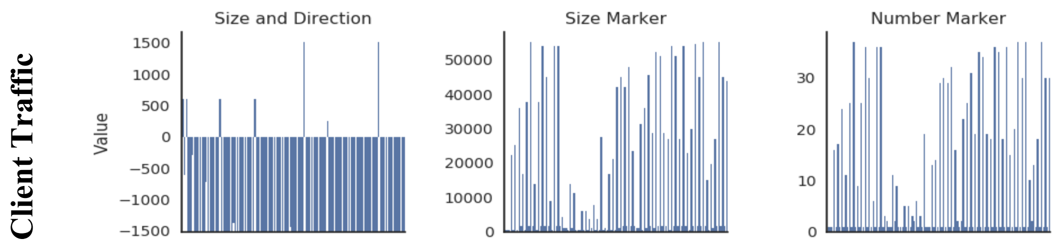
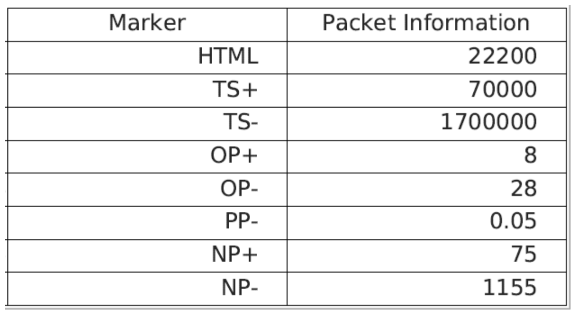


Figure 20: *Fingerprint of the client visiting an unknown website*

We see whether or not it is clear which site the client randomly chose to visit. For this specific example, we see a strong correlation between the client traffic fingerprint and the New York Mets fingerprint. The size and number markers for both fingerprints see a similar dip towards the middle of the graph, representing a more constant back and forth between client and webserver, without too many packets in one direction consecutively.



Marker	Packet Information
HTML	22200
TS+	70000
TS-	1700000
OP+	8
OP-	28
PP-	0.05
NP+	75
NP-	1155

Figure 21: *Table of relevant markers for client traffic*

Figure 21 shows the markers that are appended at the end of the packet trace for the client traffic that we captured. We can see the most correlation with the table of markers for the NY Mets homepage in Figure 19, where for example the total transmitted bytes are the same, as well as the percentage of packets incoming/outgoing, and the total number of packets sent.

We go back to the client terminal in which we ran the shell script, which should be waiting for a keyboard input saying "Press any key to see which site the client visited:". After we arrive upon a guess as to which site the client visited, we press any key on the keyboard to see whether our guess was correct.

To expand on this experiment, we can increase the number of sites, clients, marker types, etc. to make the experiment even more interesting and realistic. Using Wireshark to gain a better understanding of the packet traces is also recommended. Wireshark can be downloaded from the company's homepage. Besides downloading and installing the software, there is often no configuration necessary.

Wireshark's default file extension is a pcap file, which the Tcpdump function can save its traffic to by the following:

```
sudo tcpdump -s 1514 -i any 'port <port_num>' -U -w - | tee <name_file>.pcap
| tcpdump -nnxxXSs 1514 -r -
```

This will tell the function to listen on a certain port number, output the traffic onto the display, and to save it to a pcap file with the name that we specify.

Chapter 6

Conclusion

In this work, a private Tor network was set up on GENI, followed by the testing of a website fingerprinting attack on the network. For both parts, a step-by-step procedure was explained, and the background information to better understand Tor was introduced. To facilitate their use by other researchers, these experiments are also published on the Run My Experiment on GENI blog [18], [19].

The private Tor network topology that was presented in my thesis can be easily programmed to have more nodes, whether it is more clients, routers, web servers, or even directory servers. Creating a larger topology requires more resources, but is able to provide results more reflective of the live Tor network. Being able to easily create any topology that is required shows the flexibility and practicality of GENI.

The website fingerprinting attack that was described in this thesis, although very limited in the number of websites, shows how a correlation attack can be performed even on a private Tor network. First experimenting with an attack or defense on a private Tor network provides good feedback and a strong foundation of the research before experimenting on the live Tor network.

This thesis contributes some examples that I would like to be used as references for other researchers and students working with Tor to set up their own experiments. Future work in using GENI to set up better, more complex private Tor networks, as well as testing attacks and defenses on these networks is limitless. GENI’s versatile programmability and wide availability of resources provides a promising method of expanding research on attacks and defenses of Tor.

Bibliography

[1] P. Syverson, “A Peel of Onion,” in *Proceedings of the 27th Annual Computer Security Applications Conference*, New York, NY, USA, 2011, pp. 123–137.

[2] “Welcome to Tor Metrics.” [Online]. Available: https://metrics.torproject.org/. [Accessed: 05-Apr-2017].

[3] P. Howell O’Neill, “As Kremlin tightens reach, Tor usage spikes in Russia,” *Daily Dot*, 18-Jun-2014.

[4] “Tor Project: Bridges.” [Online]. Available: https://www.torproject.org/docs/bridges. [Accessed: 25-Apr-2017].

[5] E. Jardine, “Tor, what is it good for? Political repression and the use of online anonymity-granting technologies,” *New Media Soc.*, Mar. 2016.

[6] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, “Low-resource Routing Attacks Against Tor,” in *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, New York, NY, USA, 2007, pp. 11–20.

[7] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, “The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network,” presented at the 21st Annual Network & Distributed System Security Symposium (NDSS ’14), 2014, pp. 23–26.

[8] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, “Denial of Service or Denial of Security?,” in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2007, pp. 92–102.

[9] D. Herrmann, R. Wendolsky, and H. Federrath, “Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve Bayes Classifier,” in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, New York, NY, USA, 2009, pp. 31–42.

[10] M. Edman and P. Syverson, “As-awareness in Tor Path Selection,” in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, pp. 380–389.

[11] L. Overlier and P. Syverson, “Locating Hidden Servers,” in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2006, pp. 100–114.

[12] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, “Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, New York, NY, USA, 2013, pp. 337–348.

[13] S. J. Murdoch and R. N. M. Watson, “Metrics for Security and Performance in Low-Latency Anonymity Systems,” in *Privacy Enhancing Technologies*, 2008, pp. 115–132.

[14] R. Jansen and N. Hooper, “Shadow: Running Tor in a Box for Accurate and Efficient Experimentation,” Sep. 2011.

[15] N. Hopper, E. Y. Vasserman, and E. Chan-TIN, “How Much Anonymity Does Network Latency Leak?,” *ACM Trans Inf Syst Secur*, vol. 13, no. 2, p. 13:1–13:28, Mar. 2010.

[16] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald, “ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation,” in *Proceedings of the 4th Conference on Cyber Security Experimentation and Test*, Berkeley, CA, USA, 2011.

[17] M. Berman *et al.*, “GENI: A federated testbed for innovative network experiments,” *Comput. Netw.*, vol. 61, pp. 5–23, Mar. 2014.

[18] T. Fukui, “Anonymous Routing of Network Traffic Using Tor,” *Run my experiment on GENI*, 02-Feb-2017. [Online]. Available: http://witestlab.poly.edu/blog/anonymous-routing-of-network-traffic-using-tor/. [Accessed: 21-Apr-2017].

[19] T. Fukui, “De-anonymizing Tor traffic with website fingerprinting,” *Run my experiment on GENI*, 23-Apr-2017. [Online]. Available: http://witestlab.poly.edu/blog/de-anonymizing-tor-traffic-with-website-fingerprinting/. [Accessed: 24-Apr-2017].

[20] “Tor and HTTPS,” *Electronic Frontier Foundation*. [Online]. Available: https://www.eff.org/pages/tor-and-https. [Accessed: 17-Apr-2017].

[21] B. Schneier, “Attacking Tor: how the NSA targets users’ online anonymity | US news | The Guardian,” *The Guardian*.

[22] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The Second-generation Onion Router,” in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, Berkeley, CA, USA, 2004, pp. 21–21.

[23] H. Neal, “File:Onion diagram.svg,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/File:Onion_diagram.svg. [Accessed: 25-Apr-2017].

[24] The Tor Project Inc, “Tor Project: manual.” [Online]. Available: https://www.torproject.org/docs/tor-manual.html.en. [Accessed: 24-Apr-2017].

[25] “Tor directory protocol, version 3.” [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt. [Accessed: 24-Apr-2017].

[26] R. Dingledine and N. Mathewson, “Tor Protocol Specification.” [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. [Accessed: 05-Apr-2017].

[27] F. Liu, “How to Setup Private Tor Network,” *Programming Languages*, 09-Jan-2015. [Online]. Available: http://fengy.me/prog/2015/01/09/private-tor-network/. [Accessed: 18-Apr-2017].

[28] The Tor Project Inc, “Tor Project: Arm.” [Online]. Available: https://www.torproject.org/projects/arm.html.en. [Accessed: 09-Apr-2017].

[29] Shivam Singh Sengar, “How do traffic correlation attacks against Tor users work? - Information Security Stack Exchange.” [Online]. Available: https://security.stackexchange.com/questions/147402/how-do-traffic-correlation-attacks-against-tor-users-work. [Accessed: 09-Apr-2017].

[30] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, “Website Fingerprinting in Onion Routing Based Anonymization Networks,” in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, New York, NY, USA, 2011, pp. 103–114.

[31] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, “Effective Attacks and Provable Defenses for Website Fingerprinting,” in *Proceedings of the 23rd USENIX Conference on Security Symposium*, Berkeley, CA, USA, 2014, pp. 143–157.

[32] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster, “The Effect of DNS on Tor’s Anonymity,” presented at the Network and Distributed System Security Symposium (NDSS) 2017, 2016.

[33] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, “Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services,” in *Proceedings of the 24th USENIX Conference on Security Symposium*, Berkeley, CA, USA, 2015, pp. 287–302.

[34] T. Fukui, “Tor-experiment,” *GitHub*. [Online]. Available: https://github.com/tfukui95/tor-experiment. [Accessed: 25-Apr-2017].