Special Issue

Communications of the CCISA
Vol. 23 No. 1 Jan. 2017

Increasing bandwidth in Tor paths with multiple inter-relay connections

Te-Yu Liu, Po-Ching Lin*
Department of Computer Science and Information Engineering,
National Chung Cheng University
andyowenx@gmail.com, pclin@cs.ccu.edu.tw

Abstract

Tor is a well-known low-latency anonymous network for Tor users who surf the Internet to hide themselves. However, Tor transmission is usually considered slow for several reasons. One of them is that the connections from multiple Tor users are multiplexed into one TCP connection between each pair of adjacent onion routers in the Tor path. The overall throughput is limited because the available bandwidth of a connection is subject to congestion control and fairness with other non-Tor connections. In this work, we design and propose multiple-connection Tor (MCTor) to speed up the Tor network by *allowing multiple connections* between adjacent onion routers. MCTor will increase the overall throughput and is compatible with the Tor network. The emulation supports that MCTor can achieve our goal. We have two connection distribution mechanisms, uniform and hashing. We implement emulation to test both mechanisms in three experiments: Youtube, upload, and download. In the first experiment, we improve the throughput by 35% using the hashing mechanism. In the second, we improve the throughput by 17% using the uniform mechanism. In the third, we improve the throughput by 48% using the uniform mechanism.

Keywords: Tor, privacy, TCP fairness, multiple connections

1 Introduction

The Tor network involves volunteers who are willing to participate in the network and donate bandwidth of their hosts as relays, so that Tor users can keep their identities private on the Internet and protect transmitted data confidential until the last relay [17]. Instead of directly connecting from a user to the destination, the Tor network transmits data via a sequence of relays by establishing virtual tunnels, thus restricting anyone from tracking the original user address and watching the transmitted data at the same time. In [15], it is shown that thousands of Tor users scatter throughout the world.

Despite the benefits of Tor, data transmission in the Tor network is usually very slow, a typical disadvantage that Tor users have to endure. Three primary problems make the Tor network so slow: long Tor circuit length, misuse of the Tor network and quality of the relays

[19]. A Tor circuit means the concatenation of connections in the entire routing path, namely the Tor path, from a user via three onion routers (ORs) as the relays (to be described later) to the destination. Tor circuits constitute the foundation of Tor anonymity, and it ensures an attacker is unable to trace back to the original user, but it also increases the latency between the onion proxy (OP) on the client host and the destination significantly. Misuse of the Tor network means some ORs are misconfigured to allow peer-to-peer in the Tor network, but it is not recommended to use peer-to-peer in Tor [3]. According to [14], if Tor users want to use peer-to-peer, the appropriate OR lists will be limited by special ports for peer-to-peer. The ORs in the list will be too few to transmit all the peer-to-peer packets in the Tor network efficiently, making loading of the ORs unbalanced and decreasing the bandwidth. The last is quality of the relays. Most of Tor ORs are from volunteers contributing their network bandwidth set by the user configuration. Tor cannot ask them to donate all of their bandwidth. Therefore, the bottleneck of bandwidth is the lowest available bandwidth provided by the three Tor ORs.

For a long time, Tor was undesired by users who cannot accept its long latency and low bandwidth. Thus, it deserves the effort to increase the bandwidth and reduce the long latency of Tor. Some prior studies presented various methods to accelerate the Tor network. Kale et al. [7] reduces the congestion problem and selects the best path by the circuit congestion in Tor. Reardon et al. [13] used UDP in the Tor network, but that method changed the Tor source code significantly. In contrast, we intend to leverage multiple connections to speed up Tor. An intuitive consideration for this purpose is using multipath TCP (MPTCP) [10], which offers APIs to support multiple subflows in a TCP connection. However, there is no TLS support for MPTCP so far. MPTCP users also have to recompile their OS kernel to use MPTCP, and the overhead is too large.

In this work, we accelerate the Tor network by allowing multiple connections in each segment of the Tor path to gain more available bandwidth. This design is named multiple-connection Tor (MCTor). In the current Tor implementation, only one connection is in each segment, between the OP and the entry OR, the entry OR to the middle OR, the middle OR to the exit OR, and the exit OR to the destination [5], so there is only one circuit in a Tor path. We establish multiple circuits over the connections in each segment (actually, parts of the circuits in each segment), and distribute the Tor cells (i.e., Tor packet unit, which will be introduced in Section 2) to different circuits. Each TCP connection will gain fair bandwidth in a bottleneck link for fairness [8]. For example, suppose that the bandwidth of a bottleneck segment is *r*, *k* TCP connections are through this segment, and one of them is a Tor connection. The connection will gain the bandwidth of *r/k* due to TCP fairness. If the number of Tor connections is increased to *m*, where *m>1*, and then the bandwidth of the *m* connections will become $\frac{\mathrm{mr}}{k+m-1}$. The result is $\frac{m}{1+\frac{m-1}{k}}$ times more bandwidth for the Tor connections. Moreover, in the original Tor implementation, the data from multiple sources may be multiplexed into the same Tor circuit. If a Tor connection in the circuit experiences congestion and reduces the sliding window due to congestion control, the transmission of all the cells in the circuit will be slowed down. In comparison, if multiple circuits are allowed, the overall impact due to congestion control in a single connection will be reduced.

The remainder of this work is organized as follows. Section 2 offers the background and related work of Tor. Section 3 describes the detail and implementation of the MCTor. Section 4 describes the experiment scenarios and the performance of MCTor. The conclusion and future work are offered in Section 5.

2 Background and Related work

In this section, we will describe the background of Tor network and two studies related to this work. Section 2.1 describes the Tor environment and the circuit establishment. Section 2.2 describes how the two papers speed up Tor bandwidth.

2.1 Background

Tor makes users stay anonymous because of its encryption and connection mechanisms. Instead of connecting to the destination directly, Tor connects from the source and to the destination via three Tor ORs, as illustrated in Figure 1. Tor servers record the information of subsets of Tor ORs, including their public key and identity. The directory servers help Tor users get the connection data necessary for establishing a circuit without connecting to the middle and the exit OR (to be elaborated in Section 3.1).

To connect through the Tor network to the Internet, a Tor user has to establish her own Tor circuit and connect via three Tor ORs: entry OR, middle OR and exit OR. The entry OR, also called as the Tor guard, is the first OR that a Tor user will connect to. Only this OR knows the Tor user's real address. The middle OR is between the entry OR and the exit OR, and it hides the entry OR from the exit OR. If an attacker trying to find out a Tor user's real address compromises an exit OR or she establishes an exit OR, she will find nothing but the middle OR's address. The exit OR is the gate of the Tor network to the Internet. After the exit OR receives the packets initiated from the OP, it will repackage the packet header and replace the header information from the OP with its own. This replacement will make the destination treat the exit OR as the source instead of the real Tor user. The mechanism helps Tor users stay anonymous in the Internet and eliminates the user's source in the packet headers.

Besides establishing a circuit to hide the real source address from the outside, Tor also encrypts connections with TLS to conceal the transmitted data. The traffic passes through the Tor network is repackaged into fixed size cells of 512 bytes. The size is the minimum transmission unit in the Tor network. There are two types of cells in the Tor network: the control cells and the relay cells. (1) A control cell is always interpreted by the OR receiving it. It can be a create cell, a created cell and a destroy cell. A create cell is used when an OR initiates a connection belonging to a circuit to another OR. A created cell passes a command to next OR in the connection established by a create cell. The destroy cell is to destroy a circuit. (2) A relay cell carries data end-to-end. It is interpreted only by the OP and the exit OR. Figure 2 shows the difference between a control cell and a relay cell. A control cell consists of circID, CMD and data, but more fields are in a relay cell, such as Relay, StreamID, Digest, Len, CMD and data.

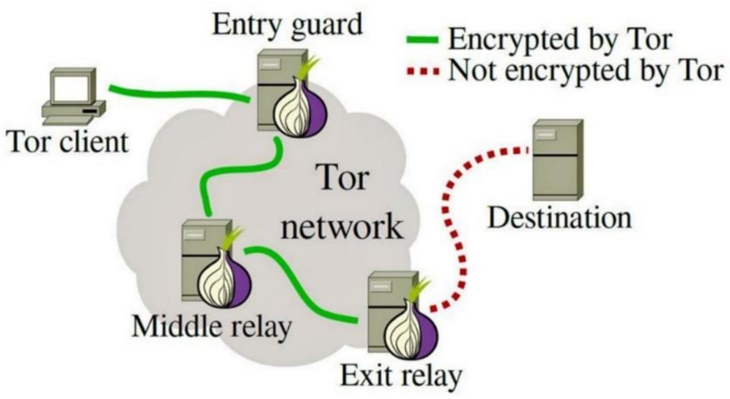


Figure 1: Tor network.

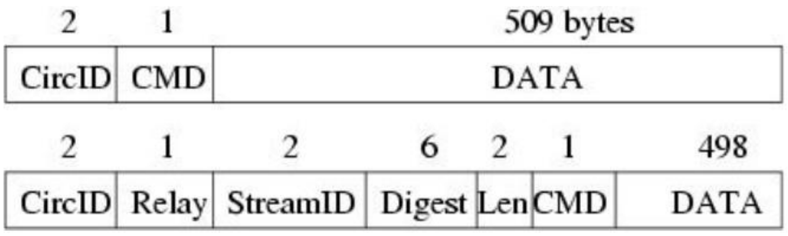


Figure 2: Tor cell structure.

2.2 Existing method

Reardon et al. [13] proposed a method to speed up the Tor network by changing the transport protocol from TCP to UDP. They found out that congestion control mechanisms are unfairly applied to all circuits when they are intended to throttle only the noisy senders. They eliminated the influences of congestion control by replacing the TCP connections between ORs to DTLS, a secure datagram (UDP based) transport protocol, but there is still a user-level TCP stack for the exit OR to understand the cells without changing the source code significantly. Each circuit also involves a single TCP connection in each segment to ensure that congestion in one circuit will not affect the others.

Kale et al. [7] proposed a new way to reduce the congestion problems and selected the best path by the circuit congestion in Tor. They implemented a control metric to detect congestion in Tor on the entry ORs, and then they applied two control cells called SWITCH cells and SWITCH_CONN cells. When the entry OR detects congestion in this circuit, it will send a SWITCH cell to inform the middle OR and the exit OR that it is time to destroy this circuit. Then the entry OR will choose a new middle OR and exit OR in this circuit. When the entry OR detects that there may be congestion in itself, it will send a SWITCH_CONN cell to the middle OR, which will extend the SWITCH_CONN cell to the exit OR. If the entry OR does not receive SWITCH_CONN from the middle OR and the exit OR, it will send an error message to the OP to tell it that there is congestion in the entry OR. OP will immediately switch its connection to another entry OR. Table 1 summarizes the comparison.

Table 1: Comparison of existing Tor acceleration techniques.

Technique	Paper	characteristic	Limitation
Speed up Tor network with UDP	[13]	replace TLS between ORs with DTLS	need to change the original Tor code significantly
Improving unfair distribution	[7]	reduce congestion by choosing the best path	cannot change the path when keeping a conection persistent (e.g., downloading files)

3 System Architecture

We accelerate the Tor network by establishing multiple circuits instead of one (thus multiple connections in each segment), so that all the ORs are expected to share more bandwidth (still subject to the bandwidth limitation allowed by Tor users) when it transmits data through the Internet. Considering Tor does not force volunteers to update their Tor program to the latest version, the design can be compatible to existing implementation by reducing to one circuit when contacting it. In this section, we will describe the Tor design and how we emulate the key Tor functions by multi-connection Tor (MCTor). Section 3.1 describes the Tor program structure and the main idea of Tor. Section 3.2 describes that main structure of MCTor and how we implement MCTor to emulate the Tor operation.

3.1 Tor Program Overview

The Tor network hides a user's identity through three Tor ORs, so that an attacker who intercepts the packets in the Tor network (except at the entry OR) cannot trace the real address belonging to a Tor user by the source IP address. The packet payloads are also invisible to the attacker because Tor encrypts the packets transmitted in circuits.

Tor exchanges AES session key with other Tor ORs by RSA. When a Tor OR publishes itself, it will send its RSA public key and identity to the Tor directory server, which saves the public key and identity to its OR list. No matter what configuration it has, a new OR in the Tor network will start as a middle OR. It will become an exit OR after it has been published longer than three hours, if there is an exit OR flag setting in its configuration. The rule that allows an OR to become an entry OR is very strict [16]. Tor directory server decides whether an OR can become an entry OR based on three requirements: (1) This OR should have large enough measured bandwidth. (2) This OR should always keep itself running most of the time. (3) This OR should be running for at least eight days.

A Tor user connects to the Tor network and builds a circuit through 11 steps for establishing successful connections. (1) The OP must connect to the Tor directory first, and will receive a list of Tor ORs, as well as their corresponding identities, RSA public keys and OR states. (2) The OP randomly chooses an entry, a middle and an exit OR from the list of ORs. (3) The OP sends a create control cell to the entry OR. (4) The entry OR returns a created control cell to the OP. (5) The OP sends another relay control cell to the entry OR with an Extend command inside. (6) The entry OR sends a create control cell to the middle OR. (7) The middle OR sends a create control cell to the exit OR. (8) The exit OR sends a created control cell to the middle OR. (9) The middle OR sends a created control cell to the entry OR. (10) The entry OR sends a relay control cell with an Extended command inside to the OP. (11) The OP begins to communicate with the destination through the Tor network. Figure 3 illustrates the process.

Tor users should gain more bandwidth if Tor uses multiple connections because TCP connections via a bottleneck router/link are supposed to share equal bandwidth due to TCP fairness and multiple connections will gain more bandwidth than only one connection. In the beginning of this work, we once considered combining MPTCP and Tor to implement multiple connections because MPTCP can create multiple subflows in a TCP connection and automatically distribute the packets over the subflows. However, MPTCP has not supported TLS encryption yet, except that the idea was discussed in an expired Internet draft [4], not to mention an implementation. Therefore, we still use multiple TCP connections instead of MPTCP in this work.

3.2 Multi-connection Tor Overview

The current Tor implementation has a large code base of around 200 thousands lines according to our own study. Instead of modifying the code, which takes great efforts

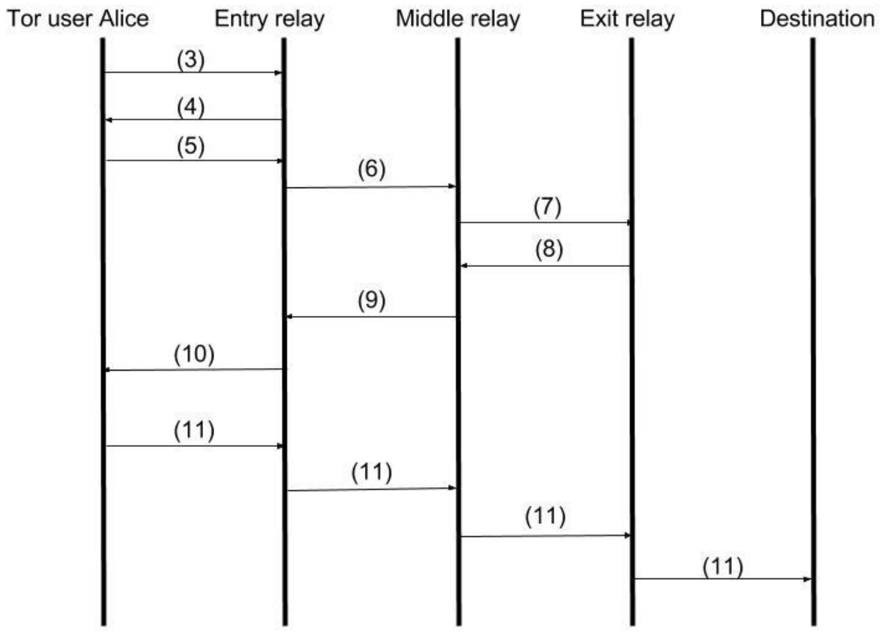


Figure 3: Tor connection process.

with many unnecessary details, we develop a much simplified version, namely MCTor, as the proof-of-concept of the main idea. The structure and operation of the MCTor program are very similar to those of Tor to emulate a Tor network with multiple connections between adjacent relays. We describe the program details in this section.

MCTor is a SOCKS5 proxy program that facilitates changing the network configuration for various experiments. This program is designed for verifying our hypothesis of gaining more bandwidth, not for users who want to completely stay anonymous in the Internet. We skip the features of the Tor program that are irrelevant to the verification, such as command cell, directory server, choosing random OR, etc. We keep two necessary features, (1) relay cell and (2) entry, middle and exit OR, but slightly change them for the MCTor environment. The relay cell is kept because we have to specify the number of TCP connections from one OR to another. For example, when a user surfs a web site with a browser, the browser can create multiple connections simultaneously to the Internet because the web site may embed images from other web sites. In MCTor, we want to hide the real number of connections from the browser. We also repackage the data from the browser and insert the stream identifier and data length into the packets. The source code is publicly available at https://github.com/andyowenx/MCTor.

Depending on relay cells, we can merge multiple TCP connections into one. Figure 4 presents the format of a relay cell in MCTor. Every packet from the OP goes through MCTor will carry a relay cell to identify itself as the TCP connection it belongs to. The first four bytes mean the stream identifier, i.e., stream_id of a connection. The next four bytes mean the data length of this cell. Following is filled with partial data belonging to this connection. The entry, middle and exit OR are necessary in the verification because we want to make our environment as similar as typical Tor does. We keep three ORs, namely MCORs, in the path like a typical Tor deployment.

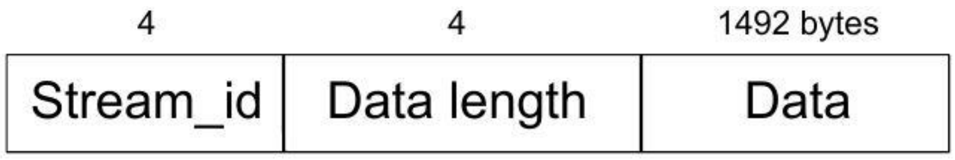


Figure 4: Format of MCTor relay cell.

MCTor involves three programs: MCTor_OP, MCTor_transmit_OR and MCTor_exit. (1) MCTor_OP is the Multi-connection Onion Proxy (MCOP) in the MCTor network. It communicates with the browser and encrypts the data from the browser three times by AES-CTR with the keys of the entry, middle and exit MCOR. Then it transmits data to the entry MCOR in the MCTor network. (2) MCTor_transmit_OR serves as the entry and middle MCORs in the MCTor network. Its function is decrypting the data with the keys of the two MCORs and transmitting the data from one hop to the next. (3) MCTor_exit is the exit MCOR in the MCTor network. It decrypts data to get the plain text, and then sends them to the destination. There are three main components in MCTor: (1) handling the socket file description with libev, (2) encrypting/decrypting data with AES-CTR, and (3) distributing data to the circuits.

Libev is a high-performance event loop. It can monitor multiple file descriptions in the same process, and can be triggered by reading/writing to a file description under monitoring. Figure 5 describes how libev works in an MCTor_OP. Libev handles file descriptions in this program based on the their attributes in the MCTor network. When MCOP receives a new TCP connection from the browser, libev will call the browser_connect_to_proxy function, which will authenticate the proxy information from the browser first and then send the destination information to the entry MCOR. To handle the communications of this connection in MCOP, another libev watcher should be set to monitor this file description. Figure 6 shows how a cell goes through the entry and the middle MCOR. When a cell comes from MCOP, the entry MCOR will simply decrypt the data part in a cell then pass it to next MCOR.

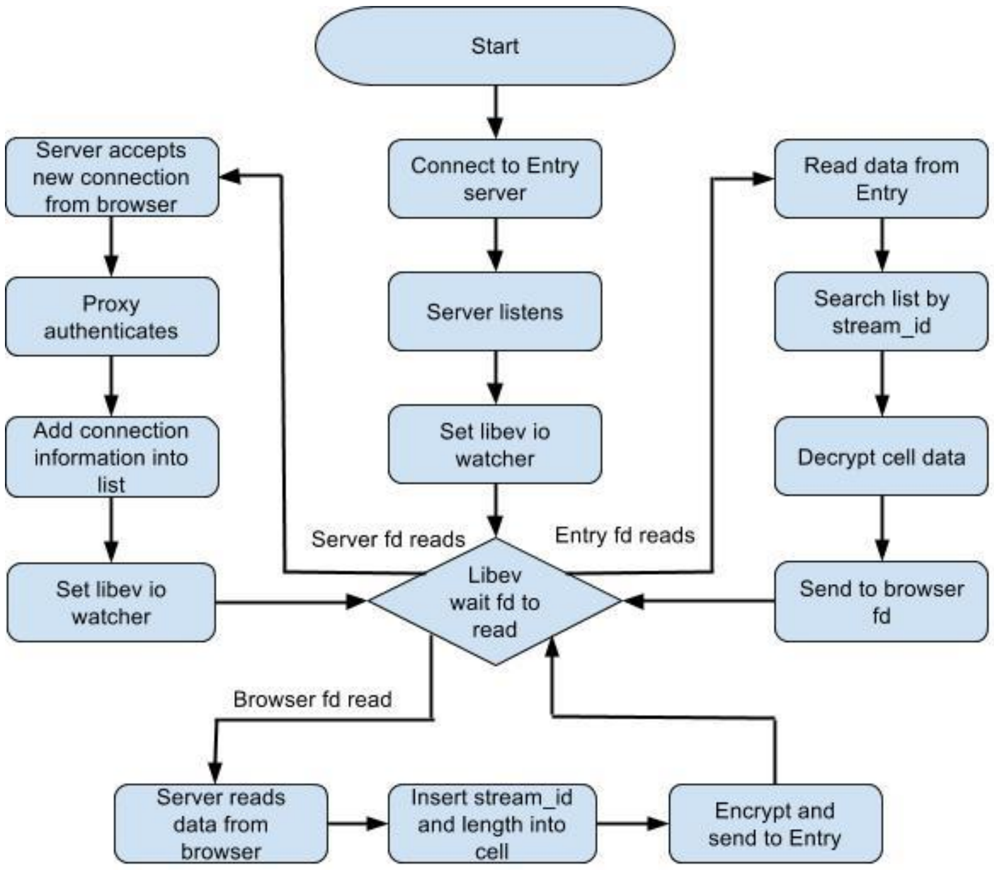


Figure 5: Flowchart of MCTor OP.

Figure 7 describes when the exit MCOR receives packets from the middle MCOR, libev will call the handle_from_middle function. After receiving packets from the file description, the exit MCOR will read the stream_id from the cell and look for the file identifier corresponding to this stream_id. If the exit MCOR cannot find it, this means this is a new connection. We record the connection information and insert it into a list for connection tracking. Then a new libev watcher is set to monitor any data receiving from it. If the stream_id is in the list, we decrypt the data in this cell and send to the corresponding file description.

Tor uses the AES-CTR to ensure that someone who captures the cells between the client and the exit OR cannot observe the plain text of the cells. We want to emulate it to make sure that this step will not affect the experimental results. Because AES-CTR encryption and decryption are implemented using the same step, we only implement one function aesctr_encrypt to achieve both encryption and decryption. Like the Tor network, we encrypt the payload in the cells three times with the keys of the entry, middle, and exit MCOR at MCOP.

Communications of the CCISA
Vol. 23 No. 1 Jan. 2017

When a cell passes through each MCOR, the MCOR program will decrypt it with its own key. After arriving the exit MCOR, the same payload will be recovered by decryption.

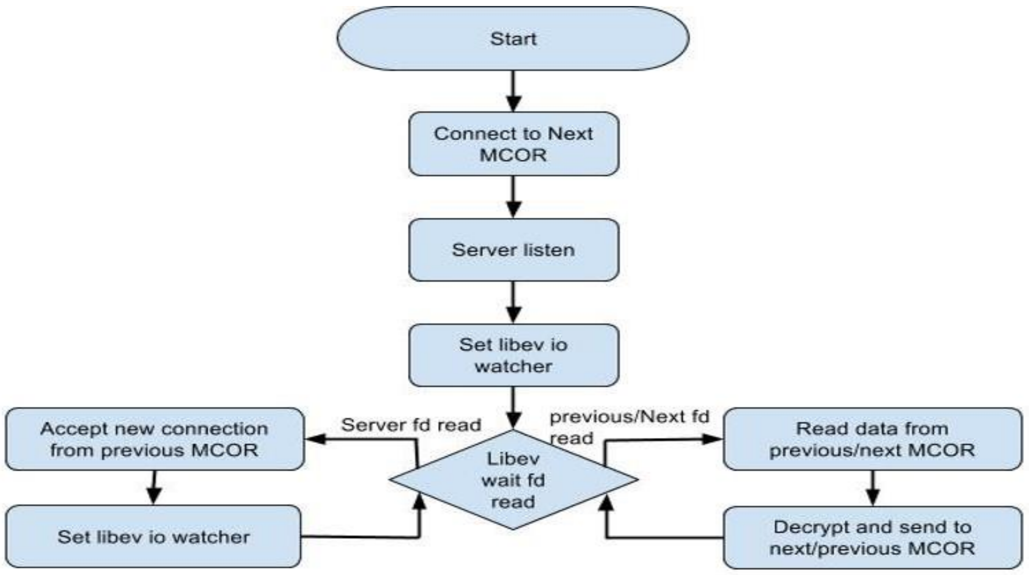


Figure 6: Flowchart of MCTor entry/middle.

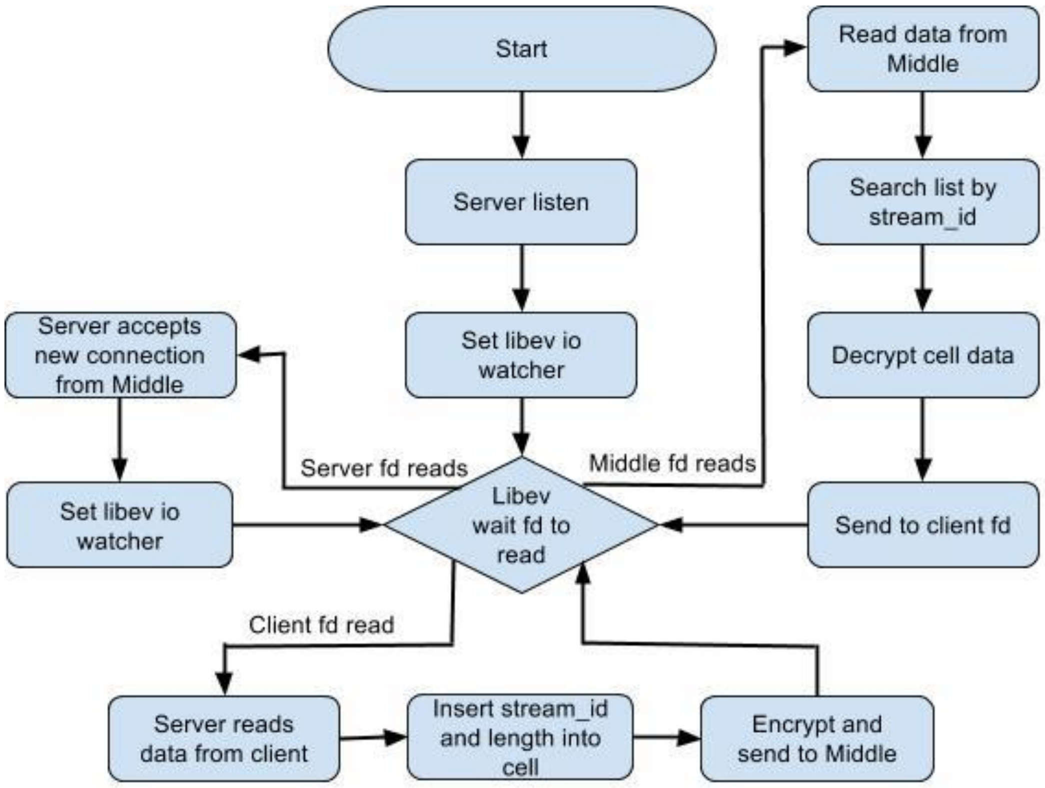


Figure 7: Flowchart of MCTor exit.

When a new connection is created by the browser, we will give it a unique stream identifier. There are three mechanisms to distribute this connection to which circuit in the MCTor network: (1)uniform, (2)hashing and (3)least loading. In the uniform mechanism, we take the remainder of dividing the stream identifier by the number of circuits. This connection will be distributed to the circuit whose index is equal to the remainder. In the hashing mechanism, we hash both source and destination IP addresses and port numbers as a pair by MD5, and then take the remainder of dividing the hash value by the number of circuits. The difference between the uniform and hashing mechanisms is the former distributes the connections in a round-robin manner, while the latter distributes connections randomly. In the least loading mechanism, we will detect the bandwidth in each circuit, and distribute a new connection to the least loading circuit. However, this mechanism is not suitable in the MCTor network. When the OP decides to distribute a new connection, it only measures the bandwidth passing through itself. The OP cannot measure the bandwidth in the entry, middle and exit MCOR. This mechanism makes its decision only by local information without the bandwidth over the entire path, so the decision is inaccurate. Therefore, we experiment with only the uniform and hashing mechanisms.

4 Architecture Evaluation

In this section, we will describe the experimental environment, scenarios and results. Section 4.1 describes the MCTor deployment for the experiment. Section 4.2 describes various scenarios and the experimental results.

4.1 Experimentation Environment

In the experiments, we use the MCTor network to emulate the Tor network with multiple connections between adjacent nodes. The client is a personal computer with Intel i7-2600 CPU and 8GB RAM. To make the experiments close to reality, we rented three Linode2048 servers from www.linode.com. They are Ubuntu 14.04 systems with 1 CPU core, 2GB RAM, 125Mbps outbound bandwidth, and 40Gbps inbound bandwidth located in Newark (US), London (UK) and Singapore. The server in Newark serves as the MCTor entry OR, that in London as the middle OR and that in Singapore as the exit OR. This path extends MCTor's routing length almost around the globe to emulate a long Tor path. Figure 8 shows the MCTor network path in the experiment.

4.2 Experiment result

We measure the data transmission time with multiple connections by downloading, uploading and Youtube watching. We will not compare the results from MCTor and Tor directly because as a Tor user, we cannot fully control the Tor path. In the Tor design, a

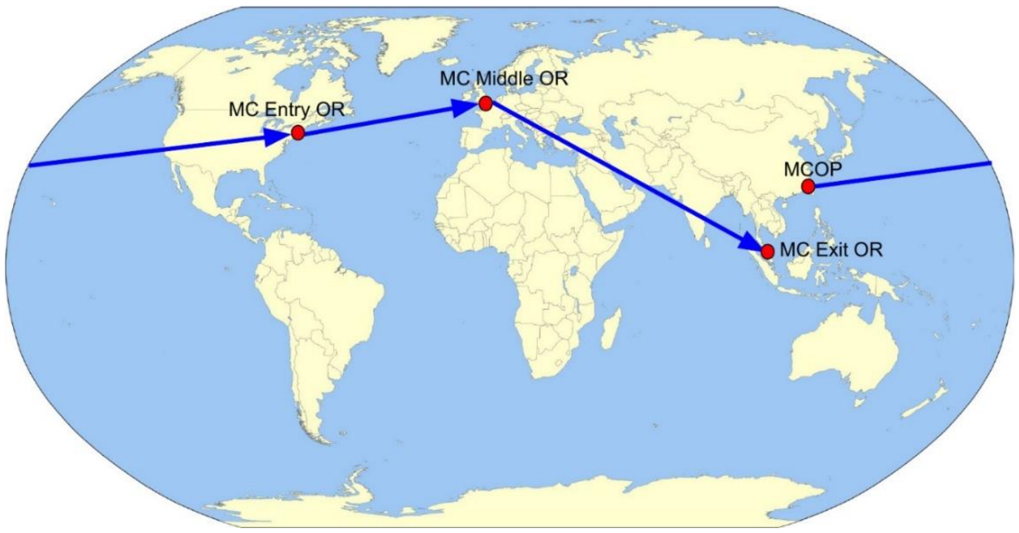


Figure 8: The MCTor path.

Tor user can decide the entry and exit ORs, but the middle OR must be randomly selected by the OP. Since we are unable to dictate equal paths in both the Tor and MCTor networks, the performance of both cannot be directly compared. Instead, we use only one circuit in MCTor to emulate the Tor network on the same path.

Before discussing the experiment, we have to explain another uncontrollable factor: Internet network congestion. In the experiment, the packets we send and receive will travel along a long Internet path, meaning the throughput will highly depend on the congestion in the path when the experiment is conducted. Thus, packet loss and Internet delay are unpredictable, resulting in some outliers in the experiment. We detect outliers by the Tukey's test [11], in which data outside the following interval will be treated as outliers.

$$[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$$

We set k to 1.5. Q_1 and Q_3 are the lower and upper quartiles. Figure 9 illustrates an example of outlier in an experiment on June 29th. We downloaded five same iso image files at same time from Avira rescue system [2] via three multiple connections between adjacent ORs in the MCTor network, and each file is 627MB long. One of the downloading processes takes 12 times longer than others, so we exclude this outlier to eliminate its impact. Therefore, it is necessary to exclude such outliers to reduce the impact from Internet congestion.

We select two mechanisms to distribute connections to the MCTor circuits in the experiments: *hashing* and *uniform*. The hashing mechanism is distributing connections to the circuits according to the mapping through a hash function, while the uniform mechanism is distributing connections to the circuits in a pre-defined order. The experiments are conducted in two scenarios for both mechanisms. The first scenario is busyness.

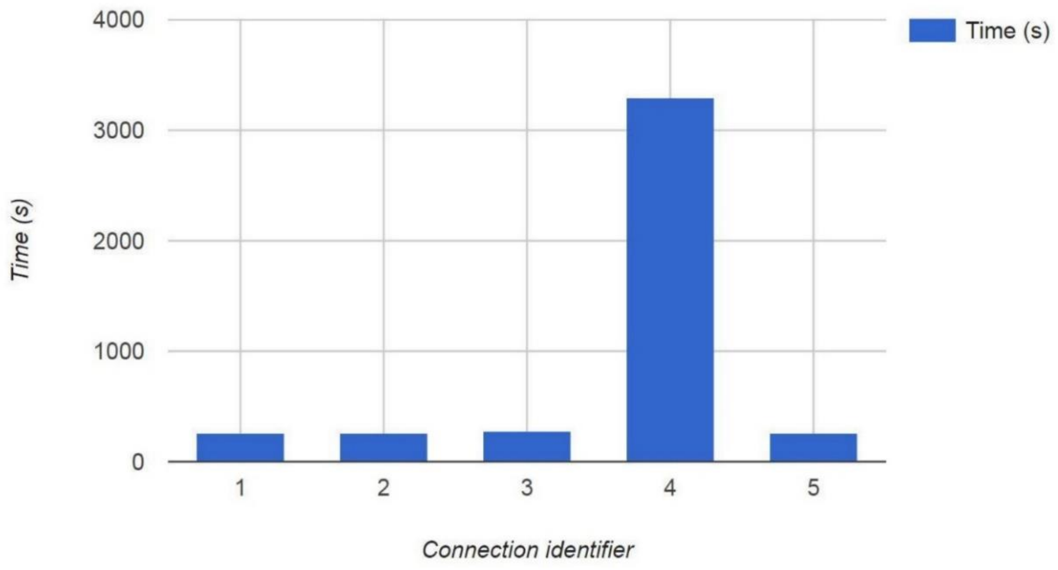


Figure 9: An example of outlier.

This goal is to know how much it will affect the MCTor network if the link of the middle MCOR is shared by non-MCTor traffic simultaneously. In the evaluation, we make the middle MCOR keep receiving 5Mbps traffic generated by iperf from another computer. The second scenario is fairness. We assume two users connect through the same MCTor path and observe whether they share a balanced amount of bandwidth. We conduct at least three times for each of the three experiments and the two scenarios. The following figures are the average data in each experiment.

We watched a Youtube video clip [1] in 480p and observed network connection information by firebug [6], which is an add-on for Firefox. Firebug can record the details of every connection, e.g., session time, request header and HTML source code from this web page. Once the video is fully loaded, firebug will record the time.

Figure 10 presents the experimental result. Every time we increase the number of connections between adjacent MCORs, the video loading time is decreasing, no matter which connection distribution mechanism, uniform or hashing, is used. With five circuits in the MCTor path, uniform connection distribution is 30% faster and hashing is 35% faster than only one circuit in the path, which is the case of the Tor network.

For the upload experiment, we upload a 235MB binary file which composes of random bytes to MEGA [9]. When the uploading process begins, the file is split automatically into multiple parts by the browser. The browser establishes five to seven connections to MEGA and sends each of file parts through these connections to MEGA. As shown in Figure 11, when the number of circuits increases in MCTor, the throughput will also increase. When comparing five circuits with one circuit, the uniform mechanism improves 17% of throughput, and the hashing mechanism improves 11% of throughput.

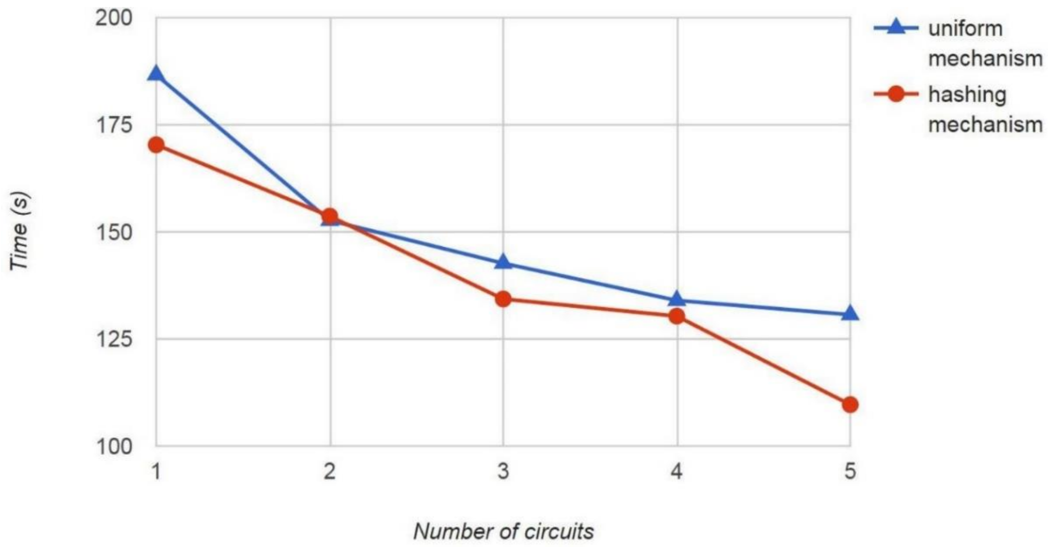


Figure 10: Uniform and hashing in the Youtube experiment.

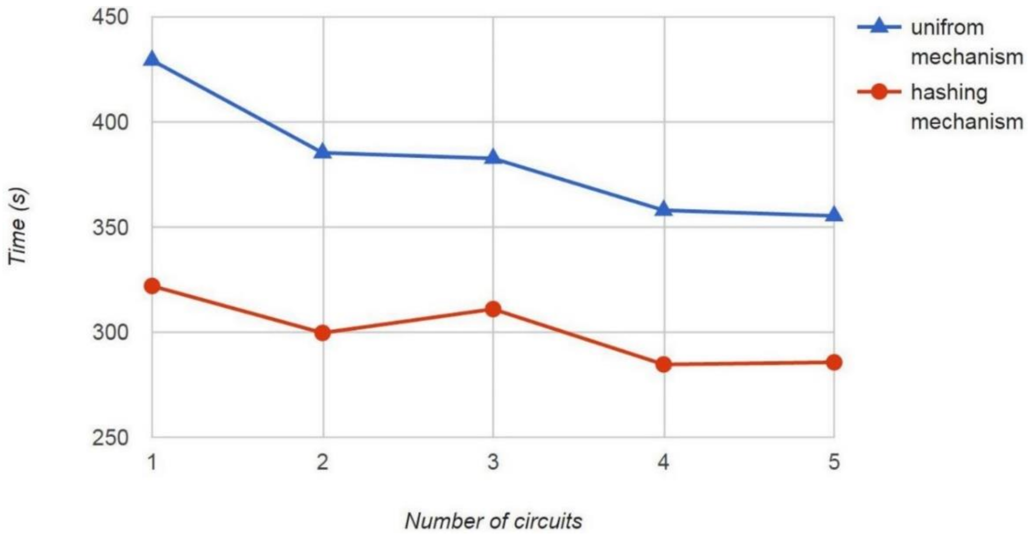


Figure 11: Uniform and hashing in the upload experiment.

For the downloading experiment, we download five same iso image files from Avira [2] at the same time. Each file is 627 MB, and we record the total time. Figure 12 presents the results of both the uniform and the hashing mechanisms. The throughput from both mechanisms increases when we add more circuits between adjacent MCORs. The downloading time in the hashing mechanism result is slightly longer than that of the uniform mechanism because the former does not equally distribute the bandwidth to each circuit while the latter does. In this experiment, up to five circuits will be established. The uniform mechanism distributes the connections to each circuit in an order, while the hashing mechanism randomly distributes the connections to each circuit. In this experiment, the hashing mechanism does not distribute every connection in order, so it makes a circuit idle. A file which the idle circuit should have handled was distributed to other circuits. This causes the hashing mechanism slower than the uniform mechanism. When there are five circuits, the uniform mechanism is 48% faster, and hashing is 43% faster for only one circuit in the experiment.

During our experiments, we make sure there are no other connections in the OP except those in the MCTor network. The result shows that users should use the uniform mechanism for downloading, and the hashing mechanism for watching Youtube and uploading. In general, users usually do multiple actions at the same time such as surfing

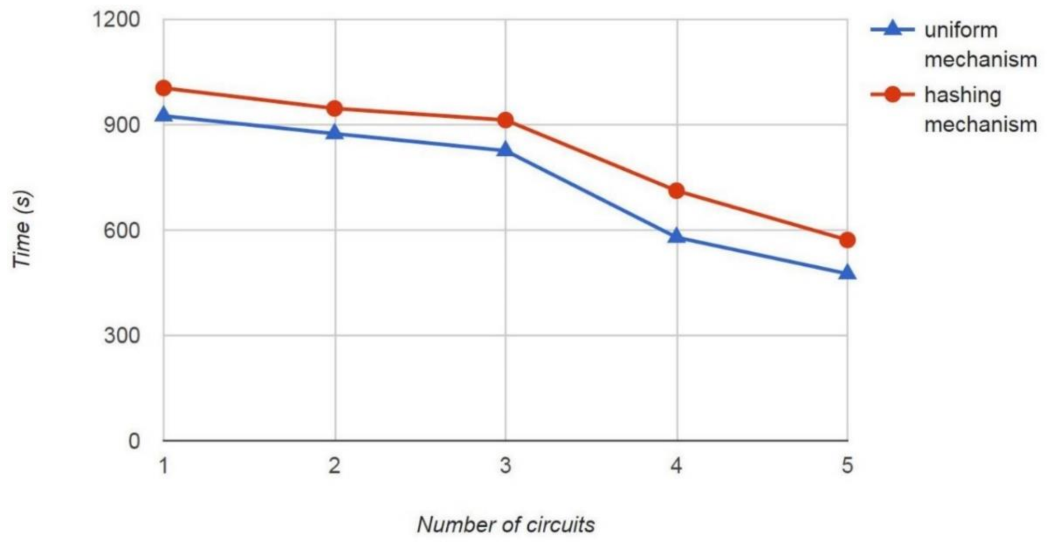
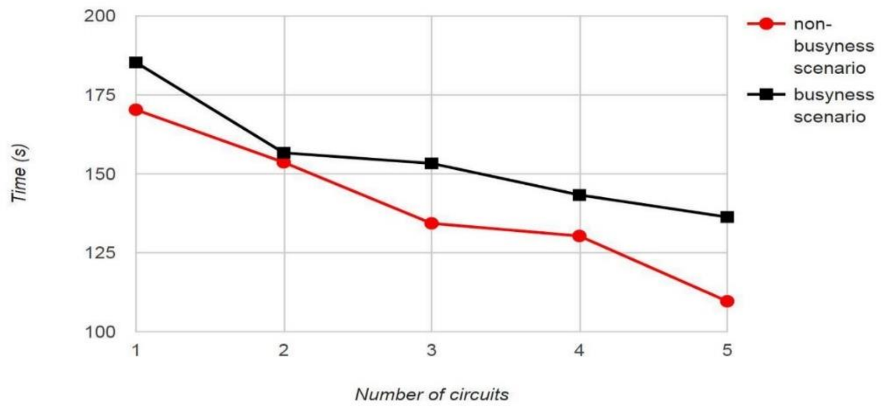


Figure 12: Uniform and hashing in the download experiment.

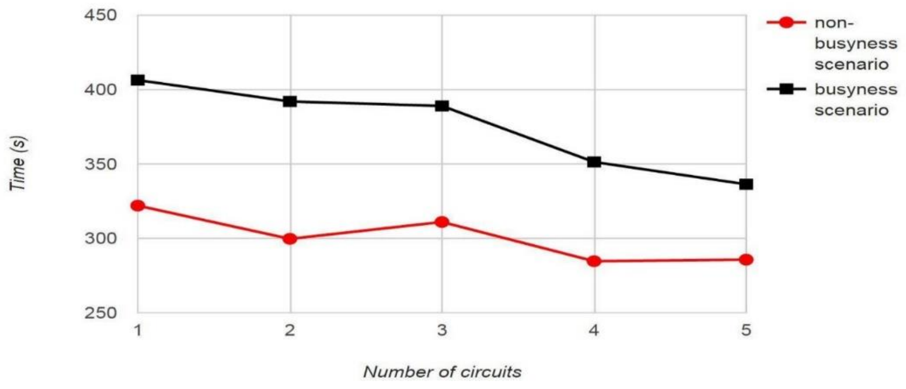
the Internet, listening music from Youtube, and downloading files from some website. In this case, there are multiple connections in MCTor. We cannot make sure that the uniform mechanism can fairly distribute downloaded files to each of circuits because we cannot force users to close other connections to guarantee the uniform mechanism will distribute download connections in order. This causes the uniform mechanism cannot perform the best. On the other hand, the hashing mechanism is not limited by that situation.

In the busyness scenario, we use *iperf* to generate 5Mbps extra traffic in the middle MCOR. Then we rerun either the uniform mechanism or the hashing mechanism, depending on which is faster in the Youtube, upload and download experiments. Figure 13 presents the results in the busyness scenarios. In the busyness scenarios of Youtube, upload and download, the throughput improvement of five circuits compared with one is 26%, 17% and 41%. The busyness scenarios with five circuits in the Youtube, upload and download experiments compared to non-busyness scenarios with five circuits is slower 19%, 15% and 43%.



(a)Busyness scenario in the Youtube experiment

(b)Busyness scenario in the upload experiment.



(c)Busyness scenario in the download experiment.

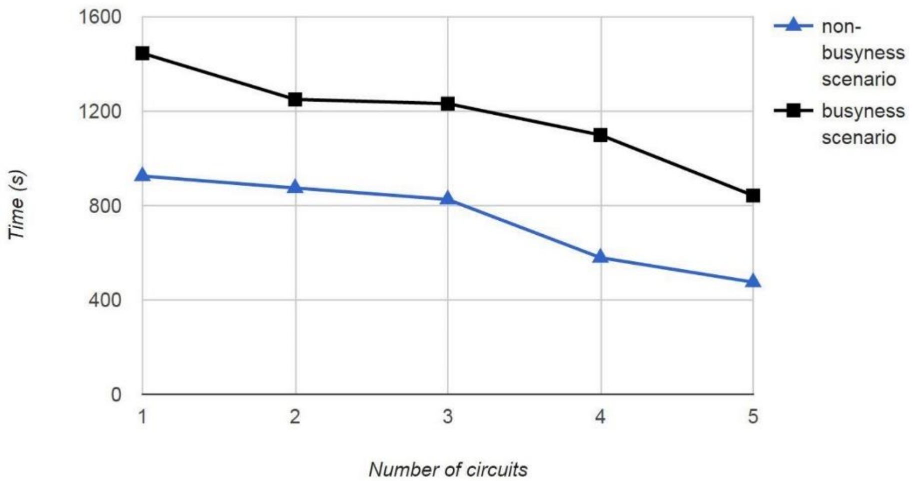


Figure 13: The busyness scenario in three experiments.

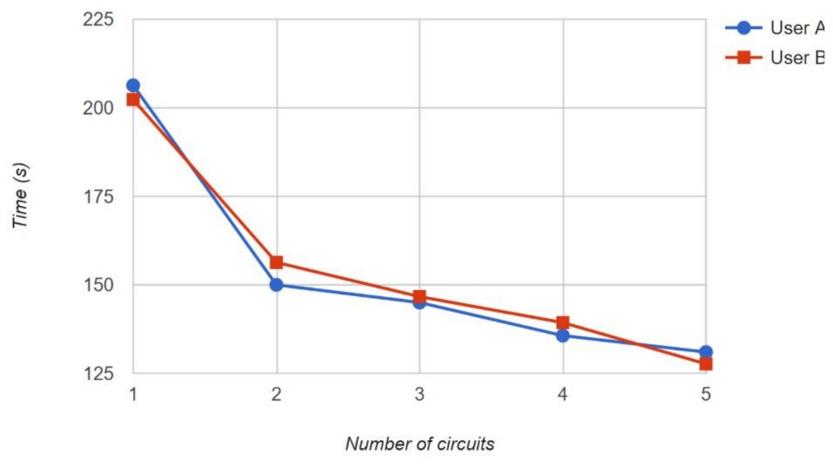
We also test with the fairness scenarios, in which two users do the completely same action in the Youtube, upload and download experiments. Figure 14 shows the fairness scenario in the Youtube, upload, and download experiments. In Youtube fairness experiment, the maximum difference between user A and user B is 4.05%. In the upload fairness experiment, the maximum difference is 11.68%. In the download fairness experiment, the maximum difference is 6.08%. Among three experiments, the maximum difference is 11.68%, and the average difference is 4.59%. We think it is acceptable for the MCTor network.

In our experiments, we prove that our hypothesis is feasible for Tor network. No matter which mechanism we choose, we can improve MCTor network throughput by increasing the number of circuits between adjacent MCORs in the Youtube, upload, and download experiments. In the busyness and fairness scenarios, we prove that MCTor network throughput can still be improved in a busy situation and MCTor can fairly distribute bandwidth to users.

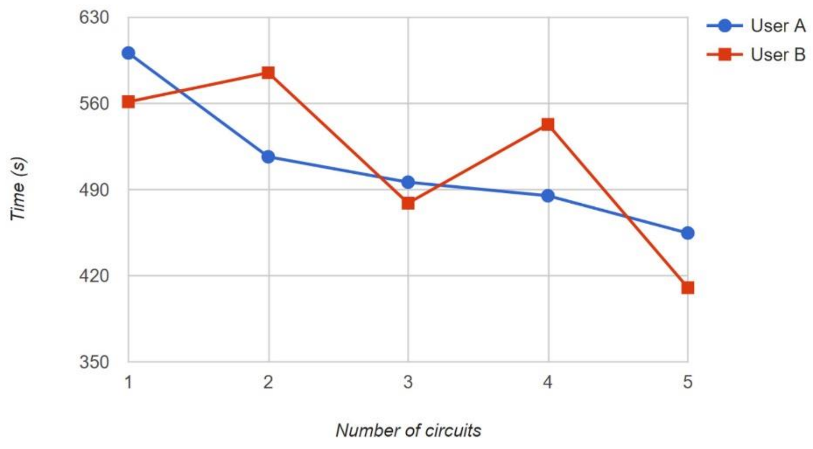
5 Conclusion and Future work

Even though Tor provides anonymity, Tor users often suffer from long delay when accessing the Tor network. In this work, we accelerate Tor by implementing multiple connections between adjacent relays, namely MCTor, which serves as a proof-of-concept implementation to verify the idea. The result shows that MCTor can significantly increase Tor network throughput. In the Youtube experiment, we improve the throughput by 30% using the uniform mechanism and by 35% using the hashing mechanism. In the upload experiment, we improve the throughput by 17% using the uniform mechanism, and by 11% using the hashing mechanism. In the download experiment, we improve the throughput by 48% using the uniform mechanism, and by 43% using the hashing mechanism.

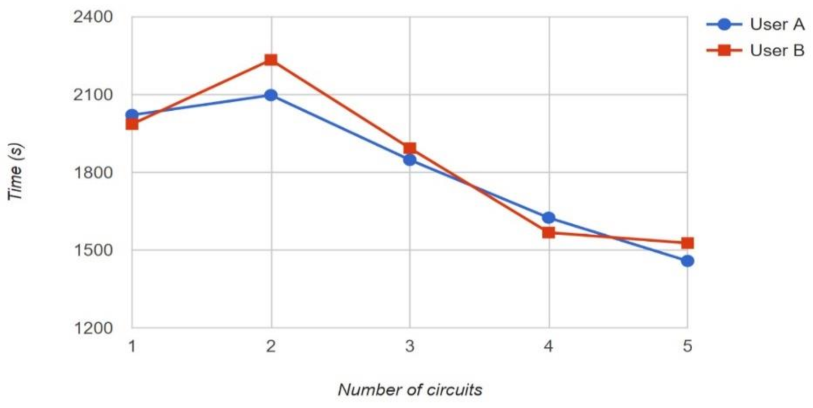
There are four issues that have not been addressed: (1) distributing the packets in a connection to different circuits, (2) experiments in the real-world Tor network, (3) simulating the MCTor network in ns-3 [12] and (4) calculating the extra loading for multiple connections in Tor network. First, the packets in a connection are distributed into only one circuit in MCTor in the current design. This design will result in load imbalance, particularly for a long connection, and cannot utilize idle circuits. MCTor should be able to distribute the packets through multiple circuits, and assemble them on the exit MCOR. This approach can be faster than using only one circuit. This can improve the throughput when there are some idle circuits in MCTor network. Second, we demonstrate that MCTor can work in a real environment. However, its integration with the Tor network still relies on the significant modification of the Tor source code, which is non-trivial given our current human resources. The source code of MCTor has been released at *https://github.com/andyowenx/MCTor*. We wish the idea of MCTor can



(a) Fairness scenario in the Youtube experiment.



(b) Fairness scenario in the upload experiment.



(c) Fairness scenario in the download experiment.

Figure 14: The fairness scenario in three experiments.

be integrated into the Tor source code by interested users in the future. Third, simulating the MCTor network in ns-3 can help us prove our hypothesis. Although we try to simplify our environment, Internet congestion is still unavoidable. However, simulating a congestion-free environment in ns-3 can avoid this problem. Fourth, multiple connections can generate extra load in both computing and network bandwidth for the Tor network. To establish more connections between two adjacent ORs, the OR servers have to load more memory to record the connection information and more network bandwidth for connections maintenance. Those can be ignored in our simple environment, but they can be an important issue in Tor network. Thousands of ORs in the Tor network can influence Internet congestion if Tor uses multiple connections. We should calculate the load carefully and make some trade-off between bandwidth improvement and the load.

References

[1] Adele - Hello, https://www.youtube.com/watch?v=YQHsXMglC9A.
[2] Avira rescue system iso file, https://www.avira.com/zh-tw/download/product/avira-rescue-system.
[3] Bittorrent over Tor isn’t a good idea, https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea.
[4] O. Bonaventure, “MPTLS: Making TLS and Multipath TCP stronger together draft-bonaventure-mptcp-tls-00,” https://tools.ietf.org/html/draft-bonaventure-mptcp-tls-00, Oct. 2014.
[5] R. Dingledine, N. Mathewson and P. Syverson, “Tor: The Second-Generation Onion Router - Onion Routing,” *Proceedings of the 13th conference on USENIX Security Symposium*, 2004.
[6] Firebug 2.0.17, https://addons.mozilla.org/zh-tw/firefox/addon/firebug.
[7] T. G. Kale, S. Ohzahata, C. Wu and T. Kato, “Improving the Unfair Distribution of Tor Circuit Traffics,” *IEEE 17th International Conference, High Performance Switching and Routing* (*HPSR*), 2016.
[8] Y. D. Lin, F. Baker and R. H. Hwang, *Computer Networks: An Open Source Approach*, McGraw-Hill, 2012.
[9] MEGA service, https://mega.nz.
[10] MultiPath TCP, https://www.multipath-tcp.org.
[11] NIST/SEMATECH, *e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/handbook/prc/section4/prc471.htm.
[12] NS-3, https://www.nsnam.org.

[13] J. Reardon and I. Goldberg, Improving Tor using a TCP-over-DTLS Tunnel, *Master's thesis, University of Waterloo*, 2008.
[14] M. A. Sulaiman and S. Zhioua, "Attacking Tor through Unpopular Ports," *IEEE 33rd International Conference on Distributed Computing Systems Workshops*, 2013.
[15] The anonymous Internet, http://geography.oii.ox.ac.uk/?page=tor.
[16] The lifecycle of a new relay, https://blog.torproject.org/blog/lifecycle-of-a-new-relay.
[17] Tor: Overview, https://www.torproject.org/about/overview.html.en.
[18] Tor routing figure, http://cdn.arstechnica.net/wp-content/uploads/2014/01/tor-structure.jpg.
[19] Why is Tor slow, https://tails.boum.org/doc/anonymous_internet/why_tor_is_slow /index.en.html.
[20] World Map, https://en.wikipedia.org/wiki/World.

Biography

Te-Yu Liu (andyowenx@gmail.com) received his master's degree in Computer Science and Information Engineering from National Chung Cheng University in 2016. His research interests include Tor network and network security.

Po-Ching Lin (pclin@cs.ccu.edu.tw) received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2008. He joined the faculty of the Department of Computer and Information Science, National Chung Cheng University (CCU), Chiayi, Taiwan, in August 2009. He is currently an associate professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.