Improving Performance in the Tor Network

Laverne Schrock
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
schr1230@morris.umn.edu

ABSTRACT

The Tor network is designed to allow people to use the internet anonymously, but at the cost of slower network access. In this paper we explore two methods of selecting better performing routes through the network, one based on measuring how busy relays are, and the other based on the amount of time it takes for data to travel from one end of the route to the other. Experimental results indicate that these techniques improve several performance metrics in the network, without endangering Tor's anonymity goals.

Keywords

Tor, Performance, Anonymity

1. INTRODUCTION

When people want to visit a website, they launch a web browser and type in the address of the desired site, for example, `google.com`. This results in a request being sent to a server on the internet, through devices operated by various third parties, and then the server sends data back to the user's computer. As this data is passed along by various machines, they are all able to read it. To prevent abuse, many modern protocols use encryption so that only the intended recipient can learn anything meaningful from the data.

While encryption prevents intermediaries from learning *what* you are sending and receiving on the internet, it does not prevent them from learning *with whom* you are communicating, and there are many cases in which individuals wish to avoid disclosing what sites they are visiting. In some nations, visiting web sites discussing political opinions at odds with those in power can attract the unwanted attention of the authorities. Even in the U.S., journalists wish to avoid revealing the identity of informants and whistle blowers. The browsing trends of law enforcement officers could reveal the subject of a secret investigation. The average user of the Internet may also wish to obscure their activities, since Internet Service Providers (ISPs) are legally permitted to sell data revealing what sites users visit [3].

The Tor network can be used to hide the true source and destination of traffic on the Internet. When using Tor, clients route their traffic through a chain of Tor relays. Each hop in the route is encrypted in such a way that each relay on the route can only determine what relays are immediately before and after it, and no relay can determine both the original source and final destination of the traffic.

Tor traffic is easily identified as such, so in order for Tor to be useful, it must be used by diverse groups of people. If Tor were only used by government agents, then using Tor would make it easier for someone listening in on the connection to know that the user is a government agent. If Tor were only used by dissidents, then law enforcement under authoritarian regimes would have an easy way of identifying such individuals. It is only when used by many people, with many different uses, coming from many different locations, and connecting to many different locations, that Tor provides real anonymity to its users.

Millions of clients connect to the Tor network every day [7]. Since traffic is being bounced across the globe instead of being routed as directly as possible, using Tor is slower than using the Internet directly. Increasing the speeds that Tor can offer to users could increase both the amount that people use it and the number of people who use it, potentially improving diversity in traffic.

In this paper, we present two modifications to the Tor network intended to improve performance for users. In Section 2, we introduce terminology and the relevant parts of Tor's implementation. In Sections 3 and 4 we present the modifications and the results of experiments indicating that these modifications would indeed improve performance in the Tor network. In Section 5, we discuss the impact that these changes have on Tor's anonymity goals.

2. BACKGROUND

The Tor network consists of three main components: the relays, the directory servers, and the clients. Tor relays are run by volunteers around the world, mostly concentrated in Europe and North America. As of October 2017, there are over 6000 relays in the network [7]. The relays report their existence, bandwidth capabilities, and other information to the central directory servers. There are 9 of these servers scattered across Europe and North America. Clients then consult these servers to learn about the about available relays and their properties. To reduce the damage that a compromised directory server would cause, clients will not act on information received from a directory server unless a majority of the servers agree with it [6].

In order for a user to contact a server over the Tor network, the client must go through several steps, the first of which is *path selection*. To select a path, clients randomly choose three relays from the list of all available relays that they

UMM CSci Senior Seminar Conference, November 2017 Morris, MN.

have learned about from the directory servers, weighted by bandwidth. Some additional restrictions will cause some paths to be discarded. For example, paths are not permitted to have the same relay twice.

The next phase is *circuit construction*. In this phase, the client initializes a circuit through the relays in the path. To do so, the client negotiates an encrypted connection to the first relay. Through that connection, the client negotiates a connection with the second, and the connection to the third relay is the negotiated through the second relay. The newly constructed circuit is then added to a pool of circuits, if it passes a quality test. In Section 4, we explain the current behavior of the test and proposed modifications to it.

Finally, in *circuit selection*, the client selects a circuit from the pool to service the request. Note that circuits can handle multiple requests simultaneously and thus the *circuit selection* phase does not remove the circuit from the pool. In Section 3 we describe proposed modifications to this stage which prioritize the selection of faster circuits.

Because constructing circuits can take a fair amount of time, the Tor client builds the pool when launched. Thus, only the *circuit selection* phase must be done when issuing a new request. Whether they've been used or not, circuits are cycled out of the pool after a while and replaced by new ones, triggering the *path selection* and *circuit construction* phase for each new one.

3. THE AVOIDING BOTTLENECK RELAY ALGORITHM

A circuit can only operate at the speed of the slowest relay in the circuit. If a relay is the slowest point in some circuit, we refer to the relay as a *bottleneck relay* or simply a *bottleneck*. A relay that is a bottleneck can only provide bandwidth to a new circuit by reducing the bandwidth allocated to its current circuits, so it would be better if clients avoided using such relays. The more thinly spread that a bottleneck's bandwidth currently is, the less it will have to offer new circuits.

The Avoiding Bottleneck Relay Algorithm (ABRA) is a set of modifications to Tor which allow clients to avoid circuits containing serious bottlenecks during the circuit selection phase. In this section, we describe the implementation of ABRA in Tor relays and clients, and present experimental results demonstrating the ability of the implementation to improve performance in the Tor network.

3.1 Creating a weight

If a relay is not currently using all its bandwidth, then it is not restricting any of it circuits. Clients should prefer this relay over relays that are a bottleneck since this relay will be able to provide at least a small amount of bandwidth that isn't coming from the bandwidth currently allocated to other circuits, thus improving the total throughput of the network.

Just because two relays have currently allocated all their bandwidth, does not mean that these relays would serve a new client equally poorly. For example, if relay A is maxed out with a single circuit running at 30 Mb/s, and relay B is maxed out with two circuits running at 20 Mb/s each, then relay A is a better candidate for placement in a new circuit since it would cap the bandwidth at 15 Mb/s, while relay B would cap it at $13.\overline{3}$ Mb/s. This is because of Tor's flow control algorithm, which tries to share bandwidth equally between its circuits [4].

If a relay is using all of its bandwidth, then it is likely slowing at least one circuit, although it is not necessarily a bottleneck on all of its circuits. If one circuit is using less bandwidth than some other circuit, then the slower one must not be limited at this relay, since the relay would happily increase the speed of the slower circuit by reducing the speed of the faster one. So when a relay's bandwidth capacity is exhausted, we expect that all the circuits restricted at this relay will be using roughly the same amount of bandwidth, and that there will be some number of circuits using various, lower amounts of bandwidth.

The goal of ABRA is to use this knowledge about the way bandwidth is divided between circuits to produce a metric indicating how much a relay will delay a circuit. In order to do this, the relay needs to measure the bandwidth of each circuit and perform statistical grouping of these bandwidths to determine on which of the circuits this relay is a bottleneck.

In order to determine how much bandwidth a circuit is using, the relay records how much data was sent over the circuit in the last g milliseconds. Every w seconds, the relay divides the maximum value recorded in this interval by g, and sets this value to be the bandwidth of the circuit.

Geddes et al. [4] performed a set of experiments with different values of g and w combined with different statistical grouping methods and found that $g = 100\ ms$ and $w = 1\ sec$ paired with the following grouping method, produced results closest to those that would be produced by an algorithm having full knowledge of the network.

Algorithm 1 Head/Tail clustering algorithm

```
1: function HeadTail(input,threshold)
2:     mean ← sum(input)/len(input)
3:     head ← {d ∈ input | d ≥ mean}
4:     if len(head)/len(input) < threshold then
5:         head ← HeadTail(head, threshold)
6:     return head
```

The pseudo-code for the Head/Tail algorithm is given in Algorithm 1 (from [5] as cited in [4]). The algorithm takes in a set of numbers and a threshold. The mean of the input is computed and all members of the input greater than or equal to the mean are placed in the *head*. If the percent of the input that is placed in *head* is less than the threshold, the algorithm makes a recursive call with *head* as the new input. Otherwise, *head* is returned.

The relay employs $HeadTail$ by passing it the set of all the bandwidths of circuits currently running through the relay. The relay then sets $weight = \sum \frac{1}{h_i}$, where h_i is the i^{th} value in the *head* returned by the Head/Tail clustering algorithm. It is important to note that *weight* will be frequently changing as circuits running through the relay come and go and as the amount of traffic passing through the circuits changes. Every five seconds, the relay broadcasts a message down each of its circuits specifying the current computed weight. [4]

Going back to our previous example with relays A and B, if we assume that the Head/Tail algorithm correctly identifies the limited circuits, then A has a weight of $1/30 = 0.0\overline{3}$, and B has a weight of $1/20 + 1/20 = 0.1$.

3.2 Using weight to inform circuit selection

As the client builds circuits and adds them to the pool of available circuits, it begins receiving a weight from each of the relays in each of the circuits. For each circuit, the client sums these weights to compute a total weight for the circuit. [4]

When the client performs a *circuit selection*, it first checks to see if any of its circuits have a weight of 0, which would indicate that this circuit contains no bottlenecks. If so, the client randomly selects from this subset, biased towards circuits with higher bandwidth. If no circuits have a weight of 0, the client randomly chooses from the pool, biased towards circuits with a lower weight. [4]

3.3 Empirical Evaluation

Because these changes modify the behavior of the relay and client components of Tor, they cannot be tested by simply deploying a few machines on the Tor network. To collect empirical evidence on the performance properties of these changes, Geddes et al. employed Shadow[1], a powerful network simulation tool designed for the testing of complex distributed systems such as Tor or Bitcoin. In order to mimic the various usage patterns found in the real world, they configured Shadow to have 500 relays, 1350 web clients, 150 bulk clients, and 500 web servers. The web clients download 320 KB files and randomly pause between 1 millisecond and 1 minute before starting the next download. Bulk clients download 5 MB files and take no breaks.

Four different metrics were measured in this experiment: total network utilization, time to first byte (TTFB), web download times (from the web clients), and bulk download times (from the bulk clients). The total network utilization was computed by having each relay record how many bytes it relayed every 10 seconds. These measurements are then summed across all 500 relays to get the total network utilization for that 10 second period. Clients determine TTFB by measuring how much time passes between initiating a download and receiving the first byte of that download.

The total network utilization was found to be on average 14% higher when using ABRA than when using unmodified Tor. All clients had improved download times with some web clients seeing improvements of almost 200%, and bulk clients consistently experiencing an increase of 5-10%. The only metric by which ABRA performed more poorly was TTFB where 12-13% of the downloads had an increased TTFB, while the rest of the downloads had the same TTFB as standard Tor [4].

4. CIRCUIT-RTT

As we saw in the previous section, ABRA provides a mechanism for clients to select well-performing circuits from their pool of currently built circuits. In contrast, Circuit-RTT replaces one of the checks that Tor performs on circuits *before* they are added to the pool. Implementing Circuit-RTT is also more straightforward than ABRA since it requires no modification of the relay software.

Circuit-RTT uses round-trip time (RTT) as its core metric for the selection of circuits. Currently, the Tor implementation measures the time to build a circuit and if it takes too long, those circuits are rejected. Circuit-RTT works on the assumption that testing the RTT of a circuit right after its construction is a good indicator of how well the circuit will serve the user. The authors of [1] performed experiments revealing the distribution of RTTs in the Tor network. They then implemented and tested a circuit selection mechanism based on the results of the statistics collected. [1]

4.1 Circuit Build Time

The current version of Tor records how long the construction of a circuit takes. Using the last 1000 build times, the Tor client estimates what percent of all possible circuits would take less time to build than this one. If the percentage is over a certain limit, then the circuit will not be added to the pool, but its build time will replace the oldest entry in the list of build times. Throughout the rest of the paper, we'll refer to this *algorithm* as CBT.

4.2 Building a Model of RTT distribution

Circuit-RTT performs the exact same role as CBT, but rather than using the build time of the circuit, it measures the RTT of the circuit and uses that metric as the criterion by which the circuit will be kept or discarded.

After a circuit is built, the client can communicate directly with any of the relays in the circuit. If a client sends a bad request, the relay will send back an error. Circuit-RTT exploits this feature by sending a request that is specifically designed to fail at the last relay in the circuit. The client records how much time passes between the sending of the bad request and the receiving of the error response and stores this value as the RTT of the circuit. [1]

With the ability to measure the RTT for a circuit, clients could simply discard circuits that have an RTT over a certain threshold. However, clients connect over Internet connections of various speeds and connect to the Internet at various locations relative to the Tor relays. To handle this variance, Circuit-RTT must, just like CBT, be given knowledge of what distribution the RTTs will fall into if the client were to take many circuits and test their RTT.

19 computers across Europe were used to study the distribution of RTTs in the Tor network. Each of these computers built 1 million circuits and measured the RTT once on every circuit. It was found that the distribution of the RTTs can be closely approximated by a Generalized Extreme Value (GEV) distribution, although the parameters of the distribution varied from client to client.[1]

4.3 Testing the selection method

Implementing Circuit-RTT requires modifying clients to record and store the RTT for circuits as described in the previous section. When circuits are discarded for any reason (e.g. age), the client does not discard the RTT measurement, but instead keeps the RTT for the 1000 most recent circuits. Working under the assumption that these RTTs can be fit to a GEV distribution, the client finds the parameters that cause the curve to best fit the collected points.

When the client builds a new circuit, it can now check what percent of all circuits will be faster or slower than this new circuit, based on the parameters for the GEV distribution that the client has computed.

The developers of Circuit-RTT deployed modified clients on 19 on computers across Europe, each constructing 1 million circuits. These clients did not build circuits and add them to a pool. Instead, for each circuit built, the clients measured the build time, the RTT, the latency to `google.com`

[1] `https://shadow.github.io/`

Table 1: Latency under CBT and Circuit-RTT

	CBT (ms)	Circuit-RTT (ms)	Percentage Change
Median	309	300	-2.9%
90^{th} percentile	541	499	-7.8%

Table 2: Bandwidth under CBT and Circuit-RTT

	CBT (Mb/s)	Circuit-RTT (Mb/s)	Percentage Change
Median	3.24	3.30	+1.9%
90^{th} percentile	1.10	1.14	+3.6%

and then discarded the circuit. They then analyzed the collected data to determine which ones would have been allowed into the pool when using CBT and which would have been allowed in when using Circuit-RTT.

As summarized in Table 1, when filtering the collected data to only contain the top 80% of circuits, using the build time of the circuit to judge its quality leaves us with circuits having a median TTFB of 309 ms and a 90^{th} percentile of 541 ms. Using the RTT of the circuit to judge its quality would result in use of circuits with a median TTFB of 300 ms and a 90^{th} percentile of 499 ms. These are improvements of 2.9% and 7.8% respectively, a statistically significant difference. [1]

When filtering the data points to only contain the top 50% of circuits, the difference between CBT filtering and Circuit-RTT filtering becomes even larger, but filtering the circuits so drastically would damage the anonymity guarantees that Tor aims to provide. They also explored using CBT filtering and Circuit-RTT filtering together, but found that this did not perform as well as Circuit-RTT alone. [1]

To determine the impact on bandwidth, the developers used a very similar experiment to the previous one. The only differences are that the experiment only used 7 computers, each one making 100,000 measurements, and bandwidth was measured instead of latency. To measure the bandwidth of a circuit, the clients downloaded a single 5 MB file which was mirrored around the world on a commercial content delivery network. The developers then considered bandwidth measurements that would be left when only keeping circuits with CBT or RTT measurements in the best 80%. As summarized in Table 2, clients achieved higher bandwidth with Circuit-RTT than with CBT, but the improvement was not as large as the improvement to latency. [1]

5. IMPACT ON ANONYMITY

In the previous two sections, we presented two different strategies for improving the performance of the Tor network. While both were shown to successfully improve various metrics of performance, we have not explored how these modifications to Tor impact the anonymity guarantees provided by the network.

5.1 Timing Attack

If an attacker outside the Tor network is able to observe all traffic entering and leaving the Tor network, then it will be able to carry out what is referred to as a *timing attack*.

Suppose the attacker wishes to know whether a particular client is talking to a particular server. This attacker can watch the traffic flowing through the network and come to the conclusion that whenever the client sends a piece of into the Tor network, we see a chunk of data with the same size come out of the Tor network a short time later and go to a particular server. Thus the attacker can determine that the client and server are indeed talking to each other despite the fact that their connection is obfuscated through the Tor network. This attack works because Tor introduces no artificial delays into its traffic, and Tor explicitly does not protect against such an attack. [2]

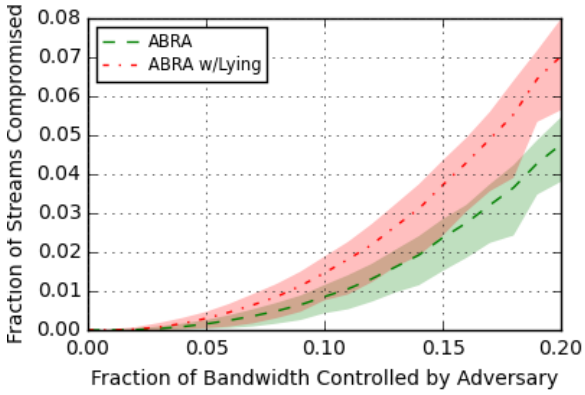

Figure 1: Circuits compromised under ABRA

Another type of attacker is one that controls some fraction of all the relays in the network. We refer to these as *malicious* relays, and we will assume that they can cooperate in their attempts to de-anonymize traffic.

If a circuit is composed of entirely of malicious relays, the attacker can trivially detect that it controls the whole circuit and has thus de-anonymized it. If only the first and last relay in the circuit are both controlled by the attacker, the attacker will not have any immediate reason to associate the traffic passing through the two relays. However, the attacker can perform the same time and size measurements that a global attacker would use to de-anonymize traffic. Thus, we call a circuit *compromised* when the attacker controls both the start and the end of the circuit [4]. This is an attack that Tor can defend against, or rather, we wish to avoid adding means by which an attacker could compel clients to choose malicious relays more than non-malicious relays.

5.2 Malicious relays with ABRA

When modified to use ABRA, clients assume that relays correctly compute their own weight and broadcast it honestly. The relays' declarations of weight are signed to prevent relays from lying about each other, but malicious relays

could simply claim a weight of 0 in order to increase their chance of being used by clients.

During the experiments described in Section 3.3, Geddes et al. [4] recorded what circuits were in the client's pool when every circuit selection choice was made. After the experiment was complete, they recorded how often clients selected a compromised circuit. They then re-ran each selection choice with the weights of all malicious relays set to 0, and recorded how often clients would have selected the compromised circuits.

The more relays that an attacker has, the more likely we are to use compromised circuits. To determine how the attacker's power grows with relation to the fraction of the network that it controls, these choices were simulated with various numbers of relays being granted to the attacker.

The results of these calculations are visually presented in Figure 1. The upper and lower bounds of the colored regions mark the the 90^{th} and 10^{th} percentile respectively, and the dashed lines represent the median. When the malicious relays were assumed to always report 0, circuits compromised by the attacker are selected more frequently. For example, when the attacker controls 20% of the network, the median percent of circuits compromised is around 5% with ABRA, and increases to around 7% when the malicious relays lie. We also see that as the attacker controls an increasing fraction of the network, its ability to compromise circuits grows more rapidly when its relays lie.

If ABRA were to be implemented in the live Tor network, the lying relays would draw traffic away from honest relays, thus driving down the weight of the honest relays. This would in turn increase the chance of clients selecting circuits containing honest relays, indirectly drawing traffic away from the malicious relays. We would thus impact of lying relays to be less than that measured in the above experiment, since the experiment only counted what how many times clients *would have* chosen a compromised circuit, assuming that the malicious relays were being dishonest for the first time.

5.3 Concerns with Circuit-RTT

Implementing Circuit-RTT only requires modifications to the Tor client, and none to the relays. However, when performing the measurement of the RTT of a circuit, recall that the client sends a request that it knows the end relay will reject, in order to measure how long it takes for the error message to get back.

If the end relay is malicious, it has no means of sending the error message back *faster*. All that the malicious exit relay can do is delay the sending of the error message, thus discouraging the client from using that circuit [1]. This is not beneficial to the attacker since its goal is to have clients create as many compromised circuits as possible.

6. CONCLUSION

In this paper, we described the basics of the Tor network and two recent efforts to improve its performance. Both ABRA and Circuit-RTT provide performance improvements in the Tor network. Circuit-RTT impacted TTFB more than it did throughput, while ABRA improved throughput more than it did TTFB. This is not surprising since TTFB and RTT are essentially the same metric and this is what Circuit-RTT uses to accept or reject circuits. ABRA favors relays that haven't used all their capacity, so it is natural that download speeds and total network throughput were improved. ABRA does not explicitly factor RTT into its metric so it isn't surprising that TTFB was not favorably impacted.

Because Circuit-RTT and ABRA modify different parts of the life-cycle of Tor circuits, it would be possible to implement both of them simultaneously and potentially compound their improvements. As the Tor network continues to gain additional users, maintaining and improving its performance will be paramount to its continued success.

Acknowledgments

7. REFERENCES

[1] R. Annessi and M. Schmiedecker. Navigator: Finding faster paths to anonymity. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 214–226, March 2016.

[2] R. Dingledine, N. Mathewson, and P. Syverson. Challenges in deploying low-latency anonymity (draft). `https://svn.torproject.org/svn/projects/design-paper/challenges.pdf`, 2005.

[3] B. Fung. Trump has signed repeal of the FCC privacy rules. Here's what happens next. `https://www.washingtonpost.com/news/the-switch/wp/2017/04/04/trump-has-signed-repeal-of-the-fcc-privacy-rules-heres-what-happens-next`, April 2017.

[4] J. Geddes, M. Schliep, and N. Hopper. ABRA CADABRA: Magically Increasing Network Utilization in Tor by Avoiding Bottlenecks. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, pages 165–176, New York, NY, USA, 2016. ACM.

[5] B. Jiang. Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer*, 65(3):482–494, 2013.

[6] N. Mathewson, R. Dingledine, K. Loesing, D. Johnson, S. Hahn, G. Kadianakis, P. Palfrader, L. Nordberg, M. Perry, I. Markin, T. Wilson-Brown, D. Goulet, M. Traudt, I. Lovecruft, G. Goodell, R. Ransom, and J. Bobbio. Tor directory protocol, version 3. `https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt`.

[7] Tor metrics. `https://metrics.torproject.org/`.