Congestion-aware Path Selection for Tor[*]

Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
{t55wang,k4bauer,ciforero,iang}@cs.uwaterloo.ca

ABSTRACT

Tor, an anonymity network formed by volunteer nodes, uses the estimated bandwidth of the nodes as a central feature of its path selection algorithm. The current load on nodes is not considered in this algorithm, however, and we observe that some nodes persist in being under-utilized or congested. This can degrade the network's performance, discourage Tor adoption, and consequently reduce the size of Tor's anonymity set. In an effort to reduce congestion and improve load balancing, we propose a congestion-aware path selection algorithm. Using latency as an indicator of congestion, clients use opportunistic and lightweight active measurements to evaluate the congestion state of nodes, and reject nodes that appear congested. Through experiments conducted on the live Tor network, we verify our hypothesis that clients can infer congestion using latency and show that congestion-aware path selection can improve performance.

1. INTRODUCTION

Tor is an anonymity network that preserves clients' online privacy. Today, it serves hundreds of thousands of clients on a daily basis [16]. Despite its popularity, Tor suffers from a variety of performance problems that result in high and variable delays for clients [9]. These delays are a strong disincentive to use Tor, reducing the size of the network's user base and ultimately harming Tor users' anonymity [7]. One reason why Tor is slow is due to the challenges of balancing its dynamic traffic load over the network's available bandwidth. In this work, we propose a new approach to load balancing that can reduce congestion, improve performance, and consequently encourage wider Tor adoption.

Path selection in Tor. The current path selection algorithm selects nodes based on the bandwidth of the nodes (adjusted by the current distribution of bandwidth in the network among entry guards, exits and other nodes), giving a higher probability of being chosen to nodes with higher bandwidth. It also takes into account a number of constraints designed to promote network diversity; the same node cannot be used more than once on the same circuit, every node in the circuit must belong to a different /16 subnet, exit policies and node flags, etc. However, peer-to-peer file sharing users, while discouraged from using Tor, may still do so and consume a significant portion of the available bandwidth [18]. Even though the number of such users is likely small, when these bulk downloaders use nodes with insufficient bandwidth, they may affect the performance of other clients using the nodes by introducing high delays due to congestion.

Latency as a congestion signal. Congestion occurs at the node level either when a node reaches its bandwidth rate limit configured in Tor, or when a node's connection to the Internet is congested. When a node is congested, outgoing cells must wait in the node's output queue. We find that this *node latency* is sometimes significantly larger than the *link latency*, which is dominated by the propagation delay between two nodes. Delays that do not originate from propagation effects have been found to be quite common [5]; they have also been found to be large [22]. From measurements and analysis of the live Tor network, we find that Tor's token bucket rate limiting implementation often contributes to congestion delays of up to one second per node. These delays are detrimental to interactive web browsing users, who are the most common type of Tor user [18].

Figure 1 demonstrates a scenario under which latency can be used as an indicator of congestion. This figure shows the round-trip times of two different Tor nodes, PPrivCom007 and eficommander. We measured these round-trip times using a one-hop circuit from the client to each node. Their mean round-trip times are comparable (about 0.53 s), but the distribution of the measured round-trip times to PPrivCom007 had higher variance: one-quarter of the measurements exceeded 0.6 s. PPrivCom007's long tail suggests that cells had to wait a long time at the node before being processed; this indicates congestion. In contrast, the cells sent to the node eficommander almost never had to wait, even though getting there took more time (for us). Traffic going through PPrivCom007 could be redirected to eficommander; this will improve the experience of those using PPrivCom007 while not harming those using eficommander.

Congestion-aware path selection. To reduce congestion and improve Tor's load balancing, we introduce *node latency* as a new metric to be used when selecting nodes to form a circuit. Our approach uses a combination of lightweight active and opportunistic methods to obtain this information. Clients measure the overall latency of their circuits and use an inference technique to extract the component latencies due to congestion for each individual node along the circuit. Live experiments indicate that a typical client's circuit latency can be reduced by up to 40% if congestion information is taken into account during path selection.

Contributions. This paper contributes the following:

1. We identify latency as a measure of node congestion and characterize how congestion varies across different types of nodes. We describe ways to observe and iso-

[*]Last updated on September 15, 2011.

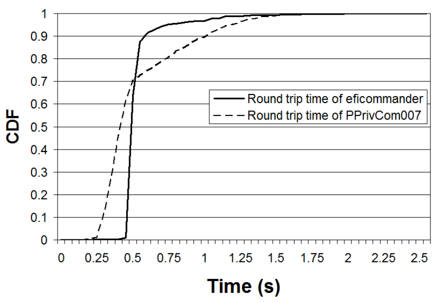


Figure 1: Round trip time of two different Tor nodes measured on 30 March 2011. 1,000 measurements were taken for each node.

late this node congestion from other sources of delay (such as propagation delay) with lightweight tests.

2. We design and evaluate a latency inference technique that attributes congestion-related latencies to constituent nodes along a measured circuit.

3. We extend Tor's path selection algorithm to avoid congested relays when building circuits. Our approach has low overhead, can be incrementally deployed, needs no additional infrastructure, and our live evaluation shows that it improves performance.

Outline. We present our findings in the following sections. In Section 2 we describe Tor's design at a high level. Section 3 presents related work. Section 4 describes our design goals. Section 5 details our methods for measuring node latencies. We describe our congestion-informed path selection algorithm in Section 6. In Section 7, we present experimental results and discuss open issues and extensions in Section 8. Section 9 includes avenues for future work and conclusions are presented in Section 10.

2. TOR BACKGROUND

Tor is the third-generation onion routing design providing source and destination anonymity for TCP traffic. A client wanting to connect to an Internet destination through Tor first contacts a *directory server* to obtain the list of Tor nodes. Next, the client constructs a *circuit* of three *Tor routers* (or nodes) and forwards traffic through the circuit to a desired destination using a layered encryption scheme based on onion routing [13]. To balance the traffic load across the routers' bandwidth, clients select routers in proportion to their bandwidth capacities. To mitigate the predecessor attack [29], the first router on the circuit (called an "entry guard") is selected among nodes with high stability and bandwidth. Clients choose precisely three entry guards to use for all circuits and new entry guards are selected every 30 days. The last router (called an "exit router") is chosen to allow delivery of the client's traffic to the destination. All data is transmitted through Tor in fixed-size 512-byte units called *cells*. More details about Tor's design can be found in its design document [8] and its protocol specification [6].

3. RELATED WORK

Tor requires a good path selection algorithm to effectively distribute its traffic load across its nodes. Different criteria have been proposed as possible factors in the path selection algorithm, such as autonomous system awareness [11], application awareness [24], or game-theoretic criteria for maximal utility [30]. In this paper, we describe a modification to Tor's existing path selection algorithm to incorporate latency, which reduces congestion and improves load balancing.

Snader and Borisov [25, 26] propose opportunistic bandwidth measurement schemes to improve Tor's current bandwidth-weighted selection algorithm. Opportunistic measurement schemes have the advantage of adding little or no overhead to the network. Our algorithm is also opportunistic. However, their work focuses on measuring bandwidth while we focus on latency.

A persistent distributed reputation system of nodes used in path selection is proposed by Bauer et al. [2] as a defense against nodes which misreport their bandwidth (as self-reported bandwidth was exclusively used at the time). Their reputation system does not include latency; we propose that latency is a good indicator for improperly measured or reported bandwidth. Due to poor load balancing, users may choose nodes which cannot handle the traffic, causing congestion.

Using latency as a path selection criterion has been investigated by Sherr et al. [23]. In their paper, a case is made for link-based path selection, which uses link-based properties (e.g., latency, jitter, loss). Panchenko and Renner [20] propose using round trip time as a link-based measure to choose paths. They give a technique to obtain round trip time and roughly analyze the increase in performance by using this criterion. In this paper, however, we look into considering latency as a *node-based* property instead of a link-based property. Link-based latency includes propagation delay, so only using link-based latency as a measure may bias path selection against circuits with nodes that are geographically far apart or on diverse networks.

Latency in Tor has also been considered from other perspectives. Hopper et al. [15] looked into how network latency can be used to deanonymize clients. Evans et al. [12] investigate using long paths to congest routers, thus revealing the identities of those connected to the router due to the change in round trip time. Since our congestion-informed path selection approach allows clients to detect congested routers, our proposal may be a defense against such attacks; we do not, however, focus on defense mechanisms in this paper, but rather on improving Tor's performance.

4. DESIGN

We designed our scheme based on the need for practical, yet lightweight latency measurements. Our design decisions and reasoning are given in this section.

4.1 Centralized or Decentralized

Tor's centralized bandwidth authorities currently use active measurements [21] to reach a consensus on routers' bandwidth. However, centralized schemes may be open to attacks that try to "game the system," where a malicious node performs better if it knows it is being measured. This causes the node's resources to be overestimated and too many clients will connect to it, which may eventually cause congestion or strengthen attacks. Furthermore, centralized schemes cannot provide clients with real-time data on the network, as this requires a large communication overhead.

To maintain low measurement overhead and mitigate an adversary's ability to influence the measurement process, we propose an opportunistic, decentralized scheme. Each client maintains their own list of congested nodes, which they use to inform their path selection.

A decentralized scheme requires the clients to perform the measurements and only use their information for themselves. Distributed schemes were proposed several times to solve different issues, such as to eliminate the need for a global witness in mix networks [10] or Tor's previous reliance on self-reported bandwidth [2]. The central directory does not need to (and in fact should not) act on the information the clients gather for themselves. Gaming is less likely since the measurements are based on real experience of the clients. Another advantage is that clients can obtain data whenever they need it without having to ask the directory authorities. A potential downside is that clients cannot do nearly the same amount of measurements we would expect a centralized measurement scheme to perform.

4.2 Bandwidth or Latency

As previously mentioned, Tor uses a bandwidth-weighted path selection algorithm informed by active bandwidth measurements of Tor nodes conducted by Tor's bandwidth authorities.

Latency has also been studied as a metric for path selection. Sherr et al. [23] propose a link-based path selection algorithm to make use of latency and similar link-based properties. We hypothesize that latency can also be a good indicator of congestion, as values above the expected distance-based latency demonstrate that a packet has waited for some time in a queue. Currently, Tor takes into account the circuit build time adaptively and disregards circuits that take too long to build [4]. This approach, however, cannot identify circuits that may become congested after they are constructed, and the client will not learn to avoid attempting to build circuits over nodes that are consistently congested. We present an improved method to use latency for circuit construction.

4.3 Circuit or Node Tests

Measurements can be performed on nodes or circuits. In either case, the ultimate goal is to obtain latency measurements on each individual node of the Tor network.

Directly testing nodes provides more accurate measurements. However, if one-hop circuits are built to measure the latency of nodes, the node can easily guess that they are being tested, which may lead to biased results. Another way is to build three-hop circuits, but to only measure the round trip time to the first hop and ignore the other two hops. However, measurements done this way will create a large burden on Tor nodes; the details are discussed in Section 5.2. Since we also want a decentralized scheme where all clients perform testing, we must be careful not to create a large burden on the Tor network, so we discard the possibility of measuring nodes directly.

Measurements on circuits have a strong advantage: they can be done opportunistically—which is to say, the client only tests the circuits built automatically by the original Tor path selection algorithm. This minimizes communication overhead, as we describe in Section 5.2. Furthermore, the client can quickly avoid circuits that are temporarily congested. Because of the inherent practical advantages of measuring circuits over measuring nodes, we will focus on measuring circuits in this paper. Some inference must be done to attribute latency correctly to each node, as we describe in Section 5.4.

5. LATENCY

In this section, we describe our latency model, our approach to measuring latency, and a technique for identifying congestion-related delays and attributing those delays to individual nodes along a measured circuit.

5.1 Latency Model

We first define a latency model for nodes. Our latency measurements on the Tor network suggest that latency measurements on a node can be cleanly divided into a *non-congested component* and *congestion time*. When a node is not congested, the latency can be attributed to propagation delays, which are nearly constant. Non-congested measurements can therefore be defined as measurements that are very close to the minimum of all measurements on the same node. For many nodes, this accounts for most of the data. When a node is congested, an amount of congestion time is added to the round trip time before it can reach the client. This amount of time is frequently much larger than the non-congested measurements.

We define the following terms with respect to a node:

t_{min}	the minimum round trip time
t_c	the congestion time
t	the round trip time
γ	a smoothing constant

t_{min} is the minimum round trip time for all measurements of round trip time of a node. It is a constant, assuming all measurements are done from the same client; the chief component of t_{min} is the propagation delay. We define the congestion time $t_c = t - t_{min}$. By removing t_{min} from the round trip time, we isolate the congestion time. γ is a small constant added to the measurements to allow for quick reactions to transient congestion, as detailed further in Section 5.4. Thus, the actual congestion time is $t_c = t - t_{min} + \gamma$.

This latency model assumes that nodes are individually measured using one-hop circuits. However, for the various reasons discussed in Section 4, we perform only circuit-level measurements from which we deduce each constituent node's t_c value, as described in the remainder of this section.

5.2 Measuring the Latency

We next discuss how circuit-level latency is measured by the client. This measurement should fulfill the following criteria:

1. It should be lightweight. There should be little burden on the network even if all of Tor's estimated 300,000 clients use this scheme simultaneously.
2. It should be indistinguishable from non-measurement traffic. Otherwise, it may be possible for malicious routers to influence the measurements.
3. It should exclude the destination server's latency. We do not want the packet to exit from the last hop.

Due to the first criterion, we discard the possibility of measuring the latency of nodes directly from the client, although it would be convenient. This is because a new connection between a client and a node requires some bootstrapping, including a new execution of the TLS protocol, which we found can take up to one second. This could create an immense burden on some nodes, especially ones that do not wish to or cannot offer too much of their resources.

Measurements of a circuit can be done in two ways: we can *actively* probe the circuit, or we can perform measurements *opportunistically* so as not to create a burden on Tor.

Active probing. One way to actively probe a circuit is to send a `BEGIN` cell telling the exit node to open a connection to TCP port 25 (SMTP) on some end server. The principle is to violate the exit policy of the exit node; port 25 is by default banned to prevent anonymous spamming. An error message is sent back by the exit node without contacting any end server. This ensures that the round trip time measured is indeed the round trip time of the circuit, without extra end-server latency. Another method to measure the round trip time is to tell the exit node to connect to *localhost*, which the exit node will refuse to. This scheme was used by Panchenko, Richter, and Rache [20]. These two methods share the same principle: by forcing the exit node to return an error message, we get the round trip time to the last node. However, a potential disadvantage of this approach is that the exit node can identify the measurement probes and attempt to influence the results. In our experiments, we use a technique that is conceptually similar: we use circuit build cells to measure the circuit latency. To extend the circuit to the final exit router, the client sends a circuit `EXTEND` cell through the entry guard and the middle router. The middle router sends a `CREATE` cell to the exit router, which after performing public-key cryptography replies with a `CREATED` cell back through the circuit to the client. The time spent performing public-key cryptography can be considered a small constant, which will later be factored out of the latency measurement.

Opportunistic probing. If only active probing is used, our scheme might add too much measurement traffic into the Tor network, particularly if all clients were to perform such measurements frequently. Thus, we also use an opportunistic approach that leverages Tor's end-to-end control cells as the measurement apparatus. The stream-level and circuit-level `SENDME` cells are sent end-to-end in response to every 50 and 100 `DATA` cells, respectively. In addition, `BEGIN` and `CONNECTED` cells are sent whenever a new exit TCP stream is established, which for web browsing clients can happen several times per web page visited. As long as the client is using the circuit, we can obtain a number of measurements without any additional burden on the Tor network.

Note that if we want the exit node to immediately send a message back without spending time contacting a server, then the measurement is slightly skewed towards the first two nodes. To be precise, the message has to travel through each link among the client and the nodes twice, and it has to wait in the queue (if any) of the first two nodes twice, but it only needs to wait in the queue of the exit node once (see Figure 2).

Overhead. The opportunistic measurements have no overhead, as they leverage existing end-to-end control cells. However, it might be desirable to augment the opportunistic measurements with additional active measurements, at some communication cost. We can obtain one congestion time entry for each member of a circuit by sending just one packet (512 bytes). Suppose the client actively probes each circuit they build 5 times over 10 minutes. This will add an average of 5 B/s of usage to each node. If 300,000 users use this scheme together, they will add a total of 4.5 MB/s of usage to Tor. This is currently around 0.5% of the total bandwidth offered by all Tor nodes, so our scheme will only add a small load to the Tor network. As will be seen in Section 6, this is enough for our scheme; the other measurements needed can be done opportunistically.

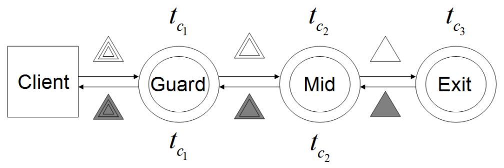


Figure 2: A breakdown of congestion in testing. The test packet (colorless triangle) is sent to the exit node and a response packet (colored triangle) is returned without going through any destination server.

5.3 Isolating Circuit Congestion

When we obtain a measurement on the circuit, we want to highlight the congestion times t_{c_1}, t_{c_2}, t_{c_3} for each node along the circuit. But first, it is necessary to separate the circuit's propagation delay from the delay due to congestion. We next describe this process.

For one round trip of the entire circuit, the time T can be dissected this way:

$$T = T_{min} + (T_c - \gamma)$$
$$T_c = 2t_{c_1} + 2t_{c_2} + t_{c_3}$$

where T_{min} is an estimate of the circuit's end-to-end propagation delay and T_c is the circuit's delay due to congestion (γ is a small constant described in Section 5.4). The difference between T_{min} and T_c is that T_{min} should be constant for the same circuit, while T_c varies depending on the extent of the circuit's congestion. In addition, T_c only includes the last node once as in our tests, as our probes do not exit through the final node. In our tests, we find that the congestion term T_c is sometimes zero, but it is often non-zero.

For each measurement of T in this circuit, we save it in a list $\{T_1, T_2, ..., T_k\}$, and after all measurements of the circuit are done, we take the lowest measurement, and let this be T_{min}. Note that the number of measurements taken per circuit should be large to ensure that T_{min} converges to the circuit's actual end-to-end propagation delay.[1] Through experimental analysis, we find that T_{min} can be correctly determined within an error of 0.05 s with 80% probability by using only five measurements—in the case that T_{min} is not correctly identified, the circuit being considered is likely to be heavily congested.

The i^{th} measurement of congestion time ($0 \le i < k$) is given by:

$$T_{c_i} = T_i - T_{min} + \gamma$$

[1] T_{min} can also be intelligently estimated using other methods. For instance, the King method [14] can be used to approximate the pairwise network latency between any two Tor nodes without probing either of the routers directly, so an approximation of T_{min} can be acquired with no additional burden on Tor or the possibility of congestion.

In Figure 2, we summarize how a single end-to-end circuit round trip time measurement is conducted and where the congestion occurs.

5.4 Attributing Circuit Congestion to Nodes

Now that we have isolated the delay due to congestion from the circuit's total delay, we need to attribute the congestion delay to the circuit's constituent nodes. The simplest solution is to take the congestion time of a circuit, divide this by the number of nodes in the circuit, and record this as an entry to the congestion time of each node equally. While simple, this is unfair to especially fast or slow nodes, as these nodes will have entries that are averaged out by other nodes. A fairer solution would be to assign a congestion time value to each node in proportion to its previously estimated congestion time.

Our approach. Each client maintains a congestion list of all known relays paired with a number L of congestion times for each relay. This list is updated as new measurements are taken. When we refer to the observed congestion time of a node, we refer to the mean of all L measurements of congestion times of the same node.

We now define how this list is updated with new measurements. Consider a three-hop circuit. Suppose the observed congestion times of nodes in this circuit r_1, r_2, r_3 are respectively $t_{c_1}, t_{c_2}, t_{c_3}$. The entry guard is r_1, the middle router is r_2, and the exit router is r_3. Next, suppose the round trip time taken for some cell to return across this circuit is T; then the total circuit's congestion time is $T_c = T - T_{min} + \gamma$. For r_1 and r_2, we assign the following congestion time:

$$t_{c_i} \leftarrow T_c \cdot \frac{2t_{c_i}}{2t_{c_1} + 2t_{c_2} + t_{c_3}}$$

Here $i = 1$ for node r_1 and $i = 2$ for node r_2. For r_3, we assign the following congestion time:

$$t_{c_3} \leftarrow T_c \cdot \frac{t_{c_3}}{2t_{c_1} + 2t_{c_2} + t_{c_3}}$$

Note that these formulas can be fully extended to a case with more hops, wherein only the exit node will have a multiplier of 1 and other nodes will have a multiplier of 2. In case there are no observed congestion times for the node yet (as the node has not been measured before), we take $t_{c_1} = t_{c_2} = t_{c_3}$ for the first set of measurements for the new node, even if other nodes in the same circuit have been measured before.

Details. A technical issue emerges when a node becomes congested after a long period of being non-congested. In this scenario, the observed congestion time would be very close to zero and the algorithm would not respond fast enough to assign a high congestion time to this node. This is where the term γ comes into play. By ensuring that the minimum observed congestion time is at least γ, we can guarantee that even nodes without a history of congestion will not be immune to blame when congestion occurs in a circuit with such a node. We empirically find $\gamma = 0.02\,\mathrm{s}$ to be a good value; this is not large enough to cover the differential between congested and non-congested nodes, yet it ensures that convergence will not take too long.

When a new observed congestion time has been assigned to a node, the node's mean observed congestion time should be updated. A simple solution would be to record the average, and the number of data points we had recorded, in order to get an overall average. However, we find that each data entry should not have permanent effect on the node's average, because the conditions of the nodes sometimes change, which affects their congestion status, and the observed congestion time should be updated without being significantly affected by the original congestion time (wherein the node was in a different situation). Therefore, we maintain a list of congestion time measurements for each node, L; when this amount of data has been recorded, we push out old data whenever new data comes in. If L is chosen to be large, then the client's preference for a node will not change as quickly, and vice versa. We find that $L = 20$ offers a good balance—a mistaken or temporary situation will not affect the average much, while $L = 20$ still allows convergence within a reasonable amount of measurements.

Advantages. This method has an advantage over the naive method. Assuming that each node has a constant congestion time, if all observed congestion times are already accurate, no further observations will change any of the predictions; i.e. there is a point of convergence (see Appendix A for more on the convergence of our method). This method is also fair, in the sense that if certain nodes are already known to cause congestion, then high congestion time values taken for circuits containing these nodes will be properly attributed to these nodes (and not to other low-congestion nodes in the same circuit).

6. MAIN ALGORITHM

Congestion can be either short term (e.g., a file sharer decides to use a certain node for their activities), or long term (e.g., a node's bandwidth is consistently overestimated or its flags and exit policy are too attractive). For short-term congestion, we want to provide an instant response to switch to other circuits. For long-term congestion, we propose a path selection algorithm that takes congestion time into account.

6.1 Instant Response

We provide two ways in which clients can perform instant on-the-spot responses to high congestion times in a circuit.

Choosing the best pre-built circuits. Tor automatically attempts to maintain several pre-built circuits so that circuit construction time will not affect the user's experience. Two circuits are built for each port being used in the past hour (a circuit can count for multiple ports). Only one of those circuits is chosen as the next circuit when the user's circuit times out or breaks. A reasonable scheme, therefore, is to test all of those circuits before choosing which to use. As stated above, those tests can be done quickly and with minimal overhead. We can also increase the number of pre-built circuits to allow the clients to choose the best among more circuits, increasing the strength of our scheme. We suggest that five active probing measurements per pre-built circuit is sufficient to choose the best, as we observe in our experiments (in Section 7) that congestion along a circuit typically manifests itself within a small number of measurements.

Switching to another circuit. While using the circuit, a client may continue to measure the circuit and obtain congestion times. This can be done with no overhead to the Tor network by opportunistically leveraging Tor's stream-level and circuit-level `SENDME` cells, or the stream `BEGIN` and `CONNECTED` cell pairs (as described in Section 5.2). This gives us the round trip time T, from which we can follow the procedure given in Section 5.3 to isolate the nodes' congestion time. If the observed congestion time is "large", the client should stop using this circuit and choose another circuit instead. Here, "large" can be defined several ways—it can be adaptive so that the client drops circuits at a predefined frequency. It can also be defined as a constant congestion time threshold (e.g., one second), so that clients will not choose to use circuits with a high congestion value.

Both instant response schemes offers a more flexible response to congestion than the original path selection algorithm, as they incorporate real-time congestion information into the selection decision. We note that clients may retain their guard nodes for a long time (currently 30 days). It might be tempting to drop a chronically congested guard node and choose another one. However, doing so introduces a trivial attack on guards and is not recommended.

6.2 Path Selection

In addition to an instant response, we also want a long-term response where clients can selectively *avoid* certain nodes if they often receive poor service from them. This can be helpful when there are nodes with poorly estimated bandwidth, when bulk downloaders customize their clients to use only specific relays, and when there are other unexpected load balancing issues that have not been resolved. Our congestion-aware path selection works as follows.

Congestion list. Each client will keep a list of all routers, each of which will be recorded with a list of their measured congestion times. The list of measured values is of size L; when new data comes in, old data is pushed out. We first note that the size of the list is not too large. If $L = 20$ entries of observed congestion times are recorded at 2 bytes (which can be per 1 ms up to 65535 ms) each, each router will take up 40 bytes, and so 2,000 routers will be 80 kilobytes—this is manageable even if there are 20,000 routers. This is also an advantage of our node-based approach over link-based selection; we can record observed congestion-related latencies for nodes in $O(n)$ space, while it would take $O(n^2)$ space to do so on pair-wise links, for n nodes.[2]

Router selection. Our scheme is designed to be built atop the current path selection algorithm in this way: when we wish to extend a circuit by one node, we pick a few routers from the list according to the original scheme (e.g., 10 nodes), and then choose one of them in negative correlation to their observed congestion times. Observed congestion times are obtained by recording both the active and opportunistic measurements done for the instant response schemes. For example, suppose that node r's observed congestion time is t_{c_r}. We define a base constant $\alpha > 0$, and use it to obtain the probability of selecting the node r for a circuit:

$$P(C_r) \propto \frac{1}{\alpha + t_{c_r}}$$

where C_r is the event of node r being chosen. α is a constant that prevents very low congestion nodes from dominating the path selection algorithm.

The effect of this scheme on the user's experience and the Tor network itself depends in part on the choice of the constant α. A smaller α will impact the load balancing more as nodes with less observed congestion become more likely to be chosen. We suggest choosing α to be approximately the mean congestion time of all observed nodes (we found this generally to be between 0.25 s and 0.5 s), so that the least congested nodes will receive about twice as many clients as they did before. This should be sufficient to ameliorate long-term congestion, because the non-congested nodes (for which t_c is close to zero) will receive up to twice as much traffic as they once did until they have no extra bandwidth to spare.

Advantages. Our approach is simple and efficient. The computation required to support latency measurement and path selection is very simple: it is nothing more than a few arithmetic operations on small numbers. Furthermore, this scheme requires no further infrastructure to support, and it is incrementally deployable in the sense that any client who chooses to use this scheme can immediately do so. In fact, even if only a portion of clients use this scheme, the performance of the remaining clients might also improve, as our proposal aims to reduce the overall congestion in the network.

7. EXPERIMENTS

We designed a number of experiments that aim to validate our assertions about latency and congestion in Tor. For all experiments, we use the Tor controller protocol [27] to connect to Tor. We use the final pair of circuit construction cells to measure the round trip time of a circuit (as described in Section 5.2). In the remainder of this section, we show that congestion is a property of Tor nodes, present experiments and results that quantify the persistence of the circuit-level latency estimates, explore the relationship between a node's consensus bandwidth and its observed congestion, and detail the performance improvements offered by congestion-aware router selection.

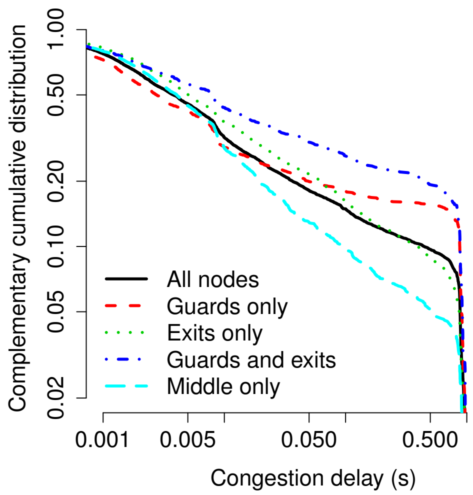
7.1 Node Congestion

We first seek to demonstrate that congestion is a property of Tor routers. For 72 hours in August 2011, we collected round-trip time data for all Tor routers that can be used on a circuit by measuring the time to construct one-hop circuits. For each node, we subtracted the node's minimum measurement (e.g., the propagation delay) to isolate the congestion delays t_c. For each measured node, on receipt of the cell building response cell, we immediately send another circuit building request cell, for a total of five times per node.

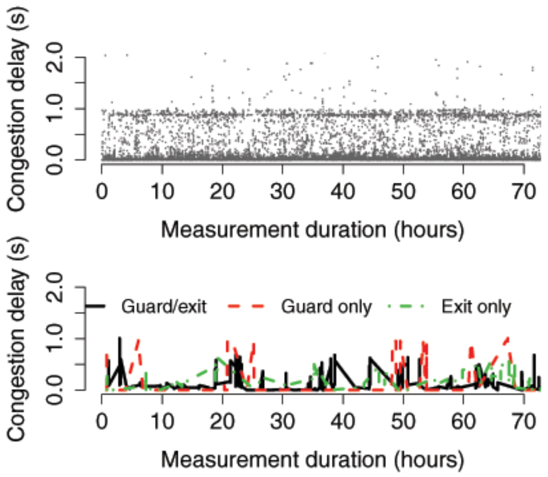
Figure 3(a) shows the distribution of congestion delays for entry guards, exits, guard/exits, and middle-only nodes. The median congestion delay is minimal (3–5 ms) across all node types; however, the tails of the distributions tell a different story. For the most congested ten percent of the measurements, nodes marked as both guard and exit experience congestion delays greater than 866 ms, and guard-only nodes have at least 836 ms of congestion delay. Exit-only and middle-only nodes tend to be the least congested. Guard nodes may be the most congested because the stability and bandwidth criteria for the guard flag is too high. Relaxing the requirements for the guard flag would enable some middle-only nodes to become guards, reducing congestion among guards. overall congestion.

Figure 3(b) shows congestion delays over the duration of our measurements for all routers (top) and for three repre-

[2]However, we note that it might be possible to extend the virtual coordinate system approach proposed by Sherr et al. [23] to capture congestion-related latencies in addition to propagation delay between nodes in $O(n)$ space.



(a) CCDFs of congestion delay for nodes with different flags (log-log scale)



(b) Congestion delays over time for all routers measured (top) and the variability for three high-bandwidth routers (bottom)

Figure 3: Analysis of congestion delays

sentative high-bandwidth (10 MiB/s) routers (bottom). We note that these delays tend to be low. However, there exists noticeable variability regardless of a node's flags or bandwidth, and many of the delays are close to one second (Figure 3(a) also illustrates these one second delays where the CCDF lines become vertical). These one second delays are the result of Tor's token bucket rate limiting with a once-per-second token refilling policy.[3] Note that our measurements are able to remotely identify these one second delays because there is no explicit pause between receipt of a circuit building response and the transmission of the subsequent measurement cell. These delays indicate that nodes are being asked to forward more traffic than they can handle, resulting in congestion. Thus, we conclude that congestion is a property of the Tor router itself, motivating the need for clients to consider congestion when selecting nodes for a circuit.

To verify that these one second delays are in fact the result of Tor's token bucket rate limiting mechanism and not some other unknown phenomenon, we conducted a simple experiment with a high-bandwidth (1 Gb/s) Tor router deployed on the live network. This experiment consists of two phases. First, we configured the router to use no explicit rate limiting. In such a configuration, the node does not exhaust its supply of tokens and, thus, it does not need to explicitly wait for its tokens to be refilled. Next, we configured the router to an average incoming and outgoing bandwidth 50 KB/s using Tor's `BandwidthRate` option. We also configured the maximum token bucket size to be the same value using the `BandwidthBurst` option. For each phase of the experiment, we measured congestion delay using the procedure described previously.

[3]Increasing the frequency with which the tokens are refilled may reduce or eliminate these one second delays. This design change is currently being discussed [28].

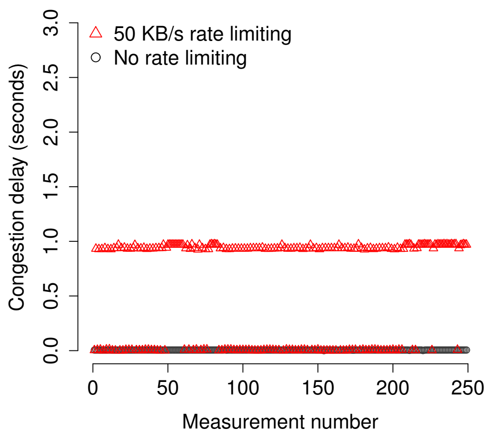


Figure 4: Congestion delays due to token exhaustion during a controlled experiment on the live Tor network

Figure 4 shows the congestion delays for the live Tor router with rate limiting is enabled (red triangles) and disabled (black circles). We see that when rate limiting is enabled, the router quickly exhausts its allotted tokens and must wait for very close to one second before the tokens are replenished. When rate limiting is disabled, we observe no such delays. Thus, we conclude that delays that approach one second are the direct result of token exhaustion and Tor's per-second token refilling policy.

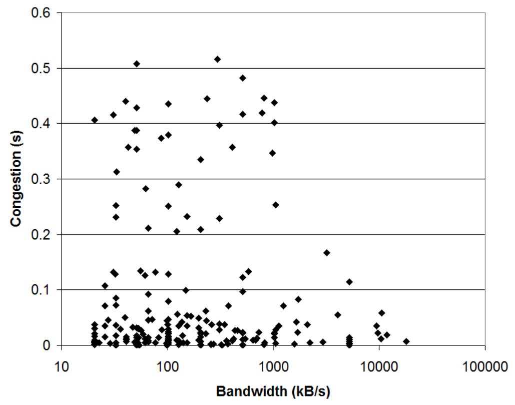


Figure 5: The congestion time of 294 nodes plotted against the bandwidth.

7.2 Latency Persistence

We are next interested in determining whether measurements of congestion persist over a long period of time, or if congestion tends to be short-lived and transient. A lack of persistence would suggest that congestion comes in bursts, and that an instant response (like choosing another circuit) would help resolve this issue.

A circuit over the same three routers was built 10,000 times, at 10 second intervals, for one day. For this circuit, we found that T_{min}, the minimum round trip time, was 0.96 s and the mean congestion time, T_c, was 0.41 s. The mean difference between any two randomly chosen latency measurements was 0.56 s. Compared to this, the mean difference between any two adjacent round trip time measurements was only 0.21 s. The probability that the difference between one data entry and the next one being less than 0.05 s was 54.2%. This suggests that circuit congestion is persistent in the short term and that switching to another circuit will help if a few measurements indicate that it is congested.

7.3 Congestion and Bandwidth

To investigate the possible relationship between a node's bandwidth and its congestion, we obtained the bandwidth consensus of all nodes in the Tor network on June 11, 2011 and looked at the top 87.5% of the list, which are candidates for the middle node. The mean bandwidth was around 750 kB/s while the median was 150 kB/s. This consensus did not contain nodes below 20 kB/s. This suggests that the bandwidth limit for most nodes can easily be reached by only a few users on a fast connection.

The latency of 294 randomly chosen nodes was tested, each 30 times over 2.5 s intervals. Each node's congestion time t_c was obtained from those 30 tests. Figure 5 shows a scatter plot for congestion time versus the log of the consensus bandwidth. As can be seen from the plot, there is very little correlation between the two.[4] The correlation coefficient (Pearson's r) between the log of the bandwidth and the congestion time was -0.00842 for this particular experiment. This suggests that simply looking at the bandwidth is not sufficient to resolve load balancing issues, thus motivating our congestion-aware path selection scheme.

[4]Dhungel et al. observe a low correlation between consensus bandwidth and *overall* router delay [5].

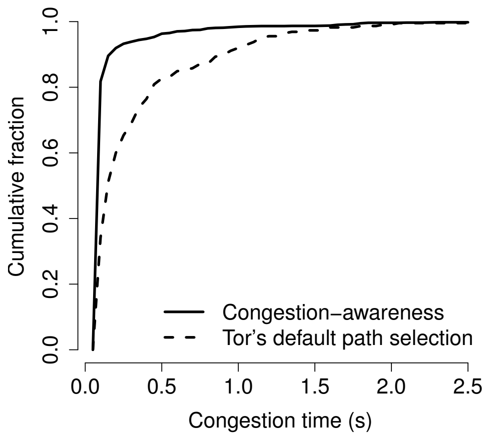


Figure 6: The client's congestion time when using Tor's current path selection algorithm compared with our congestion-aware path selection.

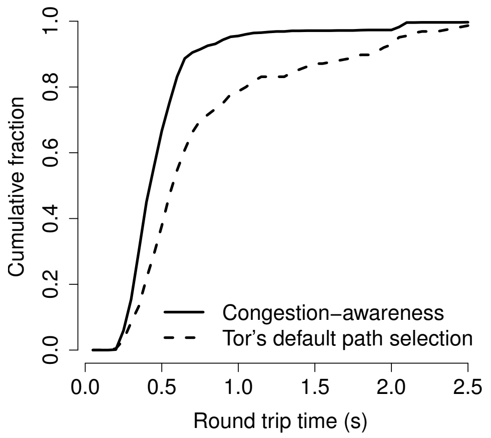


Figure 7: A client's round trip times when using Tor's current path selection algorithm compared with our congestion-aware path selection.

This experiment implies that it is not only the faster or the slower nodes that are congested. In the same experiment, we also noted whether or not a given node that was being tested was a guard, exit, a guard and an exit, or none of the above. We found that if a node was neither a guard nor an exit, its congestion time was significantly lower, while guards had the most congestion time for our experiment. This suggests that the long-term path selection algorithm may help.

7.4 Improvements of Our Scheme

We next present experiments that seek to quantify clients' latency improvements when using our scheme. Experiments are performed on both the instant response and long-term path selection components.

In these experiments, an unmodified Tor client used the current path selection algorithm in Tor. At the same time, a modified client uses the instant response components of our scheme (from Section 6.1) to determine which circuit it should use. The original client builds 225 circuits and measures each one precisely 30 times to obtain round trip times. The modified client determines which circuits it should use based on the same data.

Choosing the best pre-built circuits. We first tested how much of an improvement we would see if the client simply tested each circuit five times when building them preemptively and chose the one with the lowest congestion. For simplicity we assumed that the client always had three circuits to choose from. The original client tested each of its circuits 30 times, and took the mean of the congestion times as its experience with the circuit. The modified client chose the best among every three circuits to use by only looking at the first five measurements; after choosing the best out of three, all 30 measurements of that circuit are revealed to the modified client and it is taken as its experience of the circuit. Without the scheme, the mean circuit congestion time was about 0.276 s. With the scheme, it was about 0.119 s. We find that this large improvement was because most circuits were non-congested, except a minority where the congestion time was very high. Those circuits also clearly exhibited congestion in the first five measurements.

Switching to another circuit. We next tested how much of an improvement we would get if the client switches to a better circuit when the current one becomes too congested. This time both the original client and the modified client can see all measurements. The modified client dropped a circuit if the last five measurements had a mean of more than 0.5 s of congestion; 73 of the 225 circuits were eventually dropped. This sufficed to improve the mean congestion experienced from 0.276 s to 0.137 s.

Finally, we combined the two instant response schemes. 75 of the 225 circuits were chosen using the first scheme, and later 11 of the 75 circuits chosen were eventually dropped using the second scheme. We achieved a mean congestion time of 0.077 s, compared to the original 0.276 s. The total round trip time was reduced from a mean of 0.737 s to 0.448 s. Figure 6 shows the distribution of congestion times for the client when it used our improvements compared to the original selection scheme, and Figure 7 shows the distribution of round trip time for the same comparison.

One may worry that these schemes will add too much overhead because they drop existing circuits and build new ones. With the first scheme we are not dropping extra circuits; instead of choosing one of the available circuits arbitrarily, we measure the circuits a few times to choose the least congested one. With the second scheme, in our experiment we found that we would need to build about 26% more circuits, which is a relatively modest increase.[5]

Long-term path selection. We evaluate the long-term path selection algorithm as follows. We ran a client that builds many circuits over the entire Tor network using the original path selection scheme. In total 13,458 circuits were built, for which the round-trip time was obtained 5 times each. One-third of the circuit build times were used as testing data; the rest were used in training the client to learn the estimated congestion times for each relay. By using the long-term path selection scheme, we observed a decrease in the mean congestion time for this experiment from 0.41 s to 0.37 s over the testing data. The improvement is not as large as in the instant response schemes, because the long-term path selection scheme tackles more persistent factors which adversely affect node performance rather than short-term bursts of congestion.

[5]Circuit building cells are much rarer than data transfer cells; further, the Tor Project is working to decrease the computation required for circuit building by a factor of 4 [17].

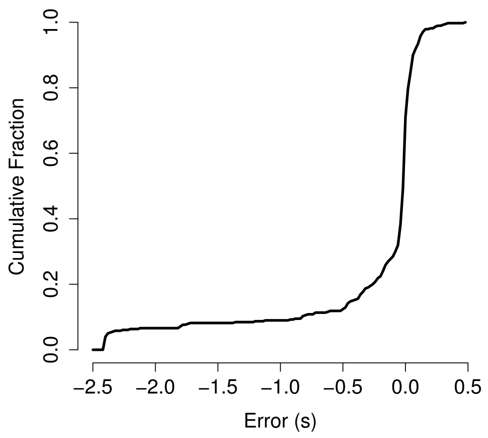


Figure 8: Distribution of errors when learning individual node congestion over a large number of trials

The long-term path selection scheme offers an improvement nonetheless because it is capable of deducing the congestion time of individual nodes while only measuring the congestion times of random circuits, allowing it to choose uncongested nodes. We performed a total of 379 trials where we compared deduced congestion (by building three-hop circuits) to directly measured congestion (by building one-hop circuits). Figure 8 shows the distribution of errors. We found that nearly 90% of the errors were within -0.5 s to 0.5 s, and 65% of the errors were within -0.1 s to 0.1 s. The scheme very rarely overestimated node congestion, but sometimes underestimated it, as shown by the large number of negative errors. The mean error was therefore -0.2 s. This may be because high congestion is somewhat random in nature, so the scheme is less accurate in predicting the extent of a node's congestion while only given a previous record.

We next investigate whether or not our scheme is capable of deducing the congestion time of individual nodes while only measuring the congestion times of random circuits over 300 nodes. This would imply that our long-term path selection scheme is able to identify and choose uncongested nodes.

A single trial consists of the following steps:

1. Build a circuit of three random routers among the 300 nodes.

2. Measure the circuit's round-trip time five times.

3. Obtain the circuit congestion by subtracting each value of circuit round-trip time by the minimum of the five values.

4. Update the recorded congestion of the nodes in the congestion list.

5. Test each node individually five times to get the node congestion directly.

6. Take the error of each node as the difference between the mean estimated congestion time and the directly measured node congestion.

We performed a total of 379 trials; Figure 8 shows the distribution of errors. We found that nearly 90% of the errors were within -0.5 s to 0.5 s, and 65% of the errors were within -0.1 s to 0.1 s. We found that the scheme very rarely assigns high congestion values to nodes which are consistently non-congested. This is essential to the accuracy of the scheme. However, there were quite a number of large negative errors. This was where the directly measured congestion value for the node was very high, but the deduced congestion value (using our scheme) for the node was not nearly high enough. The mean error was therefore -0.2 s. This may be because high congestion is random in nature, so the scheme is less accurate in predicting the exact value of congestion for a congested node. Nevertheless, it would assign a high congestion value to nodes which have a history of creating high congestion values.

8. DISCUSSION

We next discuss a variety of open issues related to our proposed congestion-aware path selection.

8.1 Client Differentiation

While Tor is not geared towards serving clients using BitTorrent, a large proportion of Tor's traffic is consumed by a small number of these clients [18]. These clients can easily congest most of the nodes in the Tor network, causing congestion problems for everyone using these nodes. A fundamental issue is that the path selection algorithm has to assume that all users are approximately the same; however, the large difference between a bulk downloaders and a web-browsing client means that this assumption will cause congestion when the file-sharing clients are assigned to low-bandwidth nodes. This difference has been a motivation for work to improve Tor's ability to handle congestion [1].

Our scheme helps ameliorate the problem of congestion introduced by bulk downloaders in two ways. First, bulk downloaders will not switch circuits because their streams are in active use, but web-browsing clients can do so. This allows web-browsing clients to make an instant response and switch to another circuit if their current one is too congested. Suppose Alice, a web-browsing client, wants to switch from a congested circuit to a circuit with non-congested nodes. The switch is unlikely to harm the users of another circuit as Alice only demands a small amount of bandwidth, while Alice herself benefits from having a better circuit. Second, clients will test each circuit they build before deciding which one to use. If certain circuits contain nodes that are already congested by bulk downloaders, then other clients will avoid them. We note that web-browsing clients generally should not mind using circuits with lower bandwidth, so long as they receive enough bandwidth to transport their relatively light traffic.

8.2 Anonymity

The list of latencies stored on a user's computer may compromise anonymity if divulged. If the list of latencies for all users is known to an attacker, he can perform an attack by only controlling the exit node, and using the lists to probabilistically guess who is connecting by checking the frequency of connections; this will give him some amount of information. Our scheme, however, gives no reason to directly divulge the list of latencies at any point.

We note that an entry guard can deduce the users' preferences for entry guards and middle nodes, but it is unable to tell which exit nodes they prefer, while the exit node is unable to tell which clients prefer them. To obtain the client's preference for exit node, the attacker requires a circuit in which the entry guard and exit node are colluding. Such a case already results in anonymity being compromised without our scheme. Our scheme does, however, cause a more lasting effect for such an attack.

8.3 Security of Our Scheme

We next consider the security of the scheme. We consider a particular attack called the *smearing attack*. The attacker first uses all of his available bandwidth to create malicious nodes. These malicious nodes attempt to give the appearance of congestion by artificially delaying cells. If a client measures a circuit containing some innocuous nodes and these malicious nodes, the innocuous nodes will be "smeared" with high observed congestion times. After a certain amount of time, these malicious nodes will be observed to have a very high congestion as well, so the smearing becomes less effective. Once a malicious node becomes rarely selected, it is taken down, and a new one is created in order to maintain the attack. This attack is continued until all innocuous nodes can no longer be smeared further (this is bounded by the amount of bandwidth available to the attacker). After all nodes are maximally smeared, the attacker can stop the attack and enjoy a larger control of the network for a while, as his nodes will now seem more attractive.

Note that nodes in Tor are less likely to be chosen if they do not have the "stable" and "fast" flags. The stable flag is a barrier for malicious nodes, as it requires the node to demonstrate high stability before they can be effective. In particular, Tor by default does not choose nodes without the "stable" flag for entry guards. We neglect this barrier in the following analysis, giving more power to the attacker.

A parameter of the attack is C, which indicates for how long each malicious node will attempt to smear other nodes before being replaced. If $C = 5$, for example, the attacker will attempt to keep each malicious node up for as long as it takes to smear other nodes five times for each client measuring the nodes, then take it down and replace it with another node. We take t_c as the mean performance of the nodes (including the malicious node) and t_{max} as the maximum time the client performing the latency measurement will wait for before timing out. The estimation is done by running a simulation with the simplifying assumption that all nodes can be selected in all positions. We did not experiment with this attack on the live Tor network.

Figure 9 shows how much bandwidth the malicious nodes must be given at any point in time in order to affect the measurements of the congestion time of the non-malicious nodes. As can be seen, an attacker can indeed smear other nodes and gain an advantage by coming up with fresh, non-smeared nodes. We note that the instant response given in Section 6.1 (switching circuits immediately) provided by

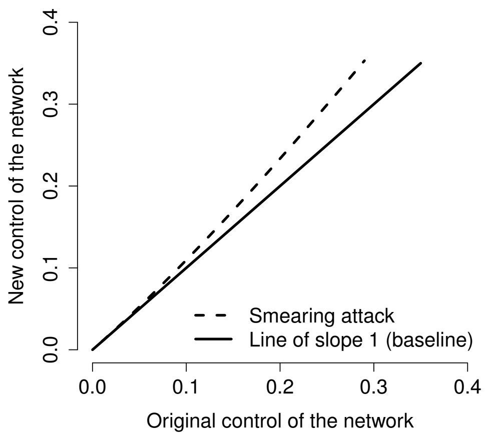


Figure 9: An estimation of how much control an attacker could gain if they used their nodes to smear other clients. The numbers given are a measurement of the fraction of the network the attacker controls. The diamond points show how much control the attacker can gain. The line is drawn as a baseline comparison when no attack is performed. We chose $t_{max} = 5000\,\mathbf{ms}$, $t_c = 500\,\mathbf{ms}$, $L = 20$, $C = 30$.

the congestion measurements makes it harder for smearing attacks to be performed, as a smeared circuit will be dropped quickly. We also note that the advantage gained is temporary—when the adversary stops performing the attack and uses all their bandwidth to acquire control of the network, clients will start measuring the other nodes' non-smeared congestion times as well, so their observed congestion times will slowly return to their non-smeared levels.

9. FUTURE WORK

In this section, we identify a variety of avenues for future work.

Centralized testing. In this paper, we investigated a scheme under which clients build their paths using their own active and opportunistic measurements of latency. This has several advantages over a centralized scheme: latency measurements are readily available to clients during their normal usage, and this allows clients to respond quickly to temporary bursts in latency (e.g., a file sharer decides to download a large file through Tor for 15 minutes). We also recognize that centralized testing has several advantages—if all clients use the same latency measurements done by a centralized tester, then there will be no loss of anonymity even if an attacker can guess their congestion list. Furthermore, a much smaller number of measurements needs to be done. As future work, we wish to investigate whether we can improve our scheme by adding or replacing certain components with centralized testing.

Defenses and attacks. Because our scheme allows clients to choose non-congested circuits, these clients may be inherently protected from congestion attacks [12,19] that attempt to identify a client by clogging their circuit. However, other attacks, such as those that rely on accurate network latency estimates [15], might become easier if congestion is reduced. Ultimately, our congestion-aware path selection algorithm's primary objectives are to reduce congestion, improve load balancing, and improve clients' quality of service, not necessarily to provide any defense mechanisms. A complete investigation of the effects of decreased congestion on prior attacks is future work.

Large-scale evaluation. Our experiments show that a single client can expect to experience improved performance when using our proposed congestion-aware path selection. We suspect that clients who use Tor's standard path selection algorithm might also experience improved performance due to an overall reduction in network congestion. However, it is possible that our proposed instant response methods that aim to help clients avoid congested circuits in real time may produce short-term oscillation where clients initially switch to non-congested nodes, but after a large number of clients switch to the same set of non-congested nodes, these nodes may become congested. As future work, we plan to investigate the potential performance benefits and other effects when this path selection technique is deployed at scale through whole-network experiments in a testbed [3].

10. CONCLUSION

Many different metrics for path selection in Tor have been proposed, some of which consider the use of latency. However, previous work treats latency as a property of a link and focuses on the delays that occur primarily due to propagation. We assume a different approach: we identify the importance of latency as an indicator of a node's congestion. To reduce congestion, improve load balancing and, ultimately, improve clients' quality of service, we propose an improved path selection algorithm based on inferred congestion information that biases path selection toward non-congested nodes.

We expect our proposal to improve the experience of any client who uses this scheme, in addition to those who do not as long as the number of clients who use the scheme is substantial; in addition, this scheme requires no further infrastructure, a very small overhead, and can be incrementally deployed as only clients need to participate.

Acknowledgements. This work was funded in part by NSERC, MITACS, and The Tor Project. We also thank Jean-Charles Grégoire, Angèle Hamel, Ryan Henry, Femi Olumofin, and Rob Smits for their valuable suggestions.

11. REFERENCES

[1] AlSabah, M., Bauer, K., Goldberg, I., Grunwald, D., McCoy, D., Savage, S., and Voelker, G. M. DefenestraTor: Throwing out windows in Tor. In *PETS* (2011).

[2] Bauer, K., McCoy, D., Grunwald, D., Kohno, T., and Sicker, D. Low-resource routing attacks against Tor. In *WPES* (2007).

[3] Bauer, K., Sherr, M., McCoy, D., and Grunwald, D. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test* (August 2011).

[4] Chen, F., and Perry, M. Improving Tor path selection. `https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:`

/proposals/151-path-selection-improvements.txt, July 2008.

[5] Dhungel, P., Steiner, M., Rimac, I., Hilt, V., and Ross, K. W. Waiting for anonymity: Understanding delays in the Tor overlay. In *Peer-to-Peer Computing* (2010), IEEE, pp. 1–4.

[6] Dingledine, R., and Mathewson, N. Tor Protocol Specification. https://gitweb.torproject.org/tor.git/blob_plain/HEAD:/doc/spec/tor-spec.txt. Accessed August 2011.

[7] Dingledine, R., and Mathewson, N. Anonymity loves company: Usability and the network effect. In *Workshop on the Economics of Information Security* (June 2006).

[8] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *USENIX Security* (2004).

[9] Dingledine, R., and Murdoch, S. Performance improvements on Tor or, why Tor is slow and what we're going to do about it. http://www.torproject.org/press/presskit/2009-03-11-performance.pdf, March 2009.

[10] Dingledine, R., and Syverson, P. Reliable MIX Cascade Networks through Reputation. In *Proceedings of Financial Cryptography (FC '02)* (March 2002), M. Blaze, Ed., Springer-Verlag, LNCS 2357.

[11] Edman, M., and Syverson, P. F. AS-awareness in Tor path selection. In *Proceedings of CCS* (2009), pp. 380–389.

[12] Evans, N., Dingledine, R., and Grothoff, C. A practical congestion attack on Tor using long paths. In *Proceedings of the 18th USENIX Security Symposium* (August 2009).

[13] Goldschlag, D. M., Reed, M. G., and Syverson, P. F. Hiding routing information. In *Proceedings of Information Hiding: First International Workshop* (May 1996), Springer-Verlag, LNCS 1174.

[14] Gummadi, K. P., Saroiu, S., and Gribble, S. D. King: Estimating latency between arbitrary Internet end hosts. *SIGCOMM Comput. Commun. Rev. 32*, 3 (2002).

[15] Hopper, N., Vasserman, E. Y., and Chan-Tin, E. How much anonymity does network latency leak? In *CCS* (2007).

[16] Loesing, K. Measuring the Tor network: Evaluation of client requests to the directories. *Tor Project Technical Report* (2009).

[17] Mathewson, N. New paper by Goldberg, Stebila, and Ostaoglu with proposed circuit handshake. https://lists.torproject.org/pipermail/tor-dev/2011-May/002641.html. Accessed June 2011.

[18] McCoy, D., Bauer, K., Grunwald, D., Kohno, T., and Sicker, D. Shining light in dark places: Understanding the Tor network. In *PETS* (2008).

[19] Murdoch, S. J., and Danezis, G. Low-cost traffic analysis of Tor. In *Proceedings of IEEE Symposium on Security and Privacy* (2005).

[20] Panchenko, A., and Renner, J. Path selection metrics for performance-improved onion routing. In *Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 114–120.

[21] Perry, M. Torflow: Tor network analysis. HotPETS, 2009.

[22] Reardon, J., and Goldberg, I. Improving Tor using a TCP-over-DTLS tunnel. In *USENIX Security* (2009).

[23] Sherr, M., Blaze, M., and Loo, B. T. Scalable link-based relay selection for anonymous routing. In *PETS* (2009).

[24] Sherr, M., Mao, A., Marczak, W. R., Zhou, W., Loo, B. T., and Blaze, M. A3: An Extensible Platform for Application-Aware Anonymity. In *17th Annual Network and Distributed System Security Symposium (NDSS)* (February 2010).

[25] Snader, R., and Borisov, N. A tune-up for Tor: Improving security and performance in the Tor network. In *NDSS* (2008).

[26] Snader, R., and Borisov, N. Eigenspeed: Secure peer-to-peer bandwidth evaluation. In *Proceedings of the 8th International Workshop on Peer-to-Peer Systems (IPTPS)* (2009).

[27] The Tor Project. Tor control protocol. https://gitweb.torproject.org/torspec.git/blob/HEAD:/control-spec.txt.

[28] Tschorsch, F., and Scheuermann, B. Proposal 182: Credit bucket. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/proposals/182-creditbucket.txt. Accessed August 2011.

[29] Wright, M. K., Adler, M., Levine, B. N., and Shields, C. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur. 7*, 4 (2004), 489–522.

[30] Zhang, N., Yu, W., Fu, X., and Das, S. K. gPath: A game-theoretic path selection algorithm to protect Tor's anonymity. In *GameSec* (Berlin, Heidelberg, 2010), Springer-Verlag, pp. 58–71.

APPENDIX

A. CONVERGENCE

Measurements of latency done under our scheme are expected to converge in some sense. To illustrate certain properties of our scheme, suppose that each node in the network behaves entirely consistently in terms of their observed congestion time—each measurement of the node will tell us the one exact congestion time of the node without error. First of all, it can be seen that if our scheme has already obtained the correct congestion time of every node, then new measurements will change nothing. This is a simple but important property which the naive scheme (always assuming each node is responsible for exactly $\frac{1}{N}$ of the congestion of the circuit, where N is the number of hops) does not have.

Furthermore, it can be easily seen that there is no stable assignment of values except that which assigns to each node its true congestion time. Consider the congestion time of nodes as unknowns to be solved in a set of equations. Testing the round trip time of any three nodes allows us to obtain a linear equation on the three unknowns representing the congestion time of the three nodes. Since any three nodes can be chosen, there are enough linear equations to solve for all congestion times, and so there is no other solution.

Now that we can see there is no stable assignment except that which is desired, it is necessary to see that convergence will eventually be reached. One way to see this is to observe the change in the total difference:

$$\Delta = \sum |t_{r,observed} - t_{r,real}|$$

It can be seen that if any three nodes are measured, their total will converge towards the true total, and since no other nodes besides those three are affected, this necessarily means that Δ will become smaller (or does not change) with each new measurement. Δ will not converge to a constant greater than zero, as there is always some test that can reveal false values; thus, eventually Δ will converge to zero.

It is also of concern how quickly convergence may be reached. One way to increase convergence time is to decrease L, the size of the list of recorded congestion times kept for each node. We note that, without the assumption that nodes behave consistently, it may not always be optimal to decrease L. One may choose to keep a longer congestion list so that temporary situations can be forgiven.