FasTOR: A Low Latency Tor Client

github.com/malik-sahab/FasTOR

Muhammad Abdullah Malik
Bachelor of Science, Computer Science
Lahore University of Management Sciences
Lahore, Pakistan 54000
Email: 18100006@lums.edu.pk

Zain Imran
Bachelor of Science, Computer Science
Lahore University of Management Sciences
Lahore, Pakistan 54000
Email: 18100260@lums.edu.pk

ABSTRACT

Today, more and more people are concerned about privacy on the Internet and are turning towards Tor [9] and other tools to achieve anonymity and privacy. However, mainstream adoption of Tor has been slow, citing a number of reasons. One of the major reason is the high delays on the Tor network, which can be as much as 5x the latency on a normal network [1].

Therefore, the goal of this study is to find a way to provide a low-latency Tor client. In this paper, we discuss our approach towards building a low latency Tor client, aptly named FasTOR. Our approach builds up on the work previously done in this field such as the LASTor project [1] The difference in our approach is that we are not only considering geographical distance when choosing relays for path selection but also the advertised bandwidth of the relays. Similar to LASTor, we are aggregating relays into clusters according to their inferred geographical location. Our modification are only at the client-side and do not require any revamping of the Tor network. By applying our algorithm and performing test runs on few of the top sites, we observe latency on our custom Tor circuit to be about 1.5x the latency of the same website being queried on the Internet paths.

Categories and Subject DescripTors

D.3.3 [**Tor, STEM, Networks, Anonymity**]

General Terms

Algorithms, Measurement, Performance, Design.

Keywords

Onion Routing, STEM- Tor Controller, Low-latency, End-to-End

1. INTRODUCTION

Tor is a second-generation onion routing technology that is known for protecting user privacy and anonymity on the world wide web, although recently it is also being used for circumvention of internet censorship in places where freedom on internet is restricted.

However, Tor has not seen widespread adoption by every day internet consumers. There are numerous reasons for that, from lack of knowledge of the tool to a rather complicated Tor set up. However, one of the major reasons is the high latency for interactive traffic on Tor. A lot of potential users are put off by the excruciatingly large loading times for websites when browsing through the commonly distributed Tor browser.

In this paper, we aim to highlight an important improvement that we want to make to the Tor client. Our approach is to optimize the path selection algorithm that the client uses during the building of the circuit. Our optimizations will choose the shortest Tor circuit to tunnel the traffic through. Current path selection algorithm goes for highest entropy by choosing random relays. A consequence of that is that the traffic may be routed all over the world before it reaches its destination, hence the high latencies. One reason for such a design is that it preserves client anonymity, which is an important motivation behind using Tor.

Therefore, we seek to optimize path selection without decreasing the entropy of path selection by a large factor. All our suggested changes and improvements are at client-side unlike previous approaches to reduce latency [7][8]. A simple update for the Tor client would enable a user to benefit from the improved performance. Therefore, we seek to answer the question: what latency improvement can a Tor client obtain without mandating modifications to the Tor network. Towards this end, we design and implement FasTOR, which differs from the default Tor client only in its path selection approach.

In implementing FasTOR, we observe how opting for inferred geographical distances of relays rather than latencies on internet lead to substantial reduction in path latencies. We implement weighted shortest path algorithm for choosing paths, with weight of the edge inverse to the distance between the relays. However, an attacker could set up a large number of relays near a location close to the chosen path in order to increase probability of him controlling one of the chosen path relays. To counter this, we are clustering relays in similar geographical locations and then running WSP on the connected graph of such clusters.

We demonstrate FasTOR's benefit in improving latency times by using it to visit the top ten websites and recording the time it takes to fetch the HTML page in each case around 10 times and then averaging the response time. We see that using FasTOR as the Tor client, it should take around 1.5x the time for the same request to travel on a path on the Internet.

2. OVERVIEW

2.1 Problem Statement

Our goal in this paper is to address the shortcomings in Tor discussed above with respect to latency and anonymity without requiring a revamp of Tor's design. Leveraging the fact that intelligence in Tor resides at the client, we seek to only modify the client-side path selection algorithm so that clients can benefit

today without waiting on updates to relays to be developed and deployed. In doing so, we respect conventional wisdom on how to preserve client anonymity in Tor, e.g., the use of an entry guard to protect against statistical profiling attacks and the need for sufficient randomness in relay selection to protect against colluding relays.

2.2 Initial Experimentation and Results

With the problem statement in focus, our first decision boils down to usage of which Tor API to use in order to select paths and make custom circuits. We used the STEM library [2] which turned out to be a flexible python library when it comes to executing the tasks mentioned above.

In order to make sure that all our results were coherent and fair our testing scheme remained same for all the tests that we ran (with or without Tor). As a result our first experiment (**test1**) was to measure latency and times over simple internet whilst fetching HTML web-pages. We used top ten websites [4]:

i. 'https://www.google.com.pk/',
ii. 'https://www.youtube.com/',
iii. 'https://www.apple.com/',
iv. 'https://www.yahoo.com/',
v. 'https://www.linkedin.com/',
vi. 'https://www.wikipedia.org/',
vii. 'https://www.amazon.com/',
viii. 'https://twitter.com',
ix. 'https://www.pinterest.com',
x. 'https://www.quora.com'

Each web-page was fetched 50 times in total. Runs were divided into loops of 10 in order to distribute tasks efficiently. Figure 1 shows the results of our test with latencies (seconds) on y-axis and websites queried on x-axis

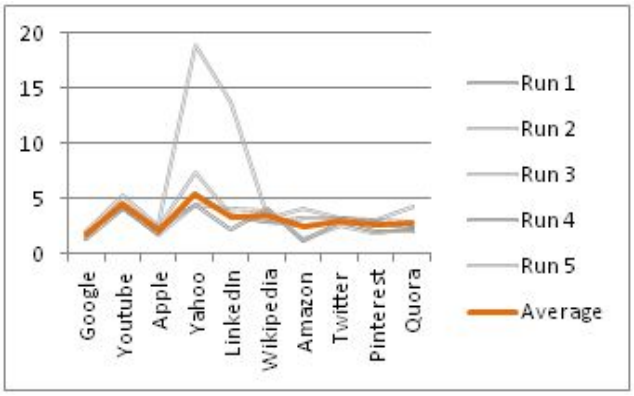

Figure 1: Fetching web-pages over the internet

In comparison we did our second test (**test1b**) following the same schema but over a Tor circuit that is pre-built and present in the list of available circuits in a control connection. Figure 2 shows these results.

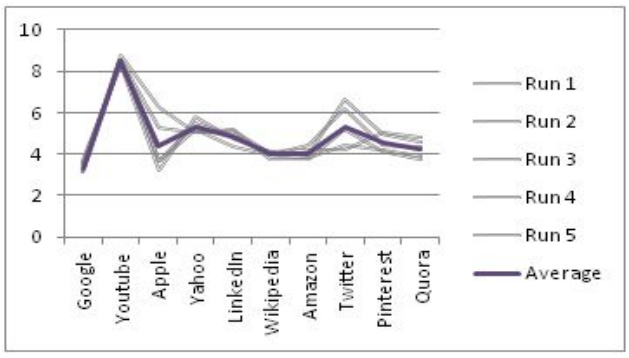

Figure 2: Fetching web-pages on Tor - Pre-built available circuits

With these initial results at hand we have a clearer view of the current state of Tor network and its latencies. Figure 3 makes a good comparison of both the experiments and shows that the Tor network has become faster compared to the fact that it used to be 5x slower a few years ago.

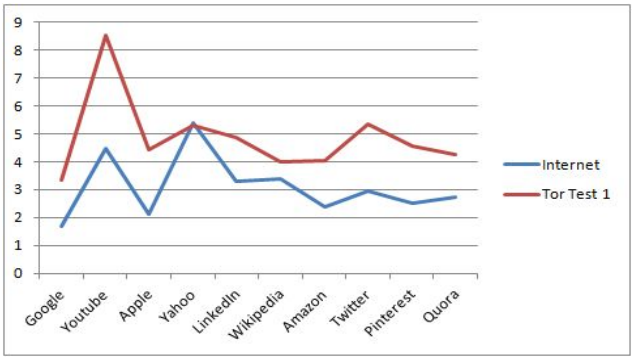

Figure 3: Comparison of Internet and Tor for fetching webpages.

We ran a third test (**test2**) which involved creating custom circuits for the first time. After populating a list of relays from the micro-descriptors consensus. This involved selecting three relays for the path.

- **Guard Relay**: This was fixed. Owing to Tor policies
- **Middle Relay**: Randomly selected from the list of relays
- **Exit Relay**: Randomly selected from the list of relays but making sure that it has the 'Exit' flag.

The results (Figure 4) show that there is no major difference between test1b and test2 showing that Tor is still selecting its relays randomly and no path selection algorithm is being used.

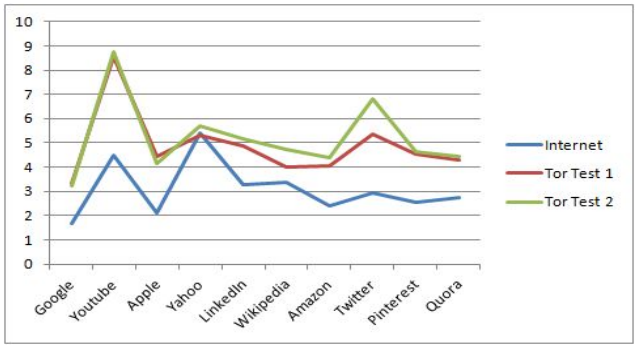

Figure 4: Comparison of Internet, Tor and Random Selection Circuit for fetching webpages.

2.3 Our Approach

Upon valuable discussion and meetings, our decision was to focus upon path selection and pre-build previous research based upon this i.e. LASTor [1]. With the vision and limitations of this paper in view we knew that there was a need to select paths based on something more than just geographical distances because intuitively, bandwidth might play a major role. The tests to follow cleared a lot of such confusions and yielded to a research that was more analytical based.

3. IMPLEMENTATION

3.1 Scripting Phase

For each control connection, we got the micro-descriptors consensus which gave us the information of present relays. Each descriptor is an object representing a relay and its valuable information that include advertised bandwidth, IP address, fingerprint etc. The descriptors were populated in a graph of 35 nodes distributed on the world map (Figure 5) [5] [6]. Each node in itself was a dictionary that had the following attributes:

- **Name**: 'NA1' short for North America 1
- **Latitude, Longitude:** This is the geographical centre of that node area
- **Relays List:** List of all relays that lie within that node area
- **Bandwidth:** Average bandwidth of relays in that node

All locations were measured using geoip library in Python [3]. In addition, all nodes on the graph were connected to each other via edges with actual geographical distances as their weights.

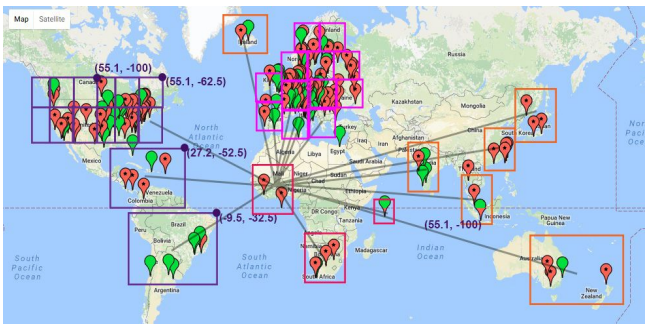

Figure 5: A view of our implementation of nodes. Showing all edges from one node from Africa.

With the graph populated. Our first approach was to run a shortest path algorithm (WSP) with a fixed length of **three**. The shortest path returns the geographical shortest path of nodes. From these nodes, a relay is selected **randomly** (to maintain anonymity) and the relay fingerprints are used to create a circuit. Fetch requests sent over this circuit set a base for our further experimentation and testing that involve various different schemes.

3.2 Testing Phase

With the above implementation and code, we ran our fourth test (**test3**). Again, the scheme was same but this time, the exit relay was near to the resolved destination address. The path was selected upon the following criteria:

- **Guard Relay**: This was fixed.
- **Middle Relay**: Randomly selected from the list of middle node selected by shortest path.
- **Exit Relay**: Randomly selected from the list of exit node near to the destination address

The results in Figure 6 were not as we predicted. With shortest path selection, the latencies should have decreased but they increased.

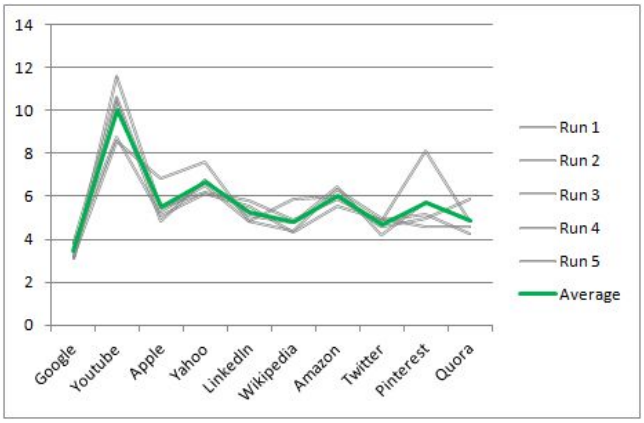

Figure 6: Fetching web-pages on Tor - Circuit built using path shortest path selection with exit node near to the destination address

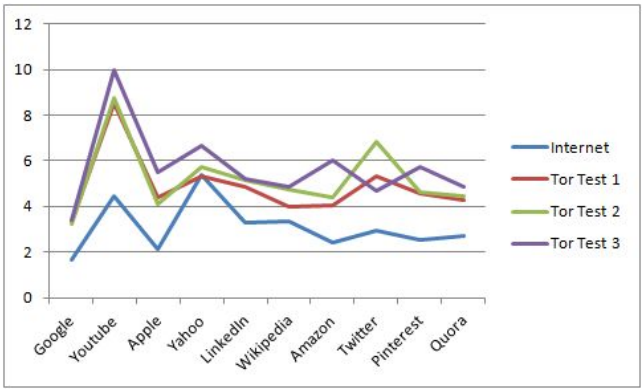

Figure 7: Comparison of all tests

With the above results, we made a few key observations. Upon printing the bandwidths of the relays that were part of the circuit,

we could observe a pattern that the **lower bandwidth relays were surprisingly taking lesser time**. Our reasoning behind this could base on the fact that higher bandwidth relays are being over-utilized and sending traffic to lower bandwidth relays could get lower latencies and in the long run lesser congestion.

This lead to modifying our second test. We conducted **test2b** to select the middle relay with **higher** and **lower** bandwidths both. The results are tabulated in Table 1.

Test 2b: Random Selection based on bandwidth		
Random	Low	High
3.738	4.56	9.861
7.822	14.98	11.407
4.616	5.88	8.474
7.579	5.12	16.55
4.937	5.793	8.431
3.928	6.338	8.653
5.495	11.531	8.753
4.784	9.678	8.335
4.482	9.118	10.891
4.411	8.675	6.851
	1.6x slower	2x slower

Table 1: Random Selection based on bandwidth

4. PATH SELECTION

4.1 The Algorithm

With the guard node and exit nodes set as the end points and values given based on each testing criterias accordingly, we take all weights of the edges that are connected to guard node. We then add to all these edges the distance from corresponding nodes to exit node. The shortest path from this list is returned. In addition we used the Haversine Formula to calculate the distance between two nodes given the latitude and longitude. [10]

$$\operatorname{hav}\left(\frac{d}{r}\right) = \operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)$$

4.2 Clustering of relays

Based on the latitudes and longitudes discovered through GeoIP, we were able to classify relays into more than a dozen clusters on the basis of their IP addresses. The clusters represented the various geographical regions around the world such as North America, Africa, Europe, South and Central America and Asia. Based on the prevalence of Tor relays, each of these geographical regions had multiple clusters of their own. For example, Europe has around 13 clusters in total in our design.

This clustering of relays helped us with constructing a compact, completely connected graph of nodes, with each node representing a geographical cluster and the edges representing the actual distance between the points of average longitude and latitude of the clusters. A side effect of this was that the WSP algorithm run time was drastically decreased which helps in reducing the initial set up time for FasTOR.

4.3 Accounting for destination

As described in Section 3.2, we ran tests where FasTOR was destination aware and therefore, chose the exit relay closest to the destination of the traffic it was carrying. However, as shown, the approach did not bore any fruits as there was not any noticeable improvement in the latencies.

5. ANONYMITY

Anonymity has not been a big focus for us in this paper. Instead we relied on Tor's design to fulfill the anonymity requirement. Nevertheless, we tried to preserve Tor's anonymity by closely following the basic Tor's design principles for security and anonymity such as going for high entropy when choosing relays. In our custom path selection algorithm, we made sure to choose a relay randomly after we had chosen a node from our geographical-based Tor graph. Even when selecting an exit node, we made sure that we made a different choice for an exit relay every time we had to build a circuit.

6. DISCUSSION

6.1 Additional Tests and Results

6.2 Selecting Guard Relay Close to Source

The results for this result show a lower latency than taking guard relay fixed in Netherlands. Here we used a guard relay present in AS1 node i.e. in Asia so that it was closest to the source. Even though it turned out to be faster but it causes anonymity issues and deviation from Tor policies.

6.3 Keeping the Tor path shortest

The results for this result were the best when it comes to latencies and that was because the packets stayed at the Tor path for the least time. Even though it turned out to be faster but again it causes anonymity issues since most of the time our traffic is outside the Tor network.

6.4 Selecting lower bandwidth middle and exit relays both

These results were not so different from **test2** leading to the inference that geographical distances and lower bandwidth only are not the ideal metrics for this issue. In addition we got to know that advertised bandwidth is very different from what the actual relay has.

6.5 Key Takeaways

There were a number of key takeaways from our study such as:

Geographical Distance is not the ideal metric to measure latencies. The paper on LASTor tried to make a convincing argument to relate latency and geographical distances but during our testing,

we did not find the relationship to be holding up.

Advertised Bandwidth and actual bandwidth are very different. There is a need to have actual state of each relay's actual bandwidth and most importantly the load it is currently serving. Micro-descriptors can hold this information for each relay.

Machine learning can be used to train over the results and datasets to look for the best 'features' that lead to smallest latencies over paths.

6.6 Limitations

We only had access to a few machines accessing internet through a local provider and an academic network (LUMS). Therefore, our measurements for query times is biased and may vary depending on the network the tests are conducted in. Any attempts to replicate the results may yield different results due to the very same reason. However, what we are confident is that the factor of throughput and latency will remain the same; i.e. the difference between measurements on an internet path versus a Tor path or measurements between differently configured Tor circuits will vary with the factor of difference.

7. RELATED WORK

Our work was inspired by Akhoondi, Yu, and Madhyastha's paper on "LASTor: A Low-Latency AS-Aware Tor Client". We initially set out to improve their approach to the latency problem.

In their paper, they developed a new Tor client, called LASTor, to demonstrate that both significant latency gains and protection against snooping ASes can be obtained on Tor today without requiring any modifications to Tor relays. Based on measurements along paths between 10K (client, destination) pairs, they showed that LASTor can deliver a 25% reduction in median path latency. To deliver these latency benefits, they showed that it is important to carefully select entry guards and account for replicated destinations. They also went on and developed a space and time-efficient technique for enabling LASTor to reliably detect the possible presence of snooping ASes on any path. Moreover, they made path selection in LASTor tunable so that a user can easily choose an appropriate trade-off between latency and anonymity.

8. CONCLUSION & FUTURE WORK

What started as an attempt to build a low-latency Tor client cumulated into a learning experience into increasing our understanding of the Tor network and its inner workings. More importantly, it realized us of the erratic nature of Tor relays and the very limited real-time information about the relays that is available to the client. Therefore, there was a lot of fluctuations in the readings that were taken and it was difficult to point those variations to any one factor. For example, we thought that choosing a relay with higher advertised bandwidth was a smarter decision but it turned out that the opposite was better. What this failure made us realise was that there is no fix formula for getting the lowest latency on Tor. Trying out multiple things and tweaking with the different factors can be the key to success.

To that end, we plan to perform machine learning on our collected dataset of the various relays and query times on the circuits that we have built during our testing. Using classifiers such as geographical proximity, advertised bandwidth and type of relay flag (if any), we can train a system to pick the best combination of relays to form a highly efficient, low latency path.

9. REFERENCES

[1] Akhoondi, Masoud, Curtis Yu, and Harsha V. Madhyastha. "LASTor: A low-latency AS-aware Tor client." *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012.

[2] "Stem Docs." Welcome to Stem! — Stem 1.5.4 Documentation. N.p., n.d. Web. 19 May 2017. https://stem.torproject.org/

[3] "GeoIP2 City Database Demo." GeoIP2 Database Demo | MaxMind. N.p., n.d. Web. 19 May 2017. https://www.maxmind.com/en/geoip-demo

[4] Alexa Top 500 Global Sites. N.p., n.d. Web. 19 May 2017. http://www.alexa.com/topsites

[5] "Get Latitude and Longitude." Latitude and Longitude Finder. N.p., n.d. Web. 19 May 2017. http://www.latlong.net

[6] "World City Map of Tor Nodes." World City Map of Tor Nodes. N.p., n.d. Web. 19 May 2017." https://tormap.void.gr/

[7] M. Sherr, M. Blaze, and B. T. Loo, "Scalable link-based relay selection for anonymous routing," in PETS, 2009.

[8] A. Panchenko and J. Renner, "Path selection metrics for performance improved onion routing," in SAINT, 2009.

[9] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second generation onion router," in USENIX Security Symposium, 2004.

[10] "Haversine Formula." Wikipedia. Wikimedia Foundation, 08 May 2017. Web. 19 May 2017. https://en.wikipedia.org/wiki/Haversine_formula