Torchestra: Reducing Interactive Traffic Delays over Tor

Deepika Gopal
UC San Diego
drgopal@cs.ucsd.edu

Nadia Heninger
UC San Diego
nadiah@cs.princeton.edu

ABSTRACT

Tor is an onion routing network that protects users' privacy by relaying traffic through a series of nodes that run Tor software. As a consequence of the anonymity that it provides, Tor is used for many purposes. According to several measurement studies, a small fraction of users using Tor for bulk downloads account for the majority of traffic on the Tor network. These bulk downloads cause delays for interactive traffic, as many different circuits share bandwidth across each pair of nodes. The resulting delays discourage people from using Tor for normal web activity.

We propose a potential solution to this problem: separate interactive and bulk traffic onto two different TCP connections between each pair of nodes. Previous proposals to improve Tor's performance for interactive traffic have focused on prioritizing traffic from less active circuits; however, these prioritization approaches are limited in the benefit they can provide, as they can only affect delays due to traffic processing in Tor itself. Our approach provides a simple way to reduce delays due to additional factors external to Tor, such as the effects of TCP congestion control and queuing of interactive traffic behind bulk traffic in buffers. We evaluate our proposal by simulating traffic using several methods and show that Torchestra provides up to 32% reduction in delays for interactive traffic compared to the Tor traffic prioritization scheme of Tang and Goldberg [18] and up to 40% decrease in delays when compared to vanilla Tor.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General - Security and Protection

General Terms

Security, Design

Keywords

Privacy, Tor, BitTorrent

WPES'12, October 15, 2012, Raleigh, North Carolina, USA.

1. INTRODUCTION

Tor is an anonymizing network designed by Dingledine, Mathewson and Syverson in 2004 [8] that provides privacy and anonymity to users all over the world. Traffic from clients is relayed through three Onion Routers before being forwarded to the destination. Tor's goal is to prevent an attacker from linking together the source and destination IP addresses of clients and learning their browsing habits. On any circuit, each Onion Router knows the address of the node before and after it but not of any other node along the path between the source and destination, thus preserving anonymity. Tor has been designed to protect against a non-global adversary i.e., an adversary who does not have control over both exit and entrance nodes of a circuit.

Tor is known to suffer from performance issues, as described by Dingledine and Murdoch in [9]. Since Tor relays are run by users, bandwidth is limited to how much a user is willing to allocate for Tor's usage. Thus when Tor is used for bulk file downloads (such as BitTorrent [5]), delays for interactive traffic (such as web traffic and SSH) increase [9]. Several measurement studies have observed that BitTorrent users represent only a small fraction (3-10%) of users in the Tor network, but use the majority (50-70%) of the bandwidth [13, 12, 3]. The disproportionate delays caused by these bulk downloads provide a strong disincentive for users to use Tor in situations where anonymity is not the main goal. Since the level of anonymization improves with the number of users, a reduction in delays for interactive traffic may result in an increase in the number of Tor users and benefit the Tor community.

Tor traffic can accumulate delays at many different processing stages as it moves through the network. The relatively small amount of traffic from interactive circuits will be disproportionately delayed due to time spent queued behind large amounts of bulk traffic in socket buffers or Tor's output buffers [7]. Additionally, since all circuits between a pair of nodes go through a single connection, when TCP's congestion control is triggered on this connection due to bulk traffic, interactive traffic on that connection will also be slowed down [15]. We discuss these delays in more detail in Section 3.

There have been several proposals to improve Tor's performance for interactive connections by using heuristics to classify circuits as interactive or non-interactive, and on each node prioritizing packets from interactive circuits before non-interactive circuits [18]. However, this approach is limited to improving the delay time that interactive packets

spend in Tor's output buffers when there is not much traffic already queued up in the buffers [7].

Our approach, which we call "Torchestra", is to create two separate connections between each pair of nodes: one for interactive traffic, and one for bulk traffic. This approach allows us to address causes of delay that continue to exist in prioritization schemes: by separating interactive traffic onto its own connection, we automatically give it greater priority in the input buffers and it will not be subject to congestion control triggered by bulk traffic.

To classify traffic as bulk or interactive, we use the exponentially weighted moving average (EWMA) heuristic introduced by Tang and Goldberg for their Tor traffic prioritization scheme [18]. The EWMA heuristic calculates a moving average of the number of cells sent on a circuit and adds greater weight to recent values. Our implementation continuously updates the EWMA value for each circuit and transfers a circuit to the appropriate connection according to its traffic flow. We describe our algorithm and implementation in Section 4.

In Section 5, we use the Tor simulator ExperimenTor [2] to evaluate our proposal using several experiments: first by comparing simple file downloads of various sizes, and then using simulated traffic emulating packet timings recorded from our own web and SSH traffic and finally, timing patterns collected on a public, non-exit Tor node. We compare the results to the vanilla Tor scheme and to the Tor traffic prioritization scheme of Tang and Goldberg [18]. When we replayed SSH and HTTP traffic collected from our own normal usage, we found between 8% to 32% reduction in delays with Torchestra compared to prioritized Tor and a 13% to 36% reduction in delays when compared to vanilla Tor. In our simulations with packet timings from Tor traffic, we observed a 2% to 25% decrease in delays for interactive traffic with Torchestra compared to prioritized Tor and a 4% to 40% decrease in delays when compared to vanilla Tor.

Finally, in Sections 6 and 7 we discuss the security properties of the scheme and potential modifications that may lead to further improvement, including the possibility of more fine-grained traffic balancing by dividing Tor traffic between many different connections.

While there are several considerations to be made when adopting Torchestra in practice, our results illustrate that our proposal has the potential to improve Tor performance significantly beyond the benefits seen by traffic prioritization schemes. In essence, we are creating parallel Tor networks and using the separate connections to obtain the performance we desire from each.

2. RELATED WORK

Reardon and Goldberg [15] were the first to propose a scheme to improve Tor's performance for interactive traffic. They describe several problems due to bulk traffic, including TCP congestion control unfairly affecting light traffic and delays due to queuing of light traffic behind heavy traffic. Their solution was to create a separate connection for every client using a user-level TCP stack and to multiplex these connections over a DTLS/UDP tunnel between every pair of nodes. In this manner, a heavy circuit can affect only itself, and not other heavy circuits or light circuits. While this would be an ideal solution, Dingledine explains in [9] that it has not yet been implemented due to licensing issues on most high quality user-level TCP stacks.

Tang and Goldberg [18] proposed "prioritized Tor", which aims to reduce the delay of light circuits by giving higher priority to interactive circuits. They did this by calculating the Exponentially Weighted Moving Average (EWMA) of the number of cells to measure the recent activity of a circuit, and then giving circuits that have a lower EWMA value and hence lower activity higher priority over other circuits that have cells ready to transmit. According to Dingledine [7], the benefit to this approach may be limited: since all circuits are using the same connection to send data to the next node, if there is already a large amount of data queued up on the socket buffer or Tor's output buffer, interactive circuits will still face high delays. Also, light circuits will face the effects of congestion control triggered by heavy circuits.

Dingledine [7] describes another option available for Tor nodes to control traffic, the PerConnBWRate and PerConnBWBurst configuration options. Both these options are used to separately rate-limit every connection from a non-relay. In this way, heavy clients which are not relays can be throttled at the entrance router itself. The issue with this approach as mentioned by Tschorsch and Scheuermann [19] is that since this form of configuration is static, it does not take care of the current load and state of the network and even if there is bandwidth available, clients will be unnecessarily throttled. To overcome this problem, Syverson, Jansen, and Hopper [17] propose adaptive bandwidth throttling. They use EWMA to classify circuits by usage, and throttle an adjustable fraction of users at the entrance node.

The inspiration for our work and the name we chose for our scheme comes from the Orchestra project of Chowdhury et al. [4], a method to manage network bandwidth in a map-reduce system by opening a number of TCP connections proportional to the amount of data to be transferred across the network in order to reduce the average job completion time. The node that has more data to transfer will open more connections and, due to TCP's max-min algorithm, will get a greater share of the bandwidth. Originally, we hypothesized that increasing the number of connections for light circuits would ensure that they get a greater assured share of bandwidth. However, Tor already implements the max-min algorithm to ensure that bandwidth is divided equally amongst connections, as well as sending out cells in round-robin order for the different circuits on a connection. It turns out that using separate connections for light and heavy circuits improves Tor's performance for more subtle reasons, which we describe in Section 3.

Tschorsch and Scheuermann [19] explain how the division of bandwidth between circuits is not fair. User-configured bandwidth is divided equally amongst connections, and each connection's bandwidth is divided equally amongst its circuits. The authors observe that the circuits that exist on connections shared between very few circuits get a larger slice of the bandwidth. They implement a solution that achieves max-min fairness between circuits and uses an N23 congestion feedback scheme to better make use of bandwidth and prevent congestion. Their observations suggest a potential problem with our scheme: since the connection used for interactive traffic likely has many more circuits than the bulk transfer connection, less bandwidth will be available to a light circuit using Torchestra than in vanilla Tor. We test the impact of this in Section 6.2.

Several measurement studies have examined how Tor is used by analyzing exit node traffic, and published statistics

on protocols and traffic distributions. In 2007, McCoy et al. measured the fraction of BitTorrent users as 10%, and their traffic consumption as 67% [13]. One year later in 2008, they measured these percentages as 3.3% and 51.5%, respectively [12]. In the most recent study we know of, in 2010, Chaabane, Manils, and Kaafar [3] observed that BitTorrent users made up only 10% of Tor connections, but used 55% of traffic, while about 70% of circuits carried web traffic, which accounted for about 35% of traffic. We use this information to design experiments attempting to replicate realistic conditions.

3. TOR TRAFFIC AND DELAYS

In this section, we discuss the potential sources of delay in the Tor network. Throughout the rest of the paper, we will refer to interactive traffic as light traffic and bulk traffic as heavy traffic.

At the exit node, Tor processes the cells from each destination and sends cells to the next node on each circuit in round-robin order. If all circuits had similar traffic patterns, this would be ideal. However, in reality, there is a great deal of disparity between the traffic patterns of different circuits. After incoming cells are processed, they are sent to Tor's output buffer, where cells from light circuits must wait behind cells from heavy circuits before being transferred over the network. These kind of delays affect the experience of interactive traffic users but will not make much difference to bulk traffic users.

As the packets are sent over the network, the connection may trigger TCP congestion control to slow the transmission rate [15]. Since cells from light circuits and heavy circuits share the same connection, the heavy traffic may trigger congestion control due to no fault of the light circuits, but the cells from the light circuits will also be slowed.

How Torchestra can help.

The main idea behind Torchestra is to separate heavy traffic from light traffic and thus prevent bulk traffic from increasing delays for interactive traffic. All of the problems mentioned are due to light traffic and heavy traffic going over the same connection. Let us consider how having separate connections might help solve each of the two problems. Regarding the queue waiting times, light traffic will no longer be forced to wait behind heavy traffic since the socket buffers and Tor output buffers (per connection) for the two types of traffic will no longer be the same. For the congestion control issue, if the two types of traffic are separated onto different connections, when TCP's congestion control algorithm is triggered due to heavy traffic, light circuits will not get affected. Thus we see that having separate connections should lead to improvement in Tor's performance for light traffic.

4. OUR ALGORITHM

In our scheme, we open two connections between every pair of communicating Tor nodes, one "light" connection and one "heavy" connection. Circuits begin on the light connection, and are transferred to the heavy connection if their measured traffic crosses a relative threshold; similarly we move a circuit back to the light connection if its traffic drops below a certain threshold on the heavy connection.

We describe our classification methods and protocols below.

4.1 Classifying a circuit

The classification of a circuit as light or heavy is done by the exit node. This is because a node can identify when it is running as an exit for a circuit without any ambiguity. In this work we have considered the case where only one node is responsible for switching a circuit, and the entire circuit is switched at once. An alternative version of this scheme could do this transfer on a node-by-node basis; see Section 6 for more information.

For a chosen window of time, the node collects statistics about the number of cells sent on each connection and on circuits using this node as an exit node. As in Tang and Goldberg [18] we want to maintain a metric for how many cells a circuit and the connection have sent recently. We use the EWMA metric they chose for this purpose. We now give an overview of EWMA and then a description of how we classify circuits as light or heavy.

4.1.1 Exponentially Weighted Moving Average

The Exponentially Weighted Moving Average (EWMA) is a statistic used to calculate the moving average while giving more weight to recent data [16]. For our application, it smoothes the input, giving more priority to recent measurements, and can be updated using a simple formula requiring no memory.

The formula used to calculate the EWMA value at time t is given by:

$$\mathrm{EWMA}(t) = \alpha \cdot Y(t) + (1 - \alpha) \cdot \mathrm{EWMA}(t-1)$$

where $\mathrm{EWMA}(t)$ is the EWMA value at time t, $Y(t)$ is the data observation at time t and $0 \leq \alpha \leq 1$ is the multiplier that determines the depth of memory of the EWMA.

The choice of the multiplier will determine to what extent changes in recent data affect the average value. The larger α, the larger the influence of $Y(t)$ on the EWMA value. We can define a parameter W which is related to the half-life of the EWMA, and define α in terms of W.

$$\alpha = \frac{2}{W+1}$$

To begin, we compute the unweighted average over the first W periods, then from period $W+1$, calculate the EWMA using the above equation.

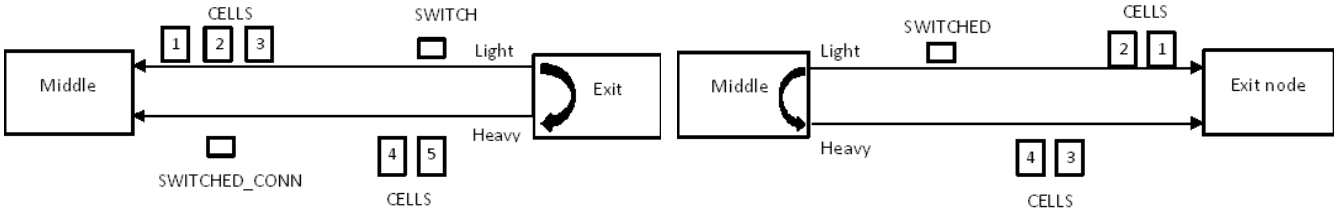
4.1.2 Circuit classification

In order to decide whether a circuit belongs on the light or heavy connection, we define two relative thresholds T_ℓ and T_h.

The node calculates the EWMA of the traffic for every circuit using the node as an exit and for every corresponding connection for a length of time; two seconds in our implementation.

After this initial period, we move a circuit from the light to the heavy connection if its EWMA is above the threshold T_ℓ times the average EWMA for the light connection. Similarly, we move a circuit from the heavy connection back to the light connection if its EWMA is below the threshold T_h times the average EWMA for the heavy connection.

In order to prevent inactive circuits from contributing to a connection's statistics and causing unnecessary transfers



(a) Inbound direction

(b) Outbound direction

Figure 1: In our connection switching protocol, the connection is transferred first on the inbound direction, from the destination to the source (in this figure, from the exit node to the middle node). No more cells are sent on the light connection after the `SWITCH` cell has been sent. The circuit is then switched to the heavy connection, and a `SWITCHED_CONN` cell is sent on this connection followed by further cells. To switch the conection on the outbound direction, the destination sends a `SWITCHED` cell on the light connection and then sends all further cells on the heavy connection.

of circuits from light to heavy, we define a third threshold T_i. If a circuit's EWMA drops below T_i times the average EWMA for the light connection, it is considered inactive and is not counted toward the connection's circuit count. In order to expedite this process, we define a higher multiplier β which is used to update the EWMA for circuits that do not send any cells in a time interval.

4.2 Protocol to switch a circuit

Once a decision has been made on the exit node that a circuit needs to be switched, the exit node initiates the protocol for the switch. The exit node is responsible for switching the circuit to the new connection between the exit and middle nodes. Once this switch is done, the middle node is responsible for switching the circuit to the new connection between the middle and entrance nodes.

New control cells.

We define three new control cells to manage the connection transfer: a `SWITCH` cell that is sent by the initiator on the old connection indicating that all further cells for this circuit will be sent on the new connection, a `SWITCHED_CONN` cell sent by the initiator on the new connection before the newly transferred circuit's cells are sent, and a `SWITCHED` cell sent by the receiver node informing the initiator that the switch is complete and it will also send further cells on the new connection.

The header in each of these control cells uses the same cell header structure as other cells in Tor. `SWITCHED_CONN` and `SWITCHED` cells have no payload. The `SWITCH` cell's payload contains a flag that indicates whether the switch has to be extended to the previous node in the inwards direction.

Since packets may arrive out of order on the two connections during switching, the node stores any out-of-order packets in a buffer. The reason a `SWITCHED_CONN` cell is required is because cells on the new connection may arrive before all the remaining cells on the old connection have been processed and this may lead to out–of–order processing.

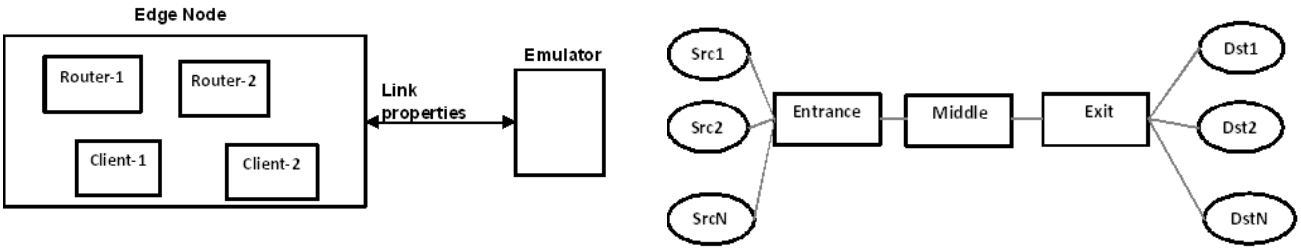
Switching protocol.

The protocol to switch a circuit from a light to a heavy connection is depicted in Figure 1. The steps are as follows:

1. Check whether a heavy connection has been created between exit and middle router. If not, create a new connection.
2. The exit node sends a `SWITCH` cell on the light connection to inform the middle node that no more cells for this circuit will be coming in on this connection. The payload in the `SWITCH` cell contains a flag that the exit node sets to inform the middle node that it needs to extend the heavy connection towards the entrance. The exit node continues to receive cells from the middle node on the light connection.
3. The exit node sends a `SWITCHED_CONN` cell on the heavy connection followed by the circuit's cells.
4. Once the middle router has received both the control cells, it sends a `SWITCHED` cell on the light connection and only then does it start processing the circuit's cells from the heavy connection. The cells that might have arrived on the heavy connection before the `SWITCH` cell arrived on the light connection are saved in a queue and these cells are processed once both the control cells are processed. This completes the circuit switch on the middle node.
5. After the exit node receives the `SWITCHED` cell on the light connection, no more cells for this circuit will be coming from the middle node on this connection. It completely switches the circuit to the heavy connection. The cells that might have arrived on the heavy connection before the `SWITCHED` cell arrived on the light connection are saved in a queue and these cells are processed once the `SWITCHED` cell has been processed.

The middle node follows the same procedure to create a heavy connection to the entrance node and switch the circuit. The middle node does not set the flag in the `SWITCH` cell and hence the entrance node does not create any extra connections to the client. A similar procedure is followed when a circuit is to be switched from a heavy connection to a light connection.

If the middle node does not support Torchestra, the `SWITCH` control cell received from the exit will be dropped. A flag is set for the circuit on the exit so it does not attempt to



(a) Physical experiment setup

(b) Virtual setup

Figure 2: Experimental setup. Routers and clients are run as virtual nodes on the physical edge node and all traffic between these virtual nodes are forced to pass through the emulator.

switch again. Thus, a maximum of one extra cell per circuit will get sent if Torchestra is not supported.

5. EXPERIMENTAL RESULTS

We performed several different experiments in order to evaluate Torchestra's performance.

5.1 Experimental setup

We evaluate Torchestra using the ExperimenTor framework [2], a Tor emulation toolkit and testbed. ExperimenTor is built on top of the ModelNet network emulator [20] and uses commodity hardware to simulate an entire network. ModelNet emulates distributed systems by allowing virtual nodes to be set up on one or more physical machines. It allows bandwidth, queuing, propagation delay and drop rate to be configured on the links between these virtual nodes to give realistic effects of the network. One machine is designated as the emulator and traffic between any two virtual nodes is forced to pass through it.

Our ExperimenTor setup consists of two machines as shown in Figure 2(a). The first machine is an edge node, on which the different virtual nodes representing routers and clients are run as separate processes. The second machine is the emulator, through which traffic between any two virtual nodes is forced to pass. Once all the virtual routers and clients are configured, ExperimenTor behaves as if each router and client is a separate node, depicted in Figure 2(b).

In our experiments, the edge node has a 2.8Ghz Intel Core processor and runs Ubuntu 11.04. The emulator has a 2.5 GHz Intel Core processor and runs Ubuntu 10.04.

Our algorithm switches a circuit when its EWMA hits a threshold value relative to the average EWMA of the connections on its circuits. The parameters we chose for our experimental threshold values are designed only to distinguish circuits with traffic falling outside of an arbitrary range of normality; these values are not carefully optimized, and more detailed measurement studies should be undertaken in order to select proper parameters. Similarly, we chose parameters for the EWMA filter to ensure a reasonable sized window of time over which the average is taken, but did not study the effect of these parameters on the performance of the algorithm. Tang and Goldberg parameterize their EWMA using a value H corresponding to the half-life of the running average. They studied the effect of different choices for the EWMA parameters on their prioritization scheme [18] and observed that a wide range of values seemed to produce good results. In the following experiments we have run the prioritized Tor scheme using $H = 66$, the value that performed the best in their experiments; we note here that the value of α we used to parameterize the EWMA for Torchestra corresponds to a much shorter half-life, and as such a more detailed understanding of appropriate parameter choices is necessary before drawing any conclusions from the results.

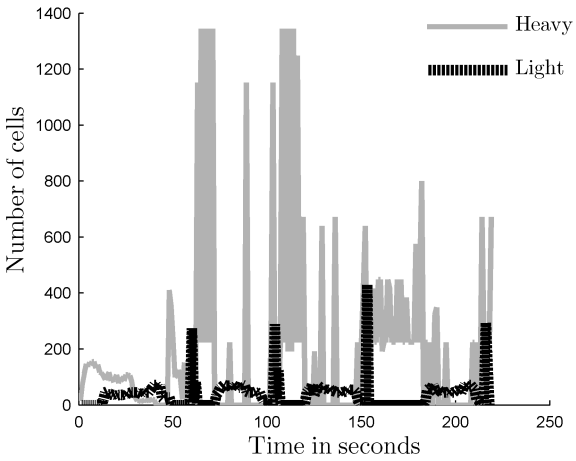
Although ExperimenTor allows simulation of the real network, our setup consists of three router nodes (which also act as directory servers) and multiple clients. The results of our experiments are preliminary demonstrations of the feasibility of Torchestra in a simple setup. We intend to carry out further experiments to confirm that Torchestra provides the same degree of improvement on a larger, more complex setup.

5.2 Experiment 1: Simple file downloads

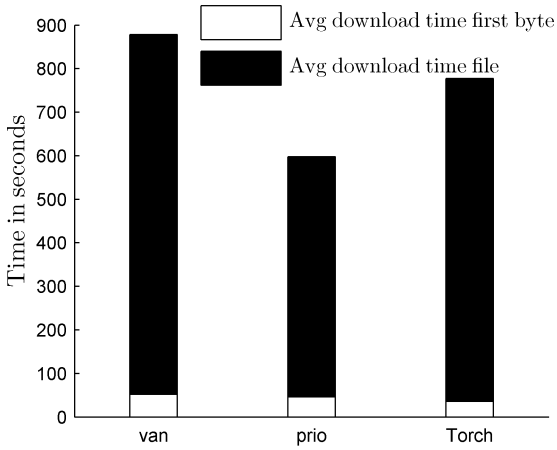
The first experiment we carried out was to simulate light and heavy traffic by having clients continuously download files of fixed size. In this experiment, our ExperimenTor setup used 3 routers and 13 clients. Seven of these clients are heavy clients, each continuously downloading a 100MB file. The other six clients are light clients, each of which downloads a 300KB file with about 50 seconds of gap between each download. This ratio of heavy clients and light clients, gaps between downloads and file sizes is designed to emulate the ratio of heavy to light traffic observed in [13]. Between every pair of routers, the bandwidth is rate-limited to 3mbps. For all other links, bandwidth is rate-limited to 1mbps.

We ran our experiment ten times each of ten minutes duration on vanilla Tor, prioritized Tor and Torchestra. We used the default parameters in vanilla Tor. For prioritized Tor, we set $H = 66$ as in [18]. For Torchestra, we used the following parameters:
$\alpha = 0.18$, $\beta = 0.18$, $T_\ell = 1.4$, $T_h = 0.3$.

This setting of α is designed so that we compute our running average over 10 time periods; as such this corresponds to a different EWMA half-life than $H = 66$. Also, as explained above, in this experiment we used slightly different parameters than in later experiments. This is because the circuits we intended to be "light" consume as much bandwidth as heavy circuits while downloading, and were correspondingly transferred to the heavy connections as often as heavy circuits. These parameters are designed to prevent



(a) Traffic pattern of heavy and light traffic



(b) Average time to download first byte and average time to download a 300KB file for light circuits.

Figure 3: Experiment 1: Modeling traffic with file downloads. We set up a simulated Tor network with light clients periodically downloading a 300KB file and heavy clients continuously downloading a 100MB file and observed that prioritized Tor has the least delay of the three schemes we compared, although the delay in receiving the first packet was lowest in Torchestra.

this. We use different parameters later on to deal with more realistic data.

5.2.1 Results

The results are shown in Figure 3.

With simple downloads, prioritized Tor has the lowest average download time. Interestingly, the time to download the first byte is the lowest in Torchestra. We hypothesize that this may be happening because the initial packets required to start the download are behind many heavy cells and hence delays will be initially high. But once the download starts, since a light circuit will take up as much bandwidth as a heavy circuit and since light cells will be given higher priority in prioritized Tor, every cell from a light circuit will be sent faster.

In order to test our hypothesis, we carried out an experiment where we deliberately introduced a variable delay ranging from 4 msec to 128 msec in powers of two between each packet sent to the light circuits. We then found the average delay per cell to reach the source client from the destination. As seen in Figure 4, with a smaller amount of delay between arriving light cells, prioritized Tor exhibits a lower transmission delay than Torchestra, but as the delay between cells grows, Torchestra's transmission delay drops below that of prioritized Tor.

5.3 Simulating web and SSH traffic

The simple file downloads we experimented with above are unlikely to represent real traffic patterns. In order to construct a more realistic experiment, we captured traffic patterns of our own web and SSH usage and replayed them as Tor traffic for light circuits, over a background of continuous file downloads for heavy circuits.

We recorded HTTP and SSH traffic from our own sessions using the network protocol analyzer Wireshark [6]. We then replayed this Wireshark traffic as light traffic for four representative intervals of 10 minutes each over a background of four heavy clients continuously downloading 100MB files. We compared the results between vanilla Tor, prioritized Tor, and Torchestra.

We used the following method to replay traffic timings:

Method for replaying traffic.

We used a Java process to simulate traffic timings. It creates a separate thread for every circuit ID to be simulated, each of which is created with a different IP address and listens on different sockets. The ExperimenTor source clients connect to each of these threads' sockets through Tor.

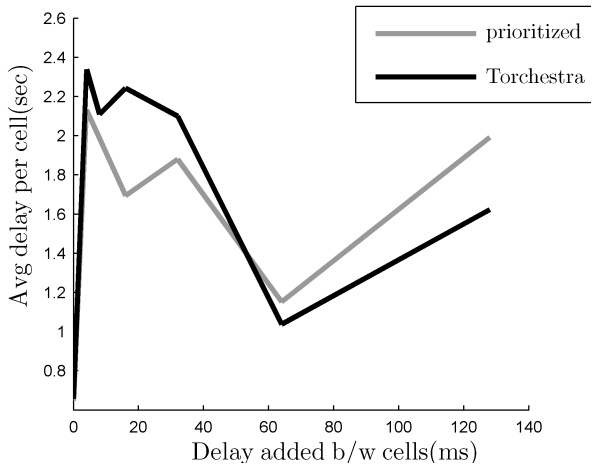
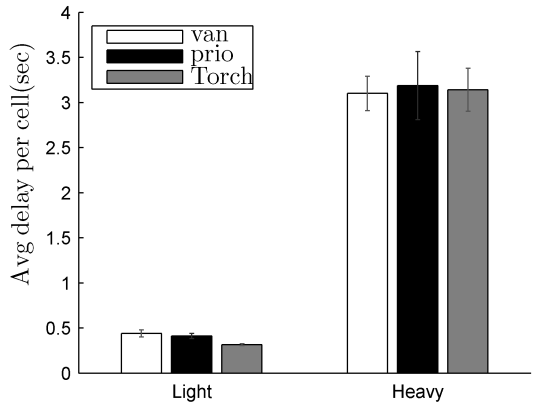
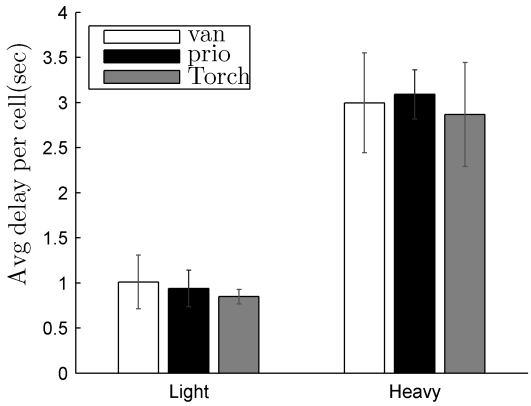


Figure 4: Experiment 1: Simple downloads with delays between cells. We wanted to test the hypothesis that prioritized Tor performed well in a simple file download scenario because both light and heavy circuits were saturating their circuits. We injected pauses between sending cells on light connections and observed that as the length of the pause increased, Torchestra's performance relative to prioritized Tor improved.



(a) Experiment 2: Replaying HTTP and SSH traffic



(b) Experiment 3: Replaying Tor traffic

Figure 5: Experiments 2 and 3: Replaying captured traffic. In order to simulate realistic traffic patterns, we captured traffic timings from interactive HTTP and SSH sessions and from a public, non-exit Tor node and used the timing data to simulate dummy traffic through our ExperimenTor setup. In both of these cases, Torchestra had the least delay. More detailed experimental results can be found in Figures 6 and 7 in the Appendix.

Every thread maintains a table of cell transmission times, and sleeps for the appropriate interval after sending a cell. Each cell consists of 498 bytes of dummy data. Tor adds 14 bytes of header information thus creating a cell of 512 bytes.

In our experiments, we measured the transmission time as the difference between the time at which a thread sends a cell and the time at which the source client receives it. The thread-switching time on our machine was about 2.7μs, and thus provides negligible overhead.

The parameters that we used in our experiment are as follows. We used the default parameters with vanilla Tor, and set $H = 66$ in prioritized Tor. For Torchestra, we used the following parameters:
$\alpha = 0.18$, $\beta = 0.36$, $T_\ell = 1.7$, $T_h = 0.3$.

5.3.1 Results

For each of our four traffic samples shown in Figure 6, Torchestra displays the least delay for light circuits. This is in contrast to the simple file download experiment above, where prioritized Tor showed the least delay. Comparing Torchestra with prioritized Tor, in each of the samples there is a 32.87%, 8.68%, 28.97%, 25.14% decrease in average delay respectively. Comparing Torchestra with vanilla Tor we see that in each of the samples there is a 36.36%, 13.17%, 30.48%, 33.9% decrease in average delay respectively. We display the average of these four cases for light circuits in Figure 5(a). In the average case there is a 23.4% decrease in Torchestra compared to prioritized Tor and a 28.5% decrease compared to vanilla Tor.

5.4 Simulating real traffic

5.4.1 Description

In both the above experiments, the heavy traffic is created artificially using wget. These traffic patterns may not match those of the real network.

In order to investigate the performance of our system on the real Tor network in a repeatable fashion, we wanted to test it on more realistic data. To this end, we collected timing information from a non-exit Tor node and used this to simulate traffic during our experiments. We ran our machine as a public, non-exit Tor node for more than a week before we started collecting data in order to ensure that our node had stabilized. We set the bandwidth on the node to be 5mbps.

We took steps to ensure that no traffic is de-anonymized and no non-metadata is collected. We collected only the circuit ID of the cell and the socket number of the connection on which cells left our node in order to determine which circuits belonged to a particular connection and the time the cells arrived at our node. Since we have collected logs only on a non-exit node, all transmitted data was encrypted and thus illegible to us; we did not examine or log the contents of these encrypted packets. When we replay traffic, we use the logged timing information to send dummy data with the same time patterns and circuit distributions, using the method described in the previous experiment. After we finished plotting our graphs we have made sure that all logs have been securely deleted.

We collected four 15 minute intervals of timing information at 6am, 12pm, 6pm, and 12am PDT. From each time interval we chose the connection with the largest number of cells and circuits to replay.

The parameters that we used in our experiment are as follows. We used the default parameters with vanilla Tor, and set $H = 66$ in prioritized Tor. For Torchestra, we used the following parameters:
$\alpha = 0.18$, $\beta = 0.36$, $T_\ell = 1.7$, $T_h = 0.3$.

5.4.2 Results

The results of these experiments are depicted in Figure 7. Torchestra displays the least average delay per cell in each case.

In the 6am case, there is a 25% decrease in average delay from the prioritized Tor case and 40% decrease from the vanilla Tor case. In the 12pm, 6pm and 12am cases there is a 8.41%, 7.43% and 2.2% decrease in average delay from the prioritized case and 8.93%, 17.9% and 4.83% decrease from

the vanilla case. We display the average of these four cases for light circuits in Figure 5(b). In the average case there is a 9.64% decrease in Torchestra compared to prioritized Tor and a 16% decrease compared to vanilla Tor.

6. DISCUSSION

There are several performance and security concerns that need to be considered in Torchestra's design. We explain the reasoning behind our design decisions and the trade-offs involved.

6.1 Connection-switching

Tor's design specifically avoids moving active connections to different circuits in order to make the system more usable for real applications. Our decision to transfer circuits to different connections might be seen as counter to this goal, and a potential cause of performance problems as the nodes must expend effort to ensure in-order delivery.

However, unlike the case of an active connection switching to a new circuit, we do not have to contend with unexpectedly switching to nodes that are potentially down or have severely limited bandwidth. There will be no extra delays incurred by cells during a circuit switch, as cells are not made to wait circuit switching protocol runs.

6.2 Are we reducing bandwidth in some cases?

One might ask whether sharing bandwidth across two connections actually reduces the bandwidth available for light circuits in some situations.

When the number of light circuits exceeds the number of heavy circuits and the amount of light traffic exceeds the amount of heavy traffic, light circuits may have less bandwidth available under Torchestra. This is because the bandwidth will be shared equally between the light and heavy connections. Tor sends out cells from different circuits on a connection in round-robin order. If there are n active circuits on a connection, each circuit will get $\frac{1}{n}$th of the bandwidth. If there are more active circuits on the light connection than circuits on the heavy connection, then as explained by Tschorsch and Scheuermann [19] this would lead to less bandwidth per circuit on the light connection which will affect interactive traffic.

In order to check for how often there are bursts of time when number of light circuits is greater than the number of heavy circuits, we carried out the following experiment.

6.2.1 Experiment to measure how many light connections we should open

As we described above, when we use a single connection each for light and heavy circuits, there is a possibility that bandwidth available to light circuits is reduced if there are bursts of time when the number of active light circuits is greater than the number of active heavy circuits. In the following experiment, we measure how often this happens.

We collected timing information from a non-exit Tor node as described in Section 5.4.1. We labelled the circuits "heavy" in order of contribution to total cells over the connection, up to a threshold of 70% of total cells. We checked for overlapping subintervals of 50msec, 70msec, 80msec and 100msec whether the number of light circuits is greater than the number of heavy circuits when there is at least one heavy circuit.

We considered 15 minute windows of time at 6am, 6pm, 12am and 12pm for different overlapping burst lengths of 50msec, 70msec, 80msec, and 100msec. In the 6am interval the percentage of cases where number of light circuits is greater than the number of heavy circuits is 3%. In the 6pm, 12am and 12pm intervals, the percentage of cases are 6.9%, 0.79% and 4.68%. Thus the fraction of cases where bandwidth may be decreased for light circuits was between 0.79% to 6.9% in our experiment. As shown in the previous section, even with this tradeoff the scheme still showed an improvement with only one connection each.

Though this situation occurs less than 10% of the time, we could presumably reduce this fraction further by opening more connections; we discuss this possibility in the next section.

Once the feature presented in [19] is integrated with Tor, no light circuits should be affected as bandwidth will be divided equally amongst circuits irrespective of the connection they are on.

6.2.2 Extending to many connections

Since we get benefits from opening a single connection for light circuits, it is natural to ask whether we can achieve further benefit from opening even more light connections to decrease the share of bandwidth for bulk traffic. In theory, we could open many connections in order to achieve any desired fractional allocation of bandwidth.

One potential problem with extending the scheme to many more connections is that each node, instead of having a socket open to n, will then have up to kn sockets open if k connections are used to share bandwidth with each node. Reardon and Goldberg [15] observe that a large number of connections may cause compatibility problems, as some versions of Windows have a limit on the number of connections allowed, 3977 outbound concurrent connections for versions prior to Windows Vista, and 16384 on Vista and later [1]. We measured the number of connections used on our (vanilla) Tor node over five different 15-minute intervals and we found the maximum number of open sockets to be 171.

6.2.3 Backward compatibility

Our circuit-switching method requires every node in the circuit to understand the protocol; therefore, it will have little benefit until a significant number of nodes have been updated. However, there is little harm in deploying it incrementally, since nodes that do not understand the protocol will simply drop the control packets and the initiating node will refrain from attempting to switch again for that circuit. As and when nodes' software is upgraded to a version that supports Torchestra, this method will automatically start working. In order to get any benefits, the entrance, middle and exit nodes should all be running versions that support Torchestra otherwise the behavior will be same as before. Thus Torchestra is backward-compatible but no benefits will be seen unless it is supported on all nodes the circuit passes through.

6.2.4 Does Torchestra compromise security?

We must carefully consider whether changing the behavior of Tor might enable new attacks. We consider different cases below.

An adversary who is not a Tor node who is sniffing a link between nodes will be able to observe when a new connection is created and when a switch happens for the first time.

They may be able to make intelligent guesses about when circuits are switched based on changes in the traffic flow between connections, and estimate the number of packets sent by the switched circuit if it occupies a significant fraction of the overall bandwidth. However, this information is already visible to an attacker on the existing Tor network who looks for changes in traffic across a link to signal when *new* connections are created.

This type of information might enable a traffic-analysis or packet-counting attack, except that in order for these attacks to be effective, the adversary should also be able to observe the connections on the guard or exit node. But an attacker who can observe both ends of a circuit can already carry out a very effective packet-counting attack.

For nodes within the Tor network, the fact that a signal is sent through the circuit to switch connections gives the other nodes in the circuit an estimate of the traffic load on the old connection at the exit node, which may allow for some de-anonymization. An attacker in control of the guard and exit node of a circuit would likely be able to correlate the timing of the circuit switches, but again, such an attacker would have enough access to do a packet-counting attack.

One could also imagine a variant of the Murdoch-Danezis [14] active attack where the attacker creates circuits through specified nodes and attempts to force a targeted circuit to switch connections. There are two considerations here. The first is the question of whether this attack provides enough of a signal in the current Tor network; the original Murdoch-Danezis attack worked in a much smaller Tor and was recently shown that Tor has grown enough to render ineffective, although the attack could be modified to work in today's Tor network [10]. We must leave an analysis of this to future work, but we note that our scheme could be modified so that circuit switches happen independently at every node in order to protect against this attack.

A final potential attack, which does not compromise security but would circumvent all attempts by Tor nodes to prioritize or rate-limit traffic based on flows through each circuit, would be for clients to split large downloads among many individually light-traffic circuits.

7. FUTURE WORK

There are several avenues to explore in future work. The first is whether we can further improve Torchestra's performance for light traffic by opening additional connections. From our experiments, we observed that light traffic might benefit from additional bandwidth about 10% of the time; however, in order for every light circuit to get more bandwidth by opening additional connections, we will have to be careful to balance light traffic across the connections dedicated to it. The second modification to be studied in future work is the possibility of allowing every node to independently decide when to switch a circuit, rather than having only the exit node decide when to switch a circuit, as we do now.

Before deploying our scheme, it should be evaluated in realistic scenarios on the real Tor network. The wildly differing results we obtained using our different experimental methods suggest that simple downloads are not an accurate way of simulating Tor traffic. A study on the actual Tor network should also include a test of the security properties of Torchestra; particularly, whether any of the theoretical attacks we outlined are feasible in practice on the real network.

A final, more methodological question, is to accurately measure the amount of delay that Tor cells experience due to all of the potential factors in the network. With a framework for these types of measurements in place, this would allow us to more carefully evaluate different schemes for improving Tor's performance under different scenarios, and better understand their limitations and avenues for improvements.

8. CONCLUSION

In this paper we investigated whether Tor's performance on interactive traffic could be improved by separating light traffic from heavy traffic on different TCP connections between Tor nodes. We classified circuits as light or heavy using the exponentially weighted moving average of the number of cells on a circuit. In our experiments we measured the average delays for interactive traffic using a variety of methods: simple file downloads, replaying traffic with the same timing patterns as in the real Tor network, and replaying SSH and HTTP traffic collected from our own usage. With simple file downloads, we found that the prioritized Tor scheme has the least overall download time of the schemes we tested, although Torchestra had the lowest delay before receiving the first byte. Replaying our own captured HTTP and SSH traffic, we found between 8% to 32% reduction in delays with Torchestra compared to prioritized Tor and a 13% to 36% reduction in delays when compared to vanilla Tor. When we simulated traffic patterns using the real Tor network, we found between a 2% to 25% decrease in the delays with Torchestra compared to prioritized Tor and a 4% to 40% decrease in delays when compared to vanilla Tor. While there are several factors to be considered and carefully evaluated before deploying Torchestra on the real Tor network, these results suggest that the simple idea of separating very heavy Tor users onto separate connections may lead to real improvements in Tor's usability for most users.

Acknowledgements

We are grateful to Kevin Bauer, Roger Dingledine, and Damon McCoy for helpful pointers and discussion, and to Rob Jansen for pointing us to his work. This material is based upon work supported by the National Science Foundation under Award No. DMS-1103803 and the MURI program under AFOSR Grant No. FA9550-08-1-0352.

9. REFERENCES

[1] Maximum socket limit on Windows. `http://smallvoid.com/article/winnt-tcpip-max-limit.html`.

[2] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald. ExperimenTor: A testbed for safe and realistic Tor experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011.

[3] A. Chaabane, P. Manils, and M.A. Kaafar. Digging into anonymous traffic: A deep analysis of the Tor anonymizing network. In *Network and System Security (NSS), 2010 4th International Conference*, pages 167–174. IEEE, 2010.

[4] M. Chowdhury, M. Zaharia, J. Ma, M.I. Jordan, and I. Stoica. Managing data transfers in computer

clusters with Orchestra. *SIGCOMM-Computer Communication Review*, 41(4):98, 2011.

[5] B. Cohen. The BitTorrent protocol specification, 2008.

[6] G. Combs et al. Wireshark. `http://www.wireshark.org/lastmodified`, 2007.

[7] R. Dingledine. Research problem:adaptive throttling of Tor clients by entry guards. `https://blog.torproject.org/blog/research-problem-adaptive-throttling-tor-clients-entry-guards`.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[9] R. Dingledine and S.J. 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.

[10] N.S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of the 18th conference on USENIX security symposium*, pages 33–50. USENIX Association, 2009.

[11] D. Gopal. Torchestra : Reducing interactive traffic delays over Tor, Master's thesis, UC San Diego. Master's thesis.

[12] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the tor network. In *Privacy Enhancing Technologies*, pages 63–76. Springer, 2008.

[13] D. McCoy, K. Bauer, D. Grunwald, P. Tabriz, and D. Sicker. Shining light in dark places: A study of anonymous network usage. *University of Colorado Technical Report CU-CS-1032-07 (August 2007)*, 2007.

[14] S.J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium*, pages 183–195. IEEE, 2005.

[15] J. Reardon and I. Goldberg. Improving Tor using a TCP-over-DTLS tunnel. In *Proceedings of the 18th conference on USENIX security symposium*, pages 119–134. USENIX Association, 2009.

[16] SW Roberts. Control chart tests based on geometric moving averages. *Technometrics*, pages 239–250, 1959.

[17] P. Syverson, R. Jansen, and N.J. Hopper. Throttling tor bandwidth parasites. *Usenix Security*, 2012.

[18] C. Tang and I. Goldberg. An improved algorithm for Tor circuit scheduling. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 329–339. ACM, 2010.

[19] F. Tschorsch and B. Scheuermann. Tor is unfair–And what to do about it. In *Local Computer Networks (LCN), 2011 IEEE 36th Conference*, pages 432–440. IEEE, 2011.

[20] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Operating Systems Review*, 36(SI):271–284, 2002.

APPENDIX

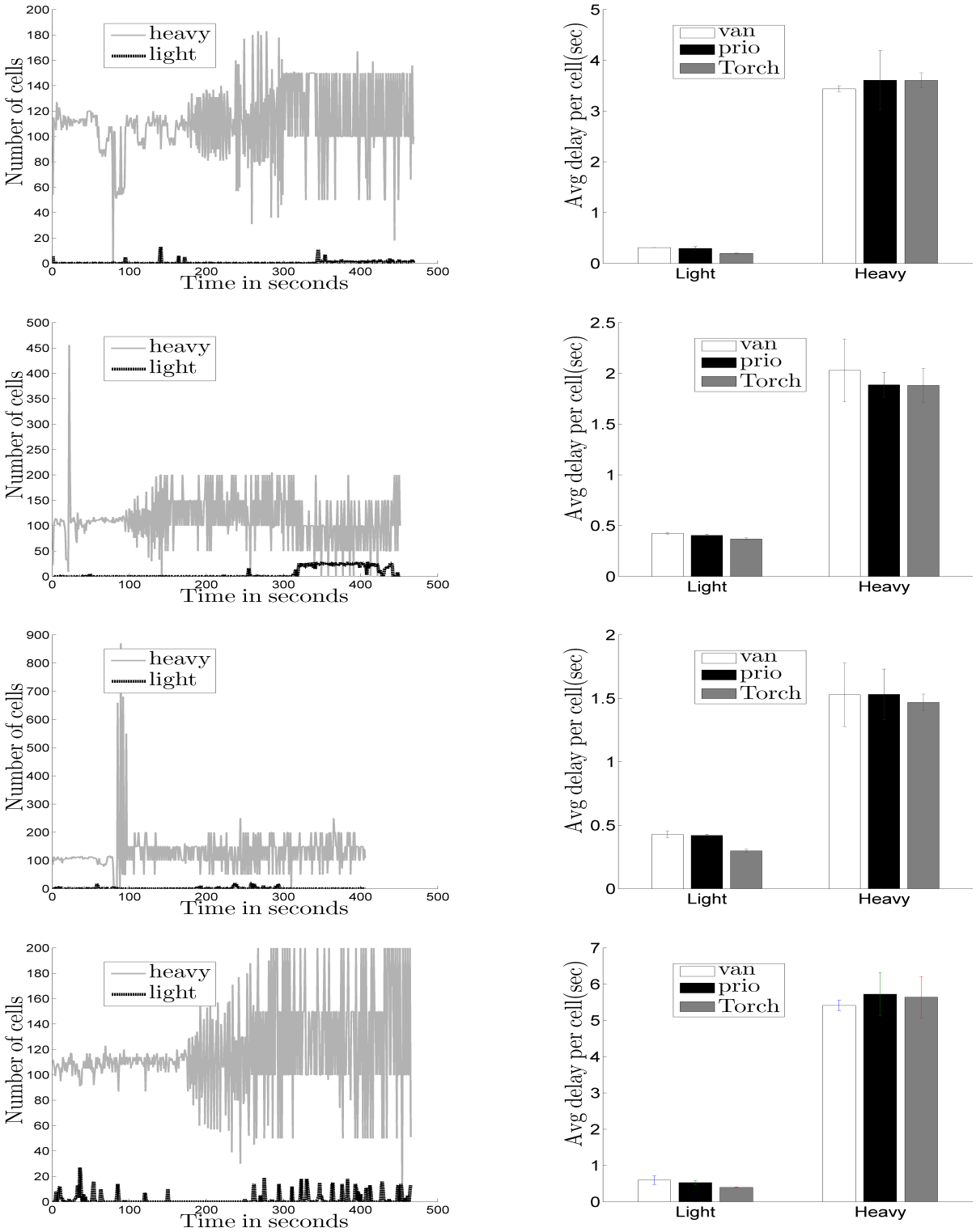


Figure 6: Experiment 2: Replaying web and SSH traffic. We collected samples of web and SSH traffic from our own usage, and replayed them through our ExperimenTor setup to compare vanilla Tor, prioritized Tor, and Torchestra. The left column shows the traffic pattern and the right column shows the average delay per cell. In contrast to the simple download experiment where prioritized Tor performed the best, Torchestra displays the least delay for light circuits in this experiment.

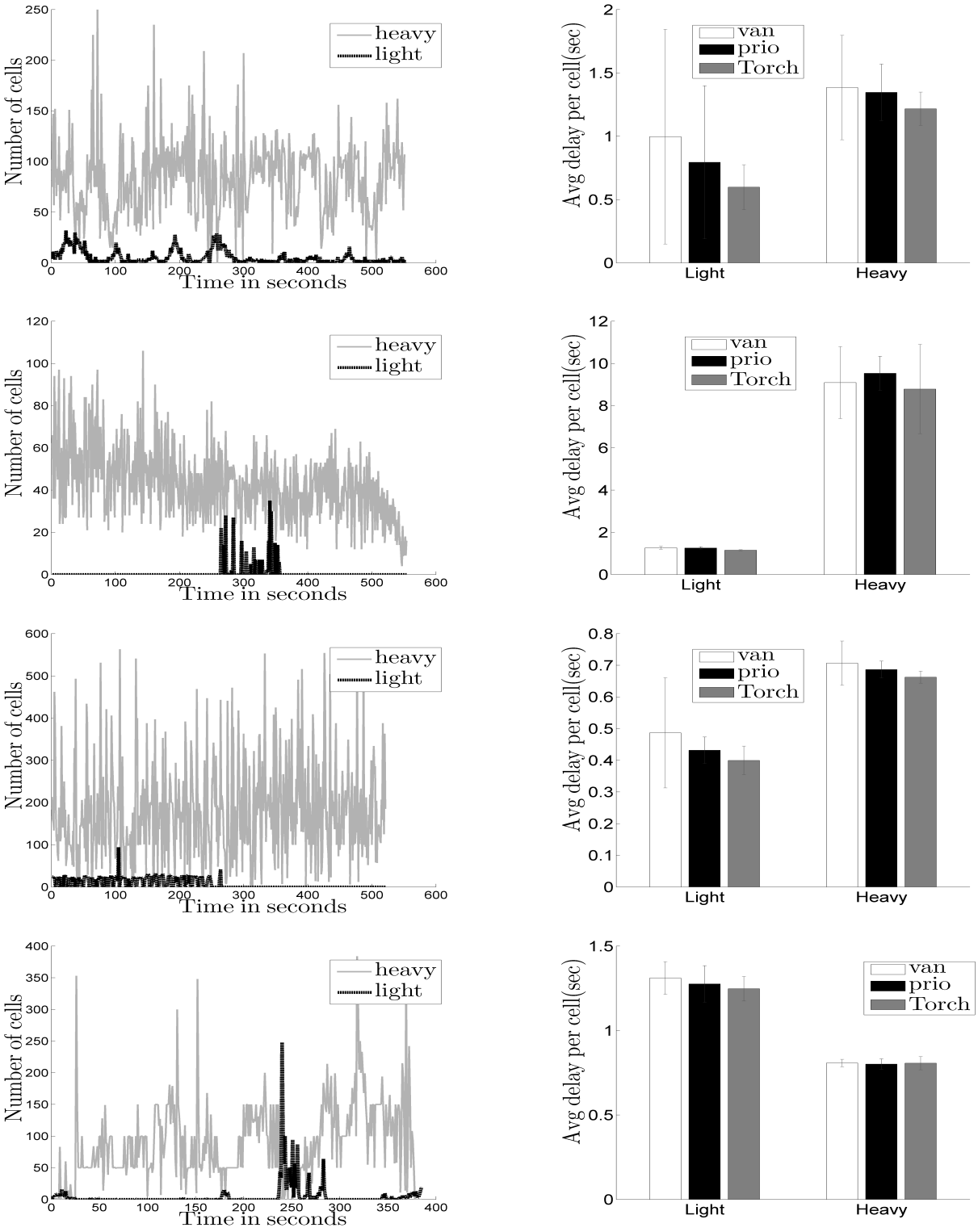


Figure 7: Experiment 3: Replaying Tor traffic. We collected timing information from a public, non-exit Tor node and used the timing data to simulate dummy traffic through our ExperimenTor setup. The left column shows the traffic patterns at 6am, 12pm, 6pm and 12am respectively, and the right column shows the average delays for vanilla Tor, prioritized Tor, and Torchestra on this data. In each case average delay per cell is the least for Torchestra.