Improving Tor Performance by Modifying Path Selection

by

MEHRDAD AMIRABADI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2014

ACKNOWLEDGEMENTS

I would like to express my appreciation to all those who have helped me in completing this thesis.

I would like to deeply thank my adviser, Dr. Matthew Wright, for his continuous support and helpful advice. I am also grateful to my committee members, Dr. Farhad Kamangar and Mr. David Levine, for their valuable comments and suggestions on my thesis.

I also wish to thank the iSec lab members, especially Mohsen Imani for his kind help during this work.

Finally, I would like to express my deepest gratitude to my parents and my sisters. Without their unconditional support I would not have made it.

November 20, 2014

ABSTRACT

Improving Tor Performance by Modifying Path Selection

Mehrdad Amirabadi, M.S.

The University of Texas at Arlington, 2014

Supervising Professor: Matthew Wright

Tor is a popular volunteer-based overlay network that provides anonymity and privacy for Internet users. Using the Onion Proxy (OP) client, users connect to a network of Onion Routers (ORs) and send their traffic through an encrypted path of three ORs. One of the main problems of the Tor network is its slow performance, and a key cause of this is the Tor path selection algorithm. In Tor, ORs are selected based primarily on their bandwidth. In this work, we improve on the Tor path selection algorithm by proposing a new algorithm that besides bandwidth, uses distance as a factor to help reduce propagation delay. In our design, we build circuits to the most popular destinations in advance. Since the Tor design does not guarantee choosing less congested ORs, we calculate the round trip time of the circuits during the circuit building phase and choose less congested circuits. We simulated our design in a discrete-event Tor simulator called Shadow. Our results show a significant improvement in performance compared to Tor, with 83% faster time to first byte and 77% faster total download times for accessing webpage-sized objects.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

A right to privacy is a human right to control one's private information [1]. Many third party companies track Internet users, collect data about individuals, and then sell that information to other companies. Considering Internet users' behavior in their online life and trackers which track users' activity on the Internet, protecting one's privacy is a growing challenge. To avoid getting tracked and identified by the companies that use this information and also to keep the private information safe, users can use the anonymity systems. These systems let users do web browsing anonymously, which means users can hide their identifiable information from the trackers.

There are many types of anonymous systems that improve users privacy on the internet such as Single hop proxies and Onion Routing systems. Single hop proxy like Virtual Private Networks (VPN) send the encrypted traffic through a middle server and then to the destination, like anonymizer.com [2]. The problem of single hop proxies is that the whole user's traffic passes through just one single hop in the path to the destination. So the proxy server knows both the client and her destination. Using these proxies cannot guarantee the complete anonymity of clients.

Tor is a popular volunteer-based Onion Routing anonymity system that has been designed to provide privacy on the Internet. Tor clients encrypt their traffic several times and send it through a *circuit* of three Onion Routers (ORs). There is not any single point in the Tor network that knows about both the source and destination of the passing traffic.

Although Tor provides stronger anonymity compared to the single hop proxies, its performance is reduced by multiple causes. Dingledine et al. described six known issues [3]: The Tor congestion control system, bulk users who are much fewer than normal users but use the main Tor resources, low Tor capacity that caused by high amount of users that are served by few number of relays, path selection, variable latency and network overhead, which impacts Tor users with lower bandwidth. In this thesis, we focus on path selection to improve Tor performance.

Tor path selection method currently is based on the bandwidth. More specifically speaking, the Onion Routers with higher bandwidth have a higher chance to be selected by Tor client software, called the Onion Proxy (OP). Since OPs select ORs without regard to their locations, each OR in the circuit could be far away from the next one. The current Tor path selection mechanism just consider node properties such as nodes' bandwidth in building the circuit. However, link properties are also important to improve the performance. In particular, having ORs far from each other adds a propagation delay to the circuits, which reduces performance.

In our design, we assign a weight to each OR calculated by including both bandwidth and distance. During circuit building, our algorithm chooses relays with higher weights, which means a good mix of bandwidth and inter-node distance. In our design, knowing the destination location is important. Akhoondi et al. use a separate circuit to 15 different DNS servers to resolve the user's request and find the closest web server geolocation [4]. But this algorithm adds an extra delay into the response time. Our solution to handle the mentioned scenario is using most popular locations based on the Alexa top 1000 websites as destinations, and create 4 circuits to them. These destinations selected using a small number of clusters in K-means. Also, we create a circuit with an exit node close to the client to use when the location of the destination is not yet known to the OP. When user sends her request, if it

contains the IP address of destination, we use the MAXMIND database [5] to find its location and choose a circuit with the exit node close to her destination. If the user's request doesn't contain an IP address, it chooses a circuit with an exit node close to the client. When we want to attach a stream to a circuit, we sort the candidate circuits based on the the total distance of each circuit. Then we choose a circuit with the lowest RTT among them and attach the stream to it.

We simulate our modified Tor in Shadow. The results for web clients show 83% improvement in time to first byte and 77% improvement in download time for the modified Tor compared to the Vanilla Tor.

In the next chapter, we consider Tor design and its problems in more details. Also, we discuss more on the related works and consider their issues. In chapter 3, we discuss more about our proposed design and considerations. In chapter 4, we show the simulation results and discuss on them. In chapter 5, we present the conclusion and the future work.

CHAPTER 2

BACKGROUND

2.1 Introduction

In this chapter, we first describe the Tor design in more detail, and then we consider prior works that have attempted to address Tor performance.

2.2 Tor Design

Tor is a volunteer-based overlay network that has been designed to provide anonymity and privacy on the Internet for users.

At the time of writing, there are more than two million users[1] [7] connecting to the Tor network simultaneously and more than six thousand ORs [8] are located around the world. Having three nodes in the path from client to her destination, makes tracking of Tor client and her destination almost impossible from any single point in the network. Besides Web browsing, there are many other services introduced by Tor developers, such as hidden services, Tor mail, and Tor chat.

There are three types of ORs: Guard nodes, Middle nodes, and Exit nodes. The number of Guard nodes, Middle nodes, and Exit nodes at the time of writing is around 2000, 4000, and 1000 respectively [8]. Figure 2.1 shows a basic schematic of the Tor network.

[1]Starting from the middle of August 2013, the number of Tor users suddenly increased from around 500,000 to more than 3.5 million users. Dingledine indicates that the inflation in Tor users was caused by a Botnet that uses the Tor network [6].

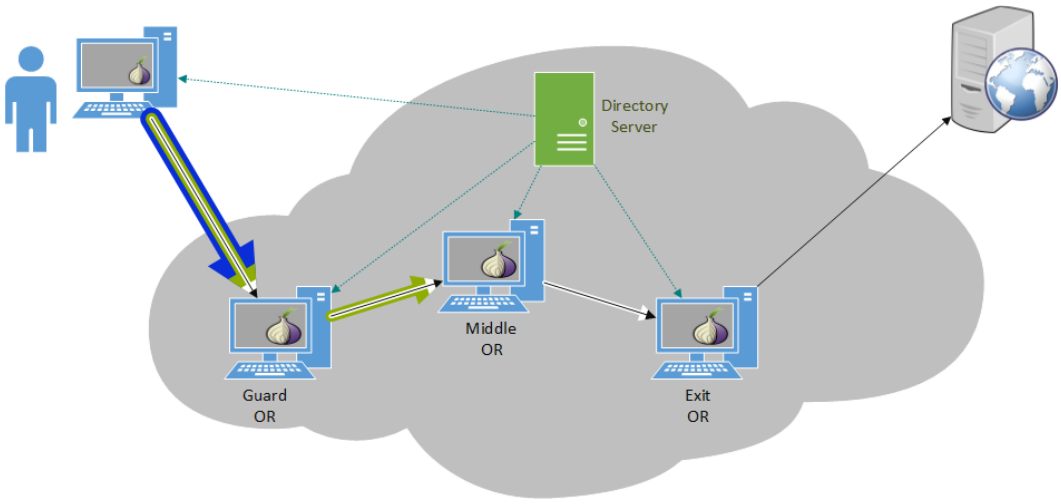

Figure 2.1. Tor Schematic.

To connect to the Tor network, users need to install Tor client software, known as Onion Proxy (OP). This software enables Tor users to build at most 14 circuits in Tor network and connect to their favorite destinations.

The OP first downloads a list of available ORs and information about them from *directory servers* figure 2.1 and this information enables the OP to create circuits. In particular, the client downloads two files with information about the ORs. The descriptors file includes information about the OR's advertised bandwidth, open ports, public keys, and exit node polices. The consensus file includes the list of ORs, flags (which specify the relay type), and bandwidth as measured by authorities. If the directory servers could not measure the OR's bandwidth, the consensus file reports its advertised bandwidth. Having a higher bandwidth increases the chance that the OR is selected by the OP.

To create a circuit, the OP first chooses the Exit node, which is the last OR in the circuit. The eligible Exit nodes in the consensus file are specified by an exit flag. The OP also checks the policy of the Exit nodes in the descriptor file to figure out whether the policies of the selected exit nodes meet the client's requirements

or not. Then the OP selects the first OR in the path, which is the Guard node. Guard nodes are selected based on their reliability. ORs with higher stability and uptime are considered as more reliable nodes. The OP picks small list of the qualified Guard nodes, extracted from Consensus file and stored in State file, for 30 to 60 days. After this period of time, it drops the Guards and select some new ones. Using same Guard nodes for this period of time helps majority of Tor users to be safe against timing attack or any other attacks that try to de-anonymize Tor users by deploying a malicious Guard node. The last OR to be selected is the Middle node. The main rule in choosing any ORs in a circuit is that they should not be in the same family or /16 subnet of other selected nodes. Using the ORs from same family makes Tor user vulnerable against some security attacks such as timing attack that try to link the Tor user and her destination. Because in the most cases ORs in a same family controlled by a same operator. Having fewer than three ORs in a circuit reduces anonymity and may enable adversaries to de-anonymize the user. On the other hand, having more than three ORs does little to increase the anonymity of Tor clients, but because of adding a propagation delay, it reduces performance significantly.

The unit of communication in Tor is a fixed-sized cell of 512 bytes. Having fixed-sized cells helps Tor to resist against fingerprinting attacks. Figure 2.2 and Figure 2.3 show the content of the control and relay cells, respectively. CircID is the circuit identifier and CMD is the command that specifies what the receiver is supposed to do with the data. Based on their commands there are two types of cells: Control cells and Relay cells. Control cells include padding and circuit management cells such as CREATE, CREATED, and DESTROY. Relay cells carry end-to-end stream data and as shown in Figure 2.3 have additional header fields: Relay header, Stream ID (an identifier for each TCP stream), Digest (a checksum for integrity checking), Len (length of payload), and the relay command. All the relay headers

and the relay payload are encrypted with 128-bit AES. The relay commands include RELAY EXTEND (to extend the circuit one hop), RELAY EXTENDED (to send an acknowledgement), and SENDME (used for congestion control).

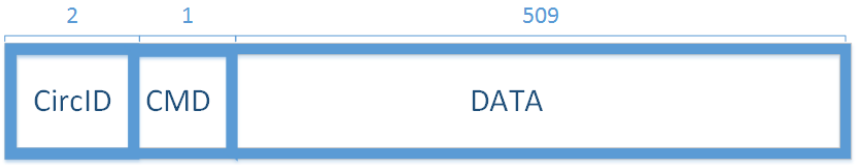

Figure 2.2. Control Cell.

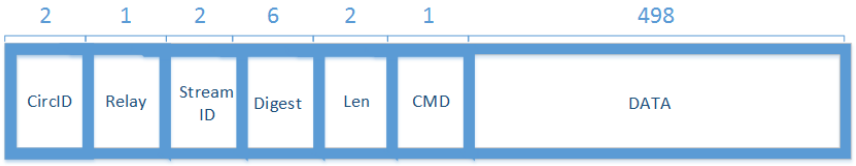

Figure 2.3. Relay Cell.

After choosing the nodes in the path, circuits get created by following these steps:

1. OP makes and sends a cell with a unique Circuit ID, the CREATE command, and the first half of a Diffie-Hellman handshake g^{x_1} as DATA encrypted with the Guard node's public key extracted from the descriptors file.
2. The Guard node responds back to the OP handshake with the CREATED cell containing the second half of the Diffie-Hellman handshake g^{y_1} and a hash of the negotiated key $K = g^{x_1 y_1}$. This key gives two symmetric keys that will be used for communicating in each direction between OP and OR.
3. After establishing the circuit between OP and the Guard node, OP sends the EXTEND relay cell to the Guard node containing the same circuit ID from

Step 1, Relay ID (Middle node ID extracted from the consensus file), CMD (EXTEND), and DATA (Middle node public key encrypted g^{x_2}).

4. The Guard node copies the encrypted g^{x_2} into a CREATE cell, chooses a new circuit ID, and sends it to the Middle node.
5. The Middle node responds back to the Guard node with the CREATED cell containing the second half of the Diffie-Hellman handshake (g^{y_2}) and a hash of the negotiated key $K = g^{x_2 y_2}$.
6. The Guard node wraps the Middle node payload into a relay EXTENDED cell and sends it to the OP.

The same procedure continues till the circuit is extended to an Exit node.

After selecting the ORs in the path, making a circuit, and negotiating keys, the OP encrypts the outgoing traffic three times and sends it to the Guard node. The Guard node then decrypts one layer of received traffic, figures out the successor node (the Middle node) and forwards the traffic to it. The Middle node performs the same procedure and forwards the traffic to the Exit node. The Exit node, after decrypting the traffic, learns the final destination of the client's traffic. If the traffic contains a host name, then the Exit node sends it to a DNS server to resolve the address. After finding the destination IP address, the Exit node connects to the destination web server and download the user's request. Then the Exit node sends back the downloaded content to the client using the same path and by following the same procedure.

The OP creates at most 14 pre-built circuits. Whenever the OP sends a new TCP request, a new stream will be created. Then the OP attaches the created stream to one of the available circuits. The OP does not attach new streams to circuits with lifetime of more than 10 minutes. After 10 minutes of using a circuit, if there are no

active streams using that circuit, OP discards the circuit. If there is a new stream and no circuit is available to handle it, OP will launch a new circuit for that stream.

2.2.1 Tor Security

If we change Tor, we need to understand the security implications of those changes. There are many security attacks that threat Tor users anonymity. Here we describe these attacks.

Timing attack are a well-studied attack on anonymity systems to link the user with her destination. This attack is classified in two types. In a *passive timing attack*, in which the adversary uses the timing pattern generated by user, while in an *active timing attack* the adversary generates a timing pattern and attaches it to the user's traffic. By analyzing the time stamps of the traffic on the entry and exit node, the adversary can detect client and her destination [9]. The adversary in this attack should be able to monitor both of the Guard and Exit node traffic. Feigenbaum et al. design a padding scheme that to some extend saves user from active adversary but it's effective against passive adversary [10].

Congestion attacks are another type of attack on Tor that enables adversaries to detect the ORs used in a path created by OP. Using congestion attack and timing analysis Murdoch and Danezis could identify all the nodes in a Tor user's path by building one circuit at a time through all the Tor ORs [11]. Evans et al. amplify this attack by using a flaw in Tor design and build long circuits that loop back on themselves [12]. In another use of the congestion attack, Hoper et al. consider the low resource attack by which adversary could predict whether two connections on a malicious website sharing same circuits. Also, in another attack they could retrieve some information about the clients who visit that website [13].

Website fingerprinting is an attack that can reveal the client's destination by passively monitoring her traffic. The strategy of the attacker in is comparing the classified packet traces of several web pages with the victim's web browsing traffic traces [14].

Mittal et al. present a throughput attack, by which an adversary who controls an Exit node can identify the Guard node in a circuit [15]. In this attack, the adversary uses a probe client to measure the throughput of each circuit and correlate it with the throughput measured on the Exit node.

Borisov et al. examined how denial of service attacks can be used to reduce the anonymity of Tor [16]. Considering the effect of Guard nodes on clients, they conclude using honest Guard nodes make Tor clients resistant against selective DOS attacks that is more effective attack, which makes users more vulnerable by only reducing the performance of the system rather than attacking the entire system. But the attack becomes more powerful when Tor clients use dishonest Guard nodes.

The Sniper attack [17] is a denial of service attack that enables an adversary to disable target relays anonymously. In this attack, the attacker causes a memory outage on the target OR by ignoring Tor's congestion control mechanism. For example, if the target is the entry node, the client (adversary) creates a circuit using that entry node and sends several SENDME cells to the exit node, which is also controlled by the adversary, and asks it to increase the window size. The exit node ignores the package window limit and sends downloaded data back to the client. The client stops reading the TCP connection and causes data to be buffered on the entry node. This causes a buffer overflow and leads to a memory outage on the entry node, which makes the OS terminate Tor.

The CellFlood Attack [18] is another type of DOS attack that targets Tor ORs. By investigating the different types of cells in communicating between OP and ORs

in circuits, Barbera et al. identified processing of a CREATE cell as an expensive process for ORs in terms of CPU usage. Since each OR can handle a few CREATE cells per second, receiving the higher rate of CREATE cell causes relays to reply with a DESTROY cell.

2.2.2 Tor Performance

Dingledine et al. identified six main causes for Tor's low performance [3]. The first problem is congestion, which is caused by the lack of a procedure to control web browsing streams and bulk transfers. This problem has two reasons. First, there is only one TCP connection for all the circuits between two relays, which is beneficial for anonymity but harms performance. Second, Tor's congestion control window is large: 512 KB for a circuit window and 256 KB for a stream window. Reducing the control windows can reduce memory usage and latency. AlSabah et al. investigate this issue and show that using smaller fixed-size circuit windows reduces the response time but increases bulk download time [19].

The second problem that reduces Tor's performance is users with high bandwidth usage. Solutions mentioned for this problem include: give priority based on recent activity, throttle some protocols on exit relays, limit the protocols supported by OPs, limit streams on OPs, limit the exit policy on exit nodes to port 80 and 443, and educate users. AlSabah et al. study the quality of service provided by Tor and found that a small number of bulk downloading users use the majority of Tor network resources [20]. To solve this problem, they propose DiffTor to classify Tor's circuits by application and assign a different quality of service to each type of application.

The third problem is the capacity of the Tor network. To increase the size of networks, Dangledine et al. suggest encouraging volunteers to run the relays, funding relays directly, scanning relays to find overloaded relays or broken exits, checking the

advertised bandwidth of the relays, speeding up listing of dynamic IP relays in the relays list, incentives to run relays, and automatically making OPs that are reachable by other OPs run as relays [3].

The fourth problem is path selection. In the current mechanism, the bandwidth of a relay is estimated based on the relay's traffic burst in the past few days. The default path selection increases the chance of selecting a relay in proportion to its bandwidth. The problems caused by this default path selection algorithm include:

- Load balancing: Fast relays have less load than the slow relays;
- Deficiency of bandwidth estimation: Snader and Borisov proposed three methods including checking the picks handled by OR in the past day, which is the current Tor bandwidth estimation method, active probing by directory authorities, and monitoring data rates in communication between relays by each individual relay [21].
- Considering Bandwidth as the only parameter to select relays: using latency as a factor to select relays could be a solution.
- Exit policies: More permissive exit policies make Exit nodes candidate for more circuits and increase their load.
- Overloaded old entry guards: Since the main factors in selecting Guard nodes are their stability and reliability, older Guard nodes have a higher chance to get selected by OPs.

The fifth problem is the ways that Tor handles variable latency. Tor rate limiting has been implemented by using a token-bucket approach to control the rate of incoming and outgoing network traffic. Also, Tor uses a predefined timeout to discard circuits, finding a more accurate value for the timeout may improve the performance. The other method used by Tor to handle the variable latency, which may reduces Tor

performance is discarding circuits when extending them is unsuccessful. Dingledine et al. suggest trying few other relays to extend the circuit before abandoning it[3].

The sixth problem is Tor network overhead, which causes trouble for users with slow connections (eg. users on cell phones or modems). Possible solutions include reducing directory overhead and improving TLS overhead by removing empty TLS application records. Ben Laurie, an OpenSSL developer, noted that Tor can remove the empty TLS application records [22]. These records are inserted for security reasons but the attacks are very expensive and they give little information to the attacker.

Jansen et al. address the current congestion problem in Tor socket management [23]. By evaluating Tor performance in both the public Tor network and the Shadow Tor simulator, they find that socket flushing and circuit scheduling are the main reasons for Tor's low performance. Tor writes data to each socket without considering the state of the socket, and because the kernel can not handle this much data, it causes a reduction in performance. To solve this problem, Jansen et al. designed a new socket transport mechanism called Kernel-Informed Socket Transport (KIST). Using this method, Tor doesn't send data to the socket if the kernel cannot send it. Also, all the writable circuits have a chance to get chosen by Tor. By evaluating KIST performance on Shadow, they conclude that KIST reduces kernel-level and end-to-end congestion. Additionally, their security evaluation of this new method shows that it is resistant to throughput attacks.

2.3 Related Works

As it's mentioned in previous section Tor's low performance causes by six different problems. One of them is the Tor path selection method. In this section, we consider the solutions that have been proposed to address this problem.

Akhoondi et al. focus on path selection to improve the Tor performance [4]. They consider the propagation delay as a controllable factor that could be improved and geographical distance as a sign of propagation delay. They set up 15 geographically distributed DNS servers on Planet-Lab nodes and send each of them name resolution requests via a separate circuit. After receiving the IP addresses from DNS servers, they locate the closest web server location to the client. Then they cluster all the available relays into 200 clusters and create all the possible cluster-level paths to the calculated location. They weight each built path based on the length of the path using the Weighted Shortest Path (WSP) algorithm to select the lowest weighted path. In the selected cluster-level path, they randomly select one node from each of its clusters and build the relay-level path. Their results show 25% improvement in Tor performance but their design also has some drawbacks. The first drawback is its network overhead, as it needs to create all possible paths to find the best path. The other problem is the cluster's bandwidth. since their clustering is based on location, it is possible to have clusters with lots of high bandwidth ORs and clusters with few low bandwidth ORs. The second drawback is that they do all of these processes for each destination. Some webpages include resources like images and advertisements from numerous locations. The OP would need to build several circuits for these resources. The third drawback is that the proposed mechanism for DNS resolution is costly and slow, as it sends the DNS request to 15 different servers distributed all over the world and waits for the responses adding a huge delay.

Wang et al. introduce *node latency* as a parameter to measure the node's congestion [24]. In their approach, the OP calculates congestion delay using both active and opportunistic methods, and then use the measured latency to avoid congested nodes in path selection to improve Tor performance. They use both the short-term and long-term congestion in their work, the short-term congestion caused by the traf-

fic of whole network and the long-term congestion caused by the relay's bandwidth. To handle short-term congestion, the OP calculates the round-trip time of each circuit, and if the mean of the measured congestion is higher than 0.5 seconds, they drop the circuit and use another one. To handle long-term congestion, they select the nodes in building the circuit based on their measured congestion times. Their results show improvement in quality of service and load balancing. In this work, the authors ignore the propagation delay and just focus on the delay caused by the congestion.

Snader and Borisov, proposed a modification in Tor path selection, which helps Tor clients choose between anonymity and performance base on their needs [21]. Using the equation below, they consider s as a parameter to choose between different levels of performance and anonymity.

$$\frac{1-2^{sx}}{1-2^{s}} \tag{2.1}$$

As s increases, OP selects high bandwidth nodes and sticks to the high bandwidth portion of the network. However, they may cause the high bandwidth relays to be over-utilized and congested, thereby decreasing expected performance of the whole network.

Sherr et al. describe Tor's default relay selection as a factor that impact the performance [25]. Authors create a model of the Tor network to evaluate the recent published papers modifying path selection. They consider throughput, time to last byte, and RTT of Tor, Snader/Borisov [21], Unweighted Tor, in which Tor relays get selected uniformly at random, Coordinate [26] where path selection is based on the relays' locations, LASTor [4], and Congestion-aware [24]. Their investigation shows that the path selection algorithms that do not consider bandwidth as a factor in relay selection have poor performance.

2.4 Evaluation

To study the Tor network, we can either model it using simulators such as ExperimenTor [27] or Shadow [28] or a private Tor network in PlanetLab. Limited experiments can also be conducted on the live Tor network.

ExperimenTor is a testbed for evaluating Tor security and performance. It features OP traffic generation and relay topologies. It also allows its users to change the behavior of the Tor clients, relays, and network configurations when simulating Tor network.

Shadow is a discrete-event network simulator that includes some plugins to simulate the whole network in a single machine. Scallion is a Shadow plugin that uses Tor source code and lets users run their Tor project experiments.

PlanetLab is a global network that helps researchers to evaluate network services by deploying them on a network of more than 1300 nodes [29]. Tor performance can be evaluated using PlanetLab by setting up a private Tor network. There are some tools that ease the management of the nodes, but in comparison with the simulators like Shadow, extensive experimentation would be time consuming.

The last option to evaluate Tor performance is using the live Tor network. Due to anonymity and performance reduction of the Tor network real users, this option is not recommended.

CHAPTER 3

DESIGN

3.1 Introduction

In this chapter, we describe our new Tor design and its improvement over the current Tor design (Vanilla Tor), at first we show the distribution of the existing Tor relays propagated in all around the world, then we explain our new relay selection algorithm and compare it to Vanilla Tor. Then we describe the way we create circuits and how we attach streams to the created circuits.

3.2 Design

Figure 3.1 shows the distribution of relays in all around the world.

Figure 3.1. Relays distributions.

Each relay is a computer that runs Tor software, configured to redirect the client's traffic to the next hop. The relay operators should determine the bandwidth

that they want to share with Tor network. We name this bandwidth, known as the advertised bandwidth.

The cumulative fraction of relays based on bandwidth is shown in Figure 3.2. 70% of the relays' bandwidth are less than 50 MBps. Tor selects relays randomly, weighted by bandwidth. The higher bandwidth a relay has, the grater the chance that the relay will be selected.

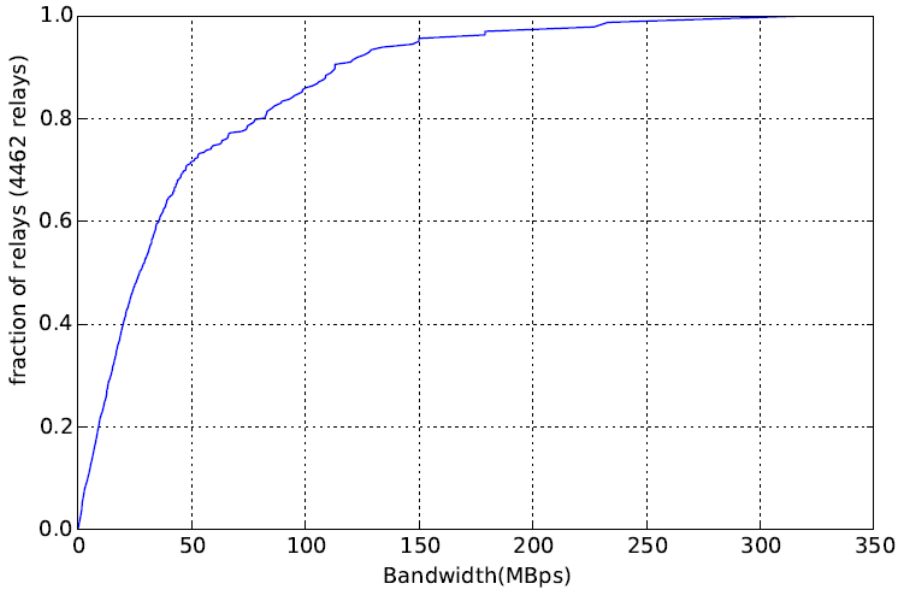

Figure 3.2. The Cumulative fraction of Bandwidth for all Tor relays.

One of the key reasons that reduces Tor performance is TorâĂŹs path selection. Tor uses bandwidth as the only parameter to select ORs in the paths. This may cause creating of expensive, high-latency paths. In this situation, the ideal solution for us is knowing the latencies of all the paths and choose the best path with lowest latency.

But the problem is knowing the latencies of all the possible paths is $O(n^2)$. In general, latency is caused by three factors: queuing delay, transmission delay, and propagation delay [4]. Queuing delay is the time that the routers is the network spend to send the enqueued packets through the Internet, and this is due mainly to congestion. Transmission delay is caused by the medium itself, i.e., it is the time to get data onto the wire. Propagation delay is the time that packets spend to travel from one router to the next. This time depends on the path length. So we prefer to use geographical distance as a rough proxy for latency. Prior work [4] failed to consider bandwidth, which is known to be necessary [25]. So our design goal is to use both distance and bandwidth to pick relays.

3.2.1 New Path Selection Method

We now describe our proposed path selection algorithm that aims to meet this goal. We calculate the weight of each relay using this equation:

$$W_T = \alpha * W_{BW} + (1 - \alpha) * W_D \tag{3.1}$$

This equation considers both bandwidth and distance in the assigned weight of a relay. In this equation, α is a parameter that we can use to tune the share of bandwidth and distance in the weights. Reducing α increases the importance of distance to the weighting while increasing α increases the importance of bandwidth. W_{BW_i} and W_{D_i} for any specific relays are computed as follows:

$$W_{BW_i} = \frac{BW_i}{BW_{max}} \tag{3.2}$$

$$W_{D_i} = (1 - \frac{D_i}{D_{max}}) \tag{3.3}$$

In W_{BW_i}, BW_i is the bandwidth of each relay downloaded from the directory server. In W_{D_i}, D is the distance that is computed based on relay's role in each path. Figures 3.3,3.4,3.5 show how we compute the distance for each type of relay. Since our points here are geographical locations, we cannot use the the Euclidean distance between two points. D is the great-circle distance between two points on a sphere from their longitudes and latitudes, so we calculate the distance between them using the Haversine formula [30]:

$$\begin{aligned} a &= \sin^2(\frac{\Delta\varphi}{2}) + \cos\varphi_1 * \cos\varphi_2 * \sin^2(\frac{\Delta\lambda}{2}) \\ c &= 2 * \arctan 2(\sqrt{a}, \sqrt{1-a}) \\ d &= R * c \end{aligned} \tag{3.4}$$

where φ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371 km).

D_{Exit} is calculated based on the distance from the Exit node to the destination (Figure 3.3). Tor builds some circuits preemptively, before the client requests a circuit to a destination, so we do not have the destination to find the best exit node. To solve this issue, we build circuits preemptively to five destinations and check every second to have a circuit to these destinations. Among these five destinations, four of them are popular destinations that we selected based on the Alexa top-1000 most popular websites. The fifth destination is the client's location. When the client attempts to visit a website using the destination's hostname instead of its IP address, it means that OP does not have the geographical coordinates of the destination. In this situation, we use the client's geographical coordinate as the destination location. To find the exact location of ORs, client destination, and client location, we use a

database provided by Maxmind[5] to convert their IP addresses to a geographical coordinates.

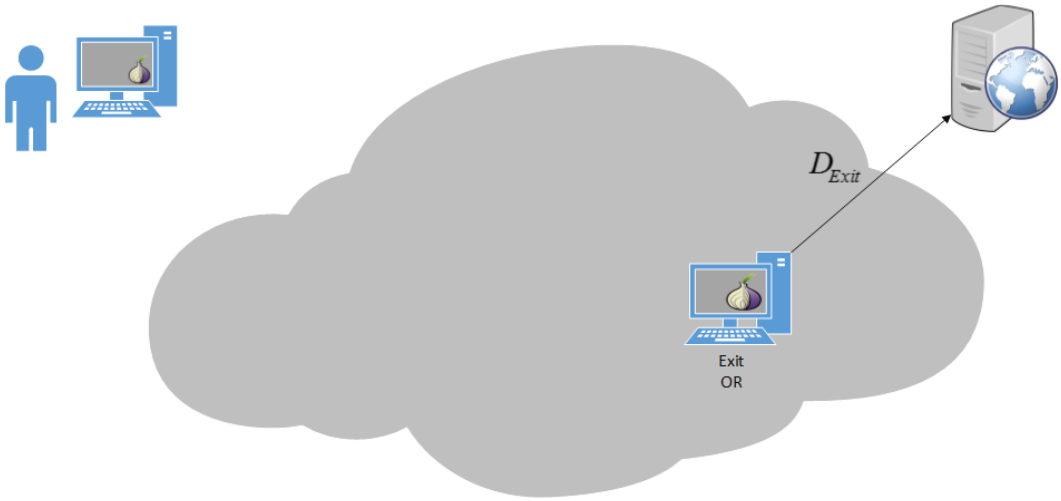

Figure 3.3. Exit node Distance.

D_{Guard} is calculated based on the distance between the client and the Guard node. (Figure 3.4)

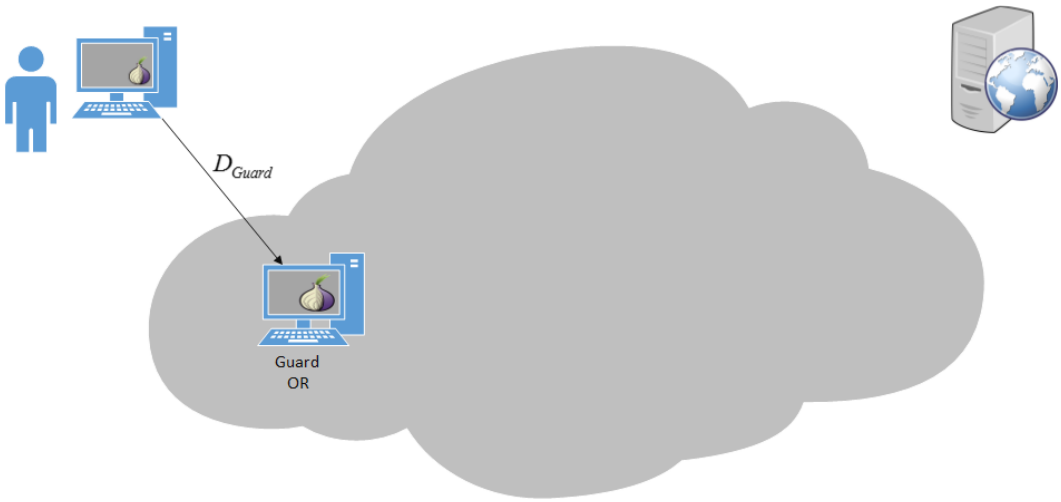

Figure 3.4. Entry node Distance.

And $D_{Middle} = D_{GM} + D_{ME}$ as illustrated in Figure 3.5.

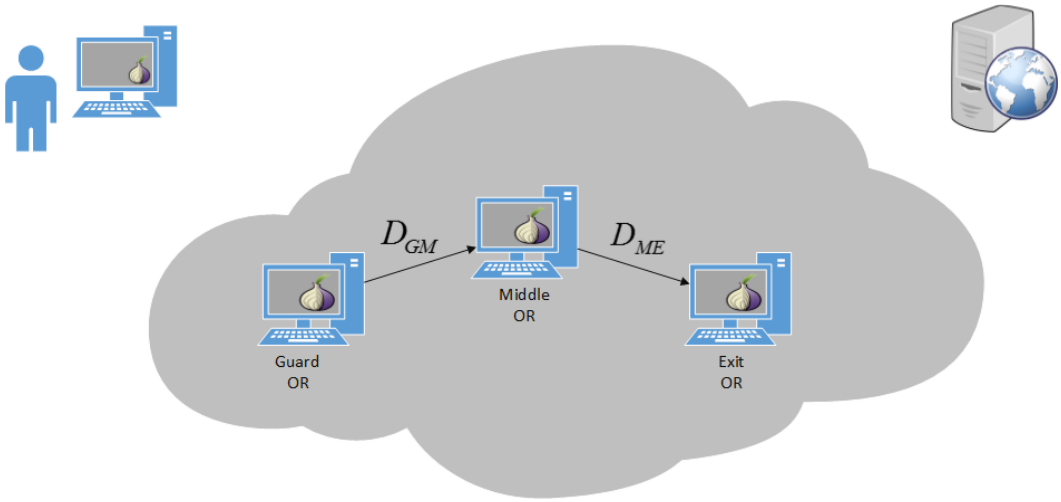

Figure 3.5. Middle node Distance.

3.2.2 Circuit Building

After choosing the nodes, we build a circuit that passes through these nodes. These circuits should not only be comprised mostly of higher bandwidth ORs, but they will have end-to-end geographical distances that are shorter than in Vanilla Tor. Another source of latency in the network is congestion; In OR selection, we ignored congestion. The OP may select a relay that has high bandwidth but it is congested. To handle this issue, we measure the round trip time (RTT) between the client and the Exit node and consider it as a sign of congestion. Higher RTTs suggest that there are congested nodes on the circuit, and we avoid these circuits in our circuit list. To calculate the RTT, we use one of the opportunistic methods mentioned in [24]. In this method, shown in Figure 3.6, we calculate the time taken to send an EXTEND cell from the client till receiving the EXTENDED cell back from the chosen exit node.

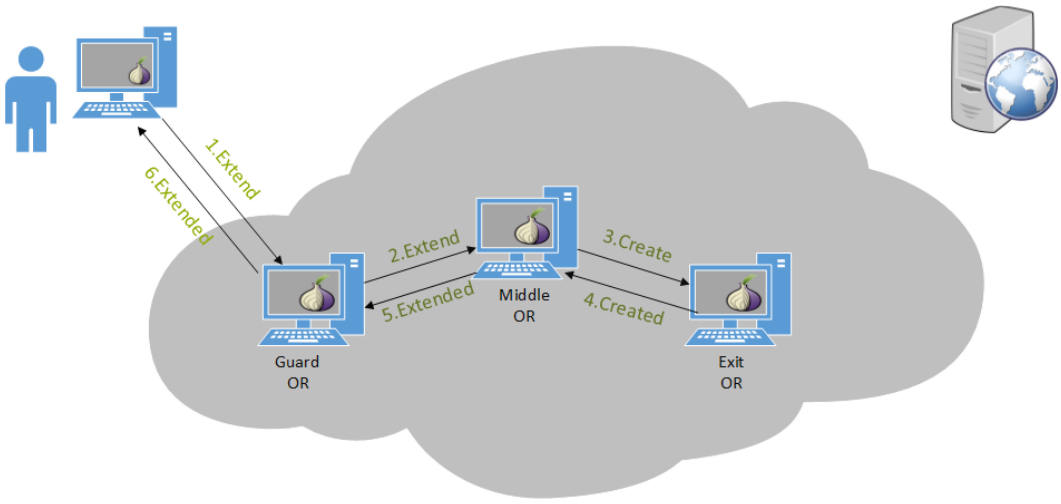

Figure 3.6. Calculating RTT using Extend cell.

3.2.3 Attaching streams to circuit

There are two scenarios to attach a stream to a circuit:

1. If we have the IP address of the destination, then we have its location from the Maxmind database. We then choose a circuit whose exit node is closer to the destination, i.e., a circuit with D_{Exit} as shown in Figure 3.7. We calculate the total circuit distance by adding the distances between client to Guard node, Guard node to Middle node, Middle node to Exit node, and Exit node to destination. $(D_{circ} = D_{Guard} + D_{GM} + D_{ME} + D_{Exit})$.
2. If we have only the web address of destination, it means that we do not know the location of destination web server. In this case, we choose a circuit with an exit node close to the client, i.e. a circuit with smaller D_{Exit} as shown in Figure 3.8. We calculate the total circuit distance by adding the distances between client to Guard node, Guard node to Middle node, Middle node to Exit node, and Exit node to client. $(D_{circ} = D_{Guard} + D_{GM} + D_{ME} + D_{Exit})$.

After building the circuits, we sort them based on their lengths (D_{circ}) increasingly. When the user's browser sends its request to the OP to open a web page, a

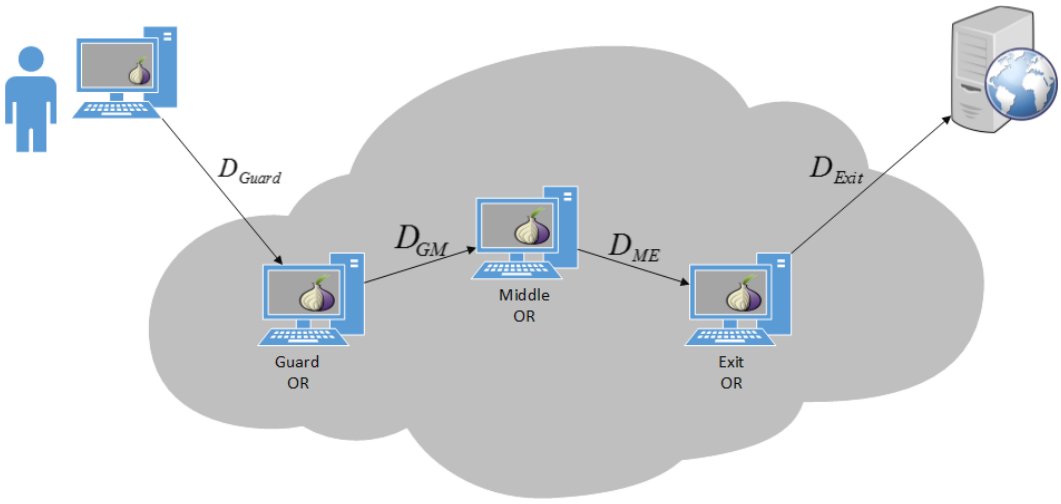

Figure 3.7. D_{circ} when we have the destination.

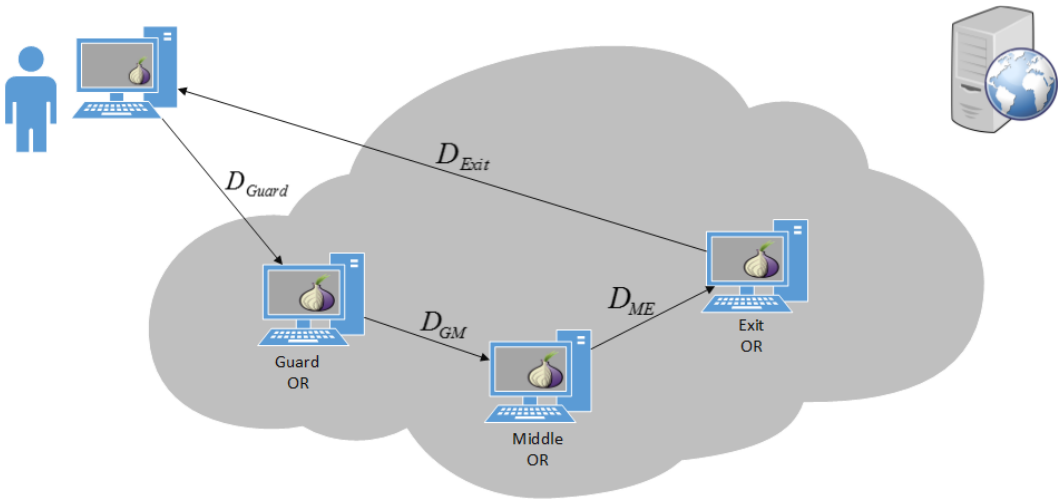

Figure 3.8. D_{circ} when destination location is unknown.

new stream will be created. To redirect the user's traffic to her destination, the OP needs to attach the TCP stream to an available circuit. This circuit in our design is the one with minimum RTT among the top three sorted circuits in the circuit list.

In addition to the mentioned algorithm, there are some other algorithms to find the shortest path between client and her destination. For example we can calculate

all the possible paths and find the shortest one like [4], but this method will add an extra overhead on the Tor network and is not desirable.

3.2.4 Finding The Most Popular Destinations

When Tor user connects to the Tor network, in Vanilla Tor, OP creates some circuits preemptively. When the Tor user sends her request, OP attaches the created stream to any available circuit. In our design, we follow a similar procedure, but we also account for geo-location. Since we are not aware of user's target destination geo-locations in advance , we create some circuits to popular destinations located in all around the world. To find the most popular destinations, we choose 10 planetLab nodes located in different areas based on the number of Tor users [7]. From those locations, we connect to the 1000 Alexa top popular websites[31] and fetch all the available links on those websites. Then we calculate the geo-location of those links. Using the K-means clustering algorithm, we classified the calculated locations into four areas as shown in Figure 3.9. To create pre-built circuits, we select Exit nodes close to each cluster.

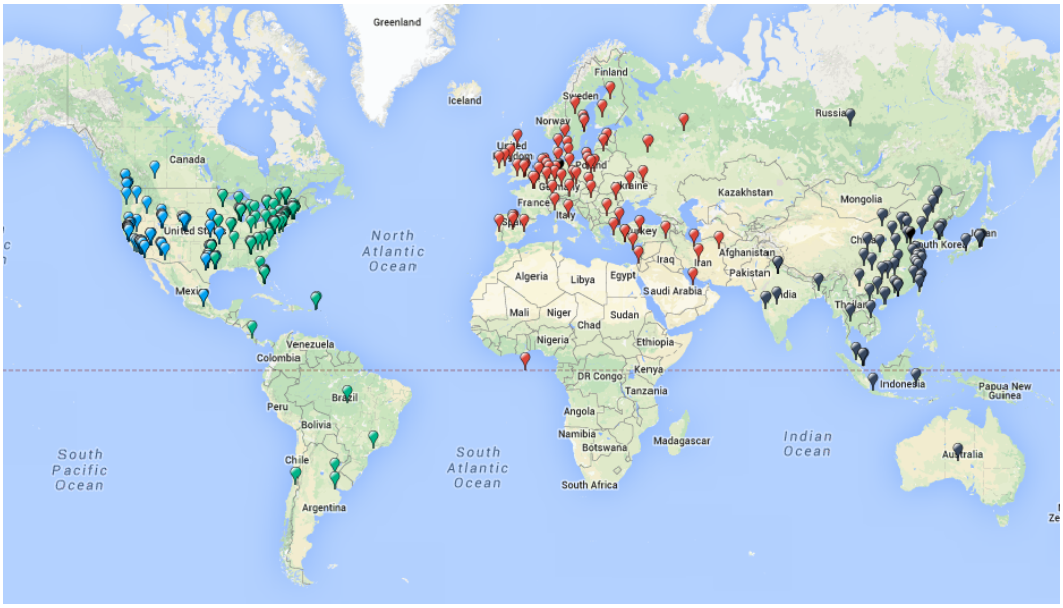

Figure 3.9. Popular websites location based on Alexa.

CHAPTER 4

EXPERIMENTS

4.1 Introduction

To evaluate our design, we use the Shadow simulator. In this chapter, we discuss about our simulation design and the simulation results.

4.2 Shadow Simulator

Shadow is a discrete-event simulator [32] for experiments on Tor designs. Shadow uses a topology file in XML format, which is a blueprint of a network that plugins work on it. This file specifies the structure of network topology and properties such as latency, jitter, and packet loss rates. Plugins are the libraries that linked to the real applications such as Tor. Shadow dynamically loads these libraries to natively execute the application code. To simulate our modified Tor code, we use the Scallion plugin, which enables communication between Tor and Shadow. Scallion also contains scripts that assist in analyzing results, generating Tor topologies, and running experiments using the generated topologies. It uses an XML configuration file to specify the Tor network topology, including the number of relays, clients, directory servers, and the specific configuration of each of them. We also use the File-transfer plugin to test Tor performance on the client side.

4.3 Simulation Environment

Our experiment was based on the *tiny* topology provided in Shadow, which includes 20 file-servers as destination, one Directory authority server, four Exit-Guard

nodes, 15 Guard nodes, 8 Exit nodes, 24 Middle nodes, 162 web clients, 18 bulk clients, 10 Shadowperf clients with 50 KiB, 10 Shadowperf clients with 1 MiB, and 10 Shadowperf clients with 5 MiB. We ran our experiments on an AMD Phenom II X4 975 Processor 3.4 Ghz with 16 GB RAM. The running time of the experiments is more than two hours for modified Tor and is around one hour for Vanilla Tor. The reason for the higher running time is the logs we put in our code for troubleshooting purposes.

We follow the procedures suggested by Jansen et al. to model clients, servers and ORs. We have three types of clients in our experiments, web clients, bulk clients and Shadowperf clients. The web clients download 320 KiB of data, which is the average size of web pages, and it waits between 1 to 20 seconds uniformly at random before starting the next download to simulate the web surfing of users [33]. The second type is bulk clients, who downloads 5 MiB of data without pausing between completion of download and starting the new one. Shadowperf clients download files like Torperf clients [34]; They repeatedly download files of size 50 KiB, 1 MiB, and 5 MiB in sequence and wait 60 seconds between downloads. Also, Shadowperf clients download each file using a new circuit with a new Guard node.

The total client load and share of each client type in our experiment with the modified code are 162 web clients with 48.72%, 20 bulk client with 35.06%, 10 Shadowperf client for 50 KiB data size with 0.44%, 10 Shadowperf client for 1 MiB data size with 6.00%, 10 Shadowperf client for 5 MiB data size with 9.78% of total client load.

4.4 Simulation Results

Here we show the output of the Scallion plugin using our modified Tor code with different values of α. We calculate performance based on Time To First Byte (TTFB)

and Time to Last Byte (TTLB), which are the time taken to send an HTTP request and time taken to receive the first byte or last byte of data back, respectively [35]. TTFB here is a measure to show how quick the response time of a web server is over the circuits. Factors that influence TTFB include: ,emory leaks, too many processes and connections, external resource delays, inefficient SQL queries, slow database calls, insufficient server resources, overloaded shared servers, and inconsistent website response times [36]. In the Shadow simulation environment, however, the whole traffic of network goes through few ORs with limited resources and thus, the most significant reason for delay is the Tor circuit's performance.

Figures 4.1,4.2,4.3,4.4 show the TTFB and TTLB of web and bulk clients respectively for different values of α. Comparing different values of α in time to first byte and time to last byte of web and bulk clients, we can conclude 0.25 is an appropriate value for α in our proposed function, which means that having higher share of distance in our function and choosing less congested circuits guarantees the performance improvement.

Figure 4.5 shows the CDF of TTFB for the web clients when receiving the first byte of 320 Kibibytes of data, using our modified Tor and Vanilla Tor. Comparing the median of users in modified Tor and Vanilla Tor shows around 83% improvement in time to first byte. Around 90% of the modified Tor web clients with $\alpha = 0.25$ have TTFB of less than 10 seconds, while around 70% of Vanilla Tor clients download the first byte in less than 10 seconds. Modified Tor with $\alpha = 0.25$ performs better than $\alpha = 0$ and $\alpha = 1$ and much better than Vanilla Tor. Since $\alpha = 0.25$ places more weight on distance than on bandwidth in our weighting function, it has more effect on TTFB. When $\alpha = 1$, OP selects nodes based on their bandwidth only, which is similar to Vanilla Tor. But the performance of $\alpha = 1$ is around 55% better than Vanilla Tor for the median of users. The reason that improves performance of $\alpha = 1$

is the stream attaching procedure in modified Tor, in which we calculate the RTT of each created circuit and choose the circuits that are less congested. When $\alpha = 0$, OP selects nodes based on their geolocations. In this case, the created circuits are shorter and have less propagation delay in comparison with other values of α. But using $\alpha = 0$ increases the chance to select the low bandwidth nodes. However, the performance improvement with $\alpha = 0$ is around 80% in comparison with Vanilla Tor.

The TTFB improvement is slightly better for bulk clients in comparison with web clients. Comparing the median of bulk users in modified Tor with $\alpha = 0.25$ and Vanilla Tor shows around 86% improvement in time to first byte. As shown in Figure 4.6, almost 95% of the Bulk clients who use our modified Tor with $\alpha = 0.25$ download the first byte of 5 MiB in less than 10 seconds, but only 60% of Vanilla Tor clients could download the first Byte of data in this period of time. When $\alpha = 1$ in modified Tor, the TTFB of the median of the users is around 64% better than Vanilla Tor. This improvement is about 85% for $\alpha = 0$, which is almost the same as $\alpha = 0.25$ for the median of bulk users.

Figure 4.7 shows the CDF of time to last byte for web clients when downloading 320 KiB. Comparing the median of web clients in modified Tor with $\alpha = 0.25$ and Vanilla Tor shows around 77% improvement in download time. When $\alpha = 1$ in modified Tor, the TTLB of the median of web users is around 48% better than Vanilla Tor. This improvement is about 76% for $\alpha = 0$. In Figure 4.7, after 20 seconds, more than 91% of the web clients with $\alpha = 0.25$ can download 320 KiB file but in the same amount of time only 65% of Vanilla Tor web users could download 320 KiB file.

Figure 4.8 shows the CDF of time to last byte for Bulk clients when downloading 5 MiByte. Comparing the median of web clients in the modified Tor with $\alpha = 0.25$ and the Vanilla Tor shows around 80% improvement in download time. When $\alpha = 1$ in modified Tor, the TTLB of the median of bulk users is around 70% better than

Vanilla Tor. This improvement is about 80% for $\alpha = 0$. In Figure 4.7, after 200 seconds, more than 90% of the bulk clients with $\alpha = 0.25$ can download 5 MiB file but in the same amount of time only 61% of Vanilla Tor bulk users could download 5 MiB file.

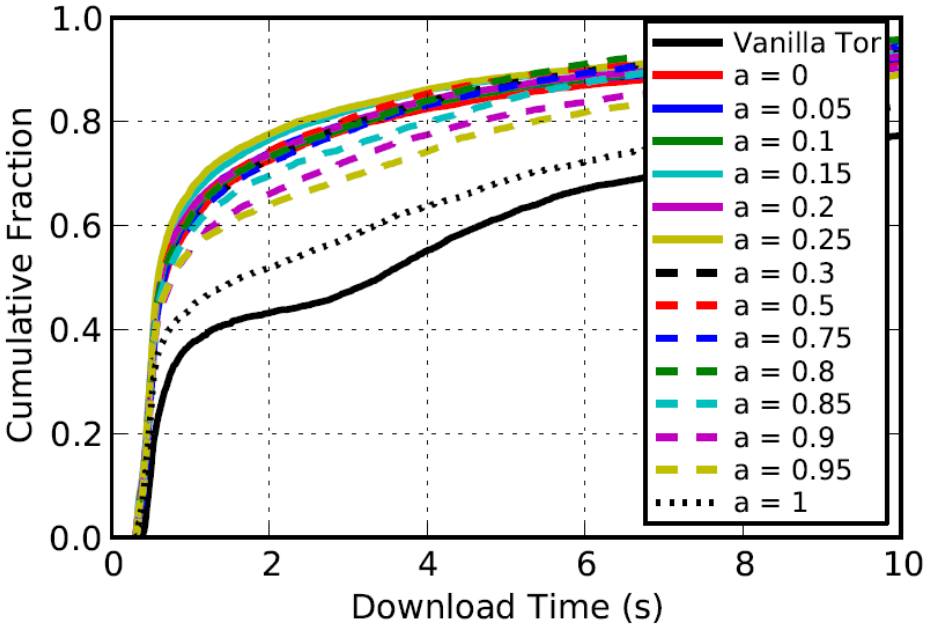

Figure 4.1. Analyses of α for web clients time to first byte (320 KiB).

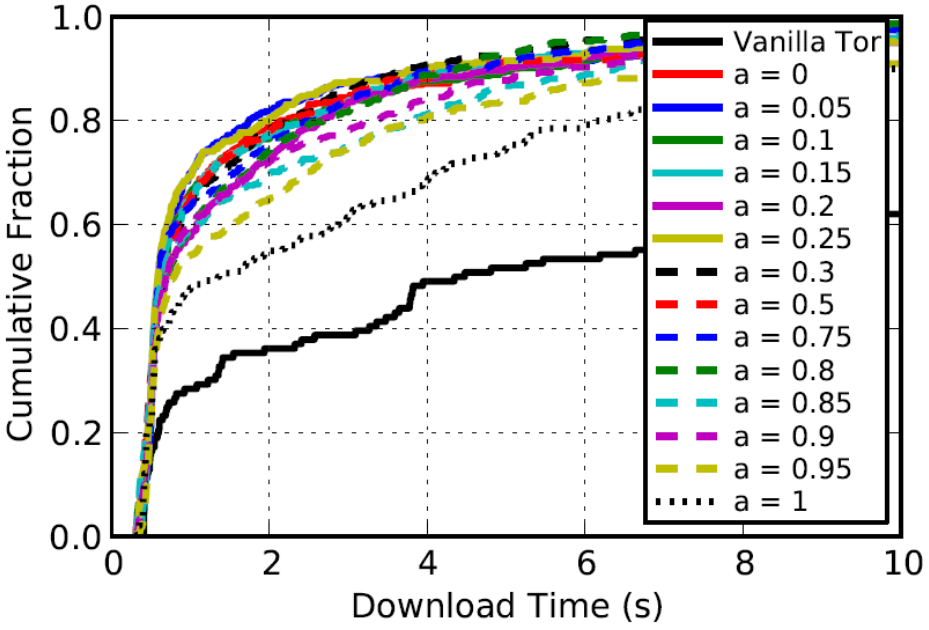

Figure 4.2. Analyses of α for bulk clients time to first byte (5MiB).

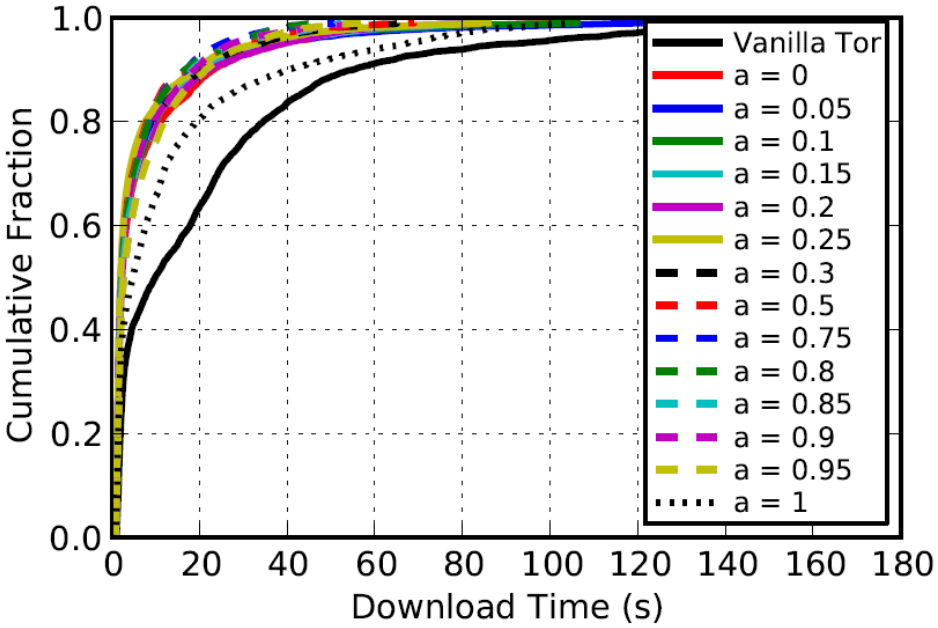

Figure 4.3. Analyses of α for web clients Time to last byte (320 KiB).

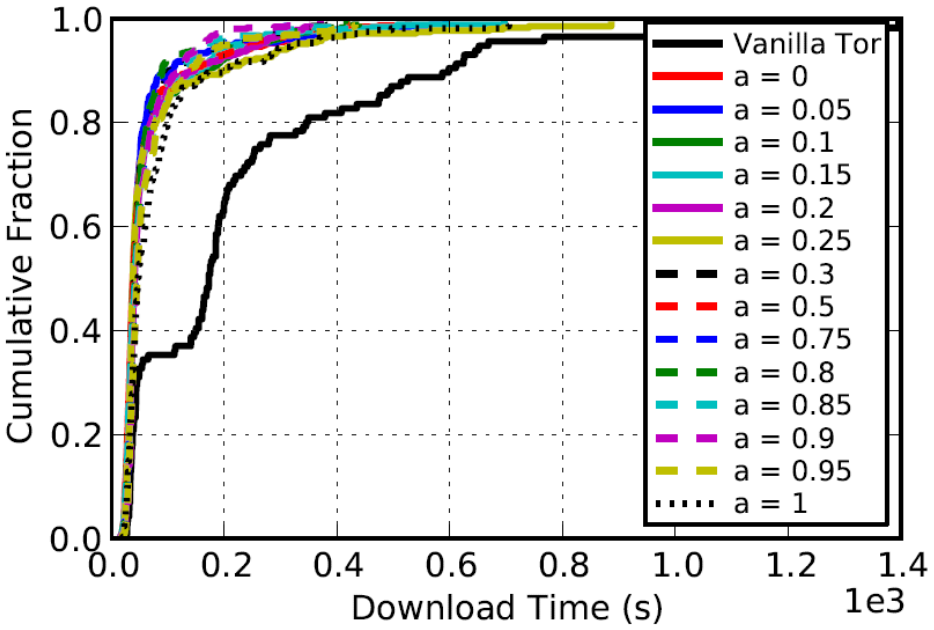

Figure 4.4. Analyses of α for bulk clients time to last byte (5 MiB).

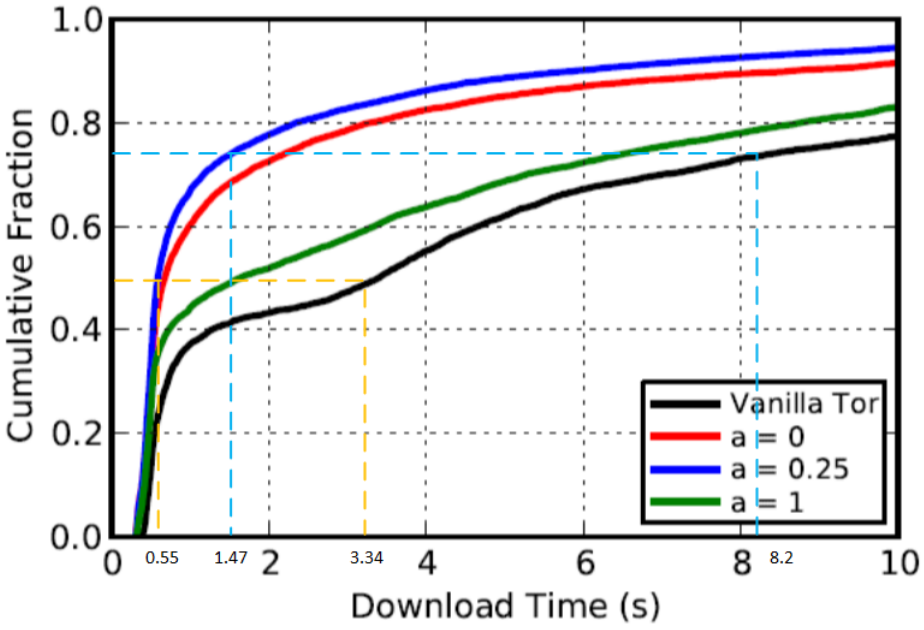

Figure 4.5. Web clients time to first byte (320 KiB).

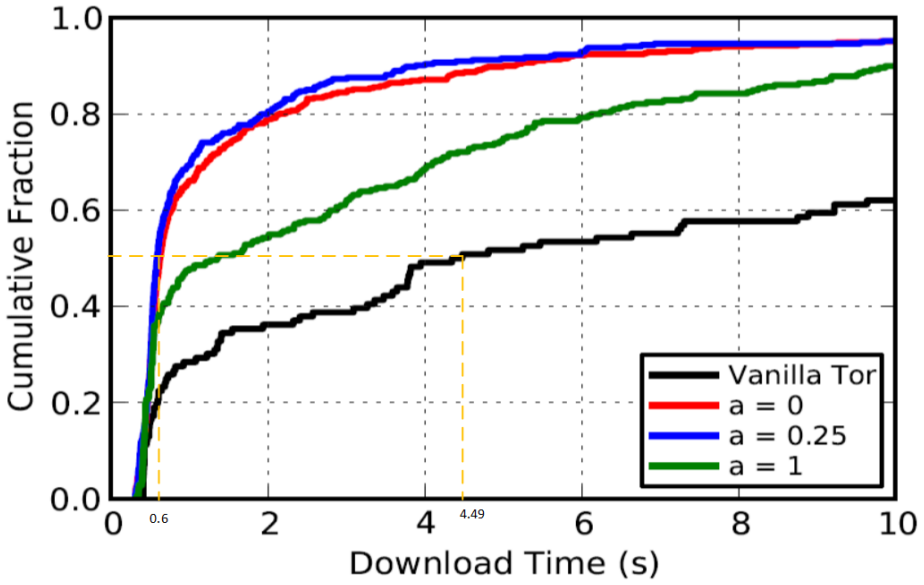

Figure 4.6. Bulk clients time to first byte (5MiB).

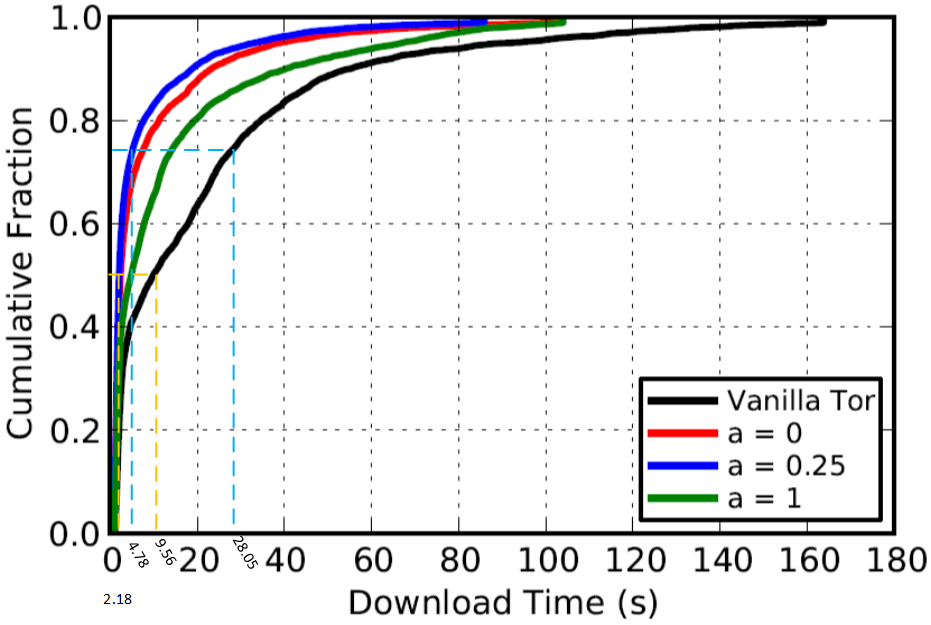

Figure 4.7. Web clients Time to last byte (320 KiB).

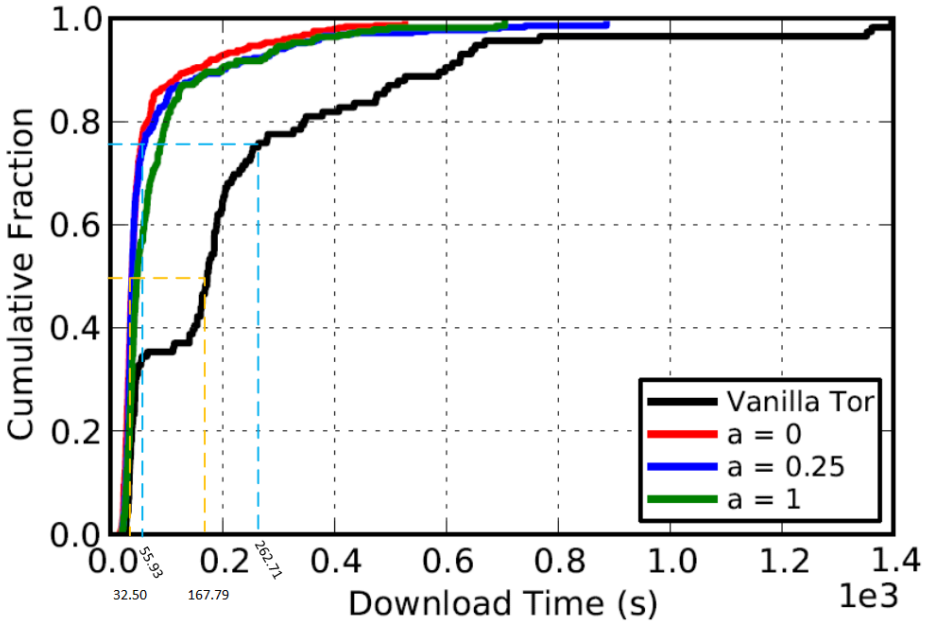

Figure 4.8. Bulk clients time to last byte (5 MiB).

CHAPTER 5

CONCLUSION

One of the reasons that reduces Tor performance is its path selection method. In this thesis, we proposed and evaluated a solution to improve Tor performance. Using a weighting algorithm, we calculate weight of each OR. In this algorithm, besides bandwidth, we include distance of ORs from each other, source, and destination. We create four pre-built circuits to the most popular destinations and one circuit with an exit node close to the client location. If the user's request contains the destination IP address, we use one of four circuits, else we use the circuit with an exit node close to the client location. Also, we calculate the RTT of the circuits to find less congested circuits. To attach a stream to a circuit, we choose the one with the minimum RTT among three candidate circuits with the lowest distance. Simulation of the modified Tor with different values of α in Shadow shows a significant improvement in performance. Comparing the results for time to first byte and download time of web, bulk, and Shadowperf clients demonstrates 0.25 is an appropriate value for α.

Using OR's location in node selection algorithm could compromise the anonymity of users to some extend. To overcome this anonymity reduction, we can use the Nikita and Borisove equation [21] and enable users to choose between anonymity and performance.

One improvement in our design could be keeping the history of the most visited websites by OP, and make OP create pre-built circuits to those destination. This solution improves the performance, but it makes some serious security issues that can risk the user's anonymity.

If we assume the length of a circuit as a distance from the Guard node to the Exit node through the Middle node, we can study the impact of shrinking and stretching of circuit lengths on the performance.

Considering the distribution of ORs in all around the world, we can investigate the effect of transatlantic and intercontinental circuits on the Tor performance. Although having a circuit in one continent may reduce the anonymity, the impact of this modification on performance might be significant.

APPENDIX A

SHADOWPERF CLIENTS

A.1 Introduction

In this section we show the results of Shadowperf clients. As mentioned before, Shadowperf clients using a new circuit with new Guard node to download files. They download 50 KiB, 1 MiB, 5 MiB files from predefined file servers and wait 60 seconds between their downloads.

Figures A.1, A.2, A.3 show the TTFB of 50 KiB, 1 MiB, 5 MiB respectively. As it's demonstrated 0.25 is an appropriate value of α in our function to improve Tor performance.

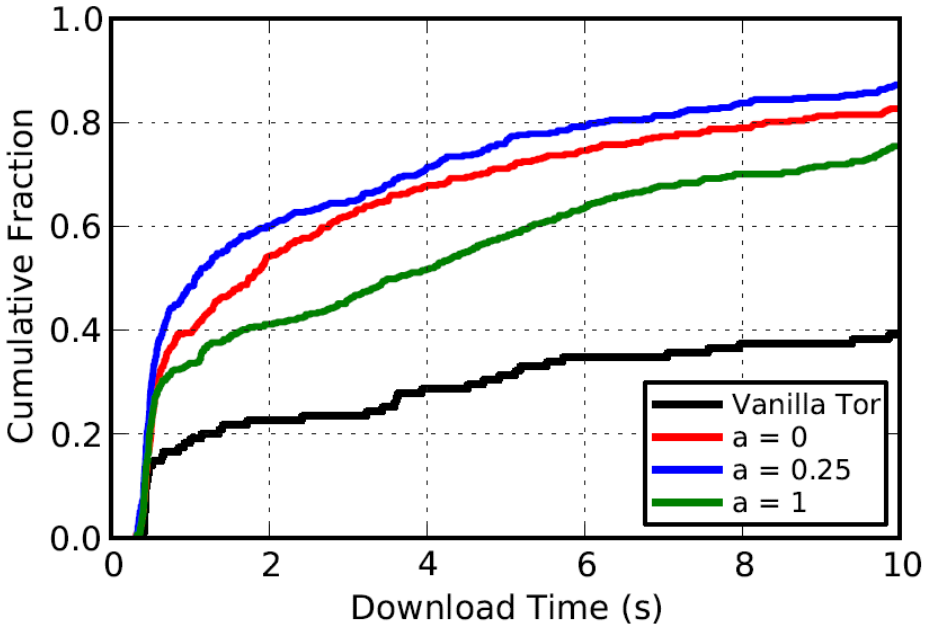

Figure A.1. Shadowperf Clients time to first byte (50 KiB).

The same as web and bulk clients, Shadowperf clients have a better time to last byte with $\alpha = 0.25$. Figures A.4, A.5, A.6 show the TTLB of 50 KiB, 1 MiB, 5 MiB respectively.

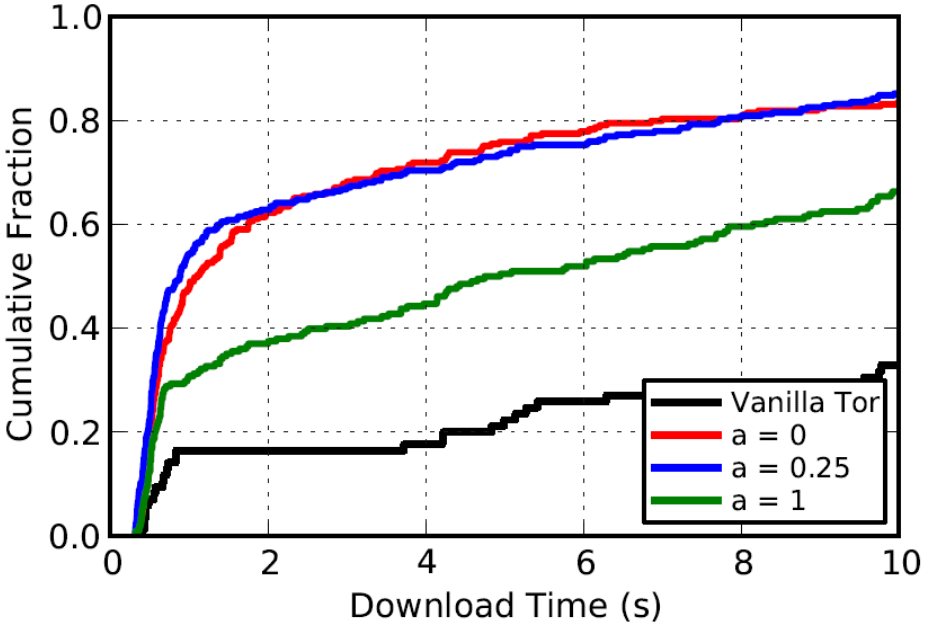

Figure A.2. Shadowperf clients time to first byte (1 MiB).

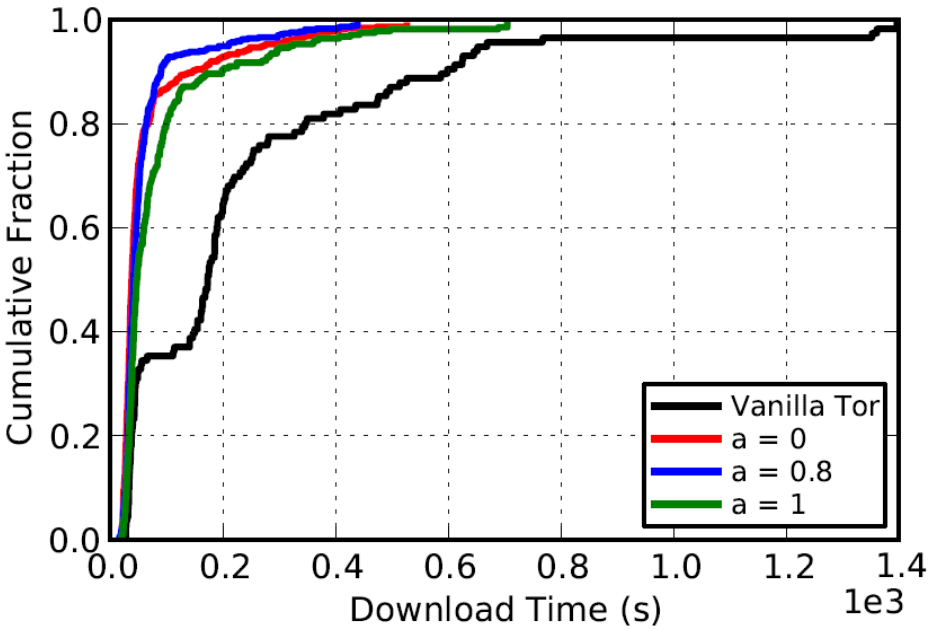

Figure A.3. Shadowperf clients time to first byte (5 MiB).

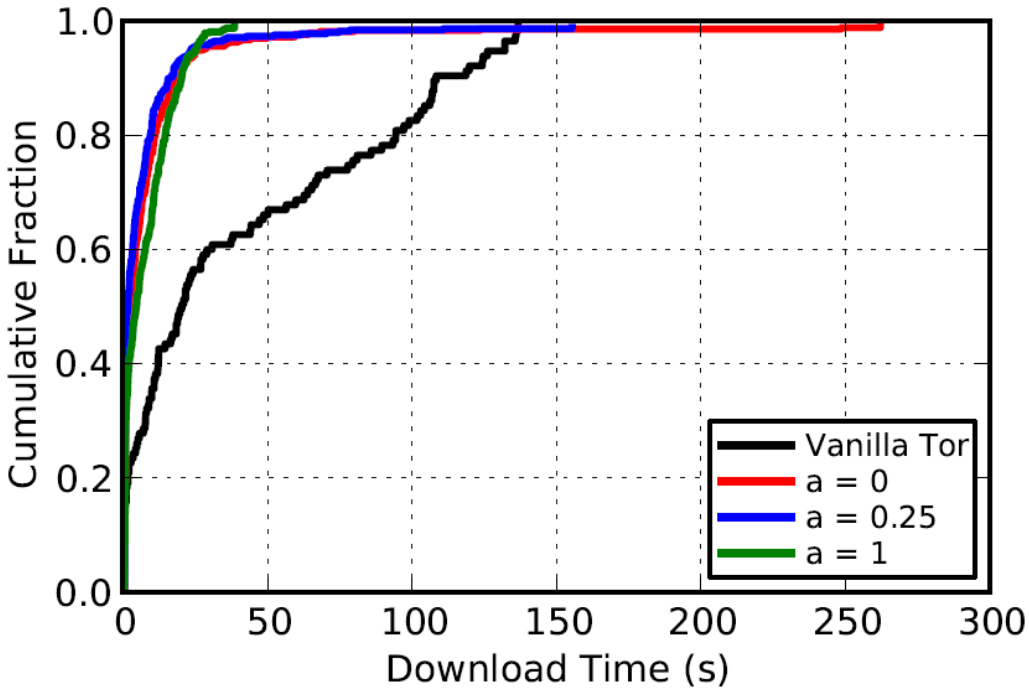

Figure A.4. Shadowperf clients time to last byte (50 KiB).

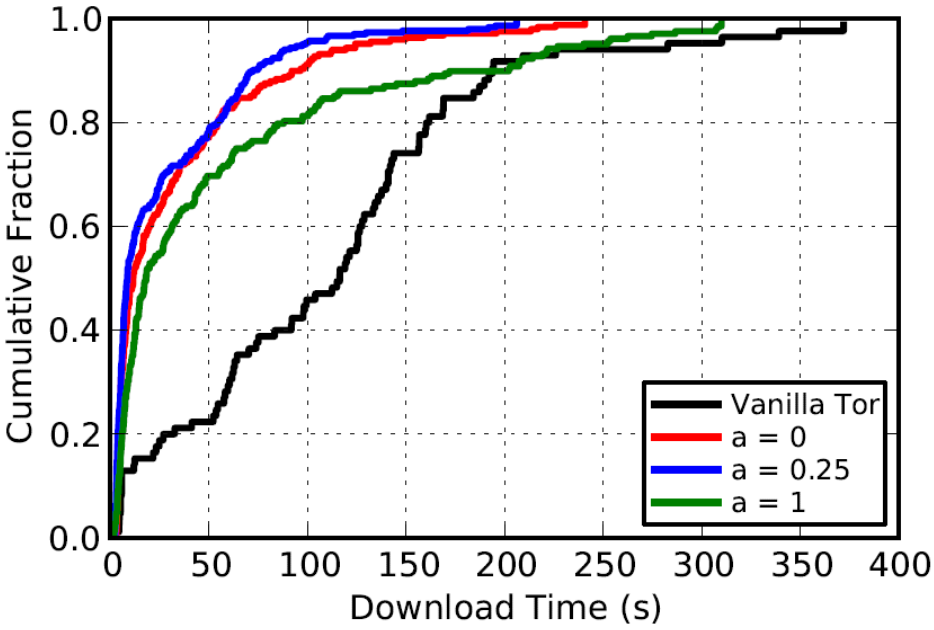

Figure A.5. Shadowperf clients time to last byte (1 MiB).

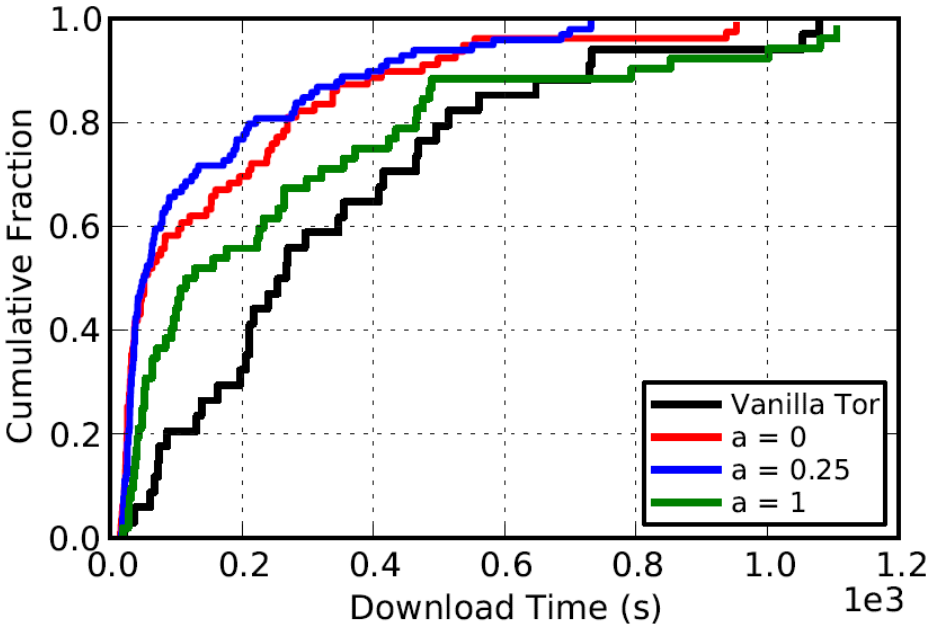

Figure A.6. Shadowperf clients time to last byte (5 MiB).

REFERENCES

[1] The right to privacy. [Online]. Available: http://en.wikipedia.org/wiki/Right_to_privacy

[2] Anonymizer. [Online]. Available: http://www.anonymizer.com./

[3] R. Dingledine and S. J. Murdoch, "Performance improvements on Tor or, why Tor is slow and what we're going to do about it," The Tor Project, Tech. Rep. 2009-11-001, November 2009. [Online]. Available: https://research.torproject.org/techreports/performance-2009-11-09.pdf

[4] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A low-latency AS-aware Tor client," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[5] Maxmind ip geolocation database. [Online]. Available: http://dev.maxmind.com/geoip/legacy/geolite/

[6] Botnet in Tor. [Online]. Available: https://blog.torproject.org/blog/how-to-handle-millions-new-tor-clients/

[7] "Tor users statistics," https://metrics.torproject.org/users.html, accessed: 2014-10-13.

[8] "Tor network statistics," https://metrics.torproject.org/network.html, accessed: 2014-10-13.

[9] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright, "Timing attacks in low-latency mix-based systems," in *Proceedings of Financial Cryptography (FC '04)*, A. Juels, Ed. Springer-Verlag, LNCS 3110, February 2004, pp. 251–265.

[10] A. Johnson, J. Feigenbaum, and P. Syverson, "Preventing active timing attacks in low-latency anonymous communication," in *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS 2010)*, July 2010.

[11] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

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

[13] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM Transactions on Information and System Security*, vol. 13, no. 2, February 2010.

[14] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *Proceedings of 23rd USENIX Security Symposium (USENIX Security '14)*. San Diego, CA: USENIX Association, August 2014.

[15] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*, October 2011.

[16] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security? How attacks on reliability can compromise anonymity," in *Proceedings of Computer and Communications Security (CCS 2007)*, October 2007.

[17] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the Tor network," in *Proceedings of the Network and Distributed Security Symposium-NDSS '14*. IEEE, February 2014.

[18] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. Keromytis, "CellFlood: Attacking Tor onion routers on the cheap," in *Proceedings of ESORICS 2013*, September 2013.

[19] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. Voelker, "Defenestrator: Throwing out windows in Tor," in *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS 2011)*, July 2011.

[20] M. AlSabah, K. Bauer, and I. Goldberg, "Enhancing Tor's performance using real-time traffic classification," in *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS 2012)*, October 2012.

[21] R. Snader and N. Borisov, "A tune-up for Tor: Improving security and performance in the Tor network," in *Proceedings of the Network and Distributed Security Symposium-NDSS '08*. Internet Society, February 2008.

[22] "Empty TLS application records being injected in tor streams," Email to or-dev@freehaven.net, in thread "Re: Empty TLS application records being injected in Tor streams" http://archives.seul.org/or/dev/Dec-2008/msg00005.html.

[23] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, August 2014.

[24] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-aware path selection for Tor," in *Proceedings of Financial Cryptography and Data Security (FC'12)*, February 2012.

[25] C. Wacek, H. Tan, K. S. Bauer, and M. Sherr, "An empirical evaluation of relay selection in Tor." in *Proceedings of the Network and Distributed System Security Symposium-NDSS '13*. Internet Society, February 2013.

[26] M. Sherr, M. Blaze, and B. T. Loo, "Scalable link-based relay selection for anonymous routing," in *Proceedings of Privacy Enhancing Technologies, 9th International Symposium (PETS 2009)*, August 2009.

[27] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald, "Experimentor: A testbed for safe and realistic tor experimentation," in *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2011)*, August 2011.

[28] R. Jansen and N. Hopper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation," in *Proceedings of the Network and Distributed System Security Symposium-NDSS '12*. Internet Society, February 2012.

[29] Planetlab. [Online]. Available: https://www.planet-lab.org/

[30] Harvesine formula. [Online]. Available: http://en.wikipedia.org/wiki/Haversine_formula

[31] Alexa. [Online]. Available: http://www.alexa.com

[32] Shadow. [Online]. Available: http://shadow.github.io/

[33] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine, "Methodically modeling the Tor network," in *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2012)*, August 2012.

[34] Torperf. [Online]. Available: https://gitweb.torproject.org/torperf.git/blob_plain/HEAD:/measurements-HOWTO

[35] Time to first byte. [Online]. Available: http://en.wikipedia.org/wiki/Time_To_First_Byte

[36] Diagnosing slow web servers with time to first byte. [Online]. Available: http://www.websiteoptimization.com/speed/tweak/time-to-first-byte

BIOGRAPHICAL STATEMENT

Mehrdad Amirabadi was born in Tehran, Iran, in 1983. He received his Bachelor of Science in Computer Hardware Engineering from the Azad University, Tehran, Iran, in 2007. He worked as a hardware engineer in Bamdad Computer, Tehran, Iran from 2007 to 2008. He also worked as a Database operator and administrator in Sadad Informatics Corporation (SIC), Tehran, Iran from 2008 to 2011. Mehrdad has been a member of iSec, the Information Security Lab at UT Arlington, since 2012. He has been a graduate research assistant and responsible for systems administration of iSec lab. His current research interests include anonymity systems, network security, and operating systems.