–Independent Work Report Fall,2014–

Applications of Random Walks in Tor

Aaron Doll

Advisor: Prateek Mittal

Abstract

Existing anonymous communication systems like Tor do not scale well: they require all users to maintain up-to-date information about all available Tor relays in the system, leading to large bandwidth expenditures. In this project, we explore uses of random walks in the Tor network, identifying two candidate designs for improving scalability, and evaluate whether either of these designs could actually improve the scalability of Tor.

The first design is a naive application of random walks that provides some scalability benefits, but nothing that hasn't been achieved through other Tor design tweaks, at the cost of a massive increase in bandwidth and circuit creation latency.

The second design is inspired by PIR-Tor [6], utilizing a client-server approach to scalable anonymity, without the high CPU cost of private information retrieval.

By obtaining relay information about only a few onion routers on an as-needed basis, this design lowers bandwidth costs and improves scalability, while providing security guarantees identical to those of Tor in its current state. We implement a proof of concept of this design, and show that there is potential performance benefits to this approach.

1. Introduction

Increasingly large parts of our lives are lived online, increasing the need for user privacy. Anonymous communication enables users to communicate with other users without revealing their identities to the recipient or a third party. Tor [4] is a deployed network for anonymous communication, consisting of over 5,000 relays and serving hundreds of thousands of users a day. Tor is widely used by whistleblowers, journalists, businesses, law enforcement and government organizations, and regular citizens concerned about their privacy.

Tor requires each user to maintain up-to-date information about all available relays in the network (global state), which can be obtained from twelve authoritative "directory servers", which publish a signed consensus to be downloaded by each member of the network. As usage of the Tor network grows, it will face scaling difficulties, as maintaining this global state becomes more cost prohibitive. There have been attempts to solve the scalability problem using peer-to-peer designs, which achieve scalability at the cost of difficult to reason about security guarantees.

Many of these peer-to-peer systems, such as Pisces [7], ShadowWalker [5], and MorphMix [8], build circuits using random walks. Each relay maintains a list of only a few other relays in the system, and to build a circuit, an initiator first selects a random neighbor and builds an onion routing circuit to it. The initiatiator then requests a list of that node's neighbors, randomly selecting one from that list, and extending the circuit to it. This process can be repeated any number of times in order to build an appropriately long random walk. All designs that utilize random walks have to mitigate route capture attacks, where a malicious node only returns other malicious nodes as its neighbors. This guarantees that all nodes after the first encountered node malicious node will also be malicious, and compromises user anonymity if the first selected hop is malicious. Pisces, ShadowWalker, and MorphMix all contain varying approaches to mitigating this attack, from social networks and a tit-for-tat exclusion policy, to since broken collusion detection mechanisms [9], to structuring the network using a DHT [5]. Unfortunately, these designs do not come with proofs of security,

In this paper, we explore two distinct designs, and evaluate whether they improve the scalability of the Tor network, without changing the security guarantees of the current system. The first is a relatively simple application of random walks. Instead of each relay containing a small list of neighbors, each maintains a list of all relays signed by a directory authority. This design allows clients to stop contacting directory authorities directly, decreasing the load on them. In the course of implementing this design, we discover that many of the benefits of this design have already been realized throughout Tor, and that the bandwidth costs of this design are therefore not worth it. Potentially, such an implementation is still useful for purposes of experimenting with random walks, or as a building block for a peer-to-peer design, but we opted to pursue our second design instead.

This design is inspired by PIR-Tor [6], a client-server approach that improves scalability by also preventing clients from having to maintain global state, except for when first joining the network. It accomplishes this through the use of private information retrieval to download information about a smaller number of relays directly from the relays themselves, preventing any information leakage about the client's relay choices, and therefore preventing route fingerprinting attacks.[2, 3] Our design is a variation of this, providing the same benefits without using private information retrieval. by having the directory authorities sign information about each router, along with an index. When a client connects to a relay, it also sends a random integer r, and the relay returns information for extending the circuit to the r^{th} relay. The client can then use that router as the next hop in their circuit. Since each router knows the next hop anyway, no additional information is leaked by having the relay send the router information to the client, and since it is signed by the directory authority, we don't have to worry about active attacks by the relays. We implement an initial proof of concept for this design, and evaluate it against an unmodified version of Tor using Shadow, a network simulator with support for custom Tor networks [1].

This paper is organized as follows. In Section 2, we discuss related work and background, such as a brief overview of Tor's current design as well as a PIR-Tor. In Section 3, we provide an overview of our first design, and evaluate whether this design would increase the scalability of Tor. In Section 4, we provide an overview of our goals for the second design, as well as an overview, inspired by PIR-Tor. In Section 5, we describe the specifics of our protocol. In section 6, we argue that our design provides the same security guarantees as Tor. In section 7, we briefly describe our implementation of our proof of concept design, as well as its limitations. The performance evaluation of this proof of concept implementation is contained in Section 8. In Section 9 we discuss future work for our implementation. We conclude in section 10.

2. Background and Related Work

2.1. Tor

Tor [4] is a deployed network for low-latency anonymous communication. Tor clients first download a complete list of relays (called the network consensus) from directory servers, and then further download detailed information about each of the relays (called the relay descriptors or microdescriptors). Microdescriptors are a fairly recent addition intended to save bandwith, containing only relatively static information such as ip address, router family, and public keys, preventing clients from having to download descriptors, which contain information irrelevant to circuit creation client side. The network consensus is signed by trusted directory authorities to prevent directory servers from manipulating its contents. A fresh network consensus must be downloaded at least as often as every 3 hours, while fresh relay descriptors are downloaded every 18 hours.

Clients select three relays to build circuits for anonymous communication. When a client starts Tor for the first time, it chooses three guard relays from among fast and stable nodes. As long as the selected guards remain available, new ones aren't chosen. The first relay in any circuit constructed by the client will be one of its three guards. Also, clients pick a circuit's final relay from the subset of the Tor relays which allow traffic to exit to the Internet, called exit relays; each exit relay has an exit policy, which lists the ports to which the relay is willing to forward traffic. The client is responsible for choosing an exit relay which can fulfill its intended purpose for the circuit. All relays are eligible to be the middle relay of a circuit, although clients prefer using relays that are not guard or exit relays, to avoid overloading these relays, since they can serve in multiple portions of circuits. Finally, Tor relays have heterogeneous bandwidths, so in order to avoid overloading slow relays, clients select a Tor relay with a probability that is proportional to that relay's bandwidth.

Recently, Tor also introduced the idea of a directory guard. Much like an entry guard, once selected, a directory guard is always contacted by the client for fetching the network consensus and descriptors or microdescriptors. This has a number of benefits, including limiting the number of accesses of authoritative directory servers required.

2.2. PIR-Tor

PIR-Tor is an architecture for the Tor network where clients do not need to maintain a global view of the system, and instead leverage private information retrieval techniques to obtain information about only a few onion routers while protecting their privacy from compromised directory servers.

According to Mittal et. al., PIR-Tor reduces the communication overhead of the Tor network by at least an order of magnitude, maintaining manageable overhead even when the Tor network scales by two orders of magnitude. Unfortunately, this reduction comes at the cost of increased CPU time for PIR relays, which while manageable, is less than ideal.

The security of their architecture depends on the security of PIR schemes which are well understood and relatively easy to analyze, as opposed to peer-to-peer designs that require analyzing extremely complex and dynamic systems, and they demonstrate that reasonable parameters for the architecture provide equivalent security to that of the Tor network.

They analyze two versions of PIR-Tor architecture, one based on computational PIR and the other based on information-theoretic PIR. In computational PIR, clients fetch only a few blocks from the PIR database, and reuse blocks to build additional circuits. While this modification has no impact on client anonymity, it slightly weakens the unlinkability of circuits. On the other hand, in information theoretic PIR, clients perform a PIR query per circuit creation and do not reuse blocks, resulting in a level of security that is equivalent to the Tor network. While information-theoretic PIR requires all guard relays to be directory servers, computational PIR is more easily integrated into the current Tor network. [6]

3. Naive Random Walk Based Approach

In this design, each relay maintains the global list of other relays signed by the directory authorities, and when contacted by a client, it sends it this list, allowing the client to choose a server at random locally. This design decreases the load on directory authorities, since clients do not need to contact them for accessing the global view of the network.

To create a circuit, the onion proxy (OP) follows the below algorithm:

3.1. Circuit Creation

1. Use its long lived entry guard to serve as the first hop of the circuit.
2. If not already connected to the first router in the chain, open a new connection to that router.
3. Choose a circID not already in use on the connection with the first router in the chain; send a CREATE cell along the connection, to be received by the first onion router.
4. Wait until a CREATED cell is received; finish the handshake and extract the forward key Kf_1 and the backward key Kb_1.
5. Download list of microdescriptors/network consensus from this node. Verify that they have been signed by directory authorities.
6. Choose a random router R from the received list of microdescriptors, extend the circuit to R.
7. Repeat this process for the nodes $R_2 \cdots R_N$

3.2. Circuit Extension

To extend the circuit by a single onion router R_M, the OP performs these steps:

1. Create an onion skin, encrypted to R_M's public onion key.
2. Send the onion skin in a relay EXTEND cell along the circuit.
3. When a relay EXTENDED cell is received, verify KH, and calculate the shared keys. The circuit is now extended.

3.3. Evaluation

The scalability benefits of this design are far outweighed by the additional bandwidth and circuit build latencies that would come along with it. Now that Tor uses the directory guard design, clients already rarely have to contact the authoritative directory servers. If we were to implement this design, it would take over three times the bandwidth of the directory guard design per circuit, which is clearly impractical. Also, downloading the entire directory every hop takes a significant amount of time, especially since directory fetches are done asynchronously, making it difficult to implement without making significant changes to how directories and circuit building works.

That being said, an implementation of this design would have its uses, but primarily for experimentation with random walks where each relay contains some subset of the relays in the network. Also, it would be useful as a primitive for use in other designs that make heavy use of random walks. That being said, without a concrete purpose in mind, it's difficult to imagine that an implementation of this design would get much use unless done in collaboration with someone with an end goal in mind.

4. PIR-Tor/Random Walk Hybrid Design Overview

4.1. Goals

The design goals for this system are nearly identical to those of PIR-Tor, and we reproduce them briefly here.

1. Scalable architecture: The design should be able to scale as the number of relays and clients increase. This improves both performance and security. This design decreases the bandwidth needed by the network, as clients are no longer required to download the global view of the network. Unlike in PIR-Tor, these bandwidth savings are obtained at little to no cost in compute time. The one requirement is that all relays must serve as directory caches, but as relays already maintain a global view of the network, this is a negligible cost.
2. Security: We aim to achieve similar security properties to the existing Tor network. We show

that our design's security properties are identical to that of the Tor network today.

3. Efficient circuit creation: We aim to minimize our design's impact on circuit creation times. Our design adds little to the circuit creation process, adding some additional signature verifications as well as accesses to the relay database at each relay.
4. Minimal changes: We aim to minimize changes to the existing Tor design, since this makes eventual implementation more realistic. This design does touch many areas of Tor's codebase, but most changes are not very large, especially relative to converting Tor to a fully peer-to-peer system. This design is incrementally deployable, since we perform all communication using modified CREATE and CREATED cells, which can be changed in backwards compatible ways.
5. Preserving Tor constraints: Circuits in this design follow existing Tor conventions, utilizing an entry guard, middle relay, and exit relay with appropriate exit policy. They also are performant, and choose relays with probability proportional to their available bandwidth.

4.2. Architecture

The key insight in this design is that clients only need to download the information for relays on an as-needed basis, immediately before they extend their circuits to them. While PIR-Tor uses private information retrieval in order to prevent fingerprinting attacks, we take our inspiration from an optimization described in the PIR-Tor paper, where they mention that if a client's entry guard is acting as the PIR server and a client is requesting a middle node, then there is no need for performing private information retrieval, since the entry guard will learn the identity of the chosen middle relay when it extends the circuit to it.

We extend this idea to the next hop by having all relays serve as directory servers, allowing clients to directly query for a next hop in the circuit. Since all relays already maintain global state, it is not a stretch to require them all to be directory caches as well. Every time a client contacts a relay *R*, it includes a random number corresponding to one of the relays in the network, and *R* simply returns the information for extending to that relay, signed by a directory authority. The client can then extend the circuit to that relay.

While guard relays must still be chosen by downloading the network consensus and descriptors/microdescriptors, this still represents a sizable improvement in scalability, since guard relays are long lived and rarely change.

5. Protocol Details

5.1. Database Organization

We make some minor changes to PIR-Tor's proposed database organization. PIR-Tor organizes relays into three databases: entry guards, middle relays, and exit relays. In our design, we organize the relays into two databases, middle and exit relays. We have no need for a separate database for entry guards, since those will always be selected by contacting a directory server, as otherwise a client would leak information about its selected entry guards. This allows clients to specify whether they need a middle or exit server from a relay in the circuit.

One complicating factor is how to deal with exit policies. In addition to the last relay being an exit, its exit policy must satisfy the client application requirements. This poses problems for our design, as clients cannot decide on an exit for themselves. In a similar, but slightly different approach to PIR-Tor, we could split the exit database into separate databases based on exit policies, and allow nodes to specify a policy when requesting a next hop. In order to make this manageable, exit policies should be standardized, not only for performance reasons, but leaking unique exit policies could serve as identifying information. If a client truly needs a non-standard exit policy, they could fallback to the current method for circuit creation, downloading all relays and doing all relay selection client side.

Tor relays have varying bandwidth capabilities, and relays with higher capacities are selected with a higher probability in order to load balance the network. Bandwidth-weighted selection is straightforward given a global view of the network, but becomes more difficult when done in a distributed manner. One possible solution is for the databases to be sorted in ascending order by bandwidth, and when clients select the random number for the next hop, they simply weight their random number selection such that low numbers have lower probabilities than higher numbers. In order to know how much to weight the numbers, clients would have to download a summary of the current bandwidth distribution. Also, in order to select random numbers bounded by the number of middle relays or exits of a certain exit policy, each client will have to obtain a count for the number of relays in each database. This information could signed by the directory authorities and obtained from a client's entry guards, or any other directory server.

Each entry in a database consists of the relay's nickname, digest, public key, IPV4 address and port, the index of the relay, a timestamp, the relay's router families if applicable, and a database identifier. This is the bare minimum for extending a circuit to a node, additional information such as IPV6 address and port, or the descriptor flags could also be useful. The router family is necessary in order to allow clients to guarantee that no two relays from the same family will be in the same circuit, which could allow the operator of that family to run end to end correlation attacks. Each entry is signed by the directory authorities to prevent active attacks by the directory servers.

5.2. Circuit Creation

The circuit creation algorithm is as follows:

1. Use one of its long lived entry guards to serve as the first hop of the circuit. If no entry guard exists, then contact a directory server or directory cache and download the full network state to choose an entry guard. This is only necessary again if your entry guards go offline.
2. If not already connected to this router, open a new connection to that router.
3. Choose a circID not already in use on the connection with the first router in the chain; send a CREATE cell along the connection, to be received by the first onion router. The CREATE cell must contain a field for a random number, which will correspond to the next hop, as well as whether the next hop needs to be an exit.
4. When an onion router (OR) receives a CREATE cell, it finishes its side of the handshake, placing key material in a CREATED cell. It extracts the random number r from the CREATE cell, as well as whether an exit server is needed. If an exit server is not needed, then it populates the CREATED cell with information for the r^{th} regular router, which includes the nickname, digest, public key, IP address, port, and a timestamp, all signed by the directory authorities. If an exit server is required, then it performs the same operation, but operating over a separate database of exit servers. The OR sends this CREATED cell back to the previous hop in the circuit.
5. The OP waits until a CREATED cell is received; finishes the handshake and extracts the forward key Kf_1 and the backward key Kb_1. Extracts the information for the next hop R from the CREATED cell. Check that received row is signed by a directory authority, and matches the row that was requested. The OP also checks that the received hop is not in the same router family as any of the current hops on that circuit, in order to help prevent the possibility of correlation

attacks.

6. Extend the circuit to R, requesting an exit server when one is needed.
7. Repeat this process until the path is N routers long.

5.3. Circuit Extension

The circuit extension algorithm is as follows:

1. Create an onion skin, encrypted to R_M's public onion key.
2. Send the onion skin in a relay EXTEND cell along the circuit. The EXTEND cell contains a CREATE cell intended for R_M.
3. Eventually, the CREATE cell reaches R_M, and R_M processes the CREATE cell as described above.
4. When a relay EXTENDED cell is received, verify KH, and calculate the shared keys. The circuit is now extended. If necessary, extract the next hop from the CREATED cell within the EXTENDED cell.

6. Security Analysis

The security properties of this design are identical to those of Tor. Each relay in a given circuit's path will only know the details of the previous and next hop, which is just as it is in the Tor network today. Furthermore, each of the nodes in a circuit are unable to mount active attacks due to the directory authority signatures on each relay, although they can decide to corrupt or refuse to respond to a CREATE request. Little true harm can be done to the client in this case, since when it receives the invalid signature or doesn't get a response, it will simply terminate that circuit and start a new walk beginning from one of its entry guards.

7. Proof of Concept Implementation

We implemented a proof of concept for this design based off of Tor v0.2.5.10, which constructs circuits using this design, albeit with some limitations. We limited our changes to circuit construction, since this is where the majority of the changes needed to be made, and although we had plans to make additional changes after we evaluated this basic implementation, difficulties with simulations using Shadow led to a delay in plans. The code for the implementation is located at https://github.com/adoll/tor/tree/drw-lib.

7.1. Implementation Design

We were able to implement the entire circuit building process, primarily by adding fields to CREATE and CREATED cells that support requesting and sending the information necessary to extend a circuit to the next hop. This enabled us to leave a large amount of the infrastructure for sending and receiving cells unchanged, and limited the bulk of our changes to src/or/onion.[c,h], src/or/circuitbuild.[c,h], and src/or/command.[c,h]. onion.c contains files for parsing and formatting CREATE, CREATED, EXTEND, and EXTENDED cells. circuitbuild.c contains the majority of the client side circuit construction logic, beginning with the function circuit_establish_circuit, as well as the function do_random_walk, which does the relay side processing. By searching the repository for do_random_walk, one can see the entry points for a relay to select a random node.

Another benefit of working solely with CREATE and CREATED cells is that these changes are backwards compatible, allowing this to be slowly rolled out if need be. Old clients will simply ignore the additional fields. Old relays will respond normally to CREATE cells that include requests for a relay, although they obviously won't incude a relay to use. Practically speaking, if the majority of relays don't support random walks, then many of the benefits would be lost, since clients would still have to maintain global state for nodes that fail to respond, but as long as most relays support it, there is little harm in having many clients still using the current design, unless the network hits the scalability thresholds, at which point it must transition to a new design, if not this one.

7.2. Implementation Limitations

First of all, we didn't modify directory servers in any way, so the databases for middle and exit nodes aren't actually implemented, merely simulated. When a relay receives a request for a next hop corresponding to index i from a client, it first checks whether it is requesting an exit or middle relay. Then, it adds either all exits, or all relays respectively to a list, sorts it by digest in order to ensure that the ordering is consistent across all relays, and then chooses the i^{th} one. When receiving a next hop, no signatures are verified, since authoritative directory servers don't sign individual relays. We also haven't implemented any bandwidth weighting, and rather than getting the number of exits and total relays from directory servers, we pass it in via flags set in the torrc.

Exit policies currently aren't taken into account, which doesn't affect our simulations since Shadow, by default sets permissive exit policies, but would preclude this implemenation from working in the real world. Also, we don't check whether a circuit contains two routers in the same family. Another relatively small problem that could be handled better in future iterations, is that currently, upon receiving a duplicate relay or encountering some other problem, we simply tear down the circuit and start over. While for ease of implementation this is certainly the way to go, in future iterations there should be a more robust error recovery procedure for these circuits.

One of the primary goals of this implementation is to prevent clients from having to maintain a global view of the network in order to create circuits. Unfortunately, at this point, this is still required by our proof of concept. There was simply too much functionality reliant on having this global view to remove all traces of it. For example, clients don't contact entry guards optimistically, instead they check the network consensus first, before establishing a connection. Also, when evaluating whether circuits can be used for certain purposes, or when calculating various metrics such as path bias, Tor repeatedly makes use of descriptors and the network consensus. Decoupling Tor from the assumption that global state will exist seems to be a rather large task.

8. Performance Evaluation

8.1. Shadow

We tested the performance of this implementation against Tor v0.2.5.10 using Shadow [1], a network simulator designed for simulations of custom Tor networks. We used the tiny-4GB topology, deployed on an Amazon EC2 m3.large instance. This topology has a total of 49 relays, 11 of which are exits. Initially we had some difficulty getting Shadow to work with our implementation, since we were using a more recent version of Tor than the last build of Shadow, which broke some Shadow functionality. After fixing that problem, we also had an encounter with one of the oddities of Shadow; namely that certain portions of the Tor source code are reimplemented by the Scallion plugin, meaning that any modification made in those portions of the Tor code, do not get executed when running a simulation. This led to a frustrating situation where our implementation worked on Chutney, another simulation tool, but not on Shadow. These difficulties somewhat limited our time for experimentation, but eventually we were able to run experiments with appropriate modifications made to v1.9.2 of Shadow.

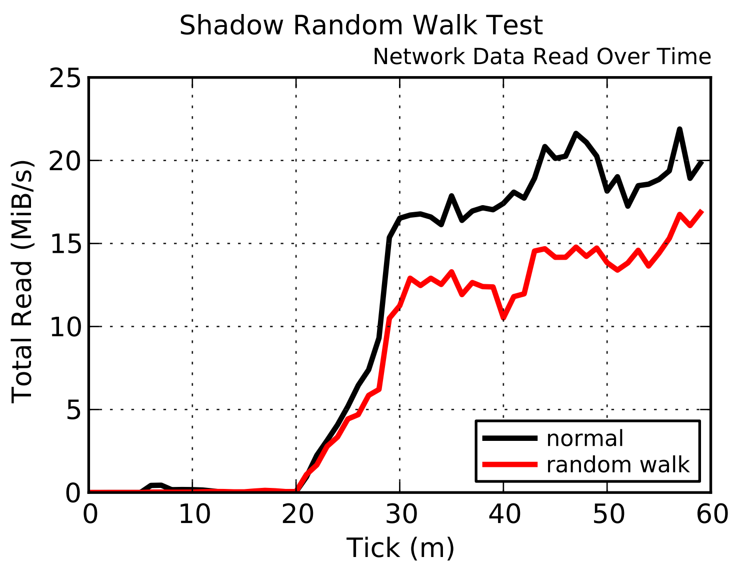

Figure 1: Read throughput vs time for both Tor v0.2.5.10, and our proof of concept implementation.

8.2. Results

In figures 1 and 2, there is an obvious gap between the throughputs reached by the random walk version, and those reached by Tor v0.2.5.10. It appears that the regular version is approximately 25% faster at the point of biggest difference between the two.

While somewhat disappointing, this is explainable by the lack of bandwidth weighting in our current implementation, and in some respects it is surprising that the throughputs are that close, despite the complete lack of bandwidth weighting in our implementation.

More promising are the results seen in figure 3. This is a graph of the fraction of requests versus the time it takes for the first byte of that request to be received. This is a good proxy for circuit construction times, and interestingly enough, our implementation has an equal or higher fraction of requests with very low times, and while vanilla Tor has an advantage as the times get longer, this is probably due to a lack of robustness in our implementation. For example, getting a duplicate router within a circuit is relatively frequent in our implementation, and this event causes the circuit to be restarted, which would greatly increase circuit construction times for the slowest circuits. There

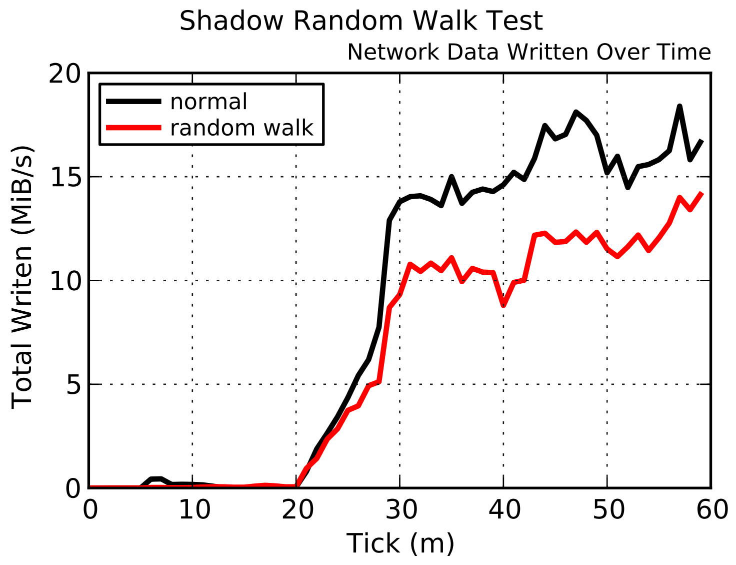

Figure 2: Read throughput vs time for both Tor v0.2.5.10, and our proof of concept implementation.

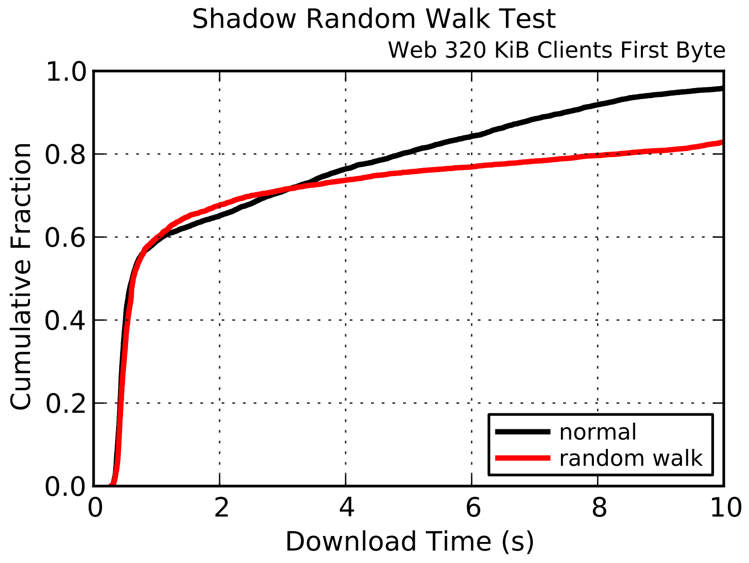

Figure 3: A graph of the proportion of web requests vs the time that it took to receive the first byte of that request for both Tor v0.2.5.10, and our proof of concept implementation.

are a few changes that could be made to make this less likely, and as the network grows in size, it becomes vanishingly less likely that a duplicate relay will be selected in the same circuit. This could also be a symptom of the lack of bandwidth weighting. As low bandwidth relays become overwhelmed due to their disproportional inclusion in circuits, they become bottlenecks, increasing latency greatly.

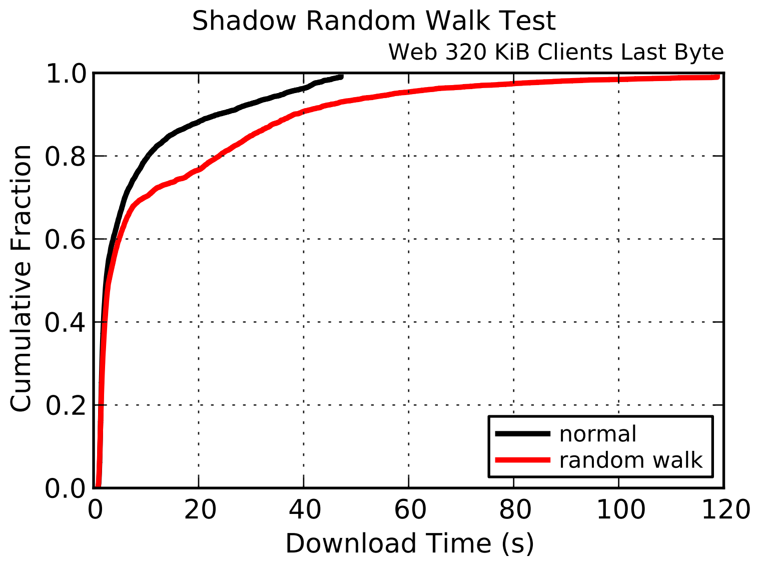

Figure 4: A graph of the proportion of web requests vs the time that it took to receive the last byte of that request for both Tor v0.2.5.10, and our proof of concept implementation.

In 4, we again see a nonnegligible gap widen between the reference implementation and our design, which I think again demonstrates the importance of weighting by bandwidth. Circuits that traverse these overtaxed hops simply take a very long time, skewing the results.

9. Future Work

While the initial results seem promising, there are many improvements that can be made to our current implementation, such as weighting by bandwidth, stopping clients from downloading the network consensus and descriptors in order to realize the full bandwidth savings, and implementing the necessary changes to directory servers' format. If we hope to make an impact and have this design implemented in Tor, we'll need to continue work on improving the implementation.

Also, Tor currently implements some basic protections against AS level adversaries, avoiding selecting relays with IP addresses within the same /16 IP prefix. How does this design function with this and other client side processing that needs to be done? Do these requirements lead to clients that are required to throw out many circuits before getting one to work? Could we split relays into different databases based on IP addresses, similar to our approach for dealing with exit policies? Or will this design provide a level of "good enough" security, where high risk users will need to download a global view of the network to ensure full security?

10. Conclusion

In this paper, we presented two designs for the Tor network where clients do not need to maintain a global view of the system, and instead utilize random walks. While our first design requires exorbitant bandwidth costs and offers surprisingly few scalability benefits, our second obtains relay information on an as-needed basis, reducing bandwidth usage signficantly at little computational cost, while still providing identical security guarantees to Tor. In order to further explore the practical considerations of this design, we implemented a basic proof of concept, which shows promise, but still requires more work if our goal is to have this design receive serious consideration for inclusion in Tor.

11. Honor Code

This paper represents my own work in accordance with University Regulations.

Aaron Doll

References

[1] "Methodically modeling the tor network," in *Presented as part of the 5th Workshop on Cyber Security Experimentation and Test*. Berkeley, CA: USENIX, 2012. [Online]. Available: https://www.usenix.org/conference/cset12/workshop-program/presentation/Jansen

[2] G. Danezis and R. Clayton, "Route fingerprinting in anonymous communications," in *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, Sept 2006, pp. 69–72.

[3] G. Danezis and P. Syverson, "Bridging and fingerprinting: Epistemic attacks on route selection," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, N. Borisov and I. Goldberg, Eds. Springer Berlin Heidelberg, 2008, vol. 5134, pp. 151–166. Available: http://dx.doi.org/10.1007/978-3-540-70630-4_10

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[5] P. Mittal and N. Borisov, "Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies," 2009.

[6] P. Mittal *et al.*, “Pir-tor: Scalable anonymous communication using private information retrieval,” in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC’11. Berkeley, CA, USA: USENIX Association, 2011, pp. 31–31. Available: http://dl.acm.org/citation.cfm?id=2028067.2028098

[7] P. Mittal, M. Wright, and N. Borisov, “Pisces: Anonymous communication using social networks,” *CoRR*, vol. abs/1208.6326, 2012. Available: http://arxiv.org/abs/1208.6326

[8] M. Rennhard and B. Plattner, “Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection,” in *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, ser. WPES ’02. New York, NY, USA: ACM, 2002, pp. 91–102. Available: http://doi.acm.org/10.1145/644527.644537

[9] P. Tabriz and N. Borisov, “Breaking the collusion detection mechanism of morphmix,” in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, G. Danezis and P. Golle, Eds. Springer Berlin Heidelberg, 2006, vol. 4258, pp. 368–383. Available: http://dx.doi.org/10.1007/11957454_21