Tor Inter-Relay Latency

Frank Cangialosi[1], Scott DellaTorre[1], Emily Kowalczyk[1], Brent Schlotfeldt[1], and Dave Levin[*1]

[1]University of Maryland

1 Introduction

As

Network latency tools such as ping and traceroute calculate the round trip time of sending small data packets from an originating host to a destination host. [3]

These tools, however, can only use the local machine as the originating host, and therefore can only generate measurements that originate from that machine.

But suppose you are able to measure the latency between two arbitrary hosts that you do not control. For example, what if you could tell a random computer A to ping some other random computer B? And what if you could do this for many computers in many countries many times? Could you craft a map of all the latencies between hosts in a network? How could you use this information to benefit the network?

Our research took these questions and applied them to the unique structure of the Tor network. Tor is a popular anonymity-preserving network, consisting of routers run by volunteers all around the world. It protects a users privacy by relaying their network traffic through a series of routers, called a Tor circuit. Tor circuits typically are generated by a Tor entry node but also can be constructed manually by the client.

Our approach was to construct specific sets of Tor circuits such that with some simple set manipulations we could find the latent measurements described above. More precisely, we constructed three circuits A, B, C where A contained nodes [W, X, Y, Z], B contained nodes [W, X] and C contained [Y, Z]. By calculating the round trip times of each, we could then subtract the round trip time of A round trip time of B round trip time of C to find the latency of [X, Y]- two unknown, intermediary relays. We took such measurements over a sample of _ Tor relays and discuss the results and implications such as _.

The rest of our paper progresses as follows. In section 2, we discuss related work on measuring latency and Tors performance. In section 3, we describe our study and how our times were gathered. Section 4 presents and analyzes our experiments results, and Section 5 concludes and discusses possible future work.

2 Related Work

In an attempt to improve path selection for Tor, Fallon Chen and Joseph Pasquale have explored

[*]Faculty Advisor

the effects of varying physical geographic distance between relays, varying relay performance flags, “fast” and “stable”, and varying number of nodes in a circuit. Their results concluded that reducing the number of relays, reducing the geographic distance between relays, and using relays with the “stable” and “fast” flags improved both performance, on the order of Kbps, and reliability. While this information provides valuable direction for how the algorithm should behave, it does not take into account the dynamic nature of the Tor network, built entirely upon volunteers [1].

Tor RTT, is a tool that allows users to compute the round trip times for Tor circuits and generate graphs from this collected data. Although Like our tool, it makes use of the Python Stem library to communicate with the Tor control socket. Presumably, a Tor user could use Tor RTT to find low-latency circuits for their computer and use this information the create a fast Tor circuit. However, running the tool and analyzing its output is a time-consuming process, and users would have to repeat it every time they changed networks. By computing just a component of the circuit RTT, the inter-relay RTT, for each pair of Tor nodes, our tool can provide valuable network information that is usable by every Tor user. With the knowledge of inter-relay RTTs, Tor clients could avoid choosing circuits containing pairs of relays with high inter-relay RTTs. This would help clients choose more efficient paths and provide Tor users with a better-performing connection. Furthermore, computing the inter-relay RTT for every pair of Tor relays is feasible, since there are on the order of one million such pairs, so this data could be made available for all Tor clients to use. Since the computation of inter-relay RTTs could provide valuable information to all Tor users, and since the Tor RTT tool doesn't provide this functionality, we decided to create our own tool. [2]

3 Procedure

At a high level, we measured the RTT between two arbitrary Tor relays, which we do not have control over, by sending traffic from our TCP client running on our local machine through different Tor circuits to our TCP server running on bluepill.cs.umd.edu and measuring the time between.

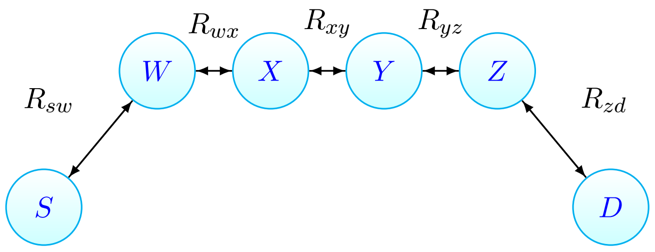
We first wrote a script that queried http://torstatus.blutmagie.de/tor_exit_query.php in order to find the list of all Tor exit nodes that had an exit policy which would allow them to connect with Bluepill on the specified port.

We then built a simple TCP server in Python, which listened for connections on a specified port, expecting to recieve one of two messages, “echo n” or “ping xxx.xxx.xxx.xxx.” If it recieved the message echo, then it acted as an echo server for the next n messages, replying with an echo of any message it recieved. If it recieved the ping message, then it opened a subprocess to run ping on the given ip address, and replied with the output. The purpose for this will be apparent in the upcoming calculations.

With these two in place, we finally created the TCP client in Python, which utilizes the SocksPy library to treat a Tor circuit as a proxy for sending TCP messages and the Stem Python library in order to gain control of Tor circuit creation. First, our client runs the exit node query script in order to obtain the most up to date list of valid exit nodes. Our client then uses the following logic in order to isolate the RTT between two specific nodes, X and Y, from this list.

For this discussion, we will follow the conventions that S represents the source, our local computer, D represents the desintation, bluepill.cs.umd.edu, $T_{S,A,B,...,Z,D}$ is the total measured time between between a message sent from S along the circuit $A, B, ..., Z$ to D, and recieved at S from D following the same circuit back, R_{xy} is the actual round trip time between any two computers, X and Y, and F_x is the port fowarding delay for node X.

1. Create a circuit of 4 randomly chosen Tor nodes, W,X,Y,Z, from the list of valid exit nodes
2. Send a 64-byte message to D, through the circuit W,X,Y,Z, measuring the time elapsed between sending a message to and recieving a response from D as T_{wxyz}. Theoretically, it should be made up of the following components:
$$T_{wxyz} = R_{sw} + R_{wx} + R_{xy} + R_{yz} + R_{zd} + F_w + F_x + F_y + F_z$$
3. Create a circuit of only the first two nodes of the previously created circuit, W and X
4. Send a 64-byte message to D, through the circuit W,X, measuring T_{wx} as in (2). Theoretically, it should be made up of the following components:
$$T_{wx} = T_{sw} + T_{wx} + T_{xd} + F_w + F_x$$
5. Create a circuit of only the last two nodes of the original four-node circuit, Y and Z
6. Send a 64-byte message to D, through the circuit Y,Z, measuring T_{yz} as in (2) and (4). Theoretically, it should be made up of the following components:
$$T_{yz} = T_{sy} + T_{yz} + T_{zd} + F_y + F_z$$
7. Send a message to bluepill, asking it to manually ping node X, response is defined as R_{xd}
8. Manually ping Y from S, result is defined as R_{sy}
9. Final Calculation:
$$R_{xy} = R_{wxyz} - R_{wx} - R_{yz} + R_{sy} + R_{xd}$$



3.1 Data Collection

We randomly chose 100 circuits to run the above procedure on. For each circuit, we repeated steps 1-9 twenty-five times. We chose twenty-five somewhat arbitrarily, aiming for a balance between statistically significant and small enough to be computed quickly. For this dataset of twenty-five times measured in milliseconds, we computed the mean, minimum, maximum, median, and standard deviation.

We then chose an X and Y to remain constant, while we randomly chose 100 combinations of W and Z. For each circuit W,X,Y,Z, we repeated steps 1-9 twenty-five times, and computed the same statistics for each dataset. The purpose of this process was to test the accuracy of our method. In theory, since all components other than R_{xy} are being subtracted away, the result should be independent of which nodes were chosen for W and Z, and should even be independent of the network speed of S or D.

4 Results

We originally hypothesized that taking measurements and calculations as described above would provide accurate figures for the latency between two Tor nodes, which could be compared in order to weight one path as more favorable to another. However, the data we collected was inconclusive in proving this hypothesis. As it turns out, the latency between Tor nodes has high variability, which often threw off our calculations. In our first data collection method, we found that about 49% of the circuits tested yielded a negative average. We also found that on average, the standard deviation was around 300, which is a statistcal formulation of the amount of variability in the set.

We found the variability to be most apparent, however, in the second data collection method. When we kept X and Y constant, only changing W and Z, we expected that the averages, mins, and maxs would vary only on the order of milliseconds between trials, because the variable components of different W and Z relays would be subtracted out. However, as shown in Figure 2, the results were completely unpredictable, and do not evidence even the slightest trend or relation. This it is not possible to conclude from the data whether or not our procedure for RTT between X and Y is sound or not.

5 Conclusion

We have suggested that knowledge of the latency between all pairs of Tor nodes in the network would decrease the overall latency of the Tor network by allowing clients to avoid paths with high latency. Although related work has attempted to improve path selection by measuring considering the RTT of an entire circuit, or properties of individual nodes, none have yet considered the specific path between a pair of nodes. We proposed that by creating multiple circuits, measuring the RTTs of each, and subtracting specific components, we could isolate the value of R_{xy} as the latency between a pair of nodes. Although our insight may be sound, our method of data collection was inconclusive in proving the success of our procedure.

Since this improvement to path selection is novel to our knowledge, there are a great number of possible directions for future work. The first is to collect a much larger sample size of RTTs for each pair, and to make more staistically sound calculations based on the data collected, which would conclusively prove whether or not our procedure is valid. The next step would be to calculate these metrics for all pairs of exit nodes in the Tor network. Given that there are around 1,000 exit nodes currently in operation, this would require calculations for about 500,000 different pairs. While large, this could still be practically and feasibly calculated over a few days. If these prior suggestions are successful, the most interesting step would be modifying the Tor selection algorithm to statistically weight paths between nodes based on this dataset, then computer response times for all circuits created with this algorithm, and compare to circuits created with the traditional algorithm.

References

[1] F. Chen and J. Pasquale. Toward improving path selection in tor. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6, 2010.

[2] ra_. Tor rtt, Dec. 2013.

[3] N. M. Roger Dingledine. Tor protocol specification. 2013.