UNIVERSITY OF CALIFORNIA, SAN DIEGO

Torchestra : Reducing interactive traffic delays over Tor

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Deepika Gopal

Committee in charge:

Professor Hovav Shacham, Chair
Professor Stefan Savage
Professor Geoff Voelker

2012

DEDICATION

I dedicate this thesis to my parents – Mangala and Gopal, my mother-in-law Ramā, my husband Aravind and my sister Vandana. Without them, none of this would have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Nadia Heninger for her guidance and advice throughout my thesis. Her passion and attention to detail has truly been an inspiration to me.

I would also like to thank Professor Hovav Shacham, Professor Geoff Voelker and Professor Stefan Savage for their feedback and agreeing to be on my thesis committee.

Kevin Bauer and Damon McCoy with their in-depth knowledge of Tor have been a great source of help throughout the project - a special thanks to them. I would like to thank Arjun Roy for answering all my Modelnet related questions. I am grateful to Brian Kantor and Cindy Moore for their assistance in setting up the Experiment testbed.

ABSTRACT OF THE THESIS

Torchestra : Reducing interactive traffic delays over Tor

by

Deepika Gopal

Master of Science in Computer Science

University of California, San Diego, 2012

Professor Hovav Shacham, Chair

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 on the internet including interactive traffic as well as for bulk file downloads. Such bulk downloads cause delays for interactive traffic as all traffic between a pair of Tor nodes goes over a single connection. The resulting delays discourage people from using Tor for normal web activity.

We propose a potential solution to this problem called Torchestra which separates interactive and bulk traffic onto two separate TCP connections between any pair of nodes. We classify a circuit as carrying either type of traffic based on the Exponentially Weighted Moving Average of its number of cells. 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 [9] and up to 40% decrease in delays when compared to unprioritized Tor.

Chapter 1

Introduction

Tor is an anonymizing network designed by Dingledine, Mathewson and Syverson in 2004 [12] that provides privacy and anonymity to users all over the world. Traffic from clients is relayed through three Onion Routers (ORs) before being forwarded to the destination. 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. In simple terms, Tor's goal is to prevent an attacker from linking together the source and destination IP addresses of clients and learning their browsing habits. 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.

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 [26]), delays on interactive traffic (such as web traffic, ssh) increases as explained by Dingledine and Murdoch in [1]. This dissuades people from using Tor on a regular basis - when privacy is not essential. Since the level of anonymization improves with the number of people using it [31], reduction in delays for interactive traffic would definitely be an incentive to get more people to use Tor on a regular basis.

Tor is known to suffer from some performance issues as described by Dingledine and Murdoch in [1]. In this thesis we focus on two issues that occur due to traffic from all circuits going over a single connection between any pair of nodes.

In 2008, McCoy et al. [6] showed that interactive traffic makes up only 30% of the overall traffic. The first potential issue (explained by Dingledine in [5]) is that it is likely that interactive traffic will be queued behind a large amount of bulk traffic in socket buffers or on Tor's output buffers (the data structure on every connection on which Tor stores cells before pushing them out on the socket) and will thus lose a lot of time. The second problem (explained by Reardon and Goldberg in [7]) is that when TCP's congestion control is triggered on a connection due to bulk traffic, interactive traffic on that connection will also get unfairly slowed down. In this thesis we investigate whether Tor's performance for interactive traffic can be improved by separating it from bulk traffic on two separate TCP connections. Throughout the rest of the thesis, we will refer to interactive traffic as light traffic and bulk traffic as heavy traffic.

The approach we have adopted, which we call "Torchestra", is to create a separate connection each for interactive traffic and bulk traffic between any pair of nodes. We use a heuristic used for a related scheme by Tang and Goldberg in 2009 [9] to find the exponentially weighted moving average (EWMA) for the number of cells sent on a circuit. EWMA provides a way to calculate the moving average of the number of cells sent on a circuit while giving priority to recent values. Our implementation continuously updates the EWMA value for each circuit and transfers a circuit to the appropriate connection as dictated by the traffic flow. As explained later, moving a circuit between connections does not add any additional latency. We have attempted to ensure that the security properties of Tor are preserved. We discuss this further in Section 4.5.4 after we have explained our algorithm in more detail.

We evaluate our proposal using several experiments: simulating simple traffic patterns, replaying dummy traffic using timing patterns collected on a public, non-exit Tor node and replaying our own web and ssh traffic. We compare Torchestra to Normal Tor and the Prioritized Tor scheme of Tang and Goldberg [9] using the Tor emulator Experimentor, created by Bauer et al. [11]. When simulating the timing patterns of traffic collected on a non-exit Tor node, 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 Normal Tor. We replayed ssh and http traffic collected from our own normal usage and 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 Normal Tor.

In the forthcoming sections of the Introduction, we summarize who the users of Tor are, some of the problems Tor faces and the issues we intend to solve. The related work section is in Chapter 2. In Chapter 3 we discuss the prelimnaries and the Tor architecture. The algorithm and protocol used to decide when to switch between connections is explained in Chapter 4. In Chapter 5 we describe our experimental setup, the experiments we performed and our results. Lastly, in Chapter 6 we discuss future work and summarize our results.

1.1 Who uses Tor?

There are different reasons why people want their privacy to be protected. Since Tor prevents the source and destination IP addresses from being linked together, any scheme which requires the cloak of anonymity is benefitted by Tor. As described by Dingledine in [4], some of these reasons are

- People should be able to express their views freely and give their opinions without fear of retribution. Bloggers and journalists come under this category. It is also useful in any online voting scheme where it is important that the identity of the voter does not leak out.
- In countries where internet traffic is monitored and websites are blocked, Tor provides a way to circumvent this tracking and overcome censorship.
- Normal users would like to maintain their privacy and access websites of their liking without fear of being profiled and in the worst case, without fear of being blackmailed.
- Any situation that allows websites to unfairly differentiate between users based on their geophysical location (source IP), can be prevented by Tor. Since Tor traffic goes through many relays scattered all across the globe, the

IP address the website sees is the exit node's IP address. Thus the website cannot be sure whether the IP address seen is that of the source or of the last node on an anonymizing network, encouraging it to provide fair service.

1.2 Motivation

We describe here some of Tor's performance issues and the motivation behind our work.

1.2.1 Why Tor is slow

Dingledine and Murdoch explain in [1] that the Tor network is slow and we summarize the main reasons below:

1. Unfair slowing down of interactive traffic

   As described by Reardon and Goldberg [7] since all circuits between a pair of nodes go over a single connection, heavy circuits that trigger TCP's congestion control directly affect light circuits which are using the same connection. Thus light circuits suffer delays through no fault of their own.

2. Interactive traffic stuck behind bulk traffic

   As described by Dingledine in the Tor project blog [5], one of the main problems with Tor today is that some users use it for high-volume transfers. On a connection, Tor sends cells out from different circuits in round–robin order and so if all circuits had similar traffic patterns and user expectations, there would be no problem. In reality, there is a lot of disparity between the rates at which different circuits send data. With bulk traffic downloads, there is data going over a circuit for long periods of time and in this case, response time is not important to the user. With interactive traffic, a much smaller file like a website needs to be downloaded and the user expects almost instantaneous responses. The issue is that when a user wants to download a web page, these cells are stuck behind the bulk traffic cells on the socket buffer or Tor's output buffer. Thus when there is heavy load on the network, light traffic can experience very high delays.

3. Disproportionate bandwidth usage

   Some clients use more bandwidth than they contribute leading to reduction in quality of user-experience for clients whose usage pattern is fairer [1].

4. Directory download overhead

   Too much network overhead is spent in downloading directory information. This especially affects low-bandwidth users [1].

5. Imperfect path selection

   Sometimes, Tor's path selection algorithm does not distribute loads fairly amongst different relays leading to some relays being underloaded and some being overloaded [1].

6. Not enough capacity

   As Tor has many users compared to the number of relays, the overall bandwidth itself is sometimes not enough, leading to delays [1].

In this thesis we focus on the first and second issues as explained in the next section.

1.2.2 Issues Torchestra tackles

The main idea behind Torchestra is to separate heavy traffic from light traffic and thus prevent bulk traffic from increasing delays for interactive traffic. To achieve this, we create a separate connection for each type of traffic and decide which connection a circuit should belong to using the Exponentially Weighted Moving Average metric. We investigate whether this will lead to improvements in the first two issues mentioned in Section 1.2.1.

Both 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. For the first 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. Regarding the second problem, light traffic will no longer be stuck behind heavy traffic since the socket buffers and Tor output buffers for the two types of traffic will no longer be the same. Thus we see that having separate connections should lead to improvement in Tor's performance for light traffic.

Chapter 2

Related Work

In this chapter we discuss the work that has been done previously to reduce delays for interactive traffic in Tor and other work that is related to the thesis.

In 2009, Reardon and Goldberg [7] were the first to propose a scheme to improve Tor's performance for interactive traffic. As explained in Section 3.1.2, congestion control of TCP that is triggered due to heavy traffic unfairly affects light traffic. They addressed this problem by creating a separate socket for every client using user-level TCP and a DTLS/ UDP tunnel between every pair of nodes. In this manner, a heavy circuit can affect only itself - not other heavy circuits or light circuits. While this is an ideal solution, Dingledine explains in [1] that it has not yet been implemented due to licensing issues on most high quality user–level TCP stacks.

Tang and Goldberg in 2009 [9] proposed a scheme Prioritized Tor where they aim to reduce the delay of light circuits by giving higher priority to interactive circuits. They did this by using the Exponentially Weighted Moving Average (EWMA) to calculate the recent activity of a circuit - circuits that have a lower EWMA value and hence lower activity are given higher priority over other circuits that have cells ready to transmit. According to Dingledine [5], the problem with this approach is that since all circuits are using the same connection to send data to the next node, if there is already a lot 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.

Another option available for Tor nodes to control traffic is to use the PerConnBWRate and PerConnBWBurst configuration options described by Dingledine in the Tor project blog entry [5]. Both these options are used to do separate rate-limiting for 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 in Tschorsch and Scheuermann [2] 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. Also, clients who run relays and use more bandwidth than they contribute will still cause problems for interactive traffic.

Chowdhury et al. [13] 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 this was our inspiration to increase the number of connections for light circuits to ensure that they get a greater assured share of bandwidth. On further study we understood that Tor is already implementing 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 reasons described in Section 1.2.2. Since Orchestra was our original inspiration, we have named our project "Torchestra".

In the paper by by Tschorsch and Scheuermann in 2011 [2] the authors explain how 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 that have very few circuits, get a larger slice of the bandwidth. They implement a solution that achieves max-min fairness between circuits and uses a N23 congestion feedback scheme to better utilize bandwidth and prevent congestion. Later in the thesis, we explain why we have chosen one connection each for light and heavy circuits. According to Damon et. al [6], the percentage of circuits carrying interactive traffic is around 90% while percentage of circuits carrying heavy traffic is 10%. Thus the light connection will have much larger number of circuits and the bandwidth available to a light circuit in Torchestra will be less than the bandwidth available to it in Normal Tor. Using an experiment, we show that only in less than 10% of the cases do light circuits get affected by the reduction of bandwidth available to them. Once the feature presented in [2] is integrated with Tor, even 10% of the light circuits should not get affected as bandwidth will be divided equally amongst circuits irrespective of the connection they are on.

McCoy, Bauer, Grunwald, Kohno, Sicker in 2008 [6] have done a study of the real Tor network by looking at exit node traffic and have measured the percentage of light and heavy circuits as well as the percentage of light and heavy traffic. We use the results published in this paper to design some of our experiments and hence we mention the statistics here. They found that the percentage of circuits that carry web traffic is around 90% and the percentage of heavy circuits is around 10%. Regarding traffic, the percentage of light traffic is around 30% and the percentage of heavy traffic is about 70%. We will refer to these statistics throughout the thesis.

Chapter 3

Preliminaries

In this chapter we give an overview of TCP, Tor's architecture, Tor's algorithm to divide bandwidth and De-anonymization attacks on Tor.

3.1 Transmission Control Protocol (TCP)

The Transmission Control Protocol [20] is a connection-oriented, end-to-end protocol which provides reliable communication. It is the transport layer of the TCP/IP suite and lies between the Application and IP layers. Before any data is sent between two hosts, a connection needs to be set up end to end. TCP guarantees that all packets are received in-order, takes care of resending dropped packets and helps prevent network congestion. It is also responsible for dividing the available bandwidth fairly between connections using the max-min algorithm. In the sections that follow we describe TCP's congestion control algorithms and its max-min algorithm.

3.1.1 TCP's slow-start algorithm

We begin by discussing TCP's slow-start algorithm as detailed in RFC 5681 [19]. In order to prevent network congestion, TCP uses the slow start algorithm to start with a single segment which is the Maximum Segment Size initialized by the receiver in the connection establishment phase. It exponentially increases the number of packets it sends. Slow-start uses a window on the sender side called the congestion window `cwnd`. After a TCP connection is established with another host, `cwnd` is initialized to a single segment which is sent to the host. Once an acknowledgment is received from the host, the sender increments `cwnd` by a single segment. The next two segments get sent and once the two acks are received, `cwnd` becomes four and thus the `cwnd` exponentially increases. The sender can transmit up to the minimum of the congestion window and the advertised window which is the number of bytes the receiver advertises that it is ready to receive. This flow control is done to prevent socket buffer overflows at the receiver. If flow control does not kick in and packets are dropped due to discarding of packets at an intermediate router, this is a hint to the sender that `cwnd` is too large and congestion avoidance algorithm takes over.

3.1.2 TCP's Congestion avoidance algorithm

Congestion avoidance is used along with the slow start algorithm. While congestion avoidance slows down the rate of packets to avoid congestion, slow start helps in getting the process started again.

Two variables are used with both these algorithms, congestion window `cwnd` as mentioned previously and a slow start threshold size `ssthresh` initialized to 65535 bytes. When a new connection is created, the slow start algorithm begins as described above. This continues until size of the receiver's advertised window size is reached or till there is packet loss. Let us consider the case of packet loss. Packet loss is indicated by duplicate ACKs or timeouts. If duplicate acks are seen, one-half of the current window size is saved in `ssthresh`. If there is a timeout, `cwnd` is set to 1 segment which effectively enables slow start again. When new data is acknowledged by the other side, we could either be in slow start mode (`cwnd` is lesser than `ssthresh`) or congestion avoidance mode. If we are in slow start mode then slow start continues till `ssthresh` is reached after which congestion avoidance takes over.

If we are in congestion avoidance mode, `cwnd` is incremented more slowly by a single segment for each RTT.

When there is a single circuit that causes congestion on a connection, congestion-control will slow down the rate at which the circuit is sending cells. But in Tor, since all circuits that span a pair of nodes go over the same connection, even if one of the circuits cause congestion, all traffic on that connection and hence all circuits get slowed down. This is especially unfair to light circuits – circuits that are not sending much data.

3.1.3 TCP's max-min fairness

We give an overview of the max-min algorithm TCP uses to divide bandwidth between connections. The max-min fairness algorithm ensures that all connections get the same share of a bottleneck. If a connection does not completely use its share, the excess capacity is shared fairly amongst the remaining connections. In other words, a source that is not able to use more than one Nth of the bottleneck's bandwidth will always be able to send at its maximum rate [23].

In Figure 3.1, we see there are 4 TCP flows – flow x starts at Node 1 and ends at Node 3, flow w and y start at node 1 and end at 2 and flow z starts at node 2 and ends at node 3. Let the bandwidth of the links between nodes be 10 mbps. In this example let us assume that if any host could send at a faster rate, it would. Between nodes 1 and 2, flows w, x and y would get a rate of 3.3 mbps. Between nodes 2 and 3, flows x and z should get a rate of 5 mbps. But since x cannot send faster than 3.3 mbps, the extra 1.7 mbps of its share gets allocated to z. Hence z can send at a maximum rate of 6.7 mbps instead of 5 mbps. This way, all the bandwidth is allocated fairly and no bandwidth is left unutilized.

3.2 Tor Architecture

Tor stands for "The Onion Router" and is a distributed, low latency, anonymizing network designed by Dingledine, Mathewson and Syverson in 2004 [12] that provides privacy to its users. As described by McCoy et al. in [6], it is primarily used for web browsing, peer to peer traffic like Bittorrent [26], instant messaging, e–mail, Telnet and FTP.

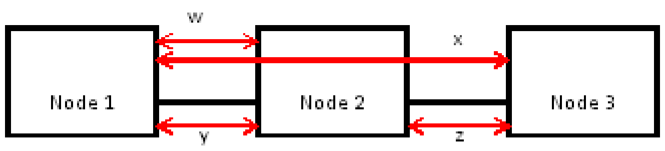


Figure 3.1: TCP max-min fairness example. Every pair of nodes has equal bandwidth links between them. Between Node1 and Node2, flows w, x and y get one-third of the bandwidth. Between Node2 and Node3, since x can only use one-third of the bandwidth, z can use up to two-third of the bandwidth.

Tor is able to provide this anonymity by sending data between the source and destination clients through a series of nodes running Tor software called relays or Onion Routers (ORs). Anyone can contribute to the system by acting as a relay. As of today there are more than 2000 relays forwarding Tor traffic.

Every client runs an Onion Proxy (OP) locally to accept TCP requests and send them through the Tor network. Before traffic can be sent out, a path to the destination needs to be created through 3 nodes called a circuit. As shown in Figure 3.2, a circuit has been created that links the Source client to the Destination client through the entrance, middle and exit nodes. The entrance node is the one that has a connection to the source client. The exit node has a connection to the destination client and the middle node sits between the entrance and exit nodes. Between every pair of nodes, a TLS connection is set up.

The creation of the circuit is done by the Onion Proxy on the client. Only the client is aware of all nodes along the circuit. It decides which nodes should be the entrance, middle and exit nodes through information received from directory servers. Directory servers are a group of trusted, well-known Tor nodes that keep track of the current network and node state. They maintain a list of active nodes and the clients can download this list. Once the client has decided on the three nodes, it sets up a connection with the entrance node and negotiates a secret key. Negotiation of the secret key between the client and each of the nodes is done

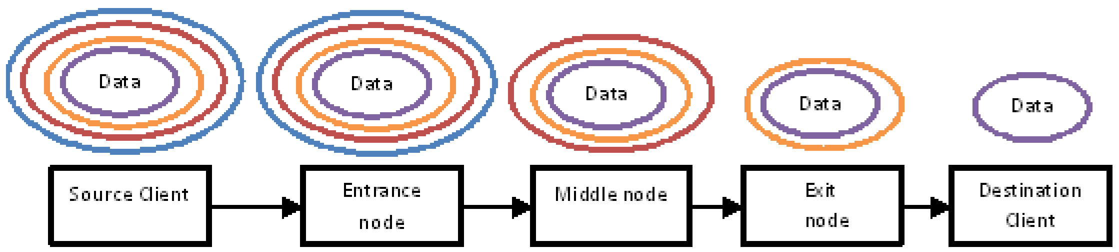


Figure 3.2: When a cell comes in on a node in the outwards direction (from the source to the destination), decryption is done once and passed on to the next node. The exit decrypts the last layer to obtain the original data. This is then sent to the destination.

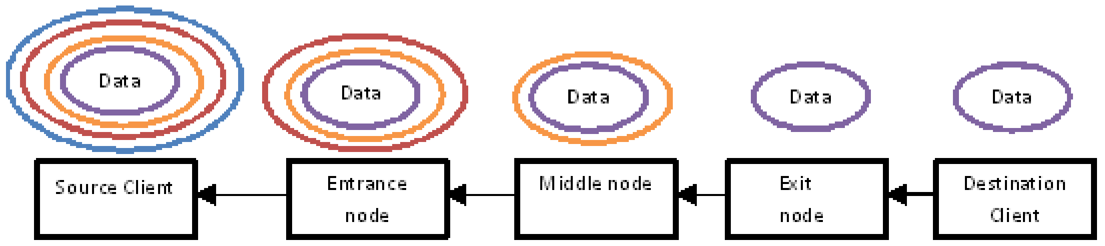


Figure 3.3: When a cell comes in on a node in the inwards direction (from the destination to the source), encryption is done once and passed on to the next node. The client receives the cell that has been sequentially encrypted by each of the Tor nodes. It then uses the secret keys it shares with each of the nodes to decrypt the cell thrice and obtains the original data

using the Diffie-Hellman algorithm [28]. In order to extend the connection to the middle node, the Onion Proxy sends an extend message to the entrance node with the middle router's address. The entrance router then sets up a connection to the middle router and the secret key negotiation between the client and the middle node is done through the entrance. In this way, the middle node is ignorant of the address of the client. The extension of the circuit to the exit node is done in a similar manner – the middle node creates a connection to the exit and the secret key is negotiated through the entrance and middle nodes. At the end of the creation of the circuit, the client has shared secret keys with each of the nodes.

Data is sent through the Tor network using fixed sized chunks of data of 512 bytes called cells. Each of these cells has a header that contains the circuit id. When a cell arrives at a node, this incoming circuit id is used to look up the output connection on which it is to be forwarded and it is replaced with the outgoing circuit id. In this manner, each of these nodes know of the previous node and the next node but do not have knowledge of any of the other nodes. The three hop design ensures that a node that is connected to one of the hosts does not even know of the node connected to the host at the other end.

As seen in Figure 3.2, the cell which is moving in the outwards direction from the source client to the destination is encrypted thrice by the Onion Proxy with each of the three secret keys it shares with each node. Every node decrypts the cell once before passing it on to the next node. The exit node decrypts the last layer to obtain the original data which it then forwards to the destination client. The destination has no knowledge that data has travelled through the Tor network. If a secure protocol like https is not being used, the exit node is in a position to look at the unencrypted data passing through it. Thus we see that since each node peels away a layer of encryption, this is called Onion Routing.

As seen in Figure 3.3, when traffic is travelling in the inwards direction towards the source client, one layer of encryption is added by every node. Since the Onion Proxy on the client has all the three shared keys, it decrypts the cell three times to obtain the original data.

Tor uses special control cells for different purposes as described in the Tor specification [21] like creating or destroying a circuit, flow control, etc. Any cell, be it control or data contain the following fields – the circuit id of the circuit, a command field specifying the type of cell, a length field which specifies the length of the payload and finally the payload itself. The structure of the cell is shown in Figure 3.4. For instance, to create a new circuit, the Onion Proxy sends a `CREATE` cell to the entrance node. The command would be "CREATE" and the payload would contain the first half of the Diffie-Hellman handshake. The length and circuit id fields would contain values as specified above.

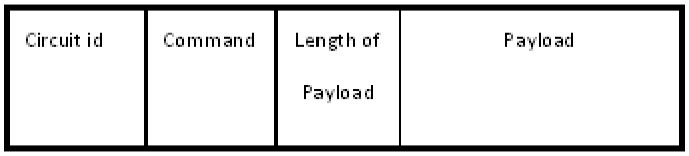


Figure 3.4: Structure of a Tor cell

3.3 Tor's algorithm to divide bandwidth

There are configuration options available to specify the bandwidth Tor is allowed to use of the node's bandwidth. Tor divides this user-configured bandwidth equally amongst all connections and implements a max-min algorithm so that if any of the connections are not making use of their fair share, the extra bandwidth can be used by connections that need it.

For different circuits sharing the same connection, a similar max-min scheme is followed where cells from different circuits are pushed out on the connection using round-robin scheduling. Thus all circuits that have data to send get equal share of the connection's bandwidth and effectively, bandwidth that would have gone to inactive circuits gets redistributed amongst the active circuits.

3.4 De-anonymization Attacks

Tor's goal is to provide complete anonymity to internet users. Without an anonymity network, even if the data is encrypted, an attacker who can monitor network traffic can keep track of the websites a client is accessing as the source and destination IP is in the clear. This can lead to de-anonymization of users. By sending data through a series of three relays that only know about the previous and next nodes, Tor overcomes these issues and provides anonymity against a non-omnipotent adversary. But there are still certain attacks that can be performed that break Tor's anonymity. We mention two of them here.

The first kind of attack discussed here is the cell-counting attack. If an adversary has control over all three Tor nodes that a circuit passes through, then the client can easily be de-anonymized. The Tor threat model therefore assumes that the adversary does not have a universal view of all traffic. A slightly more subtle attack is possible if the adversary controls the entrance and exit nodes that a circuit passes through. The way Tor works is to send data over TCP in fixed sized cells of 512 bytes. If data less than the cell size is available, extra bytes are padded to the cell size and sent over the network. As in other mix networks such as Mix-Net [15] and Mix-Master [16], Tor does not send dummy cells to fool an adversary nor does it intentionally add any delays before sending out a cell. Due to this reason, Tor is vulnerable to packet counting attacks if the adversary has control over the exit and entrance nodes. Such packet counting attacks have been discussed by Serjantov et al. in [29]. Since the number of cells leaving the exit node is equal to the number of cells entering the entrance node, if an adversary has control over more than one node, he can with very high probability recognize when a circuit uses both his nodes as an exit and an entrance. Since the exit node knows the destination and the entrance node knows the source, the anonymity of the client will be compromised. We did a simple packet-counting experiment over Experimentor on our local network where different clients download files from different destinations over Tor. In each of these cases we were able to match source and destination clients with 100% success. In order to prevent this type of attack, Tor will need to add delays or dummy cells. But this would add delays to interactive traffic and since Tor is a low-latency system, it does not protect against such a global adversary.

Another type of attack as shown by Murdoch and Danezis [17] uses the property that if high-volume traffic is pumped through a server, latency of all other traffic going through that server increases. Using a non-global adversary colluding with a malicious server, the adversary can, with high probability correctly guess which nodes a circuit passes through. The way this works is that the server sends data in a certain pattern and the adversary polls Tor nodes to check which nodes' traffic patterns match that of the server's. The nodes that match are most probably the ones used by the circuit. Though anonymity decreases, this kind of attack does not identify the source client.

Though there are a couple of attacks possible, Tor makes a trade-off between performance and latency and is one of the most widely used anonymity networks today.

Chapter 4

Our Algorithm

In this chapter, we describe the algorithm used to classify a circuit as light or heavy and the protocol used to move it from a light connection to a heavy connection or vice-versa. Between every pair of Tor nodes, we maintain two connections, one for light circuits and the other for heavy circuits. We classify a circuit as light or heavy based on the Exponentially Weighted Moving Average (EWMA) of the number of cells on the circuit. When the EWMA value crosses a certain threshold, the circuit is moved to the appropriate connection.

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 and we want only one node to be responsible for switching a circuit.

For a chosen window of time of two seconds, statistics are collected about the number of cells sent on a connection as well as the number of cells sent on each individual circuit. Only circuits using this node as an exit node are considered. As in Tang and Goldberg [9] we want to maintain a metric for "how many cells a circuit and the connection have sent recently" and 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 (EWMA)

EWMA is a statistic used to calculate the average value over a number of periods T while giving more weightage to recent data. This has been established by Roberts, 1959 [22]. The choice of the multiplier will determine to what extent changes in recent data affect the average value. The formula used to calculate the EWMA value at time t is given by:

$$EWMA(t) = \alpha.Y(t) + (1 - \alpha).EWMA(t - 1)$$

where $EWMA(t)$ is the EWMA value at time t, $Y(t)$ is the data observation at time t and α is the multiplier that determines the depth of memory of EWMA, it lies between 0 and 1.

There is a direct correlation between the number of periods T over which the moving average is calculated and the multiplier α value. This is given by the formula:

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

When period $T = 1$, α will also be 1 and hence $EWMA(t)$ will be set to the most recent value. The higher the multiplier, the larger the influence $Y(t)$ has on the EWMA value. Hence, in order to smooth out the effects of bursts of data, a lower multiplier value is chosen.

Initially, the Simple Moving Average (SMA) is calculated over the first T periods. The Simple Moving Average refers to the average calculated over a certain number of periods T, where every value has equal weight. From period $T + 1$, the EWMA value is calculated by applying the above formula. Consider an example where we want to find the EWMA for the number of cells a circuit is sending over a number of time periods. Let the number of periods $T = 3$.

1. First we calculate the multiplier $\alpha = \frac{2}{3+1} = \frac{2}{4} = 0.5$
2. Next we find the Simple moving average at the third period
3. From the 4th period onwards, the EWMA value is calculated using the above formula

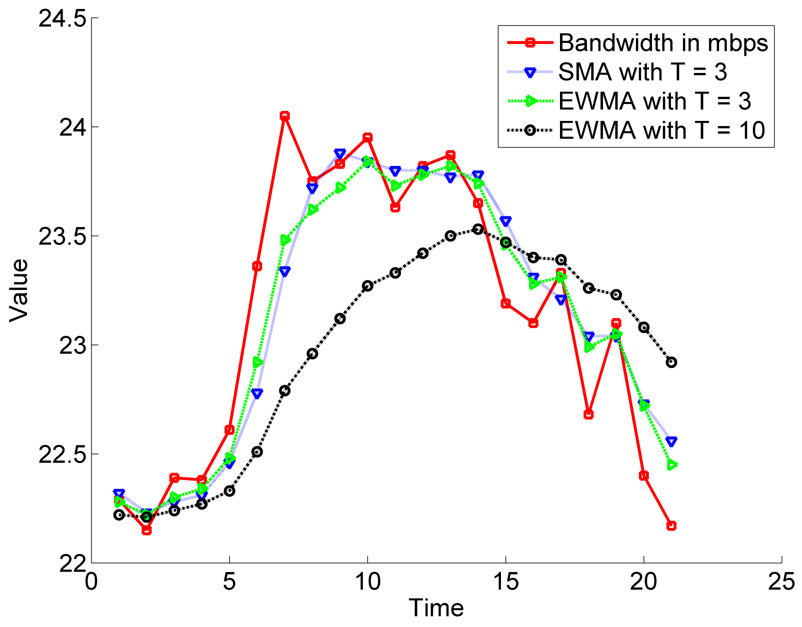


Figure 4.1: Comparison between, SMA and EWMA with different multipliers. EWMA with $T = 3$ is more sensitive to changes in bandwidth compared to SMA with the same T. EWMA with $T = 10$ smoothes the bandwidth variations.

Figure 4.1 illustrates the Simple Moving Average and EWMA for different parameters on a sample function [33]. We show the corresponding EWMA values with $T = 3$ or $\alpha = 0.5$ and $T = 10$ or $\alpha = 0.18$. The Simple Moving Average values are calculated with $T = 3$. Since the Simple Moving Average gives equal weightage to all values, we see that it changes much slower with fluctuations in bandwidth compared to the corresponding EWMA values with $T = 3$. When T is 10 we see that the changes in bandwidth are smoothed out.

4.1.2 Circuit classification algorithm

We now describe our algorithm which is used to decide whether a circuit should be classified as light or heavy.

Let the cells that have come in on the connection during a time window be N_{conn} and the cells that have come in on the circuit during the same time window be N_{circ}. Let the old and new EWMA values on the connection be $EWMA_{oldconn}$

and $EWMA_{newconn}$ and let the old and new EWMA values on the circuit be $EWMA_{oldcirc}$ and $EWMA_{newcirc}$. When the time window expires, the EWMA on the connection is calculated as follows:

$$EWMA_{newconn} = \alpha.N_{conn} + (1 - \alpha).EWMA_{oldconn}$$

Similarly, for every circuit belonging to this connection, the EWMA on the circuit is calculated as follows:

$$EWMA_{newcirc} = \alpha.N_{circ} + (1 - \alpha).EWMA_{oldcirc}$$

If every circuit on the connection were contributing equally towards the number of cells on the connection, then suppose there are 'n' circuits, every $EWMA_{newcirc}$ should approximately be $\frac{1}{n}(EWMA_{newconn})$.

We classify a circuit as heavy when the circuit's $EWMA_{newcirc}$ is greater than $\frac{1}{n}(EWMA_{newconn})$ and crosses a certain threshold i.e., it is sending that many more cells than it should be sending. We call this threshold $Thresh_{light}$.

Similarly, for circuits on the heavy connection, we classify a circuit as light when its $EWMA_{newcirc}$ is less than $\frac{1}{n}(EWMA_{newconn})$ and is lower than a certain threshold. We call this threshold $Thresh_{heavy}$.

In order to prevent continuous switching of a circuit between a light and heavy connection, a check is done before switching to make sure the $EWMA_{newcirc}$ is not too low for the heavy connection or not too high for the light connection. If these checks do not pass, the circuit is not switched.

On a light connection, if a circuit is very light, i.e., its $EWMA_{newcirc}$ is much lower than than $\frac{1}{n}(EWMA_{newconn})$, then it is not counted towards the total number of circuits on that connection. If a circuit is unnecessarily counted as active even when it is not sending any cells, the other active circuits will appear heavier than they are and an attempt will be made to move them to the heavy connection. We call this threshold $Thresh_{verylight}$.

Even when a circuit stops sending, since the EWMA for every circuit is calculated on a connection when the time window expires, the EWMA of a circuit will keep dropping, until it is removed from the total circuit count as explained in the preceding paragraph. In order to expedite this process a higher multiplier β is used when a circuit does not send any cells in a time interval,. This is again done to prevent unnecessarily counting a circuit in the total circuit count. The algorithm will work the same way for cells going in the inwards and outwards direction.

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 protocol for the switch is started. When a circuit is being switched from a light connection to a heavy connection, a `SWITCH` cell is first sent out on the light connection indicating that further cells belonging to this circuit will come in on the heavy connection. The first cell sent on the heavy connection for this circuit is a `SWITCHED_CONN` cell followed by the rest of the cells. Once the middle node has received both these control cells, it starts sending cells to the exit on the heavy connection after sending a `SWITCHED` cell on the light connection. This `SWITCHED` cell informs the exit that the middle node has completely switched to the heavy connection and that the exit can also complete the switch. We will now go through each of the steps in more detail when a circuit is to be switched from a light connection to a heavy connection. The protocol is demonstrated in Figures 4.2 and 4.3.

1. A check is done to see whether a connection marked as heavy exists from the exit router to the middle router. If not, a new connection is created.

2. Once the heavy connection has been set up, the exit sends a `SWITCH` cell on the light connection which informs 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 which the exit node sets, to inform the middle node that it needs to extend the heavy connection towards the entrance

3. The exit then sends a `SWITCHED_CONN` cell on the the heavy connection followed by the circuit's cells. The reason a `SWITCHED_CONN` cell is required is because cells on the heavy connection may arrive before all the remaining cells on the light connection have been processed and this will lead to cells

being processed out of order and dropping of cells. One thing to note is that the exit continues to receive cells from the middle node on the light connection.

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 have arrived on the heavy connection on the middle node 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. Thus the circuit has completely switched connections on the middle node.

5. When the exit node receives the `SWITCHED` cell on the light connection, this is its cue that 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 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. Thus cells are processed in the correct order.

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.

Thus we see that while doing the switch, at no point is traffic stopped. Suppose 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 switch again. Thus, a maximum of one extra cell per circuit will get sent. This is only in 10% of the circuits as these many circuits have been found to carry bulk traffic.

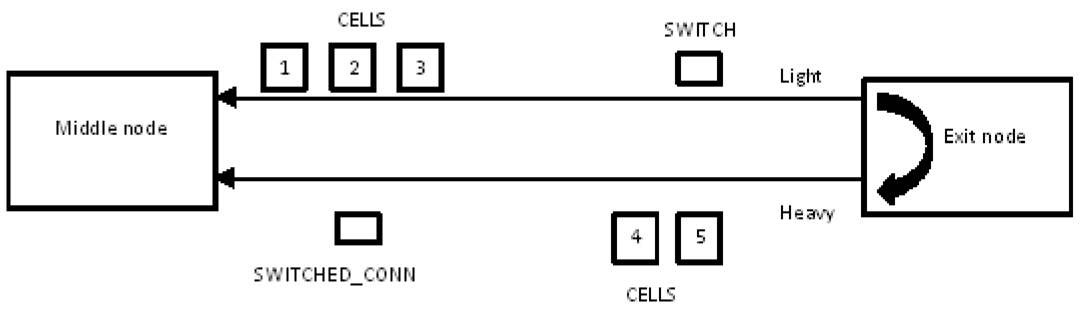


Figure 4.2: This depicts the Torchestra protocol when cells are travelling in the inwards direction from the destination to the source or in this figure, from the exit node to the middle node. No more cells are sent on the light connection after the `SWITCH` cell has been sent. The circuit is then switched to the heavy connection – a `SWITCHED_CONN` cell is sent on this connection followed by further cells.

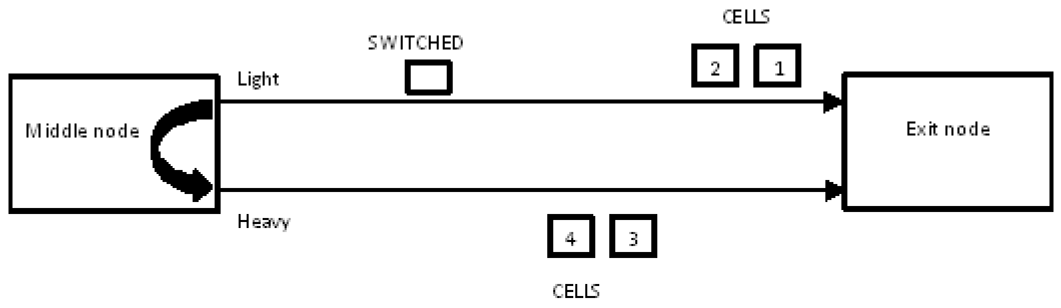


Figure 4.3: This depicts the Torchestra protocol when cells are travelling in the outwards direction from the source to the destination or in this figure, from the middle node to the exit node. No more cells are sent on the light connection after `SWITCHED` cell has been sent. The circuit is then switched to the heavy connection and further cells are sent on this connection.

4.3 Structure of control cells

The header in each of these control cells is no different from the cell header structure described in 3.2. `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. When the exit sends the `SWITCH` cell, this flag is set implying that the middle node needs to create a new connection to the entrance if one does not exists and then switch the circuit appropriately. But when the middle node sends the `SWITCH` cell to the entrance, the flag is not set ensuring that the entrance does not create a new connection or switch any circuit on the connection going to the client.

4.4 Tuning different parameters

When deciding on whether a circuit should be classified as light or heavy, the following parameters can be tuned. We explain what each of these values are:

The multiplier α ($0 <= \alpha <= 1$)

This value is the multiplier used for the calculation of EWMA for circuits and connections. If this value is very high, bursts of cells will cause light circuits to unnecessarily be moved to the heavy connection. On the other hand, the lower the value, the longer it takes to move a heavy circuit to the heavy connection. We have chosen a relatively low value since we do not want to unnecessarily move a light circuit to a heavy connection. Even if a light circuit does prematurely switch, eventually it will get moved back. But we want to avoid unnecessary switches because during the period the light circuit is on the heavy connection, its delays will be high. With this value, even though a circuit takes longer to be moved to the heavy connection, once it is moved, there is a higher probability that it is truly a heavy circuit.

The multiplier β ($0 <= \beta <= 1$)

This value is the multiplier used for the calculation of EWMA for light circuits that do not receive any cells in a time interval. The rationale behind choosing a different value from α is that, we want to more quickly decrease the EWMA value of circuits that are no longer sending any cells, so that their EWMA value drops below $Thresh_{verylight}$ which is the threshold for counting a circuit as active. If a circuit is unnecessarily counted as active even when it is not sending any cells, the other active circuits will appear heavier than they are and an attempt will be made to move them to the heavy connection.

Time Window

After a period of time expires, the EWMA values for the connection and all the circuits on it are calculated. We have selected this to be 2 seconds.

$Thresh_{light}(0 < Thresh_{light} <= 100)$

When the ratio of a circuit's EWMA value and the connection's EWMA value is $Thresh_{light}$ percent greater than what it should be, the circuit is moved to the heavy connection. We have set this value to be 70.

$Thresh_{heavy}(0 < Thresh_{heavy} <= 100)$

When the ratio of a circuit's EWMA value and the connection's EWMA value is $Thresh_{heavy}$ percent lesser than what it should be, the circuit is moved to the light connection. We have set this value to be 70.

$Thresh_{verylight}(0 < Thresh_{verylight} <= 100)$

When the ratio of a circuit's EWMA value and the connection's EWMA value is $Thresh_{verylight}$ percent smaller than expected, the circuit is no longer counted as active. The reason we have this threshold is that if a circuit is unnecessarily counted as active even when it is not sending any cells, the other active circuits will appear heavier than they are and an attempt will be made to move them to the heavy connection. We have set this value to be 90.

4.5 Performance Considerations

In the following subsections we discuss 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.

4.5.1 Bandwidth sharing for different traffic patterns

Between a pair of nodes, it may so happen that there are only light or heavy circuits. In this case, no circuits need to be switched and hence no extra connection will get created. Due to TCP's and Torchestra's max-min algorithm, the bandwidth between a node that has a single connection with a second node will be half of the bandwidth this node shares with a third node when it has two connections to it. It may appear that the circuits in the latter case enjoy more bandwidth compared to the circuit in the former. But even in the case with two connections, the bandwidth available to any circuit going between the two nodes is limited to the bandwidth of the connection it is on. Thus we see that the bandwidth available to circuits does not get affected even when there is only a single connection between two nodes.

4.5.2 How many light connections should we open?

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. One might also ask whether having a single connection reduces bandwidth available for light circuits. As described by McCoy et al. in [6], the percentage of circuits that carry web traffic is around 90% and the percentage of circuits that carry Bittorrent traffic is around 10%. Hence one might expect the greatest improvement for light traffic to be obtained if the ratio of the number of light connections to the number of heavy connections is 9:1. This would ensure that light circuits continue to enjoy bandwidth in proportion to the number of circuits.

The issue is that instead of having only one socket open to each of the other Tor nodes it is connected to, every node will now have ten connections to the aforementioned nodes and hence the total number of connections increases ten times. As explained in Reardon and Goldberg's paper [7], certain versions of Windows have a limit on the number of connections allowed. According to [30] and Microsoft forums [32], OS versions before Microsoft Vista support up to 3977 outbound concurrent connections for each IP address. On Vista and further versions, 16384 outbound connections are supported. We measured the number of Tor connections over five different 15-minute intervals and we found the maximum number of sockets to be 171. Increasing this by 10 times is almost half of the maximum number of connections supported on versions before Vista and such a solution may not be scalable going forward. We did a study to determine what the ratio of light connections and heavy connections should be so that light circuits do not suffer, as well as to ensure that minimum number of connections are opened.

Over a period of time according to McCoy et al. [6], light traffic makes up around 30% of overall data and heavy traffic makes up around 70% of overall data. When we consider a single connection each for light and heavy circuits, the bandwidth available to light circuits is reduced by 50%. But previously, on average since the light circuits were using only 30% of the bandwidth, they should still have more bandwidth than required.

The one case light circuits may have less bandwidth in Torchestra is when the number of light circuits exceed the number of heavy circuits. 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. When a connection each is used for light and heavy circuits, the bandwidth available to each connection is reduced. So suppose there are bursts of time when the number of light circuits is greater than the number of heavy circuits, in Torchestra this would mean that number of circuits on the light connection is greater than the number of circuits on the heavy connection. As explained by Tschorsch and Scheuermann [2] 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 experiment described in Section 5.2.1. Our results show that this situation occurs less than 10% of the time. Keeping in mind scalability issues and the advantages circuits obtain when a separate light connection is used, we have decided to use a single connection each for light and heavy circuits.

4.5.3 Backward Compatibility

The method we have suggested in this paper will be backwards compatible with Tor nodes using older versions of the software. When an attempt is made to switch a circuit to the heavy connection, the exit sends a `SWITCH` control cell to the middle node (described in 4.2). If the middle node does not support Torchestra, the cell will be dropped and the exit will not try to switch the circuit again. 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 backwards-compatible but no benefits will be seen unless it is supported on all nodes the circuit passes through.

4.5.4 Does Torchestra compromise security?

In Torchestra, circuits are divided between light and heavy connections based on the EWMA of their number of cells. Since the cells are TLS encrypted before they are sent out on a link, the circuit id is not in the clear and hence an adversary who is not a Tor node but is merely sniffing the link will not know when a circuit switches to a different connection, except for when a new connection is created and a switch happens for the first time. A concern may be that an attacker who is in control of the entrance and exit nodes of a circuit can de-anonymize it if a switch occurs on both nodes within a certain interval of time. But if an attacker has control over both entrance and exit nodes for a circuit, a packet-counting attack would work just as well and so Torchestra should not worsen the situation.

With Torchestra, when a switch happens the entrance node can come to certain conclusions regarding the traffic going through the exit node. Since the threshold is static and the entrance node knows the number of cells on the circuit that has switched connections, it can calculate the number of cells on the exit for the connection the circuit was previously on. Though we are not aware of any attacks this may cause, we mention this here for completeness.

In general, having fewer connections might reduce the security to that of a smaller Tor network. This needs to be studied further, but as most of the attacks this enables are equivalent to packet counting, it poses an interesting tradeoff.

Chapter 5

Experimental Setup and Results

In the following sections we explain our experimental setup, the experiments we run and our results.

5.1 Experimental Setup

In this section we describe our experiment's physical setup, our data collection methodology on the public Tor node and our method for replaying traffic.

5.1.1 Physical setup

We evaluate Torchestra using the Experimentor framework by Bauer et al. [11] which is a Tor emulation toolkit and testbed. Experimentor is built on top of Modelnet by Yokum et al. [14] and uses commodity hardware to simulate an entire network. Modelnet emulates distributed systems by allowing virtual nodes to be setup 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 5.1. The first machine is an edge node which runs the routers and clients as different virtual nodes by running them as separate processes and the second machine is the emulator – traffic between any two virtual nodes is forced to pass through it. Once all the virtual routers and clients are configured, Experimentor behaves as if each router and client is a separate node as shown in Figure 5.2.

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

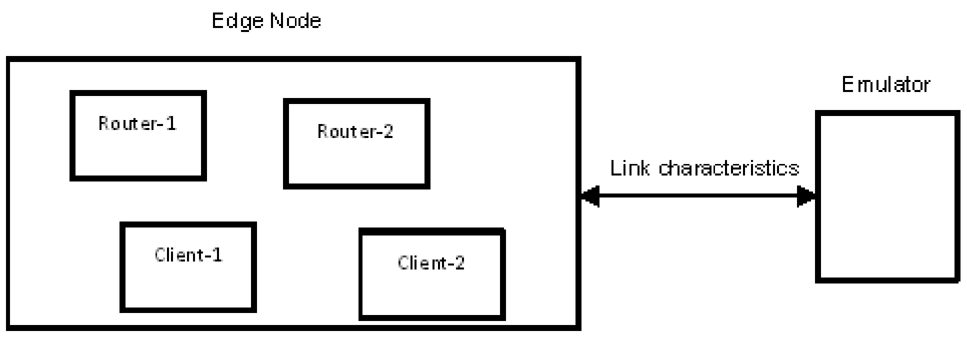


Figure 5.1: Experimentor Physical setup. Routers and clients are run as virtual nodes on the Edge Node and all traffic between these virtual nodes are forced to pass through the Emulator.

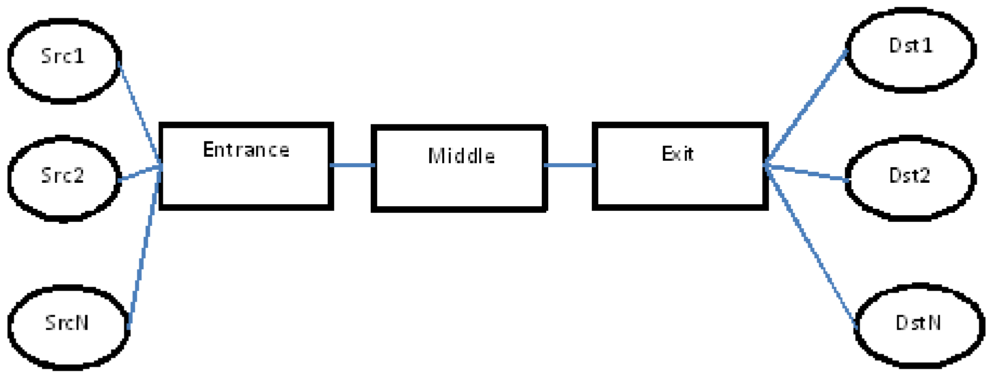


Figure 5.2: Experimentor Virtual setup

Though Experimentor allows simulation of the real network, our setup consists of 3 relays (which also act as directory servers) and multiple clients. Since we will be comparing interactive traffic delays with different features switched on, we have ensured that the setup is identical between runs.

5.1.2 Data Collection Methodology

In order to investigate the performance of our system on the real Tor network, 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, the socket number of the connection on which cells left our node so we know which circuits belonged to a particular connection and the time at which 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 timing information in the logs to send dummy data with the same time patterns and circuit distributions. After we finished plotting our graphs we have made sure that all logs have been securely deleted.

5.1.3 Method for replaying traffic

In two of our experiments – “Simulating real traffic” and “Simulating web and ssh traffic from the author’s computer” we simulate traffic patterns by sending data out at the same time intervals as the collected logs. Here are the steps we use to perform this simulation:

1. The time intervals that we want to simulate is fed to our java process.
2. A separate thread is created for every circuit id in the logs. Each of these threads is created with a different IP address and listens on different sockets.

3. The source clients created through Experimentor connect to each of these threads' sockets through Tor.
4. Every thread maintains a table for when a cell needs to be sent. After sending a cell consisting of dummy data, the thread sleeps for an interval equal to the current timestamp and the next timestamp in the table.
5. Here a cell consists of 498 bytes of dummy data. Tor adds 14 bytes of header information and a 512 bytes cell gets sent through the network.
6. Thus, every thread is sending data at time intervals relative to the real Tor network.
7. We measured the time it takes to switch between threads on our machine and it took 2.7 microseconds. Thus, the overhead due to switching between threads is negligible.
8. The time at which a thread sends a cell's worth of data and the time at which the source client receives a cell's worth of data is recorded
9. For every cell, the difference between these two time values is calculated - this gives the amount of time the cell took to travel through the network.
10. The difference calculated for all the cells is added up to get the overall delay for all cells and then the average delay per cell is calculated.
11. We compare this average delay across different modes.

5.2 Experiments

In this section we first describe the experiment we carried out to find the percentage of cases where the number of light circuits is greater than the number of heavy circuits explained in Section 4.5.2. We then run different types of experiments to compare interactive traffic delays on Torchestra, Normal Tor and Prioritized Tor by Tang and Goldberg [9].

5.2.1 Experiment to measure how many light connections we should open

Description

As we described in Section 4.5.2, 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 heavy circuits. In the following experiment, we measure how often this happens. Our experiment consists of the following steps:

1. Timing information was collected on the non-exit Tor node as described in Section 5.1.2.

2. For every connection, circuits that send more than 70% of the total cells are labeled as heavy while the rest are labeled as light.

3. Overlapping subintervals (bursts) of 50msec, 70msec, 80msec and 100msec are considered in the 15 minute interval.

4. In each of these sub-intervals, we checked whether the number of light circuits is greater than the number of heavy circuits when there is at least one heavy circuit. This is calculated as a percentage of all cases when there is at least one light cell.

Results

We considered 15 minute windows of time at 12am, 12pm, 6am and 6pm for different overlapping burst lengths (50msec, 70msec, 80msec,100msec). Traffic distribution for light and heavy circuits for the interval starting at 12pm is shown in Figure 5.3. 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

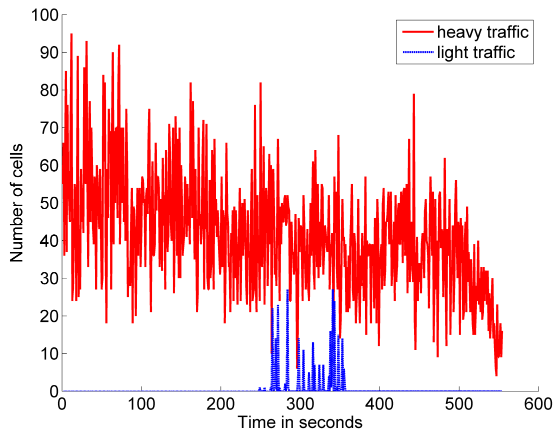


Figure 5.3: Graph depicting the traffic pattern at 12pm.

0.79% to 6.9% in our experiment. As shown in the following sections, even with this tradeoff the scheme still showed an improvement with only one connection each.

5.2.2 Simple files download experiment

Description

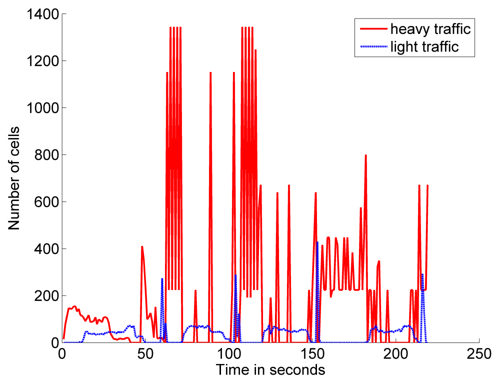
In this experiment, we simulate traffic by continuously downloading files of fixed size. All our experiments have been carried out on Experimentor as described in the previous section. In this experiment, we use 3 routers and 13 clients. Seven of these clients are heavy clients and each of them continuously downloads a huge file of 100MB through the course of the experiment which runs for 10 minutes. The other six clients are light clients each of which downloads a small file of 300KB with about 50 seconds of gap between each download. We have chosen this ratio of heavy clients and light clients in order to ensure that traffic from light clients is less than the traffic from heavy clients.

We compare between three modes – Normal Tor, Prioritized Tor and Torchestra. For light clients, in each of the modes, we compare the time it takes to download files as well as the time it takes to download the first byte. We have run each of the experiments 10 times and have taken the average. Parameters for each of the 3 modes are:

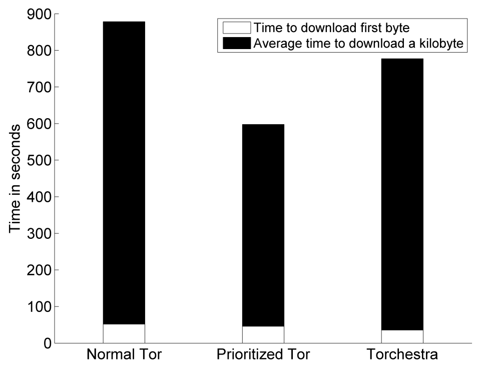
- In Normal Tor, we have not made any changes.
- For Prioritized Tor we have used H = 66 as done in [9].
- For Torchestra, we have tuned the following parameters -
  - α - 0.18
  - β - 0.18
  - LIGHT_PERCENT_UPPER_BOUND - 0.4
  - HEAVY_PERCENT_LOWER_BOUND – 0.7
  - Number of light clients - 6
  - Number of heavy clients – 7
  - Each light client downloads two 300KB files during the length of the experiment
- Between every pair of routers, the bandwidth has been rate-limited to 3mbps. For all other links, bandwidth has been rate-limited to 1 mbps.

Results

A graph of the light and heavy traffic patterns is shown in Figure 5.4 in (a). From the stacked bar graph in Figure 5.4 in (b) we observe that the average time to download a kilobyte is lower in Prioritized circuits. But we also see that the time to download the first byte is lower in Torchestra. This can also be seen in Figure 5.5 which is a plot of the cumulative sum of delays between cells on the source client. The reason this could be happening is that the initial packets required to start the download are stuck behind 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 will get sent faster.



(a) Graph depicting traffic pattern of heavy and light traffic



(b) Graph depicting time to download first byte and average download time per KB for light circuits. Average download time is lowest for Prioritized Tor but time to download the first byte is lowest for Torchestra.

Figure 5.4: Simple files download experiment

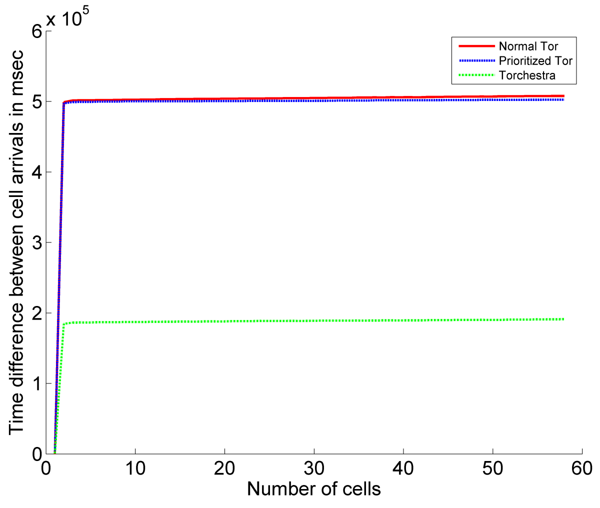


Figure 5.5: Graph depicting cumulative sum of delays between cells on a light client in the Simple files download experiment. Initial delay is lowest for Torchestra.

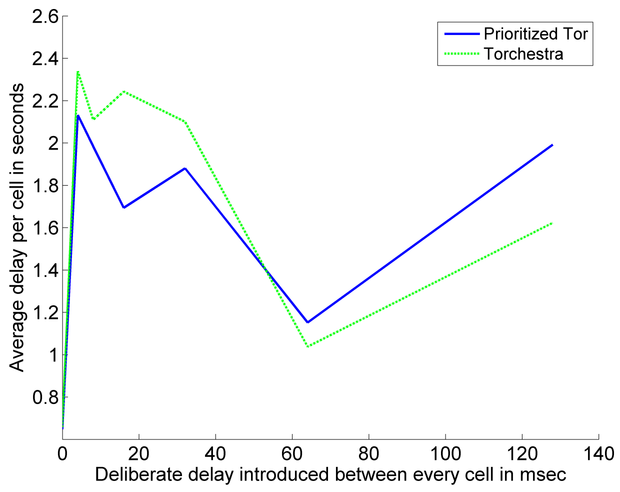


Figure 5.6: Graph depicting average delay per cell when deliberate delays are introduced in the Simple files download experiment. With up to 32 msec of deliberate delay, Prioritized Tor does better. For greater deliberate delays, Torchestra does better.

In order to test this, we carried out an experiment where we deliberately put varying delays from 4 msec to 128 msec in powers of 2 between sending for light circuits. We then found the average delay per cell to reach the source client from the destination. As seen in Figure 5.6, with deliberate delays of up to 32 msec between cells, Prioritized Tor does better but with greater delays, Torchestra does better.

5.2.3 Simulating web and ssh traffic from the author's computer

Description

The aim of this experiment is to use the web and ssh traffic patterns we generated using our own traffic as light traffic and compare the results between different modes. We performed this experiment in order to use more realistic light traffic in our experiments. Since http and ssh traffic are known to be interactive traffic, we wanted to see the kind of delays it would face in the presence of bulk traffic that we simulate using wget.

We recorded http and ssh traffic on our laptops using Wireshark which is a network protocol analyser [27]. We then ran the experiment as described in the Experimental Setup, Section 5.1 and we replayed the captured Wireshark traffic for four representative intervals of 10 minutes each. The replayed traffic represents light traffic and 4 clients continuously downloading 100MB files represents heavy traffic. We compared the results between Normal Tor, Prioritized Tor and Torchestra.

Results

In each of the four cases shown in Figures 5.7 and 5.8 we see that the average delay per cell in Torchestra is the least. 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 Normal 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.

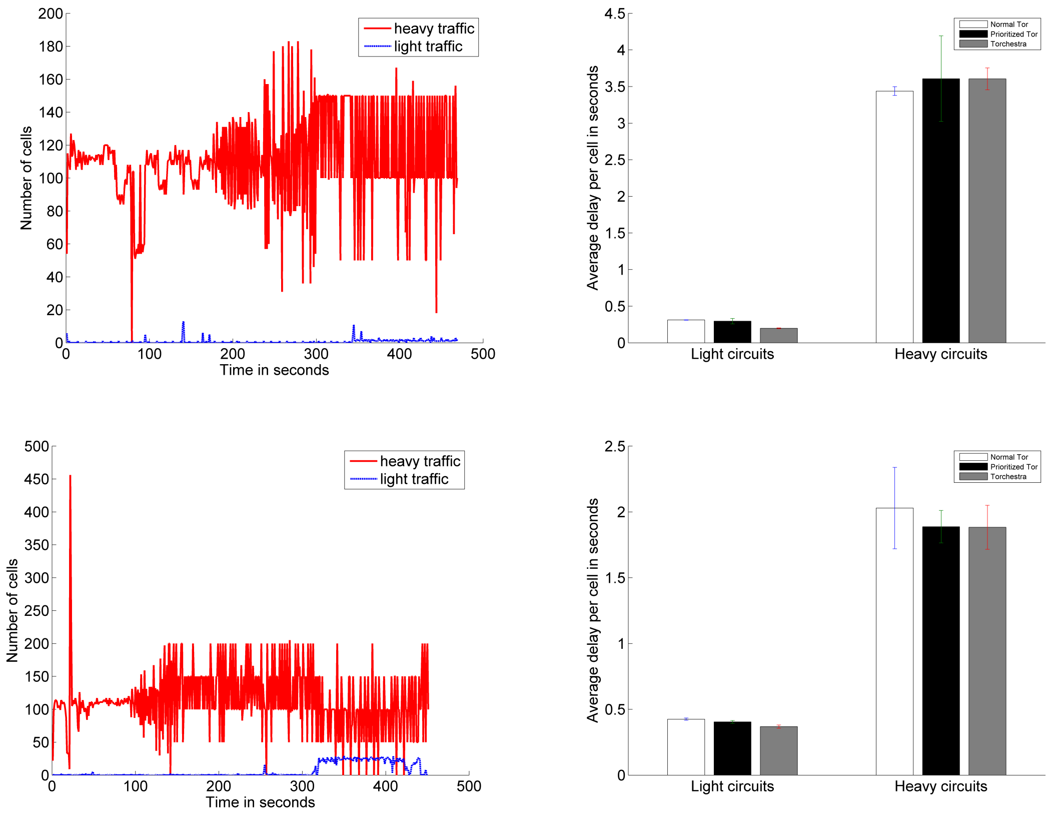


Figure 5.7: Replaying web and ssh traffic collected on the laptop for the first two samples. The left graphs show the traffic distribution and the right graphs show the average delay per cell. In each case average delay per cell is the least for Torchestra.

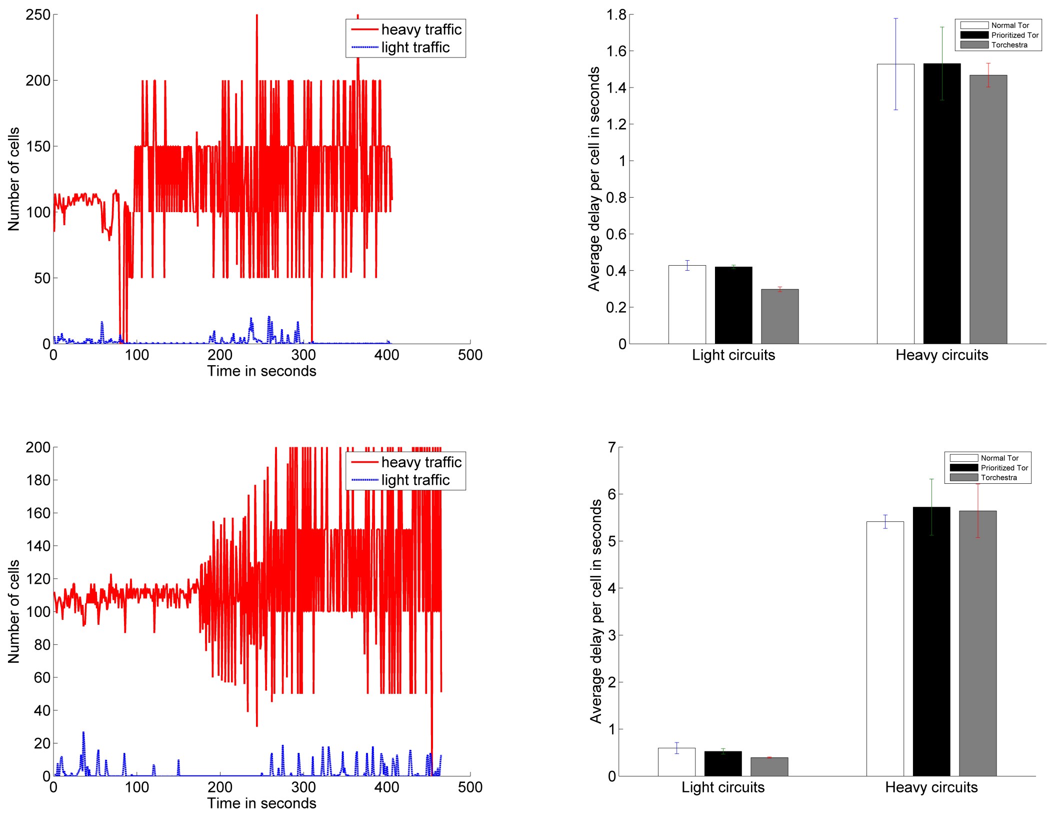


Figure 5.8: Replaying web and ssh traffic collected on the laptop for the next two samples. The left graphs show the traffic distribution and the right graphs show the average delay per cell. In each case average delay per cell is the least for Torchestra.

5.2.4 Simulating real traffic

Description

In both the “Simple files download” and “Simulating web and ssh traffic from the author’s computer” experiments, some part of the traffic is created artificially using wget. These traffic patterns may not match those of the real network.

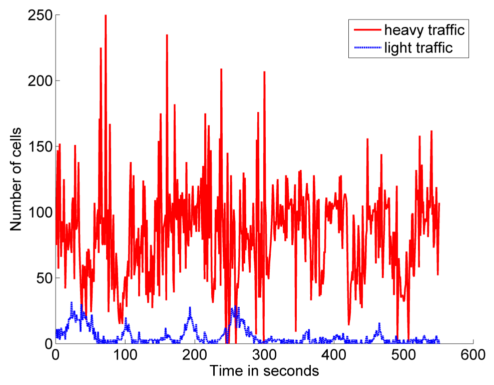
In order to get a true comparison, we collected timing information over a period of 15 minutes on the real Tor network. We then simulated this traffic in our test setup by sending dummy data with the same timing patterns using the steps described in the Experimental Setup, Section 5.1. This way, traffic simulated will be much closer to traffic on the real Tor network. Time windows of 15 minute intervals are considered during different parts of the day at 6am, 6pm, 12am and 12pm. From each time interval we chose the connection with the largest number of cells and circuits to replay.

Parameters for each of the 3 modes are -

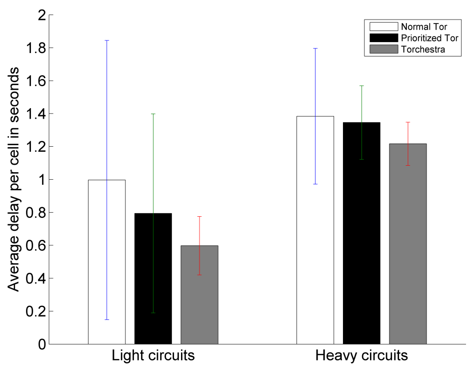
- In Normal Tor, we have not made any changes.
- For Prioritized Tor we have used H = 66 as done in [9].
- For Torchestra, we have tuned the following parameters
  - α - 0.18
  - β - 0.36
  - LIGHT_PERCENT_UPPER_BOUND - 0.7
  - HEAVY_PERCENT_LOWER_BOUND – 0.7

Results

In each of the experiments run at at 6am, 6pm, 12am and 12pm we show the traffic patterns for light and heavy traffic as well as the average delay per cell in Figures 5.9, 5.10, 5.11 and 5.12. We see that Torchestra does better than Prioritized Tor and Normal Tor in each of the experiments. In the 6am case, there

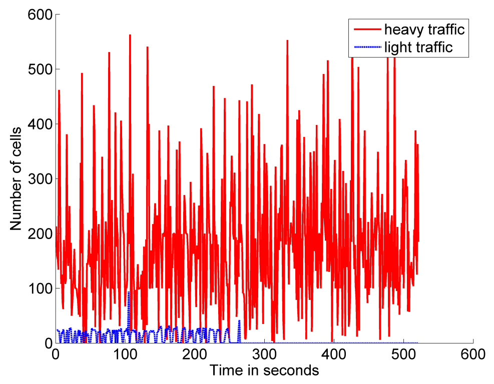


(a) Graph depicting the traffic pattern on Tor network at 6am.

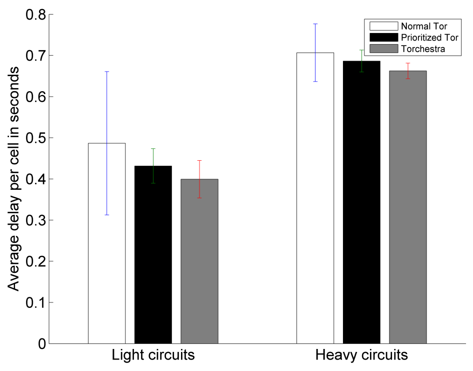


(b) Bar Graph depicting the average delay per cell in each of the modes at 6am. Average delay per cell for light circuits is lowest for Torchestra.

Figure 5.9: Simulating traffic from 6am

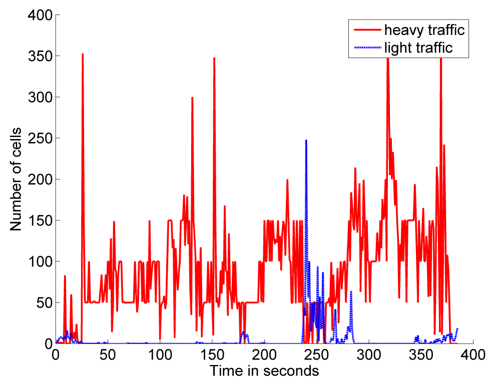


(a) Graph depicting the traffic pattern on Tor network at 6pm.

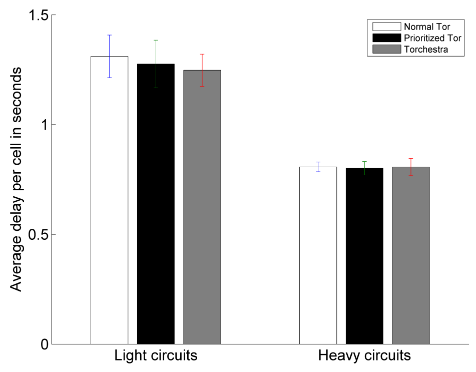


(b) Bar Graph depicting the average delay per cell in each of the modes at 6pm. Average delay per cell for light circuits is lowest for Torchestra.

Figure 5.10: Simulating traffic from 6pm

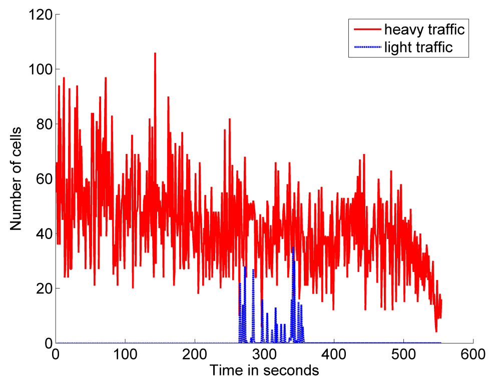


(a) Graph depicting the traffic pattern on Tor network at 12am

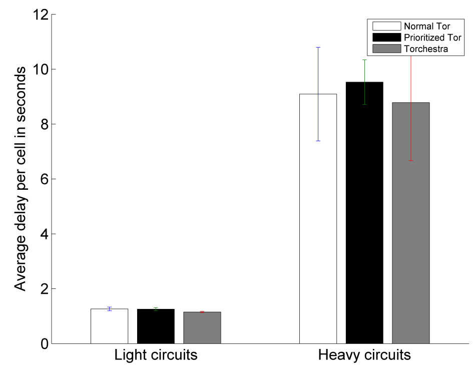


(b) Bar Graph depicting the average delay per cell in each of the modes at 12am. Average delay per cell for light circuits is lowest for Torchestra.

Figure 5.11: Simulating traffic from 12am



(a) Graph depicting the traffic pattern on Tor network at 12pm



(b) Bar Graph depicting the average delay per cell in each of the modes at 12pm. Average delay per cell for light circuits is lowest for Torchestra.

Figure 5.12: Simulating traffic from 12pm

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

Chapter 6

Future Work and Conclusion

6.1 Future Work

In Section 5.2.1 we showed that in less than 10% of the cases the number of light circuits is greater than the number of heavy circuits and in these cases light circuits will have reduced bandwidth. As future work we would like to evaluate whether increasing the number of light connections makes any difference to interactive traffic.

6.2 Conclusion

In this thesis we investigated whether Tor's performance on interactive traffic could be improved by separating light traffic from heavy traffic. 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 and used a variety of experiments – simple file downloads, replaying traffic with the same timing patterns as in the real Tor network and replaying ssh and http traffic collected on our personal machines. With simple file downloads we found that the average delay per kilobyte is the least with Prioritized Tor but the time to download the first byte is the least with Torchestra. 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 Normal Tor. Replaying the http and ssh traffic collected on our laptops, 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 Normal Tor. Torchestra is backwards compatible with versions that do not support it but in order for interactive traffic to see reduction in delays, all three nodes along the circuit should support it.

Bibliography

[1] Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. `https://www.torproject.org/press/presskit/2009-03-11-performance.pdf`

[2] Florian Tschorsch, Björn Scheuermann. "Tor is unfair – And what to do about it" lcn, pp.432-440, 2011 IEEE 36th Conference on Local Computer Networks, 2011

[3] W.B. Moore, C. Wacek and M. Sherr. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In Proceedings of the 27th Annual Computer Security Applications Conference (2011), AC-SAC 11, pp. 207-216

[4] Roger Dingledine. Tor introduction. `https://svn.torproject.org/svn/projects/presentations/slides-ucsd10.pdf`

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

[6] Damon McCoy, Kevin Bauer, Dirk Grunwald, Parisa Tabriz and Douglas Sicker. Shining Light in Dark Places: A Study of Anonymous Network Usage. University of Colorado Technical Report CU-CS-1032-07, August 2007.

[7] J. Reardon and I. Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In USENIX Security Symposium (USENIX), 2009

[8] Joel Reardon. Improving Tor using a TCP-over-DTLS Tunnel. Masters thesis,University of Waterloo, Waterloo, ON, September 2008.

[9] C. Tang and I. Goldberg. An Improved Algorithm for Tor Circuit Scheduling. In ACM Conference on Computer and Communications Security (CCS), 2010

[10] Can Tang. An Improved Algorithm for Tor Circuit Scheduling. Masters thesis, University of Waterloo, Waterloo, ON, 2010.

[11] 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

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. Proceedings of the 13th USENIX Security Symposium, 2004.

[13] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with Orchestra. In SIGCOMM, 2011.

[14] A. Vahdat, K. Yokum, K. Walsh, P. Mahadevan ,D. Kostic, J. Chase, AND D. Becker. Scalability and accuracy in a large-scale network emulator. SIGOPS Oper. Syst. Rev. 36 (December 2002), 271-284.

[15] D. Chaum. Untraceable electronic mail, return addresses,and digital pseudonyms. Communications of the ACM, 4(2),February 1981

[16] U. Moller, L. Cottrell, P. Palfrader, and L. Sassaman. Mix-master Protocol Version 2. Draft, July 2003.

[17] Steven J. Murdoch and George Danezis. Low-Cost Trafc Analysis of Tor. In IEEE Symposium on Security and Privacy, pages 183-195, 2005.

[18] W. Richard Stevens. TCP/IP Illustrated, Volume 1

[19] TCP Congestion Control RFC 5681 `http://tools.ietf.org/html/rfc5681`

[20] Transmission Control Protocol RFC 793 `http://tools.ietf.org/html/rfc793`

[21] Tor specification. `https://gitweb.torproject.org/tor.git/blob/c18bcc8a55dfaef21637b9f6f38d05610b6ab50c:/doc/spec/tor-spec.txt`

[22] S.W. Roberts, Control Chart Tests Based on Geometric Moving Averages, Technometrics, 1959.

[23] J. Crowcroft and P. Oechslin. Differentiated end-to-end Internet services using a weighted proportionally fair sharing TCP. ACM Computer Communications Review, 28:53-67, 1998.

[24] TCP max-min slides. `http://s95349177.onlinehome.us/macn/EEN634_013008.ppt`

[25] TCP max-min flow control. `http://people.cs.umass.edu/~arun/653/notes/tcp_opt.pdf`

[26] The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html

[27] Wireshark. http://www.wireshark.org/

[28] W. Diffie and M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976), 644-654.

[29] A. Serjantov and P. Sewell, Passive attack analysis for connection based anonymity systems, in Proc. ESORICS, Oct. 2003, pp. 116131

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

[31] Tor project overview. https://www.torproject.org/about/overview

[32] Microsoft forum. http://social.technet.microsoft.com/Forums/en-US/winservergen/thread/fb7a6ef7-5a70-43a4-b0fe-c0252877467b/

[33] EWMA example. http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_averages