# An Improved Algorithm for Tor Circuit Scheduling

Can Tang
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
c24tang@cs.uwaterloo.ca

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

17 April 2010

**Abstract**

Tor is a popular anonymity-preserving network, consisting of routers run by volunteers all around the world. It protects Internet users' privacy by relaying their network traffic through a series of routers, thus concealing the linkage between the sender and the recipient. Despite the advantage of Tor's anonymizing capabilities, it also brings extra latency, which discourages more users from joining the network.

One of the factors that causes the latency lies in Tor's circuit scheduling algorithm, which allows busy circuits (those with continuous traffic) to crowd out bursty circuits (those with short bursts of traffic). In this work, we propose and implement a more advanced scheduling algorithm which treats circuits differently, based on their recent activity. In this way, bursty circuits such as those used for web browsing can gain higher priority over busy ones such as used for bulk transfer; the performance for most activities over Tor is improved, while minimal overhead is incurred. Our algorithm has been incorporated into the latest build of Tor.

## 1 Introduction

Tor [3] is a distributed anonymizing network that provides privacy for its users. The network is formed by volunteers all around the world running Onion Routers (ORs). The ORs publish their related information, such as bandwidth and exit policies, to a set of centralized servers called directory authorities. These directory authorities will negotiate with each other and reach a consensus about the ORs. An end user runs an Onion Proxy (OP) locally to tunnel application requests through Tor; the OP downloads the consensus from the directory authorities, randomly picks a set of ORs, choosing them weighted by their reported bandwidths, and builds circuits through them. Then application traffic is relayed through the circuit, using layered encryption. In this way, a single OR only sees its previous OR/OP and next OR/OP, but not the actual sender-recipient relationship, and the privacy of the user at the transport level is preserved.

The anonymity provided by Tor relies on the size of the anonymity set. Currently, there are around 1500 ORs [1], and an estimated quarter million Tor users. Tor's rapid expansion period ended by the end of 2007 (in terms of the number of ORs) [10]. After that, Tor entered a relatively stable stage: the number of ORs joining the Tor network roughly equaled the number leaving it. One of the obstacles for Tor's further expansion is its performance issues.

There are many causes for Tor's performance issues. Dingledine and Murdoch [4] identified several main reasons that Tor is slow. One of the causes is that bursty circuits do not co-exist with busy circuits very well. When multiple circuits are sharing a single connection between two ORs, busy circuits such as those for bulk transfer will greatly degrade the performance of the bursty ones, such as those for web browsing. Although this effect is inevitable due to the limited bandwidth resources, we want to allocate the resources more fairly; that is, to give higher priority to the circuits with low throughput or short bursts of traffic, and make them faster. This is reasonable from the application point of view as well: circuits for web browsing or instant messaging are usually sensitive to delays, while those for bulk transfer are usually not.

Our approach is to calculate the exponentially weighted moving average (EWMA) for the number of cells sent in each circuit. When selecting the circuit to process, we always pick one with the lowest EWMA value, and flush the cells in that circuit. Newly created circuits and bursty circuits will usually have a low EWMA value, and so they will be prioritized.

In Section 2, we introduce the incentives and rationality of our proposal, briefly demonstrate the mechanism of Tor circuits, and propose our improvement. In Section 3, we show the results of some experiments under different scenarios, with analysis of the results and the overhead. Section 4 examines the effect of different parameter values on the performance of our system, and Section 5 explores the performance improvements on Tor's hidden services. Section 6 relates the compatibility of our algorithm with the existing Tor network. Section 7 provides some related work on improving Tor's performance. We discuss possible future work in Section 8, and conclude in Section 9.

# 2 Prioritizing Interactive Circuits

In this section, we propose our improvement on Tor's circuit scheduling algorithm. First, we state the incentives for our proposal; next, we describe the process by which Tor ORs select which data is to be transmitted. Finally, we describe our improvement over the existing system.

## 2.1 Incentives

Most users of Tor experience its performance issues: it incurs much higher latency than direct connections. Although the multi-hop architecture inevitably brings extra latency, the experienced latency is higher than this effect can explain.

One factor for Tor's bad performance is its limited capacity: the ORs are run by volunteers, usually on consumer computers, with limited bandwidth. Within this limited capacity, there are abuse issues observed in the Tor network. According to McCoy et al. [12], a small number of BitTorrent connections consume a very high proportion of Tor bandwidth. These connections make Tor unusable for many potential users.

Tor provides anonymity by mixing a specific user into a crowd of users; therefore, the degree of anonymity Tor provides depends on the number of users. Higher latency will discourage more users from joining the network. Hence, the performance issues do not only affect user experience, but also degrade Tor's security properties.

The uses of Tor can be divided into interactive streams and non-interactive streams. Interactive streams include web browsing, instant messaging, SSH, and telnet, while non-interactive streams include bulk file transfer such as FTP and BitTorrent. Interactive streams are usually delay-sensitive: users click on a link to a webpage and wait, expecting it to appear on the screen in seconds, while non-interactive streams are not: BitTorrent users expect the file download to be completed in hours or even days; they can tolerate higher delays. We aim to improve Tor's performance by making ORs process interactive streams first. This will give interactive users (the majority of Tor users) a better experience, and will make little difference for non-interactive users.

## 2.2 How Tor's Circuits Work

As described in Section 1, a client runs an OP locally. The OP randomly selects several ORs to form a path, then builds a Tor circuit through this path. Each circuit is used by only one client (OP). Between each pair of ORs on the path, a Tor connection is established. If multiple circuits use the same two ORs in sequence, they will share a single connection between the two ORs. Based on the current number of users and ORs, and the ORs' capacities, we can infer that usually connections will be shared by multiple circuits, especially for the connections between high-bandwidth ORs. That is, each OR will have simultaneous connections to a number of other ORs (but only one connection to any *given* OR), and each connection will transport data for a number of circuits.

All Tor traffic is relayed in fixed-size (512-byte) cells. A cell consists of a header field and a payload. The header contains metadata about the cell, such as the circuit identifier. When a cell arrives at an OR, the OR decrypts the cell, extracts the information necessary from the header, and then pushes the cell into the output queue for its circuit (the *circuit queue*). The time cost of this process is negligible [17] as a fraction of the overall time for a cell to be processed by an OR. Circuits with non-empty circuit queues are called *active* circuits.

Each connection has an *output buffer*; data written to that buffer will be transmitted to the next OR in FIFO order. As multiple circuits generally share a single connection, the cells in the circuit queues must be multiplexed into the output buffer.

When there is room in the output buffer, the OR will select an active circuit, and move some cells from its circuit queue to the output buffer. If all of the cells are moved, the circuit is marked inactive.

The contribution of this work is to change how Tor decides from which active circuit to select cells. The previous algorithm was simply to select active circuits in round-robin fashion. We show that by making a more judicious selec-

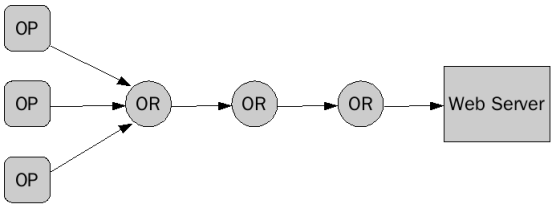


Figure 1: The Tor circuit structure for Experiment 1

tion, the performance of Tor for interactive circuits can be notably improved, while minimally affecting performance for circuits performing bulk data transfer, which tend to be delay insensitive in any event.

In order to prioritize interactive circuits, we need to decide, for example, which circuits are using HTTP, and which circuits are using BitTorrent. Unfortunately, we cannot determine the application protocol by looking directly at the content of the cells, since all of the cells are encrypted except for at the exit OR. On the other hand, circuits using HTTP may also perform bulk transfer, and we want to deprioritize them as well. Thus, the amount of traffic sent recently should be an appropriate criterion on which to base our scheduling decision. We should mention that we do not want to block BitTorrent or similar applications by blocking the port number at exit ORs, since Tor is intended to be application neutral; additionally, port number blocking can be easily circumvented by those file sharing applications.

### 2.3 Circuit Selection based on EWMA

We want to have a metric for "how many cells a circuit has sent recently", and base the circuit selection decision on this metric. The metric needs to represent an average value over a period of time of the activity for a circuit, and also needs to decay over time, since we do not want the activity from long ago to have a large impact on the current decision for a circuit. EWMA seems to be a good choice for this metric.

First, we assign each circuit a cell count value, representing the average number of cells sent recently. Every time we wish to flush some cells to the connection's output buffer, we calculate the decayed cell count value for each circuit, based on the EWMA equation that supports irregularly-spaced observations:

$$A_{t+\Delta t} = A_t \cdot 0.5^{\frac{\Delta t}{H}}$$

where $A_{t+\Delta t}$ is the new cell count value, $A_t$ is the old cell count value, $\Delta t$ is the elapsed time since the last observation and $H$ is the "half life" parameter; that is, $H$ determines the interval after which the previous average is reduced by half.

After the calculation, the OR picks the circuit with the smallest cell count value, and flushes that circuit's cell queue to the output buffer of the connection; the cell count value is updated correspondingly:

$$A'_{t+\Delta t} = A_{t+\Delta t} + C_{t,t+\Delta t}$$

where $C_{t,t+\Delta t}$ is the number of cells sent in the interval $(t, t + \Delta t]$.

The value of $H$, as well as a switch for turning our whole algorithm on or off, can be set in the Tor configuration file.

As the equation shows, circuits with low or bursty traffic will have low cell counts. These circuits are likely to be those which are still in their creation phase, as well as circuits for web browsing or instant messaging, which are exactly the circuits we want to prioritize. For each OR, cells in interactive circuits will wait for less time in the circuit queue than without prioritization. The performance of interactive circuits, on the whole, will be improved. As we show later, the performance of circuits which are deprioritized suffers only minimally.

## 3 Experiments and Results

### 3.1 End-to-End Timing Analysis

The experiments in this section were performed on PlanetLab. PlanetLab [16] is a research network consisting of nodes distributed globally, much like Tor. We selected a set of five nodes from PlanetLab, and ran a private Tor

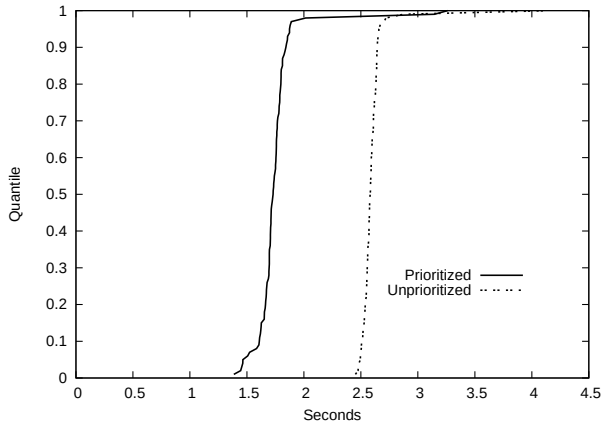


Figure 2: CDF for time cost of downloading a small file for unprioritized and prioritized Tor

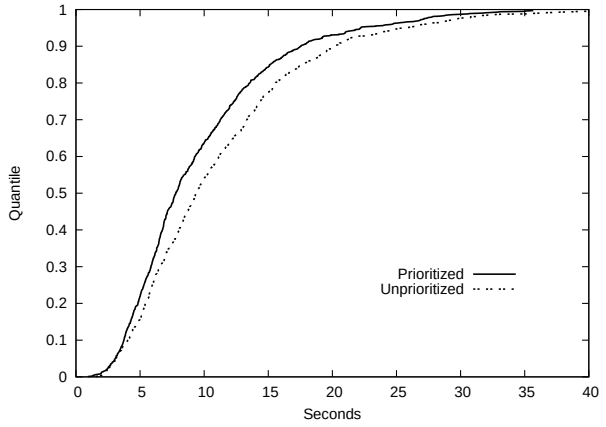


Figure 3: CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under traffic simulation

network on them. A typical Tor circuit consists of three ORs. The OR directly connecting to an OP, the OR directly connecting to the server, and the OR in the middle are called the entry OR, exit OR, and middle OR, respectively. In our experiment, we picked two nodes as directory authorities and three nodes as ORs; we ran our modified Tor on the three ORs.

Experiment 1 was designed to measure the time cost of downloading a small file (simulating web browsing), while there are competing bulk transfer circuits. According to [22], the average web page size (including HTML, CSS and images) grew from 93KB to over 312KB, from 2003 to 2008. We picked 300KB as the file size we use for our experiment. The file is hosted on the same machine as the exit OR, in order to eliminate the variance introduced by the connection between the web server and the exit OR.

We configured three local clients, who select the same route, as shown in Figure 1. Two clients were performing bulk transfer. We tried to download the 300KB file using the other client, and recorded the elapsed time. We used $H = 66$ for the algorithm, which means that after every 66 seconds, the old cell count will decay by a factor of 0.5. We performed 100 downloads for both unprioritized Tor (the stock Tor) and our prioritized version of Tor. The cumulative distribution function (CDF) of the results is shown in Figure 2.

In Experiment 1, we observed an average of 32% decrease in the time to download a small file while there are simultaneous competing bulk transfers — the median time decreased from 2.60 seconds to 1.75 seconds.

During the experiment, when switching between unprioritized and prioritized Tor, we switched on the algorithm for all three ORs at the same time. But does prioritizing one of the ORs contribute the most to the effect, or do all three of them contribute equally to the improvement? In order to investigate this issue, we selectively turned on the algorithm on individual ORs, and repeated the experiment to see the effect. We discovered that switching on the

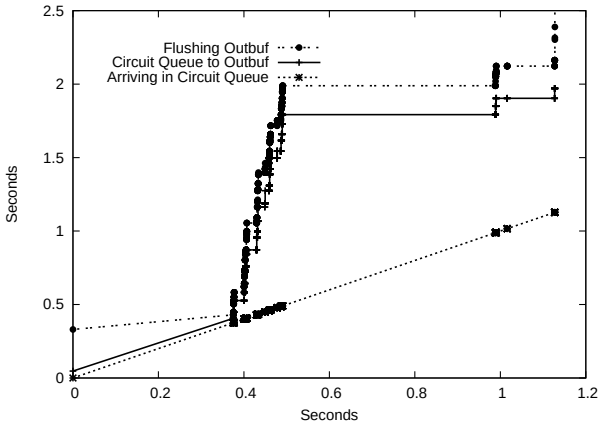


Figure 4: Time points for a cell's life cycle in the middle OR, unprioritized

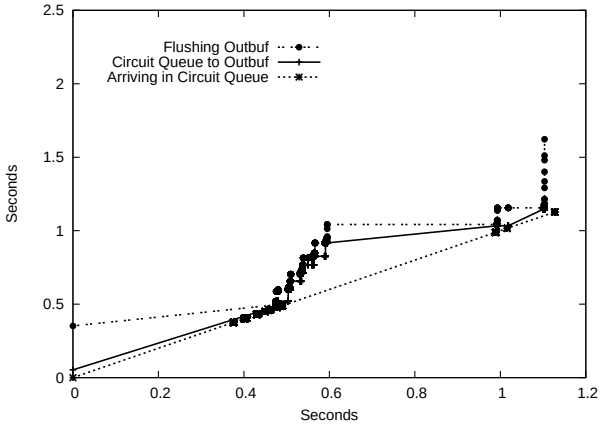


Figure 5: Time points for a cell's life cycle in the middle OR, prioritized

algorithm on the entry and exit ORs does not have noticeable effects, while by turning on the algorithm on the middle OR, we immediately see the improvement which is close to the overall improvement we obtained. The reason is that for the nodes we chose on PlanetLab, the link between the middle OR and the entry OR is slower than the link between the middle OR and the exit OR, so that cells (being sent from the server to the client) accumulated at the middle OR. This indicates that the effectiveness of the algorithm highly depends on the conditions of individual ORs; when an OR's output link is congested, cells will wait in the circuit queue for longer periods, and our algorithm will make a more noticeable difference.

For the real Tor network, the traffic distribution is quite different from the previous experiment. McCoy et al. [12] identified the exit traffic protocol distribution of Tor in 2008. HTTP accounted for 92.45% of all connections, and 57.97% of the total bytes sent; SSL accounted for 4.06% of all connections, and 1.55% of the total bytes sent; BitTorrent accounted for 3.33% of all connections, and 40.20% of the total bytes sent. Other protocols such as Instant Messaging, E-Mail, FTP, and Telnet accounted for less than 1% of both connections and bytes sent. From the cited results, we can conclude that BitTorrent consumed a disproportionately large amount of bandwidth compared to other protocols. This justifies our intention of giving interactive streams higher priority. On the other hand, in reality, the ratio of busy circuits is not as high as in our previous experiment; i.e., there are not as many low-priority circuits. To see how much improvement we can get, we created a traffic simulator which randomly generates network traffic according to the connection-to-throughput ratio in the above statistics.

In Experiment 2, we ran the traffic simulator on multiple clients, to simulate the Tor network. We downloaded a small file and recorded the time cost. There were 1000 attempts for both unprioritized and prioritized Tor. The results are shown in Figure 3.

Due to the large variance in the network conditions, the results have large variance in both unprioritized and

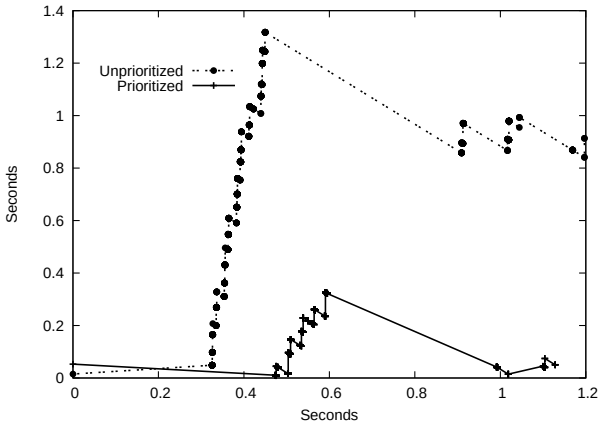


Figure 6: Time cells wait in the circuit queue in the middle OR

prioritized Tor (ranging from 1 second to almost 1 minute). However, from the CDF graph, we can still find 10–20% improvement at each quantile.

## 3.2 Fine-grained Timing Analysis

In Experiment 3, we examine the life cycle of a cell in an OR, check how much time it spends at each stage, and see where we have improved.

Remember that when a cell reaches an OR, it enters the circuit queue to which it belongs, and waits to be flushed to the output buffer of the connection. When the output buffer is empty or has flushed some cells, one circuit will be selected to flush its cells to the output buffer. Then the cells wait in the output buffer until they are flushed to the socket.

The testbed setting is similar to Experiment 1. We use libspe [18] to record the time points related to the cells: when cells enter the circuit queue, when they are moved from the circuit queue to the output buffer of the connection, and when they leave the output buffer. We record those time points of cells in the middle OR (as we mentioned, using our algorithm on the middle OR showed the most improvement in our experiment), for both unprioritized and prioritized Tor.

Figures 4 and 5 show the results for a single download of a small file. The x-axis indicates the time a cell enters the circuit queue, and the y-axis indicates the time the cell enters the circuit queue (the straight diagonal line), the time the cell moved from the circuit queue to the output buffer, and the time the cell is flushed from the output buffer. Thus the gap between the two lower lines indicates the time a cell spends in a circuit queue, while the gap between the upper two lines indicates the time a cell spends in the output buffer, waiting to be written to the socket. We can see that a cell spends most of its life cycle in the circuit queue waiting to be flushed. The prioritized Tor greatly reduced this duration.

Next, we recorded the time cells spent in the circuit queue. We see significant improvement in these durations; in fact, the average duration spent in the circuit queue decreased from 653 milliseconds to 115 milliseconds. The actual improvement in latency is even greater than the improvement in the average duration. Figure 6 shows the amount of time cells wait in the circuit queue, which is just the difference between the lower two lines in each of Figures 4 and 5. We can see that as cells arrive, more cells are queuing up within the OR, making the latency higher. Our improvement is more noticeable when the cells are queuing up: the decrease in latency is as high as 1 second, in this case. This explains the results we obtained in Experiment 1.

## 3.3 Experiments on Live Tor Nodes

In order to test the effectiveness of our algorithm on the real Tor network, we need to perform the experiments on running Tor nodes, with real Tor traffic going through them. In this section, we describe our live experiments and results, and analyze the limitations.

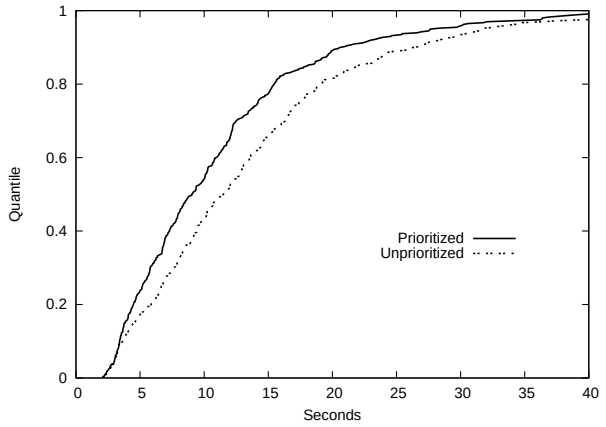


Figure 7: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network

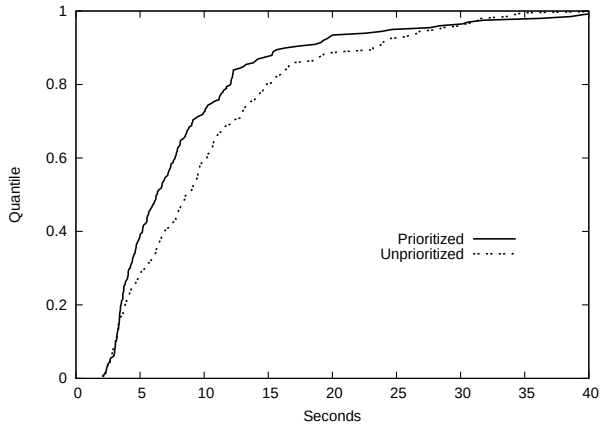


Figure 8: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during afternoons ET

### 3.3.1 Bandwidth Requirement for Live Tor Nodes

The effectiveness of our algorithm depends on whether multiple circuits are sharing a single connection between a pair of ORs. According to [1], as of March 2010, there were around 1500 ORs in Tor network, with the highest reported bandwidth as much as 15 MB/s.

We downloaded the descriptors of all the ORs, and calculated the sum of the advertised bandwidth. The total bandwidth was 440 MB/s. We assume that there are 250,000 simultaneous Tor users (circuits), who select ORs randomly using their bandwidth as weights. Assume that we have a pair of ORs, each having bandwidth $BW$ MB/s. The expected number of circuits between this pair of ORs is then $250000(\frac{BW}{440})^2 = 1.29(BW)^2$. In order to see several circuits between the pair to make the prioritization effective, we require $BW \approx 2$ MB/s at least.

### 3.3.2 Testbed Setup

Initially, we planned to run three ORs on selected PlanetLab nodes, and let them join the Tor network. However, among the PlanetLab nodes, few could reach such a high bandwidth requirement. In fact, most nodes have bandwidth lower than 100 KB/s. With these nodes, it is hardly possible to attract multiple circuits within the connection.

We found one PlanetLab node at Princeton University that has bandwidth as high as 1 MB/s. However, the daily usage on the node is limited to 10 GB. Since the startup process of an OR requires several hours to complete (including publishing descriptors, computing consensus by directory authorities, advertising bandwidth and re-advertising

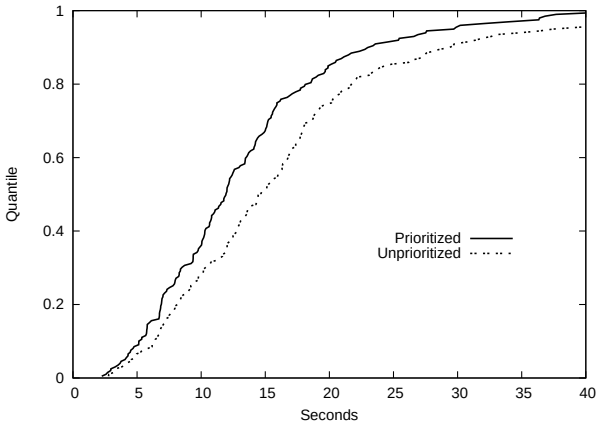


Figure 9: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during midnights ET

bandwidth after attracting traffic), the throughput limit makes it impossible to perform any tests.

Instead, we used `gurgle.cs.uwaterloo.ca` (nickname: `planetgurgle`), a machine located at the University of Waterloo, which has a comparably high bandwidth of 3 MB/s. We also selected two existing ORs with high bandwidth: `blutmagie` with 10 MB/s, and `coldbotTorHosting1` with 10 MB/s. We used `blutmagie` as the entry OR, `planetgurgle` as the middle OR, and `coldbotTorHosting1` as the exit OR. In this way, the estimated number of circuits between `planetgurgle` and `coldbotTorHosting1` is $1.29(3 \times 10) \approx 39$, which is more than enough for our purpose.

The disadvantage compared to using PlanetLab nodes is that we could only control one OR in our circuit (the middle OR), thus the results will only show the improvement of prioritizing at one hop. However, as we will see later, the results are still noticeably in our favour.

The target file to fetch is hosted in the University of Waterloo, with a size of 87 KB. We did not introduce artificial bulk transfer traffic, so this experiment is representative of normal user experience. We used webfetch [2] to fetch the file using our configured circuit. There is a 20-second break between every successive fetch.

### 3.3.3 Experimental results

We performed the experiment during different periods in a day, executing 250 downloads with `planetgurgle` configured in each of the unprioritized and prioritized modes. The CDF of the results are shown in Figure 7.

It is interesting to note that this graph is quite similar to our traffic simulation tests on PlanetLab (Figure 3). With our prioritization algorithm enabled, the median time decreased from 11.49 seconds to 9.04 seconds.

One phenomenon we observed is that during different periods in a day, the test results differ markedly. The latencies are much higher in the afternoons ET (Eastern Time; the timezone of New York and Toronto) than around midnight ET. This may indicate that most Tor users come from the other half of the globe. Indeed, according to [9], only a small percentage of Tor users come from North America and South America combined.

In order to better observe the effectiveness of our patch, we divided the results into two groups: a "fast" group which were performed during afternoons and a "slow" group which were performed around midnight. The CDFs are shown in Figure 8 and Figure 9.

The figures indicate that under various network conditions, our algorithm makes observable improvement for bursty HTTP downstream traffic.

## 3.4 Effects on Bulk Transfer

Our new scheduling algorithm should not degrade the performance of bulk transfer to any noticeable extent. By Little's Law [7], $L = \lambda W$, where $L$ is the queue length (average number of cells in the queue), $\lambda$ is the arrival rate (long term throughput), and $W$ is the average time a cell spends in the queue (latency). Our algorithm only changes the order

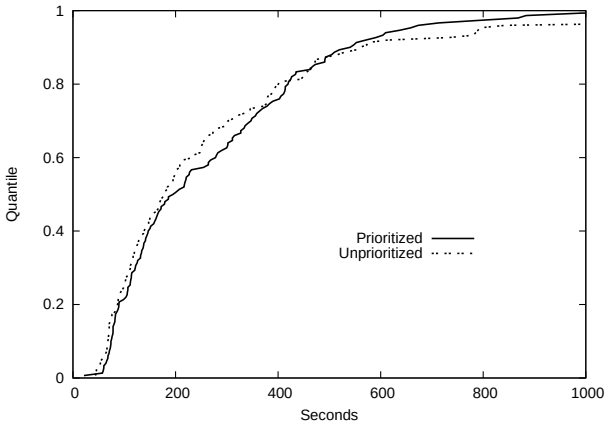


Figure 10: CDF for time cost of bulk download using unprioritized and prioritized middle OR, in the real Tor network

of cells within a queue, thus does not change $L$ and $W$. Under the assumption that the buffers are large enough, the long-term throughput for bulk transfer should stay the same.

We also experimentally compared the performance of bulk transfer circuits on our live node. We used our Tor client to continuously fetch a 4 MB file hosted at the University of Waterloo. There were 150 trials for each of unprioritized and prioritized Tor. the results are shown in Figure 10.

From the CDF, we see very little effect of our algorithm to the bulk transfer. The average time cost is 296 seconds for unprioritized Tor, with standard deviation 361 seconds, and 270 seconds for prioritized Tor, with standard deviation 213 seconds. There is no statistically significant difference in the performance of unprioritized and prioritized Tor. Additionally, as we mentioned earlier, bulk transfer usually takes at least several minutes to complete, and users doing such transfers will have more tolerance of the increased delay if they ever do notice it.

## 3.5 Overhead

The overhead for our scheduling algorithm mainly lies in the computation of EWMA values, and the cost of acquiring the current system time. This requires extra CPU resources compared to the stock Tor. However, most Tor nodes are limited by the network capacity, not by their CPUs [17]. Our scheduling algorithm will not degrade the performance of those nodes.

However, the Tor maintainers reported to us that the busiest Tor nodes are in fact CPU-limited. For these nodes, we need to make sure that prioritized Tor does not perform worse than the stock Tor. When we completed the first version of the scheduling algorithm, and performed local experiments (with very high network capacity, unlike the Tor network), we found that it did in fact perform worse than the stock Tor, and used a high ratio of CPU resources. We identified that the frequent calls to `gettimeofday` accounted for the majority of the time so consumed. When each cell is flushed, we need to know the system time in order to correctly update the EWMA values, but system calls at this frequency become a burden to the CPU.

We observed that during each write event (when cells are flushed into the output buffer), the differences in time of flushing for each of the cells in that write are usually in the handful of microseconds range. Since we do not need precision to the microsecond level for the calculation of EWMA values, we modified the algorithm so that we only acquire the system time at the beginning of the write event handling process, and store the time value. The subsequent acquisitions of system time use the cached value instead. In this manner, we reduced the total number of `gettimeofday` system call by two orders of magnitude.

After the optimization, we again performed a local experiment to find the overhead. The experiment was performed on a commodity desktop computer, with AMD Athlon 64 X2 Dual Core 5600+ processor, 3.2GB memory, Ubuntu 8.04 operating system. We ran all the Tor nodes, including three ORs, two directory authorities, and two OPs locally. The web server was also hosted locally. This setup maximally stresses the CPU. We performed the experiment in which the two clients simultaneously fetch a 5MB file from the web server. There were 200 trials for both unprioritized Tor and prioritized Tor. During the experiment, the CPU usage went up to 100%, so that our nodes were indeed CPU-limited.

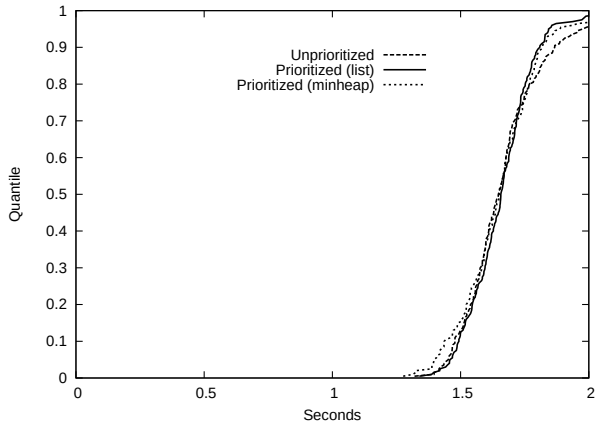


Figure 11: CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under CPU-limited scenario

The CDF of the results is shown in Figure 11 ("Unprioritized" and "Prioritized (list)").

The results showed that the average time cost is 1.66 seconds for unprioritized Tor, with standard deviation 0.15 seconds, and 1.69 seconds for prioritized Tor, with standard deviation 0.24 seconds. There is no statistically significant difference in the performance of unprioritized and prioritized Tor in the local experiment, which means that even in the rare scenario that the Tor node is CPU-limited, the scheduling algorithm will not make it significantly slower.

Nonetheless, Nick Mathewson of The Tor Project has optimized [11] the implementation of our algorithm to further reduce the overhead. Instead of using a circular linked list, the active circuits are kept in a minheap-based priority queue. Further, the computation of the EWMA cell counts is optimized; noting that only the *relative*, and not the *absolute* EWMA cell counts matter (since their purpose is just to pick the active circuit with the lowest cell count), an arbitrary reference time point is picked, and cell counts are computed relative to that time. That is, rather than decaying every circuit's stored cell count value by a factor of $0.5^{\frac{\Delta t}{H}}$, and then adding 1 for every cell sent, a single new value $V$ (representing the current weight of one cell, as compared to a cell sent at the reference time point) is updated by multiplying it by $0.5^{-\frac{\Delta t}{H}} = 2^{\frac{\Delta t}{H}}$. This value $V$ is added to a circuit's cell count for each cell it sends. This saves the traversal of the list of all circuits (not just active circuits) and a computation of the decayed cell count for each one, which in our version occurs every time a cell is sent. Every so often, the cell counts for all circuits can be renormalized by dividing them all by $V$, and resetting $V$ to 1. This has the effect of updating the reference time point to the current time. It is important to note that this optimized computation maintains exactly the same circuit-selection behaviour as our our unoptimized implementation.

Mathewson's patch not only reduces the load of the EWMA computation, but also reduces the time cost of picking the highest-priority circuit when many circuits co-exist in a connection. We performed an experiment to test the overhead with this patch in the CPU-limited scenario. The results are also shown in Figure 11 ("Prioritized (minheap)"). The average time cost is 1.65 seconds, with standard deviation 0.16 seconds, also not a statistically significant difference. This version of our algorithm has been committed to the the latest version of Tor.

# 4 Fine Tuning of the Algorithm

The parameter $H$ in our algorithm determines how far back in time we want to look to calculate the cell counts for the circuits. This time horizon should be chosen to distinguish bursty HTTP circuits from circuits for continuous data transfer. For the PlanetLab experiments, the value of the parameters do not matter much, since the goal is to make HTTP circuits always have higher priority over bulk transfer circuits, and any parameters within a reasonable range will satisfy.

However, for the Tor nodes on the live network, the conditions are more complex. HTTP circuits are not only competing with bulk transfer circuits, they are competing with each other as well. The parameters should meet the requirement of distinguishing the two sets in practical scenarios. On the other hand, the standards may differ from OR to OR, however, depending on the capacity and the network condition. For example, if an OR is slow or $H$ is set too

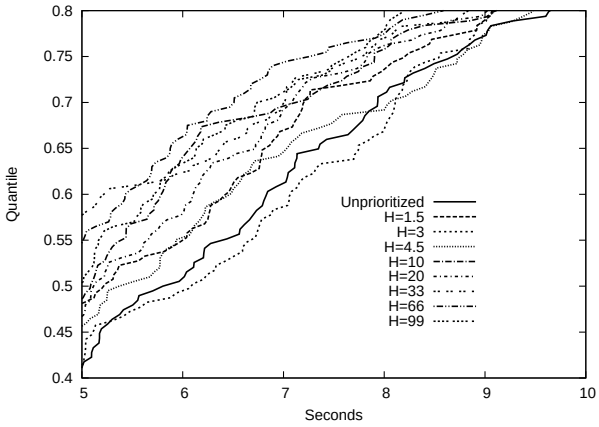


Figure 12: Comparison of performance for different values of $H$: CDF

small, the algorithm will quickly forget the circuit's past activity. A bulk transfer circuit will quickly drop to the same cell count as a newly created HTTP circuit, and compete with it. On the other hand, if the OR is fast or $H$ is set too large, a newly created bulk transfer circuit will be prioritized over an HTTP circuit created long ago.

In this section, we experiment with different values of the parameters, to examine the effects of the parameters on HTTP traffic.

## 4.1 Testbed Setup

The testbed setup is similar to the setup of our live Tor network test. We selected a variety of parameter values for the middle OR, `planetgurgle`, and tested the performance.

We used git version 0.2.1.24 of Tor on the middle OR. In the configuration file, the parameter `CircuitPriorityHalflife` is the $H$ value we mentioned earlier, which represents the interval after which the cell count for each circuit is decreased by half.

In this experiment, we randomly select a value for $H$ from the set $\{-1, 1.5, 3, 4.5, 10, 20, 33, 66, 99\}$ for our middle OR (-1 indicates unprioritized), and fetch a small file hosted at the University of Waterloo. We repeat this until each value has 200 datapoints, and collate the results.

## 4.2 Experimental Results

The results for the download times for different values of $H$ are shown in Figure 12 and Figure 13. Because of the density of the lines, we only show a fraction of the whole CDF in Figure 12. For ease of visualizing the data, in Figure 13, we show the 25th, 50th, and 75th percentile latencies for a range of different $H$ values (the curves) as well as for unprioritized Tor (the horizontal lines).

The figures show that smaller values of $H$ (1.5, 3, 4.5) perform only marginally better, if at all, than unprioritized Tor. This makes sense, as the past behaviour of a circuit will quickly be forgotten, and the bulk transfer circuits will have cell counts low enough to compete with HTTP circuits. The largest value 99 does not perform much better than unprioritized Tor either, since for our HTTP circuit, the behaviour will accumulate to a large cell count value and lose priority. The other values, 10, 20, 33 and 66, seem to be good values for our OR.

The results match our assumption: the $H$ value only need ensure that bulk transfer circuits will have higher cell counts than HTTP circuits; a wide range of values can satisfy this requirement. A value around 20 or 30 will be likely to satisfy most ORs in the Tor network. A global default value can be set in the directory authorities' consensus, so that OR operators do not need to manually configure it. Even if an OR operator misconfigures this value, for example, by setting it to 1.5, the performance will not be greatly harmed, as shown by our experiment.

However, as the Internet and the Tor network evolve, different protocols will start to use Tor, and the traffic distribution will not stay constant. The parameter should be regularly re-evaluated.

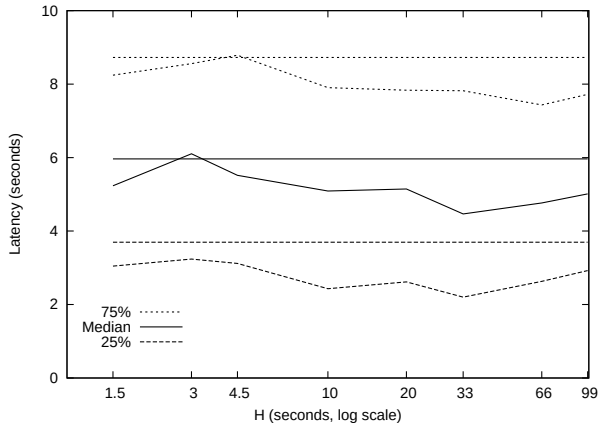


Figure 13: Comparison of performance for different values of $H$: latency vs. $H$. Values for unprioritized Tor are shown by horizontal lines.

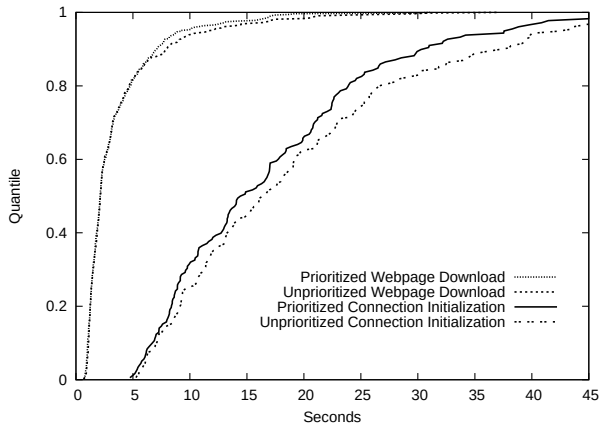


Figure 14: CDF for time cost of connection initialization and webpage fetching for Hidden Services, for unprioritized and prioritized Tor

# 5 Effects on Hidden Services

Tor's Hidden Services [3] mechanism allows users to provide TCP-based services, such as operating web servers, without revealing the server's IP address. Instead of publishing its IP address, the hidden server publishes a *server descriptor*, containing a signed list of *introduction points* to the directory authorities. The client downloads the descriptor, connects to one of the introduction points, selects one OR as the *rendezvous point*, builds a circuit to it, and tells the hidden server the rendezvous point selected. Then the server connects to the rendezvous point, and begins communicating with the client.

Because of the different design goals, the underlying network traffic for hidden services is much more complex than for regular public services. Accessing hidden services involves more than a dozen ORs and multiple rounds of negotiation; as a result, it is much slower than accessing public services through Tor.

The most time-consuming part of the process for a client to access a hidden service is circuit creation. According to statistics [8], the average time for hidden server circuit creation is 33.8 seconds, much higher than that of public services. After the circuit creation, the communication between the client and the server becomes almost as fast as accessing public services through Tor, as we will show later through our experiments.

Our EWMA algorithm is ideal for prioritizing the circuit creation phase of Hidden Services, since the command cells for circuit creation have small sizes (512 bytes each), and they are the first cells sent by a circuit. These cells should get the highest priority almost independent from the value of the EWMA parameter.

We tested the performance improvement of Hidden Services, using a similar approach to our experiments in Section 3.3, above. We used $H = 33$ for our middle OR. The hidden server is a desktop machine at the University of Waterloo. The server is also configured to use `planetgurgle` as the middle OR for all of its circuits.

We modified webfetch to support SOCKS4a, in order to resolve URLs for hidden services. We also instrumented webfetch to separately record the time for connection initialization and the time for webpage fetching. For each client instance, we perform four fetches of the target webpage. The latter three fetches will not include circuit creation, and so should be much faster than the first one. We performed 200 groups of tests (200 circuit creations). The connection initialization time for the latter three fetches is very short (averaging 2.0 seconds) compared to the first fetch. Since we are mainly concerned about the circuit creation, we only show the connection initialization time for the first fetch, and the download time for webpages. The CDF is shown in Figure 14.

As we notice, there are improvements in the connection initialization time in each quantile. The average time decreased from 19.3 seconds to 17.1 seconds (11.4%). This time consists of mostly circuit creation. Note that only two hops out of seven are prioritized (`planetgurgle`). We expect more improvement if all the ORs in the circuits are prioritized.

# 6 Discussion

## 6.1 Compatibility with the Existing Tor

For any upgrade of a distributed system the size of Tor, compatibility is a fundamental issue to consider. Requiring simultaneous upgrades for all Tor nodes would be a great resisting force to the implementation. Fortunately, since our algorithm only changes the order that cells are multiplexed from different circuits within the OR, it does not require any change in other ORs. Consequently, each OR can be upgraded individually, and each upgrade would make some improvement, as can be seen by the results of our experiments.

Our algorithm can also be turned on and off conveniently, by setting parameters in the Tor configuration file, and switched at runtime by sending a `SIGHUP` signal to Tor to reload its configuration file.

## 6.2 Effects on Security Properties of Tor

An important question is naturally whether these improvements in performance would enable an attack not previously present. We point out for emphasis that Tor is known to be insecure against an adversary that can see both ends of a circuit [6, 13, 20]. The most apparent avenue for attack is for an attacker, who can see just one end of a circuit, to try to determine whether the other end is an OR that has been upgraded to use this protocol, or not, by observing the performance of the circuit.

Here, Tor's large variance in performance comes in handy. Although our method provides a noticeable improvement, the improvement is still small as compared to the very large variance; see Figure 7. It would seem to be as easy for an attacker to learn in the stock Tor, for example, whether the OR at the other end of the circuit had high or low bandwidth.

One may also contemplate an attack in the style of [5], wherein the attacker constructs his own circuits through various ORs in the network in order to observe interference with a target circuit. This attack is already possible in stock Tor [5]; the fixes to Tor made as a result of that paper (limiting circuits to eight hops) only prevent the bandwidth amplification portion of the attack.

# 7 Related Work

A number of works examine Tor in an attempt to improve its performance; we give an overview of some of them, and other work related to our improvements, in this section.

## 7.1 Tor over DTLS

One area of investigation in improving Tor's performance is in fighting the improper application of TCP's congestion control mechanisms, which degrade Tor's performance. Between each pair of ORs, multiple circuits may share the same TCP connection, and their traffic is multiplexed within this connection using the same socket. When the number

of unacknowledged packets in the socket buffer exceeds the socket's congestion window, TCP's congestion control mechanism takes effect, and TCP will back off until more acknowledgements are received. This mechanism is desirable if there is only one circuit using the connection; however, if there is more than one circuit, one circuit sending too much data and triggering congestion control will cause cells on other circuits to be prevented from being transmitted as well.

Reardon and Goldberg [19] addressed this problem by using a TCP-over-DTLS tunnel. Instead of using a TLS/TCP connection between each pair of ORs, a DTLS/UDP connection is established, to prevent the congestion control mechanism incurred by one circuit from preventing other circuits from sending data. On top of DTLS/UDP, a user-level TCP connection is established for each circuit, to guarantee in-order delivery, congestion control, and flow control on a per-circuit basis.

Reardon and Goldberg's work concentrated on a fair application of the congestion control mechanism: the fault of one circuit should not affect other circuits. In comparison, our approach aims to be fair on resource allocation among circuits: circuits that consume few resources recently should be prioritized over other circuits.

### 7.2 Opportunistic Bandwidth Measurement

Tor relies on ORs' self-reported bandwidth values as weights to make router selection decisions. This is not necessarily accurate, and also may encourage malicious ORs to report a higher bandwidth to attract traffic. Snader and Borisov [21] proposed an opportunistic bandwidth measurement algorithm to replace the self-reported bandwidth; this is more accurate, and responds to changing load conditions quickly, while at the same time preventing low-resource routing attacks. In this way, Tor's bandwidth resources are allocated more efficiently, and the overall performance is improved.

Snader and Borisov also proposed a mechanism for users to tune Tor's parameter to select between circuits for higher performance or higher anonymity, while incurring very little cost of the other property.

Their work is orthogonal to ours; using both at the same time should evince added benefits.

### 7.3 Internet QoS Schemes

Several QoS schemes exist for today's Internet, including Integrated Services, RSVP, Differentiated Services, MPLS, and Constraint-Based Routing. [23] There are also schemes for prioritizing bursty traffic in ATM networks. [14] Our approach is similar to those efforts, in the sense that Tor is an overlay network and ORs act like Internet routers; we wish to improve the QoS of HTTP traffic by adjusting the scheduling policies within ORs.

Our EWMA algorithm is simple and effective in our situation. Nonetheless, incorporating ideas from other QoS techniques into Tor would be an interesting avenue for future work. One caveat is that in Tor, the *contents* of the packets are unavailable to the ORs, so QoS methods based on protocol analysis or deep packet inspection would be unsuitable for our use.

## 8 Future Work

### 8.1 Prioritizing Connections within an OR

In the fine-grained analysis of a cell's time spent within the OR, we observed that our modifications resulted in a reduction in the amount of time a cell spends waiting to be flushed from the circuit queues of interactive circuits to the connection output buffer. We also observed that the cells still wait in the connection output buffer for a noticeable amount of time. If this time can be reduced, hopefully interactive circuits will benefit more in reducing the latency.

One possible approach is to reduce the size of the FIFO output buffer, so that circuits are selected by priority closer in time to when the cells flushed from them are sent over the network. This will reduce head-of-line blocking within the output buffer, and get cells from high-priority circuits onto the network faster.

Another direction is to prioritize connections within an OR as well, by assigning higher priority to the idle connections. A problem to consider is how to guarantee fairness, since a busy connection may not be doing a bulk transfer; it may simply contain many circuits, all doing web browsing. Slowing down this connection is not the desired behaviour. Thus, besides watching the connection's activity, we might also need to watch the circuits within it as well.

### 8.2 Gaming the EWMA Algorithm

Since the prioritization decisions are based on the behaviour of each circuit, a user can modify her bulk transfer protocol to open many circuits and transfer parts of the file with each circuit in a bursty way. Each circuit will then have a lower EWMA value, and will be prioritized over HTTP and other interactive protocols.

Because our EWMA algorithm does not significantly degrade the performance of bulk transfer, there is no strong incentive for those bulk transfer users to implement such a modification. However, such modification can make a specified protocol prioritized over any other protocol on Tor. We should note that our algorithm does not introduce this attack, since a user can still utilize multiple TCP connections over multiple circuits on her protocol to make it more competitive, even with unprioritized Tor. Indeed, using multiple TCP connections on the regular (non-Tor) Internet will yield an unfair share of bandwidth. [15] Investigations into countermeasures will be a good direction for future work.

## 9 Summary

In this paper we examined one source of Tor's performance issues. One of the factors that contributed to the bad performance for interactive streams is the unfair scheduling algorithm among circuits: interactive circuits will be greatly slowed down because of co-existent non-interactive circuits on the same connection. We proposed an EWMA-based scheduling algorithm to prioritize the interactive circuits, and performed experiments on PlanetLab as well as the actual Tor network. The results show that under realistic network traffic, the interactive streams in prioritized Tor performs about 10% to 20% better, in terms of latency. The algorithm is completely compatible with the current Tor network: the ORs can be upgraded gradually, it can be turned on and off easily and on the fly, and the benefits will be seen immediately. Also, the algorithm brings little overhead, even on CPU-limited ORs.

## Acknowledgements

We thank Ryan Henry, Femi Olumofin, and Greg Zaverucha for their comments on a draft of this paper. We gratefully thank The Tor Project for their financial support, and also for incorporating our results into the main Tor code. We finally acknowledge MITACS and NSERC for their financial support as well.

## References

[1] Tor Network Status. `http://torstatus.kgprog.com/`, 2009. Accessed April 2010.

[2] Tony Aiuto. webfetch. `http://tony.aiu.to/sa/webfetch/`, 2004. Accessed April 2010.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[4] Roger Dingledine and Steven Murdoch. 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`, 2009. Accessed April 2010.

[5] Nathan Evans, Roger Dingledine, and Christian Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proceedings of the 18th USENIX Security Symposium*, pages 33–50, August 2009.

[6] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.

[7] John D.C. Little and Stephen C. Graves. Little's Law. `http://web.mit.edu/sgraves/www/papers/Little's%20Law-Published.pdf`. Accessed April 2010.

[8] Karsten Loesing. *Privacy-enhancing Technologies for Private Services*. PhD thesis, University of Bamberg, 2008.

[9] Karsten Loesing. Measuring the Tor Network. http://metrics.torproject.org/papers/directory-requests-2009-06-25.pdf, 2009. Accessed April 2010.

[10] Karsten Loesing. Measuring the Tor Network from Public Directory Information. http://freehaven.net/~karsten/metrics/measuring-tor-public-dir-info-final.pdf, 2009. Accessed April 2010.

[11] Nick Mathewson. gitweb.torproject.org. http://gitweb.torproject.org/tor.git?a=commit;h=06e8370c33d6ccb73d55e9e8c3d2673c48d7b328, 2009. Accessed April 2010.

[12] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining Light in Dark Places. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium*, pages 66–67, 2008.

[13] Steven J. Murdoch and Piotr Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Proceedings of the 7th Privacy Enhancing Technologies*, pages 167–183, Ottawa, Canada, 2007.

[14] Matt W. Mutka and Jose Roberto Fernandex. A Burst-Level Priority Scheme for Bursty Traffic in ATM Networks. In *Proceedings of Sixth International Conference on Computer Communications and Networks*, pages 11–16, 1997.

[15] Reinaldo Penno, Satish Raghunath, Vijay K. Gurbani, Richard Woundy, and Joe Touch. LEDBAT Practices and Recommendations for Managing Multiple Concurrent TCP Connections. http://www.ietf.org/id/draft-ietf-ledbat-practices-recommendations-00.txt, February 2010. Accessed April 2010.

[16] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[17] Joel Reardon. Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo, 2008.

[18] Joel Reardon. libspe. http://crysp.uwaterloo.ca/software/, 2009. Accessed April 2010.

[19] Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In *Proceedings of the 18th USENIX Security Symposium*, pages 119–133, 2009.

[20] Andrei Serjantov and Peter Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Proceedings of ESORICS 2003*, pages 116–131, October 2003.

[21] Robin Snader and Nikita Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Proceedings of 16th Annual Network and Distributed System Security Symposium*, 2008.

[22] WebSiteOptimization.com. Average Web Page Size Triples Since 2003. http://www.websiteoptimization.com/speed/tweak/average-web-page/, 2008. Accessed April 2010.

[23] Weibin Zhao, David Olshefski, and Henning Schulzrinne. Internet Quality of Service: an Overview. www.cs.columbia.edu/techreports/cucs-003-00.pdf, 2000. Accessed April 2010.