

Improving Tor Onion Routing Client Latency

Stephen Rollyson

stephen.rollyson@gatech.edu

CS-4980 Research Advisor: Constantine Dovrolis

ABSTRACT

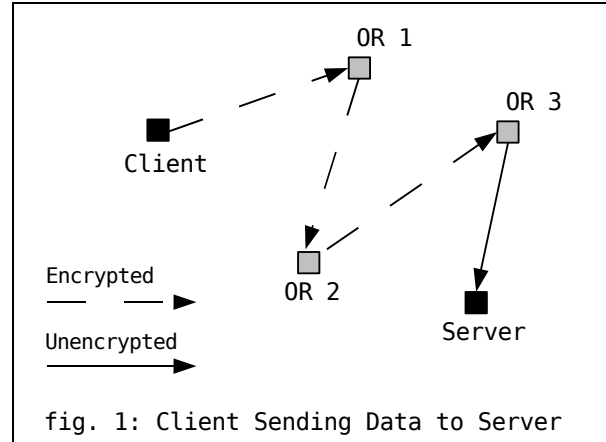
Tor, the largest deployed anonymous routing system to date, is a beneficial tool for political dissidents, whistle blowers, and lawbreakers alike. It has become popular due to its low-latency approach in providing anonymity. However, clients still experience high-latency connections due to the circuit-building code in Tor. After a review of Tor and its circuit-building method, we propose a way to improve Tor client latency by using measurements of latencies between routers in the Tor network. Our approach is to change the way clients choose circuits in the Tor network to provide a best effort attempt at improving client latency without sacrificing anonymity.

1 Introduction

Anonymous systems are not a novel concept in the realm of computer networking. There are many reasons why people would want to exchange information anonymously, be it protection from political persecution, a method of espionage, or simply as a way of harassing others without fear of repercussion. Our research will not be an attempt to argue for or against such systems, nor will it be a proposal of a new anonymous system to replace others. Instead, it will focus on an already existing low-latency anonymous routing system called Tor, which has been both implemented and tested by thousands of users worldwide.

Tor, the most recent iteration of onion routing (explained later), is a unique anonymous system in that it eschews many overly-protective and widely accepted techniques of anonymity in favor of low latency. Its creators hope that its low latency design will make it more usable for interactive applications (like remote shells, not necessarily real-time applications) and will encourage many computer users to join the network. One of the key ingredients of anonymous systems is the size of the population a user can hide in. It goes without saying that the number of users an anonymous system has is directly proportional to the anonymity of each individual user, especially if users are located in diverse locations or legal jurisdictions.

Tor relies on a hierarchical system of servers, onion routers (ORs), and users to provide anonymity. Firstly, there are a number of “directory servers,” which provide a listing of routers in the network to the users. These directory servers are maintained by trusted individuals and rely on consensus to allow new onion routers into the network (protecting against router infiltration via a small percentage of compromised directory servers). Next, there are the less-trusted “onion



routers,” which provide a way to transport encrypted information from a user to its destination and back. Finally, there are the users/clients of Tor, who typically choose routes of three router hops to send information through. It should be noted, of course, that Tor is not a peer-to-peer network, as many clients elect not to run onion routers themselves.

In order to send information through the Tor network, a client first chooses three onion routers from the router listings on the directory servers. Next, the client establishes an encrypted link to the first router to send data through. After that, the first router establishes an encrypted link to the second. Likewise, the second router establishes an encrypted link to the third. In this way, a client can send a message with layered encryption (hence the name “onion routing”). The client sends an encrypted message that the first router can decrypt, which contains instructions to send the encrypted payload message to the second router. In a similar fashion, the second router decrypts that message which contains an encrypted payload intended for the third router. Thus, information transferred between the client and the first router has three onion-like layers of encryption. Figure 1 is a pictorial description of the onion routing structure.

Often, these messages sent from the client to the third router contain TCP packets that the client would like to send to another host using the three-hop Tor circuit as a proxy. Thus, a client establishes standard TCP connections as though it were the third router. TCP packets sent back to the third router from said host will be transferred back to the client using a reversed version of this onion encryption method.

The three-hop Tor routing system actually prevents the first router from knowing the third router and vice-versa, thus making it neigh-impossible for a host to know where a TCP connection actually came from. As

a side benefit, the client's data (when sent through the Tor system) is also triple-encrypted making traffic fairly secure at the client's end. However, if the original TCP connection was meant to send plaintext to the target host, the last-hop router will still send plaintext to the host like any standard proxy server would. Naturally, this TCP connection can reveal the client's identity, especially if the client sends identifiable data over plaintext (which can be intercepted or captured by a compromised last-hop onion router).

2 Tor's Current Method of Route Selection

Currently, clients choose routes in a manner that ensures anonymity at the cost of latency. It should be noted that clients can have a list of preferred routers, but this isn't the default configuration. As such, we'll assume that no preferred routers have been chosen when we explain how a client chooses its path.

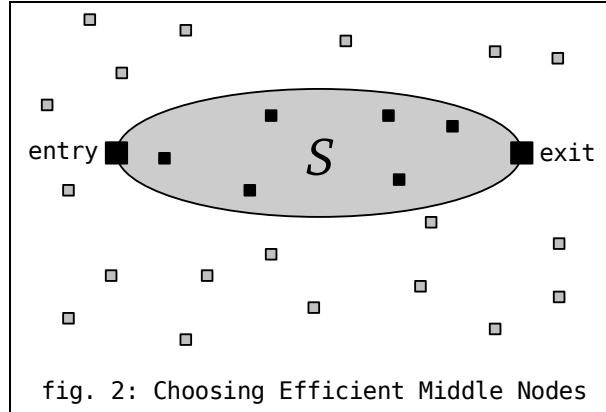
The first and second onion routers are picked from a list of running routers provided by the directory servers. The first two onion routers in the client's route are chosen at random using OpenSSL's cryptographically strong `RAND_bytes()` function. The third exit router is chosen from an eligible list of exit routers filtered by exit policy (allowed outgoing connection ports). This choice is also made using `RAND_bytes()`, but it is weighted based on the exit routers' advertised bandwidth for load balancing.

The benefits of random route choices include the fact that routes are often chosen to include onion routers in multiple countries and jurisdictions. This has its obvious legal benefits but it also hurts client performance. For instance, the first router might be in Asia and another might be in South America. This introduces fairly high latencies in the Tor routes. In the next section we propose a best-effort method of route selection that should improve route latency.

3 Improved Method of Route Selection

Our proposed method of client-side route selection requires that the Tor directory servers provide a list of router-to-router latencies for clients. This server-provided latency list and suggestions on how to implement it are discussed more thoroughly in the following section, "Directory Server Latency Storage."

Prior to the current algorithm design, we attempted a method of route selection that involved picking all three routers in the typical three-hop route in an effort to make the entire route as efficient as possible while maintaining a minimum of anonymity. However, this iteration of the design clashed with the entry guard design and experimental predicted requirements for exit nodes in the 0.1.2.3-alpha build of Tor. More information on exit node predictions is available in the references section under "Tor Path Specification." Because of these issues, the proposed algorithm is limited to picking only the middle node of a circuit in an efficient manner.

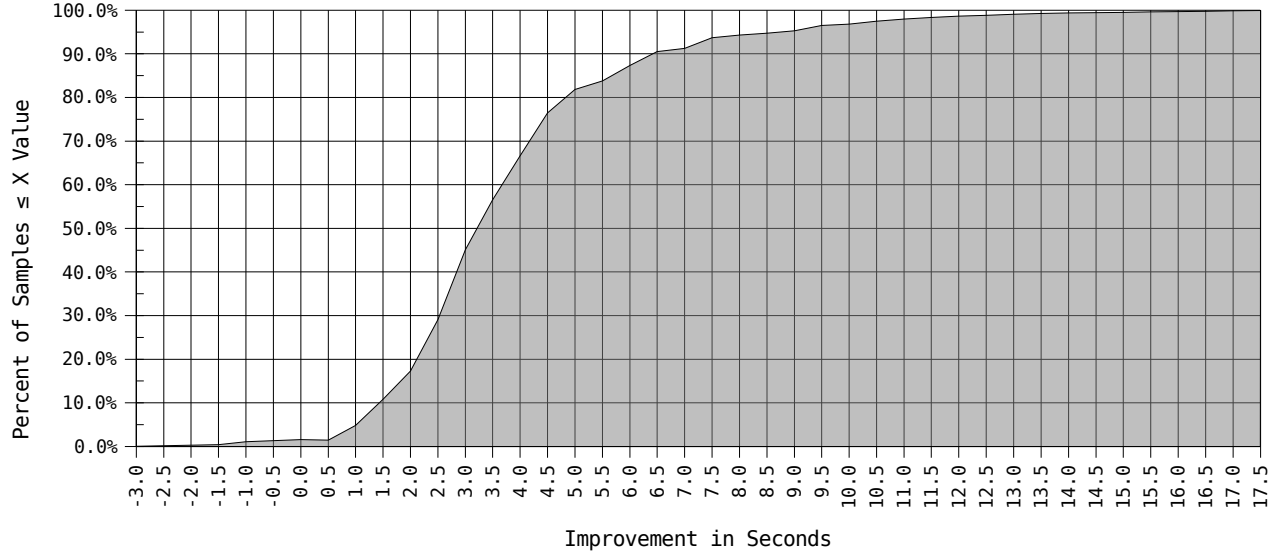


Picking a middle node that improves the entry-to-exit route latency over the current method is actually quite easy. For security reasons, it is best to pick an efficient middle node from a pool of candidate middle nodes so that the entry node and exit node will not be able to discover one another should one of them be compromised. The size of this pool should be a decent percentage of the total number of routers in the Tor network. Assuming S is the set of candidate routers and R is the set of all routers, you can guarantee this by a simple boolean statement: $|S| > \max(5, \text{percentage} * |R|)$. The minimum size of the pool, 5, and the percentage of R are values that are entirely up to the implementation of this algorithm. Larger values of the size of the pool and percentage will approach the current route selection's anonymity guarantees and load balancing while smaller values will tend to get better latencies. Figure 2 is a pictorial description of the candidate set of middle nodes, S .

Once this restriction on the size of S has been established, we can use an incrementing entry-to-exit route latency threshold L starting at 0 ms to choose the routers in S . All the routers that have an entry-to-middle-to-exit latency that satisfies L will be placed within S . After S is a satisfactory size, the client can use `RAND_bytes()` to choose a router in S as the one it will actually use in the circuit. If this algorithm creates load balancing issues on the middle node routers, we can weight the middle nodes with their advertised bandwidths in the same manner as exit nodes.

4 Directory Server Latency Storage

A list of router-to-router latencies poses its own problems. Firstly, this introduces a new security threat in that a router might lie about its own latencies so that it is always placed within the pool of candidate middle nodes. For instance, the compromised router might say it has a latency of 0 ms to all other nodes. This can be remedied by having newly introduced routers tell the other routers to do the latency measurements and post the results on the directory servers. In addition, a larger pool of candidate middle nodes will negate the effectiveness of this attack. Secondly, a list of router-to-router latencies



would be very large and might bog down the directory servers with client requests. For example, some of the test data we used indicated that a one-way latency matrix containing data for 1400 routers (double the size of the Tor network at the time of writing) would occupy around 9 to 10 MB. Clearly, a latency matrix is not very scalable. GNP would be a good way to condense this information [Ng]. Rather than having a one-sided latency matrix with a file size on the order of $O(N^2)$, GNP condenses the latencies to a k -dimensional Euclidean space. The latencies would be represented using k coordinates (where k is some constant value) and would take up space on the order of $O(N)$. Naturally, this space reduction would result in reduced accuracy of the measurements. However, the improved route selection algorithm doesn't require extremely high accuracy to be effective.

5 Client-side Analysis & Comparison

Measuring the effectiveness of this algorithm was a fairly difficult task. Since the directory servers do not provide router-to-router latencies in any fashion, we used King to estimate latencies between servers. King is a tool that estimates latencies between arbitrary remote hosts [Gummadi]. It picks domain name servers near each host and uses recursive DNS queries to find the latencies between the domain name servers, returning the DNS-to-DNS latency as the estimation. We modified King so that it would timeout after 3 seconds (which usually occurred if one of the domain name servers did not support recursion or if an associated domain name server could not be found). Because of the lengthy time to get latency measurements, we had to limit the pool of possible Tor routers to 50 and the minimum size of S to 2. Even with these restrictions, choosing a route took anywhere from 10 to 15 minutes.

Once we had a working implementation, we selected the top 200 websites from Alexa.com's traffic rankings and filtered out 11 unresponsive websites, leaving 189 of the most trafficked websites on the Internet. Then, we used the unix utility *wget* with the *-spider* option to measure the round trip latencies through Tor. In effect, the measurements were taken by using *wget* to retrieve the HTTP header from each website. We took the round trip times (RTTs) of the 189 websites eight times, four with the current route selection algorithm and four with the improved algorithm. After that, we took the average RTTs of both sets and recorded the differences in a table ($average_time_{TOR} - average_time_{IMPROVED}$). The chart at the top of the page is a cumulative distribution of these 189 differences. As you can see, 80% of the differences were between around 1.5 seconds and 6.5 seconds. Sometimes the old route selection had better latency, but we feel that this is largely due to the random nature of route selection. Other pertinent values are the average RTT of the old Tor method, 5.9 seconds, the average RTT of the improved Tor method, 2.15 seconds, and the average improvement that these numbers indicate, 3.75 seconds.

6 Conclusion

In this paper, we presented a way to improve Tor client latency by better route selection. Changing the middle node candidate pool size directly correlates with how random the route selection is. A larger candidate pool size strengthens anonymity whereas a smaller candidate pool size corresponds to lower client latency. Our hope is that this algorithm will be implemented in Tor and make it more usable for interactive applications. A better client experience will likely increase the number of Tor routers and thus improve anonymity for everyone in the Tor network.

7 *References*

- Dingledine, Roger, Mathewson, Nick, and Syverson, Paul. “Tor: The Second-Generation Onion Router”, Usenix Security Symposium 2004, San Diego, CA, August 2004.
<<http://tor.eff.org/svn/trunk/doc/design-paper/tor-design.pdf>>.
- Dingledine, Roger and Mathewson, Nick. “Tor Path Specification.”
<<http://tor.eff.org/svn/trunk/doc/path-spec.txt>>.
- Dingledine, Roger, Mathewson, Nick, and Syverson, Paul. “Challenges in deploying low-latency anonymity (DRAFT).” <<http://tor.eff.org/svn/trunk/doc/design-paper/challenges.pdf>>.
- Gummadi, Krishna P., Saroiu, Stefan, and Gribble, Steven D. “King: Estimating Latency Between Arbitrary Internet End Hosts”, SIGCOMM IMW 2002, Marseille, France, November 2002.
<<http://www.mpi-sws.mpg.de/~gummadi/king/king.pdf>>.
- Ng, T. S. Eugene and Zhang, Hui. “Predicting Internet Network Distance with Coordinates-Based Approaches”, INFOCOM 2002, New York, NY, June 2002.
<<http://www.cs.cmu.edu/~eugeneng/papers/INFOCOM02.pdf>>.