Electronic Preprint for Journal of Information Processing Vol.23 No.5

Regular Paper

Reducing Congestion in the Tor Network with Circuit Switching

Kale Timothy Girry[1,a)] Satoshi Ohzahata[1] Celimuge Wu[1] Toshihiko Kato[1]

Received: December 5, 2014, Accepted: June 5, 2015

Abstract: The Tor network is a distributed circuit-switching overlay network, which provides anonymous communication by using voluntarily running onion routers around the world. Tor is vulnerable to network congestion and performance problems because circuit traffics with different rates are competing to transfer their data through a single TCP connection. A large fraction of available network capacity is consumed by the bulk users' traffic, resulting in increasing delays for the light interactive users. The unfair distribution between the circuit traffics of bulk and light users are contributing to bottleneck in the Tor routers. This problem increases the end-to-end latency and reduces the quality of communication in Tor, which discourages many users from using and joining the network. As a result, the degradation of Tor performance does not only affect the users' experience, but also degrade the anonymity of Tor. In this work, we discovered that the current Tor design encountered problems from several performance and deployment issues relating to lower network capacity. To improve the problems in Tor, we applied the circuit switching method and addressed the short-comings of limited network capacity, by connecting the congested OR to higher bandwidth ORs. The proposed method is evaluated on our setup testbed environment and partly in the live Tor network. The experimental results showed that TCP socket buffers and Tor network capacity are better utilized and the overall end-to-end latency is reduced.

Keywords: onion router (OR), onion proxy (OP), circuit congestion, TCP buffers, Tor buffers

1. Introduction

The onion routing (Tor) network is a distributed overlay network designed to anonymize TCP-based applications like web browsing, secure shell, and instant messaging [1]. Tor consists of onion routers (ORs), which are operated by volunteers around the world to relay the user's traffic. Tor clients work as onion proxies (OPs) and build circuits through the ORs in the network to dispatch their traffic. The goal of Tor is to protect its clients OPs from any adversaries who can easily identify the users' physical locations and information they access through the Internet. Tor uses a layered encryption scheme based on onion routing to strengthen the anonymity service.

Although Tor gained wide popularity, it is vulnerable to network congestion problems due to unfair distribution of traffic between the bulk and light traffics competing to send data through a single TCP connection. This problem imposes greater latency to majority of Tor web users, than they would experience without using the Tor network [2]. In addition, high latency and low communication quality of Tor discourage many users from joining the network, which leads to degradation of anonymity services provided by Tor.

To help the wide usages of Tor, incentive-based schemes have been introduced to encourage users to join and donate bandwidth to the network to reduce the traffic pressure on the routers [3], [4]. Many users donating ORs to relay traffics would result in increasing capacity of the Tor network, but few users would lead to lower network capacity and increase Tor network-wide congestion [5].

Tor decentralized congestion control with the techniques of end-to-end acknowledge messages, and detect congestion on network edges by sending less data until the congestion reduces [1]. The problem of this design is the total latency along the circuit which contributed to longer delay of reply message to detect congestion, and to repair packet losses. Tor does not react quickly when applying its end-to-end window based flow-control algorithm. Therefore, other congestion controls techniques have been proposed to address the problem of flow-control in Tor [6], [7]. The major problem with Tor is the increase of congestion due to a small fraction of users using greedy file-sharing application such as BitTorrent that consumed a lot of network bandwidth.

Reardon and Goldberg [8] pointed out that bulk circuits are often multiplexed with light circuits in the same TCP connection, and result in unfair application of the TCP congestion control of the shared connection on all circuits. They proposed TCP-over-DTLS, where every circuit gets a separate user-level TCP connection, and DTLS is used for encrypting and securing the communication between routers. Unfortunately, TCP-over-DTLS faces drawbacks in deployment and performance problems issues that hinder its adoption. In addition, for any pair of Tor routers to use TCP-over-DTLS, both onion routers need to be upgraded which would result in complexity of transport design.

Tang and Goldberg [9] proposed a method to address the problems based on prioritizing Tor traffic of users in each OR to reduce the high congestion and, focused on improving the quality

[1] The Graduate School of Information Systems, Department of Information Network Architecture, The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] tgk@net.is.uec.ac.jp

of services for the interactive user traffic. They implemented their method of calculating the exponentially weighted moving average (EWMA) of the number of cells to measure the recent activity of a circuit, and then giving higher priority to circuits that have lower EWMA value, which classified as light interactive traffics to have their cells transferred. The problem with their approach is bulk and light circuits are still using the same TCP connection to send cells to the next OR. Therefore, the competition of sending cells still occurs since the incoming rates of different circuit traffics are not equal. Furthermore, since the prioritize approach in Tor only serves larger bandwidth for the higher priority traffic than that of the lower one in the limited bandwidth of OR, the benefit of this approach is still limited.

The aim of this paper is to address the unfair distribution of bulk and light traffics on a congested Tor circuit. We focus on applying the circuit switching method to reduce the network congestion by increasing the capacities of the Tor routers. This approach allows us to address the causes of higher delays on constrained OR that multiplex multiple incoming circuits. We evaluated the effectiveness of the circuit switching method and offer the following contributions;

- We showed why the active circuit switching approach can increases the capacity of OR and reduce the circuit congestion.
- We improved the congestion on the selected ORs along the circuit by applying the control schemes to monitor the buffer occupancy in the entry ORs.
- Our control schemes can easily monitor the congestion state in Tor and helps to improve the end-to-end throughput and latency problems.
- We showed a simple circuit switching method with small modification to stock Tor protocol.
- The proposed circuit switching approach is evaluated on our setup testbed environment and partly in the live Tor network.

The rest of the paper is structured as follows. We provide the related works in Section 2 and discuss the background of Tor network in Section 3. We elaborate on our proposed method in Section 4 and evaluate it in Section 5. Finally we conclude our study and state our future work in Section 6.

2. Related Works

There are several proposals that aim to increase the total number of ORs to increase the network capacity and reduce the congestion in Tor [3], [4], [10]. Although the network capacity of Tor could be increased with new relays, we believe this approach is a short-term solution. This is because faster network attracts bulk users which can consume more network resources. Hence, the congestion continues to occur on any selected ORs in Tor.

Dingledine [11] describes a method which tunes available options of Tor nodes, specifically PerConnBWRate and PerConnBWBurst configuration options, to control the traffic and reduce the effect of buffer overflow. Both these options are used to separately rate-limit every connection from a non-relay. When applying the options, the heavy clients can be throttled at the entrance router. The issue with this approach is that the configuration is static. This approach does not take care of the current load and state of the network. Even if there is bandwidth available, clients could unnecessarily throttle.

Tschorsch and Scheuermann [12] pointed out the problems of previous work and explained how the division of bandwidth among the circuits is not fair. User-configured bandwidth is divided equally among connections, and each connection's bandwidth is divided equally among the circuits. They implemented a solution which achieves max-min fairness between circuits and uses the N23 congestion feedback scheme [7], [12] to better make use of bandwidth and prevent congestion. However, their approach raises a problem when opening many light circuits more than a heavy circuit [7]. Less bandwidth is available to a light circuit if one TCP connection accommodates many light circuits traffics [12].

Gopal and Heninger [13] proposed a method called "Torchestra" to solve the unfair distribution between the bulk and light users in Tor. In their proposal, two TCP connections are used for OR-to-OR communication. One TCP connection is dedicated for light circuits and another is dedicated for heavy circuits. They used the EWMA proposed by the Tang and Goldberg [9] to classify circuits into light and heavy categories. They suggested that having separate connections might help solve the unfairness problems in Tor. The problem to their approach is opening two connections for light and heavy traffics between the same pair of ORs that are highly congested can still degrade the performance of light interactive traffics. In addition, both ORs can causes overhead (TCP kernel memory overflow and mismanagement of traffic flow) if accommodating many light and bulk traffics.

3. Background

In this section, we provide an overview of the Tor network and its current transport design. We identify the problems in Tor and elaborate on the problem effects that degrade the performance of Tor.

3.1 Tor Overview

The Tor network is a second generation of onion routing overlay network [1]. A client connects to a destination through Tor first contact a directory server to obtain the list of Tor nodes. Next, the client constructs a circuit of three Tor routers (or nodes) and forwards traffic through the circuit to a desired destination using a layered encryption scheme based on onion routing [14]. To balance the traffic load across the routers' bandwidth, clients select routers in proportion to their bandwidth capacities. Each onion router (OR) runs as a user-level process without any special privileges. All relays in Tor communicate using pairwise TCP connections. A pair of the relays communicates with several ORs at once and all circuits between the pair are multiplexed over their single TCP connection. Each circuit between ORs carries traffic for multiple services that the user is accessing. Each Tor user locally runs software, which is called an onion proxy (OP). Onion Proxy (OP) presents a SOCKS proxy interface to local applications, such as web browsers.

3.2 Circuit Constructions

When an application makes a TCP connection to OP, the OP

splits the TCP segment into 512 bytes of fixed-size cells, and forwards them over to Tor network. The client (OP) selects a first OR (entry OR_1), and makes a TCP connection as well as the transport layer security (TLS) on that connection. Tor entry OR_1 multiplexes circuit from one or more OPs into a single outgoing TCP connection. The OPs then instructs entry OR_1 to connect to a second OR (OR_2). The OP then instructs OR_2 to contact a third OR_3 and the TCP connection is made between OR_2 and OR_3. The last OR_3 performs de-multiplex circuits from the OR_2 and, opens the TCP connection to the responder web server. Then the OR_3 reports a one byte status message back to the client application proxy.

The ORs communicate with one another and with client OP via TLS connections with the ephemeral keys. To identify different circuits, each OR assigns different circuit IDs to the circuits. By default, Tor uses three hops circuit to enhance the OPs anonymity. If privacy is not an issue, the number of hops can be reduced to one or two ORs circuit with specific exit node. Reducing the number of hops improves the circuit performance, without experiencing many unexpected delays through ORs.

3.3 Tor Router Architecture

Figure 1 describes the Tor router architecture that includes the circuit and TLS in the application layer and the host receiving and sending TCP socket buffers for a single onion router [1], [8]. The TCP stack is responsible for providing hop-by-hop congestion control, reliability and in-order delivery for TLS data. The TLS conceals data and encrypts the segments of the circuit connections to provide hop-by-hop authentication, integrity and confidentiality between nodes. The circuit cryptography layer is responsible for confidentiality, providing label-switching routing and performing demultiplexes and multiplexes of multiple circuits carried by the TLS [1], [8]. An input buffer inside Tor is where Tor places cells from incoming circuit traffics it reads from the TLS socket. Each cell is processed and pushed into the corresponding FIFO circuit queue, then further transferred into the output buffer. The output buffer inside Tor is used for cells that have been processed and are waiting to be dispatched to corresponding sending TCP socket.

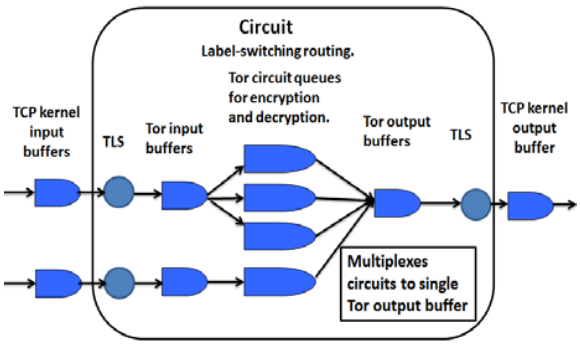

Fig. 1 Tor router architecture's protocol stacks of a single OR. Cell processing involves encryption/decryption in the FIFO circuit queues. The TCP kernel input buffer holds the data from other OR and transfers them into Tor application. TCP kernel output buffer holds the data from the Tor application and transfer them to the outgoing traffic.

3.4 Performance Problems in Tor

3.4.1 Tor Design Problem

Reardon and Goldberg [8] have identified some problems with Tor design when Tor's OPs and ORs communicate with each other using TCP connections. Every entry OR to other ORs has TCP connection multiplex circuits from several Tor clients.

Figure 1 shows the Tor OR multiplexing the incoming circuits' traffics to a single outgoing TCP connection. The single OR ensures that the flow of traffics appear in the correct order in which the component streams are multiplexed.

Reardon and Goldberg [8] pointed out that this design can potentially hinder the performance of interactive circuits carrying light traffics. They believe that multiplexing TCP streams over a single TCP connection is unwise and results in an unfair behavior of TCP's congestion control mechanism. It results in multiple data streams competing to send data over a TCP stream which gives more bandwidth to circuits that send more data. The Tor scheduler gives each byte of data the same priority regardless of its sources. A busy circuit that triggers congestion control causes light interactive circuits to struggle to have their cells sent, which result in increasing delays.

3.4.2 Causes of Congestion Delays in Tor

Joel Reardon [15] stated that the encryption computation in Tor does not represent the largest source of delays, since the OpenSSL's reads require 30 microseconds and writes only requires 40 microseconds. The time Tor takes to process 90% of cells is between 8 to 10 microseconds. For this reason the cryptography computational delays is eliminated and not a cause of delay in Tor.

The delay in Tor depends on two components, the delay through the node and the delay attributed to the latency of traversed overlay links [16], [17]. Delays occur at the node level when a node reaches its bandwidth limit, and when a node's connection to the Internet is congested. When a node is congested, the out-going cells must wait in the node's output queue. Some studies [2], [6], [16], observed that the Tor's token bucket rate limiting implementation often contributes to congestion delays of up to one second per node. These delays are detrimental to Tor router performance.

We also studied the node based delays in Tor [18] to find out the causes of congestion in Tor and, observed that the node based delay is significantly larger than the propagation link delays from the OPs to the exit OR. The average delay in a single node is 1.12 ± 1.13 seconds and the average propagation delays from the OP to exit OR is 0.38 ± 3.36 seconds. Packet queuing due to bottleneck in the node kernel buffers causes the node delay to increase and affects the overall congestion delays in Tor. The end-to-end latency delay is 2.94 ± 2.37 seconds and the average delays of packet queued in the node buffers is 0.82 ± 0.69 seconds.

From the measurement results, we concluded that the main cause of congestion delays in Tor is due to bottleneck occurring in node when multiple circuits are competing to send data to a single TCP connection.

4. Proposed Method

To solve the above problem, we focused on improving the

overall network capacities and the end-to-end throughput by switching the Tor circuit to different TCP connection that has lower congestion. We took a different approach from the related works [11], [12], [13], by applying the circuit switching of active bulk traffic that consumes a lot of bandwidth resources to another TCP connection. We applied the circuit switching approach and improve the flow rates of both light and bulk traffics because the available capacity along the circuit is improved [*1].

4.1 Metrics for Congestion Detection

We applied two metrics to evaluate and monitoring the unfair distribution of multiple circuit traffics across congested ORs. One is the output buffer occupancy in Tor application level, and the other is the number of TCP socket un-writable events in the transport level.

4.1.1 Tor Output Buffer Occupancy

This scheme can incur a significant overhead for determining incoming and outgoing circuit rates occupying the output buffer in Tor. By default, Tor has a buffering capacity for the instantaneous incoming and outgoing circuit traffics in the application level for input and output buffers occupancy, and the receiving and sending TCP kernel buffers as shown in Fig. 1. We applied this metric to monitor the OR throughput, which we denoted as the available capacity of OR, and the expected throughput of a circuit traversing the selected OR. We measured the relative rates for each circuit by measuring the buffer occupancy of all incoming and outgoing traffics.

We applied the per-circuit connection cell exponential weighted moving average (EWMA) to monitor the output buffer occupancy over time. This metric is used to classify circuits into light and bulk traffics that overflow the Tor output buffer. The per-circuit connection EWMA algorithm that we used is computed in much the same way as the EWMA algorithm that was proposed by Tang and Goldberg [9]. The difference is, whenever the circuit's cell counter is incremented, the cell counter of the outgoing connection to which that circuit belongs is also incremented. Our control scheme is based on this incremented value to identify which circuit contributed to overflowing the output buffer [*2].

We continuously update the EWMA value for each circuit when performing our experiment. We rely on the observation that bulk connections have higher EWMA values than light traffic connections, because bulk clients always steadily transferring data. When the OR output buffer is overloaded and the sending TCP socket cannot receive any more data, the circuit traffic that produces the congestion is switched to another TCP connection that was preemptively built by OP at the beginning of circuit creation. This gives a better choice of switching the bulk circuit that causes the bottlenecks in the selected ORs.

4.1.2 TCP Socket Un-writable Event

Joel Reardon [15] has fully detailed the effects of peer acknowledging for data return slowly even if the data was transferred properly. Since the TCP output buffer is where TCP maintains a perfect record of all unacknowledged data. This record is used to generate packet retransmission if a packet dropped. The TCP kernel output buffer usually swells up until acknowledgements are received. In a worst case, the sending TCP buffer size grows to a point where more memory cannot be allocated, and consequently the operating system reports the socket as unwritable.

In this work, we monitor this situation when un-writable events occur to indicate the TCP socket buffer cannot receive any more new data. We detect the socket as un-writable when executing the node monitor algorithm, and observe the writing events for the outgoing circuit reported as zero byte. This indicates the sending TCP socket cannot accept any new data because the buffer is overfull and congestion occurs at a single TCP connection.

4.2 Congestion Detection Algorithm

The classification of a circuit with heavy traffic is done at the entry OR. In this work we consider the case where only the entry ORs are responsible for switching a circuit and the entire circuit is switched at once. It is simple and effective to work at the entry ORs to determine the attribute circuit congestion to other ORs. In addition, the current distribution of bandwidth in the Tor network shows that the entry guards ORs have the higher probability of being chosen due to higher bandwidth and stable performances [6]. Therefore, applying the classification of circuit traffics and switching method at the entry OR is more effective to detect the congestion along the circuit.

For chosen buffer occupancy over time, the entry OR collects statistics about the number of cells sent on each circuit queue to the corresponding output buffer in Tor.

We modify a small portion of stock Tor algorithm to calculate and estimate the EWMA periodic samples of instantaneous output buffer occupancy B for each incoming circuit in Tor, which is expressed in Eq. (1).

$$B_t = \gamma * B_{t-1} + (1 - \gamma) * B_{sample} \quad (0 < \gamma < 1) \tag{1}$$

B_{sample} is the current collected number of sample cells sent from the circuit queues and occupied the output buffer in Tor.

The EWMA parameter γ is the fractional weight of the previous buffer estimated in the EWMA estimator. We used it to estimate the current buffer occupancy and determine the depth of kernel memory usage in the allocated output buffers in Tor. The larger γ has a greater influence on the number of cells occupying the output buffer by the incoming circuits.

Algorithm 1 shows the pseudo code to classified circuit traffic. We define α and β to be the minimum and maximum buffer threshold values [*3], below α is considered as buffer is underfull (line 21) and above β is considered as buffer overfull (line 9). If the current buffer occupancy B_t is within the acceptable range of minimum and maximum ($\alpha * B_{t-1} \leq B_t \leq \beta * B_{t-1}$) (line 18), we do not take any action to switch the circuit that has higher EWMA value. However, if the Tor output buffer rises above the maximum threshold ($B_t > \beta * B_{t-1}$) (line 9) or the sending TCP

[*1] The light circuit traffics can easily transfer their cells without any need to queue behind the bulk traffics when congestion control occurs.

[*2] Note that the transferred cell EWMA value is for each client circuit and does not affect other circuit EWMA value.

[*3] Note that the buffer occupancy thresholds also show how aggressive the flow rate of each incoming and outgoing TCP connections.

kernel buffer cannot accept any new data $S \rightarrow 0$, we switch the circuit that has the higher EWMA value to different connection. This approach allows the circuits with lower EWMA value (light interactive circuit) to have better throughput performance.

We have experimented the switch timing of circuit traffics for appropriate alpha α and beta β used in our control metrics. The most appropriate minimum and maximum thresholds for the four OPs circuits is when $\alpha = 0.1$ and $\beta = 0.5$. These threshold values have smaller circuit switching time of 1 second and provide better throughputs for all bulk and light circuits. Note that these threshold parameters are fractional of the total allocated memory for each output buffer size in the OR, which is 32 KB. Details of the experiments are shown in Section 5.3.1.2.

Algorithm 1 Decision pseudo code to estimate the output buffer occupancy and switches the bulk circuit traffic.

```
1.  α = 0.1, β =0.5
2.  C ← getConnectionList()
3.  L ← L:length()
4.  M ← getMetaEWMA()
5.  B ← getOutputBuf
6.  S ← getSocketBufEvent

7.   If L > 0 then
8.   M ← M: increment
9.       if (B_t > β * B_{t-1} or S → 0 ) do // buffer overfull ||
                                              socket un-writable
10.          for i ← 1 to L  do
11.           if C[i].ClientConnection() then   // get connections
12.              if C[i].EWMA > M then        // compare circuit-
                                               with high EWMA-
                                               value.
13.                Switch C [i] → B[i]      // switch connection-
14.              end if                      with high EWMA-
15.           end if                         to corresponding-
16.        end for                           output buffer connection.
17.      end if
18.      if (α * B_{t-1} ≤ B_t ≤β * B_{t-1})
19.       Do nothing              // accept the current buffer usage
20.      end if
21.      if (B_t < α * B_{t-1})          // buffer underfull
22.        L ← L: increment          // OR can accept new circuits
23.        Drop_circuit(circID,conn())   //OR can drop of any-
24.       endif                        inactive circuit to free-
                                       space.
```

4.3 Circuit Switching Procedures

This subsection explains in details the proposed analysis of circuit switching. We explained where the congestion detection point and the circuit switching initiator point.

Figure 2 shows an example of the switching method we applied on OP_2 circuit as the bulk circuit traffic that switched to different TCP connection. The flow of traffics is from the client OPs to the web server. In this environment, the communication between the OPs and the exit OR uses Tor protocol over TLS protocol. Then the communication between the exit OR and the web server uses TLS protocol. The web server accepts the TLS session resumptions [25] to resume connections after switching of circuit, and continue receiving the packets from the exit ORs.

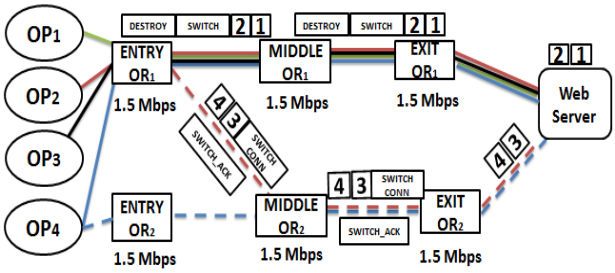

Fig. 2 Circuit switching approach and experiment setup. Solid lines indicate the initial circuit built by all the OPs. The dashed lines indicate the switched circuits passing through different TCP connections. The configured bandwidth on all the ORs is 1.5 Mbps.

In the proposed approach, the entire client OPs build two circuits to the web server and one of the circuits is selected for data transfer. The switching of circuit to a different TCP connection happens in the entry ORs. The entry ORs sends the control message to other ORs when its current output buffer rises above the maximum β threshold and sending TCP socket is un-writable.

To make this approach easier to implement, we applied the method of Gopal and Heninger [13] of connection switching protocol between two opening TCP connections. The similarity to their work is that we consider only single OR to be responsible for switching a circuit and, the entire circuit switches at once. We applied two control cells that Gopal and Heninger [13] defined to manage the connection transfer, which are SWITCH and SWITCHED_CONN cells that sent by the entry ORs when congestion occurs. A SWITCHED_CONN and SWITCH_ACK cells have no payload. The SWITCH cell's payload contains a flag that indicates that all further cells for the old circuit will be sent on the new TCP connection. Hence, the ORs on the old circuit can discard the corresponding circuit ID.

The differences are, first, we defined a SWITCHED_ACK message that sent by the receiving node to inform the initiator (sending OR) that the switch is completed and can send further cells on the new circuit. The second difference is we distributed the circuit traffics from the entry OR to different ORs instead of using the same pair of ORs. When the entry OR switch the circuit to new TCP connection, the old circuit is discarded after the entry OR sends the DESTROY cell, which contains a single octet to the adjacent OR on the appropriate direction's CircID. This process of sending the DESTROY cell has already been implemented in Tor protocol specification [19]. Third difference is only the bulk circuit traffics that causes the congestion to occur (i.e., circuit traffics with higher EWMA value) is switched to new TCP connection, based on our two control schemes. After switching the bulk circuit traffics to the new TCP, the OP owning the bulk circuit continues to send and receive cells from its current pending requests. We described our circuit classification and switching of active circuit via single entry OR (entry OR_1) and, via different entry OR (from entry OR_1 to entry OR_2) as shown below.

4.3.1 Mechanism to Switch the Circuit via Single Entry OR

This subsection explains the mechanism of switching the cir-

cuit via a single entry OR. Figure 2 shows the switching of OP_2 circuit at the entry OR_1.

The steps are as follows:

1. First, we allowed the entry ORs and middle ORs to relay traffics within Tor network based on the circuit selected by all the OPs. At the initial stage, the OP_1, OP_2, OP_3 and OP_4 are multiplexed on the entry OR_1 and passes through the middle OR_1 and exit OR_1.
2. Before the entry OR_1 switches the circuit to the separate TCP connections, the entry OR_1 monitors the incoming circuit traffics and checks if there is no congestion occurs on all the ORs along the circuit. The Algorithm 1 keeps track of the buffer occupancy to see if it stays within the acceptable range between minimum and maximum thresholds and whether the TCP socket is un-writable.
3. When the entry OR_1 detects the OP_2 circuit is flooding the output buffer occupancy beyond the maximum threshold β, the congestion occurs. The entry OR_1 switches the current circuit of OP_2 that send the cells with sequence number 1 and 2 to a different TCP connection. The remaining cells with sequence number 3 and 4 are transferred through the other circuit that the OP_2 preemptively built. The process of switching circuit involves the entry OR_1 sends a SWITCH cell on the old bulk circuit (OP_2 circuit) with right CircID to inform the middle OR_1 that no more cells for this circuit are coming on this connection. The middle OR_1 then sends the SWITCH cell to the exit OR_1 and discard the circuit when receiving the DESTROY cell. When the exit OR_1 receives the SWITCH and DESTROY control cells, the exit OR_1 first sends the cells with sequence number 1 and 2 to the web server then it tears down any associated edge connections for the corresponding circuit ID. The cells with sequence number 1 and 2 which are received by the web server are kept in the input queue and, processed when the remaining cells with sequence number 3 and 4 arrive from the middle OR_2 and exit OR_2.

   Next we describe the details of OP_2 packet received in the web server from exit OR_1 (sequence number 1 and 2) and exit OR_2 (sequence number 3 and 4). Tor uses Transport Layer Security (TLS) session resumption [25], which enables the Transport Layer Security (TLS) to resume sessions of sending the packets with sequence number 3 and 4 to the web server, after the client OP_2 sends the "Stream Identifier" of the packets and relay begin cell that contains the IP address and port number of the web server. Tor applies this approach to resume sending of data to the web server after the active circuit traffics are switched to exit OR_2. Furthermore, the TCP segments that both exit OR_1 and exit OR_2 nodes sending to the web server has the "Stream Identifier" header [21], which holds the length of the relay payload, relay command and stream identity. Initially, when the client OP_2 constructs a new TCP connection through the exit OR_1 to reach the web server, the OP_2 chooses the "Streams Identifier" for all the streams data which will be sent. The OP_2 constructs the relay begin cell with a payload encoding the IP address and port number of the destination host web server (i.e., after receiving handshake reply message from the web server). Therefore, as the exit OR_1 receives the streams data (sequence number 1 and 2) in Tor gateway layer, the exit OR encapsulates the cells, and sends it out to the appropriate host via the TCP/IP connection to reach the web server. Note that TCP layer is responsible for providing in-order delivery, and reliability for TLS incoming and outgoing data. When the entry OR_1 switches the circuit to middle OR_2 and exit OR_2, the OP_2 client resumes the session, which includes sending Stream Identifier and relay begin cell. When the exit OR_2 receives the information of relay begin cell, it sends the packet with sequence number 3 and 4 to the same web server IP address and port number. With the same information encoding in the relay begin cell, the packets sent by the exit OR_1 and exit OR_2 can be successfully delivered to and processed (reordering of packets) by the web server.
4. After step 3, the entry OR_1 then sends a SWITCH_CONN cell on the new TCP connection followed by the circuit's cells to the middle OR_2. Once the middle OR_2 received the cell, it sends a SWITCH_ACK cell back to entry OR_1 (the initiator). The entry OR_1 instructs the middle OR_2 to send the SWITCH_CONN cell to the exit OR_2, and the exit OR_2 sends back the confirmation SWITCH_ACK cell back to middle OR_2.

   Next, we explain the details of communication between the exit ORs and the web server. In Tor application layer, the entry OR_1 gateway protocol instructs the gateway protocol at the exit OR_1 to send the packets (sequence number 1 and 2) to the web server when the connection is ready. The web server can then successfully accept the data after the exit OR_1 sends the packets. In addition, as soon as the exit OR_1 receives the SWITCH and DESTROY control cells, the exit OR_1 tears down any associated edge connections for the corresponding circuit ID to web server. Furthermore, since the connection between exit OR_2 and the web server are preemptively built by client OP_2 through middle OR_2 and exit OR_2 nodes. When the exit OR_2 receives the SWITCH_CONN control cell and sends back the SWITCH_ACK cell indicating the connection is ready to be used, the entry OR_1 (at the gateway layer in Tor) instructs the exit OR_2 to transfer the remaining packets with sequence number 3 and 4 to the right destination web server. The web server then receives the TCP segments and sends the ACK message back to the exit OR_2. Since Tor uses the Transport Layer Security (TLS) session resumption [25] to resume connections after switching of circuit, the Transport Layer Security (TLS) can resume sessions of sending the packets with sequence number 3 and 4 to the web server via exit OR_2.

   Note that our method can still work for general applications even if the server does not accept or uses the resumption of TLS session. In cases that the server does not uses the TSL connection, if Tor maintains the same exit OR connection to the server, the switching of circuits only occurs within the Tor network. In addition, since our circuit switching procedures also use the default Tor protocols (i.e., all the OPs circuit can multiplexes and de-multiplexes at the single

exit OR_1 to reach the web server), the connection between the exit OR and the server can still be maintained to transfer data.

4.3.2 Mechanism to Switch the Circuit via Different Entry OR

This subsection explains the mechanism to switch the circuit via different entry ORs. Figure 2 shows the switches of OP_4 circuit from the entry OR_1 to entry OR_2.

The steps are as follows:

1. Initially, the OP_1, OP_2, OP_3 and OP_4 are multiplexed on the entry OR_1 and passed through the middle OR_1 and exit OR_1.
2. When the entry OR_1 detects the OP_4 circuit is flooding the output buffer occupancy beyond the maximum β threshold, and the congestion occurs. The entry OR_1 switches the current circuit of OP_4 via another entry OR_2 that OP_4 preemptively built. The mechanisms to switch the OP_4 circuit to different entry OR_2 are explained in the next step.
3. The entry OR_1 sends a SWITCH_CONN control cell to the middle OR_2, and the middle OR_2 extends the SWITCH_CONN cell to the exit OR_2. If the entry OR_1 does not receive any SWITCH_ACK in reply back from the middle OR_2 and exit OR_2, or the SWITCH_ACK remains unattached for "SocksTimeout" (default 2 minute), at the entry OR_1 [21], the entry OR_1 discards the control cell and sends an error message back to the client OP_4 as appropriately to close the SOCKS connection. Client OP_4 will immediately switch its circuit to another circuit via entry OR_2, which was preemptively built and successfully connected to the web server [21]. Tor build circuits preemptively, which means number of circuits are kept ready even if there is no data to be sent yet through entry OR_2, middle OR_2 and the exit OR_2 [*4]. When a circuit passing through entry OR_2 is ready to be used, Tor (client OP_4) attaches the request's stream to the circuit and sends a BEGIN, BEGIN_DIR and RESOLVE ORs pending cell as appropriate to attach on the entry OR_2, middle OR_2 and exit OR_2. This configuration of stream attached to new circuit was made by default in Tor algorithm to continue on with the transfer TCP streams [21]. Since the proposed method also work with the default Tor algorithm, this switching of circuit to different entry ORs is made possible for all client OPs.

4.4 Circuit Selection Method

For the circuit selection method, we have configured the Tor auto-circuit algorithm [20] in the client OPs to expose the functionality of TorCtl library. This allows us to tune the OP operations timely, such as query the runtime and available bandwidth and, control the number of circuits to build with specifying the number of hops.

In the first part of our evaluation, the circuit selection method is manually configured since we want to observe if the circuit switching method of active TCP connection can improve the unfair distribution between the bulk and light circuit traffics. The dynamic selection of circuit in the live Tor network will be our future work.

In this paper, we concentrated only on evaluating the effect of the circuit switching. We carefully consider the case that changing the behavior of Tor circuit selection [21] might enable new attacks. Therefore, we make sure that we do not degrade the anonymity of Tor too much by ensuring our setup entries ORs to be guard nodes. Since our current work is focused more on the performance issues in Tor, we will address the anonymity of Tor in relation to this work in the future studies.

In the second part of our evaluation, we evaluated our proposed method partly in the live Tor network, where our entry ORs are configured to run in our testbed environment and the middle and exit ORs are selected from the public Tor network. We selected the good route through the middle and exit ORs that have a reliable and higher OR capacities to ensure fairness between the bulk and light circuit traffics before switching, and after switching the bulk traffics. Our experimented OPs constructed circuits with a specified distance between each OR. The ORs are selected using uniform selection method, one-by-one from the current list of running ORs in the live Tor network to distribute the traffics. There are other studies in Tor that shows Tor path selections to improve the fairness and the performance in the Tor Network [6], [23], [24].

5. Performance Evaluation

In this section, we evaluated the performance and show the possible improvement of our circuit switching method. Firstly, we show our method evaluation that we analyzed in the testbed environment. Secondly, we evaluated the proposed method partly in a live Tor network to observe the effectiveness of our approach.

5.1 Experiment Setup

5.1.1 Network Setting

Figure 3 shows the experiment setup for our evaluation. All the ORs were running on the commercial 1 Gbps network in our University laboratory. The client OPs were running on another network in our university. The OPs and ORs were running on Ubuntu OS 14.04.2 LTS CPU 2.60 GHz 32 bit with 4 GB of RAM. We maintained the same memory and CPU for all the ORs to gain similar performance across the entire network. We run our setup ORs in the Tor network for more than a month to gain stability in performance. We configured our setup ORs

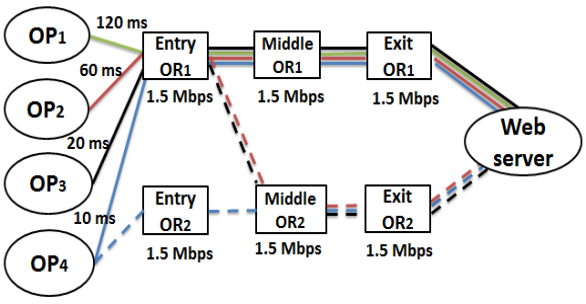

Fig. 3 The measurement method, solid lines indicate the initial circuit built by all the OPs. The dashed lines indicate the switched circuit passing through different TCP connection. The configured bandwidth for all the ORs is 1.5 Mbps.

[*4] Note that the connection to the web server is maintained by the OP_4 after circuit construction.

at the entry and middle position to relay traffic within Tor and, the exit ORs opens the TCP port 80 to reach our web server (https://www.dropbox.com/home). We work with the Tor stable source code 0.2.4.19 for all the OPs and ORs in our testbed. The ORs were running in the Internet with the bandwidth capacity of 1.5 Mbps. The entire OPs were uploading a 12.3 MB file simultaneously to the web server. The solid lines indicate the initial circuit built by OPs and the dashed lines indicate the switched circuit built after the old circuit is congested. The entries ORs are configured to open two TCP connections for OPs circuit to route their traffics through the Tor network. This is to improve the flow rates of data, TCP window sizes and the end-to-end TCP throughput in proportion to RTTs along the circuit. We use Netem emulator tool to enforce delay effects for the incoming TCP traffics to all the OPs. The delays enforced to OP_1, OP_2, OP_3 and OP_4 to the ORs are 120 ms, 60 ms, 20 ms and 10 ms, respectively. These settings enable us to configure circuit as bulk and light traffics through the Tor network.

5.1.2 Parameters Settings

The parameters setting of the measurement evaluations are explained below.

Socket Buffer Size: We configured the TCP socket buffer size varying from minimum 8 KB to average 16 KB. To measure the TCP socket buffer usage over time, we executed the libevent echo socket script to output the kernel memory usage. The confirmation results can show us that connecting any constrain ORs to higher TCP capacities can indeed increase the throughput for the entire Tor network.

EWMA Parameter (γ): In our experiment, γ is the fractional weight of the previous buffer estimate in the EWMA estimator for the current buffer occupancy of the incoming circuit traffics. We experimented with $\gamma = 0.7$. By setting the gamma parameter as $\gamma = 0.7$, we are able to distinguished between the bulk and light circuit traffics. This was done for per-circuit output buffer allocated for incoming traffic on the selected OR.

5.2 Measurement Method

We performed the measurements for network capacity, end-to-end throughput and latency for default 3 hops circuit through ORs setup in our laboratory testbed. From the experiment topology in Fig. 3, we allocated both the entry OR_1 and entry OR_2 to calculate the EWMA of transfer cells to each circuit output buffer every 15 seconds. We made this choice to increase the time to properly measure each circuit cell count and, time that cell transfer to corresponding output buffer in Tor. The EWMA algorithm gives us the estimated value of cells occupied by each Tor output buffer.

5.2.1 Circuit Traffic Analysis

Our analysis focuses on reducing the congestion in Tor by increasing the capacity in the Tor network. We measured the circuit traffic before congestion and when reaching the congestion state based on the buffer usage over time and socket un-writable event. We collected timing information from our Tor ORs using libpse [22] and recorded the time points when the cells enter the circuit queue, when they are moved from the circuit queue to the output buffer of the connection, and when they leave the output buffer. We performed 60 repeated measurements during uploading a 12.3 MB file to the web server.

5.2.1.1 OR Capacity

We used node monitor program script that runs on each OR to analyze the OR capacity, which changes over time in terms of data flow rates. The node monitor program output the OR capacity that shows the number of bytes that a relay can forward per second on all circuits going through it. We calculated the changes in ORs capacity by taking the differences of capacity "after switching" the active bulk circuit and, "before switching" the circuit traffics $C_{after} - C_{before}$. The increase of OR capacity after switching shows an improvement to our proposed method.

5.2.1.2 End-to-end Throughput

To measure the circuit end-to-end throughput from the OPs to the web server during upload, we used a Tor bandwidth monitor (TorCtl script) to measure the OR connections events. In addition, we pushed a bandwidth probe packet through the TorCtl port and measured the end-to-end throughput from OP to the web server. We performed 60 repeated measurements after 10 minutes intervals when upload is completed [*5].

5.2.1.3 End-to-end Latency

The end-to-end latency is measured by sending a single cell of 512 bytes (a TCP data packet which is part of an HTTP request) from the OP to the web server via a 3-hop circuit made through the n nodes (entry OR, middle OR and the exit OR). The average end-to-end latency varies depending on the extent of circuit's congestion, and the both side directions of all selected nodes. We obtained the delays experience by a single cell along the circuit and through the ORs (due to queuing and processing delays in the node) by applying the measurement method of Wang et al. [6].

Practically, we used the node monitor program to confirmed the node based delays by capturing the incoming and outgoing transfer rates of the data from the receiving TCP socket to Tor input buffer (upstream), and from the Tor output buffer down to the sending TCP socket (downstream).

Circuits that are multiplexed over a single TCP connection can be affected when a packet drops occur. Hence, each incoming circuit can experience increased end-to-end latency when this problem occurs. This means that packet is eligible to be dropped twice as often in the remaining incoming circuits. We illustrate this problem by showing the effective drop rates, which is the ratio of packets dropped to the total number of packets reported during measurements.

The results of our experiment measurements are shown in the next subsection.

5.3 Experiment Results

5.3.1 Analysis of Circuit Switching

We performed a series of experiments on our experiment testbed by transferring files to analyze and show the effectiveness of the circuit switching method. In addition we measured the performance for each client circuit before it was congested and after congestion occurred.

During upload of 12.3 MB file to the web server, we detected congestion occur at the entry OR_1 because OP_2 and OP_3 circuits

[*5] Note that we only measure our own circuit that we build through our setup ORs.

Table 1 EWMA values for each circuit with moving average of cells to the Tor output buffer.

Gamma	0.7			
Circuit ID	OP_1 Circuit	OP_2 Circuit	OP_3 Circuit	OP_4 Circuit
Avg. EWMA values	0.021 ± 0.01	0.051 ± 0.01	0.064 ± 0.02	0.013 ± 0.02

are largely occupying the output buffer, which goes above the maximum threshold β and the TCP socket cannot accept any more new data.

5.3.1.1 EWMA Values for each OP Circuit

Table 1 shows the output of EWMA values for each client circuit which transfers cells to corresponding output buffer at the entry OR_1. The table result shows the unfair distribution of cells transferred from all the OPs. OP_2 and OP_3 are higher compared to OP_1 and OP_4 circuit. The result matches with the previous findings [2], [6], [8], [16], [18] that delays on ORs are the principal contributor to Tor overall congestion delays. More importantly, we observed that although we enforce link delays to each respective OP circuit to entry OR_1, the OP_2 and OP_3 still transfer data quicker to obtained higher EWMA values of transferring cells compared to OP_4 circuit. The variance of delays on all ORs affects the performance of OP_4 circuit though it has a very lower link delay. It is very likely that the Tor circuit scheduling algorithm itself plays a role in the variation in delays across a given router.

5.3.1.2 Time to Switch Circuit

To evaluate the appropriateness of α and β on switch timing in Algorithm 1, we performed an experiment for different values of α and β thresholds and made decisions based on switch timing when active bulk circuit switches to different TCP connection on entry OR nodes. Switch timing is the time when sending the last cell on the old circuit path up to when the first cell is sent to the new circuit. Circuit switching occurs when there is a bottleneck in the output buffer occupancy in Tor (i.e., when the cells queue rises above the minimum α and maximum β thresholds values in the entry ORs), which are monitored by the EWMA algorithm. We evaluated the switch timing by uploading a 12.3 MB file for 60 repeated times. The threshold values for α and β are directly configured in the Algorithm 1 that runs in the entry ORs.

Figure 4 shows the cumulative distribution of circuit switch timing to evaluate the appropriateness of α and β used in Algorithm 1. We observed that 50% of circuit switch timing when $\alpha = 0.05$ and $\beta = 0.1$ is less than 1 second. This switch timing indicates the time it took the entry OR_1 to successfully switch the active bulk circuit to different TCP connection and continues on with the data transmission at the position of data discontinued in the previous circuit. When we configured the $\alpha = 0.1$ and $\beta = 0.5$, 50% of switch timing of active bulk circuit to different TCP connection is also less than 1 second, whereas 50% of circuit switch timing of $\alpha = 0.3$ and $\beta = 0.75$ is less than 2.15 seconds. When further configured the $\alpha = 0.5$ and $\beta = 0.9$, 50% of circuit switch timing is less than 3 seconds, which is significantly higher compared to other threshold values. We observed that when increasing the minimum α and maximum β threshold values, the switch timing also increases due to more cells queue up in the

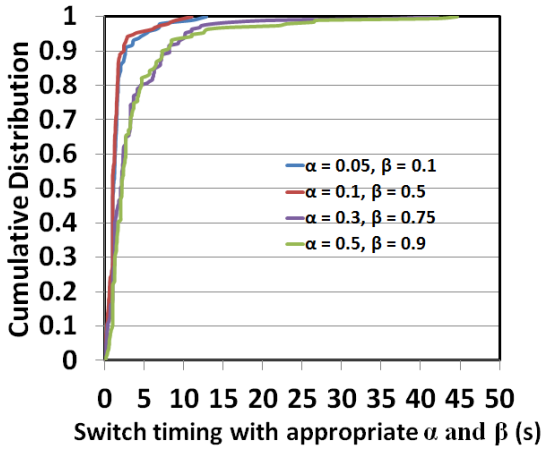

Fig. 4 Cumulative distributions of circuit switch timing with appropriate α and β used in Algorithm 1.

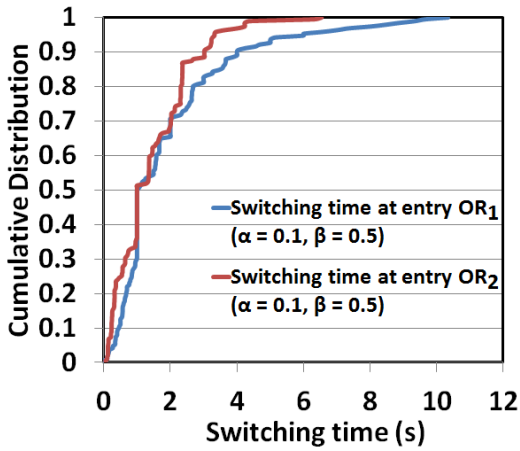

Fig. 5 Cumulative distributions of switching time for all the circuits switched to different TCP connection at the entry OR_1 and entry OR_2 nodes, when $\alpha = 0.1$ and $\beta = 0.5$.

Tor output buffer. As a result, the entry OR_1 requires more time to process the cells and send it to the corresponding output buffer. On the other hand, when lower the α to 0.05 and β to 0.1, the circuit switching delay is low and the transfer rates of cells passing through the entry OR is also low. This is because the output buffer occupancy quickly reached the maximum threshold value $\beta = 0.1$ (3.2 KB). Furthermore, when configuring the $\alpha = 0.05$ and $\beta = 0.1$, a single entry OR cannot accommodate large number of incoming circuits due to smaller output buffer size. As a result, we selected the most appropriate minimum α and maximum β thresholds values as $\alpha = 0.1$ and $\beta = 0.5$, because it has smaller average circuit switch timing required to switch the active bulk circuit to a different TCP connection.

After obtaining the appropriate α and β, we tested the appropriateness of α and β to whole laboratory testbed experiment for the entry OR_1 and entry OR_2 to obtain the average delays of circuit switching during uploading a 12.3 MB file to the web server. We performed 60 repeated measurements at the entry OR_1 and entry OR_2 nodes for all circuits, and observed the switching times of active circuit to different TCP connection. We combined all the measurement results and calculated the distribution of circuit switching time to obtain the average delays. **Figure 5** shows the cumulative distribution of switching times at the entry OR_1 and

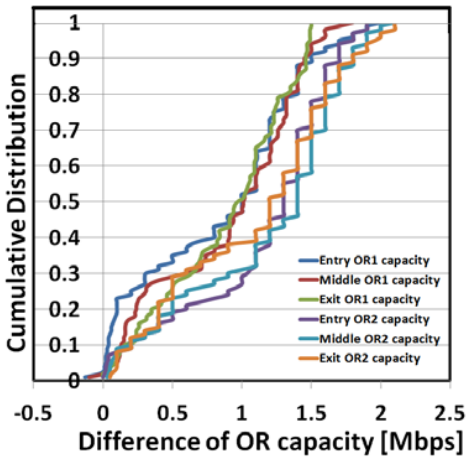

Fig. 6 Cumulative distribution of all OR capacities after and before switching the circuit, $C_{after} - C_{before}$.

entry OR_2 nodes, when $\alpha = 0.1$ and $\beta = 0.5$. The results show that 50% of circuit switching time at the entry OR_1 and entry OR_2 are both less than 1 second to successfully switch the circuit and continue on with the data transmission at the position of data discontinued on previous circuit.

5.3.2 OR Capacities After and Before Switching the Circuit

To improve the flow rates and reduce the congestion along the multiplexed circuit, the entry OR_1 switches both OP_2 and OP_3 to different TCP connections, which goes through the middle OR_2 and the middle OR_2 further transfer the traffics to the exit OR_2 (see Fig. 3). The OP_2 and OP_3 get a new circuit path through middle OR_2 which improves the capacity of the network and reduce the bottleneck on entry OR_1. We performed 60 repeated measurements during uploading of 12.3 MB file to the web server, to obtain the reliable results of ORs capacities changes [*6].

Figure 6 shows the differences of OR capacities for entry ORs, middle ORs and exit ORs. We observed the changes in Tor router capacities before and after bulk circuits switched and, calculated the differences of capacities by $C_{after} - C_{before}$. The difference shows the improvement of network capacities.

In overall tests that we performed, 50% of the increased ORs capacities after taking the differences of OR capacities $C_{after} - C_{before}$ for entry OR_1, middle OR_1 and exit OR_1 are less than 1 Mbps, 1.02 Mbps and 1.01 Mbps, respectively.

We also measured the entry OR_2, middle OR_2 and exit OR_2, under the same condition for all the OPs circuits, and observed the results consistently showed significant improvements of ORs capacities after switching the bulk circuits. 50% of increased capacities at the entry OR_2, middle OR_2 and exit OR_2 are 1.3 Mbps, 1.4 Mbps and 1.2 Mbps. The slight improvement in capacities for all the ORs after switching the bulk circuit indicates the throughput of light circuit traffic is improved, and able to transfer quickly the data at any point in time. The results showed that after switching the circuit gain better capacities compared to before switching the bulk circuit [*7].

When OP_4 circuit traffic load switches to all three OR_2 nodes, the load of traffic for OP_4 circuit "before switching" and "after switching" is measured by each OR capacity and throughputs. The end-to-end throughput for old OP_4 circuit (before switching) is 88 ± 24 KB/s. This result of throughput is lower and indicates a bottleneck occurs along the selected three OR_1 nodes. The average capacity of entry OR_1 before switching the OP_4 circuit (solid blue line) is 0.98 Mbps, for middle OR_1 is 1.01 Mbps and for exit OR_1 is 0.78 Mbps. After switching the OP_4 circuit to all three OR_2 nodes (blue dash line), the average capacities for entry OR_2, middle OR_2 and exit OR_2 are 1.98 Mbps, 1.81 Mbps and 1.87 Mbps respectively. Since all the OR_2 nodes are less congested due to handling less traffics compared to all three OR_1 nodes, the end-to-end throughput for OP_4 circuit after switching (blue dash line) to new circuit path is improved by 120 ± 13 KB/s. Note that the initial capacities of all three OR_2 nodes (i.e., before OP_4 circuit traffic passes through all three OR_2 nodes) are 3.28 Mbps for entry OR_2, 3.21 Mbps for middle OR_2, and 3.07 Mbps for exit OR_2. Our approach also improves all three OR_1 nodes capacities after switching the OP_4 circuit to all OR_2 nodes, by increasing the capacities to 1.98 Mbps for entry OR_1, 2.03 Mbps for middle OR_1 and 1.79 Mbps for exit OR_1. The differences of the OR capacities after switching and before switching gives the results in Fig. 6. This improvement in OR capacities helps to improve the flow rates of the current OP_4 circuit passing through all three OR_2 nodes and increases the overall network capacities.

Table 2 Comparisons of Tor multiplex circuits with the combination results of separate TCP connections for all OPs after circuit switched. The effective drop rate is the ratio of packets dropped to the total number of packets we observed in the experiment. The end-to-end throughputs are measured from the OPs to the web server.

Configuration	Average end-to-end throughput (KB/s)	Effective drop rate
Tor multiplex circuits	150 ± 50	0.04 %
All OP circuits after bulk circuit switched	280 ± 100	0.09 %

In overall experiment measurements, we observed in Fig. 6 that 98% of the measurement shows increases of capacities for all ORs after bulk circuit is switched. Only 2% of our test shows no improvement. At one instance, the exit OR_1 and entry OR_2 have the capacity of 0 Mbps, which indicates the OR capacity before and after switched are the same. Moreover, the experiment results show that if we can appropriately select the circuit to switch based on the higher capacity of ORs, we can be able to improve the performance. Furthermore, by connecting congested TCP connection to higher ORs capacity improves the capacity of the network and flow rates of data.

5.3.3 End-to-end Throughput

Table 2 shows the impact of increasing the network capacity to average end-to-end throughputs, for all the circuits from the client OPs to the web server. **Figure 7** shows the cumulative distribution of end-to-end throughput for all circuits. For both results, we compared the Tor multiplex circuits with the results of circuit switching approach. Note that circuit switching results for all OPs are measured beginning from the initial stage when the light and bulk circuits are multiplexed to single TCP connection, to when the bulk circuit is switched to different TCP connection.

[*6] The flow of traffic is from the client OPs to the web server.
[*7] Before switching the bulk circuit is when all the incoming circuit traffics are multiplexed to single outgoing TCP connection.

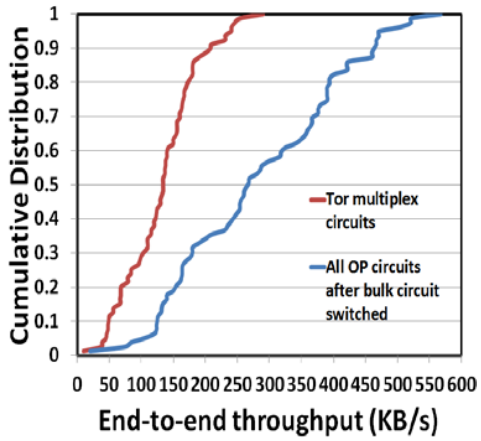

Fig. 7 Cumulative distribution of end-to-end throughput from the client OPs to the web server.

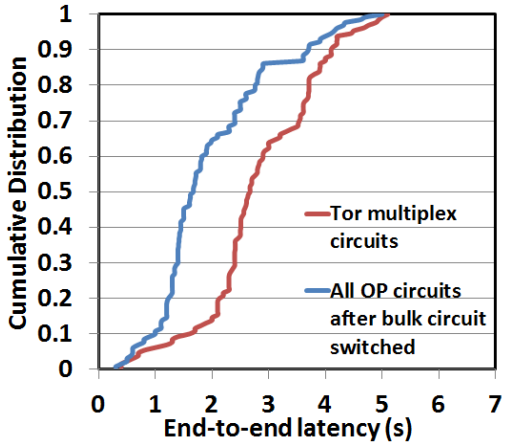

Fig. 8 Cumulative distribution of end-to-end latency from the client OPs to the web server.

In Table 2, we showed the effective drop rate of multiplex circuits compared to circuit switching to different TCP connection. The effective drop rate is the percentage ratio of packets dropped to the total number of packets we observed in the experiment. From the result, we observed the ORs that switched the circuit to different TCP connections for all OPs gain better average end-to-end throughput compared to the Tor multiplexed circuits. However, there are issues that arise in our evaluation method which we need to consider. The average end-to-end throughput for switching the active bulk circuit to different TCP connections showed higher variances (95% confidence interval is quite large). The reason is 0.09% of packet drop occurs during switching the active bulk circuits with higher EWMA values to different TCP connection.

Therefore, the packet losses became a problem for our control scheme because it contributes to additional delay. However, the benefit of our approach is packet loss does not affect throughput in total. The obvious reason is that each TCP sub-flow of the circuit traffic is not affecting each other in-terms of queuing delay or packet drops in the output buffers. As a result, the average end-to-end throughput is still improved. Figure 7 shows that 50% of end-to-end throughput for Tor multiplexed circuits is less than 134 KB/s, whereas 50% of end-to-end throughput for switching the OPs circuits at the entry ORs after congestion occurred is less than 264 KB/s. The distribution of end-to-end throughput for circuit switching approach at the entry ORs shows the improvement in network throughput of the total cells sent along the light and bulk circuits.

In case of the Tor multiplex circuits, Tor suffers an unreasonable throughputs reduction when 0.04% packet drops occur as shown in Table 2. Packet losses or reordering causes the socket to indicate no data is available to read even if other circuits have their sequential cells available in the buffers. On the other hand, packets' sitting in the output kernel buffer and causes the socket to un-writable because of no space.

5.3.4 End-to-end Latency

Figure 8 shows the cumulative distribution of the end-to-end latency for Tor multiplex circuits compared with the circuit switching approach for all the clients when congestion occurs. Tor multiplex circuits referred to all the OPs circuits that are multiplexed in entry ORs, middle ORs and exit ORs. The end-to-end latency is measured for every 60 measurement tests when uploading a file. The distribution of end-to-end latency for all circuits when switching the bulk circuit traffics to different TCP connections shows, 50% of end-to-end latency is less than 1.68 seconds, whereas 50% of end-to-end latency when multiplexing all the circuits to one TCP connection is less than 2.65 seconds. From the result, the circuit switching approach enhances the flow rates of packets and has lower end-to-end latency in comparison to Tor multiplex circuits.

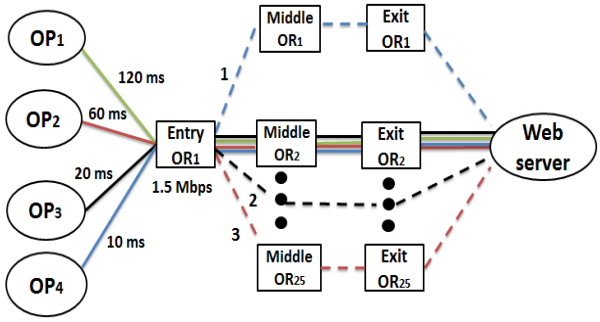

Fig. 9 Experiment topology for circuit switching method in a live Tor Network.

Testing done on our setup ORs shows that when increasing the capacity of network by applying circuit switching method, the end-to-end throughput in the Tor network increases and the overall end-to-end latency decreases. Our next approach is to test our method with the public ORs in the live Tor network, and observe if the control metrics we applied to detect the congestion on the entry ORs is effective.

5.4 Analysis of Circuit Switching in a Live Tor Network

In this subsection, we discuss and show the effectiveness of switching method for improving the end-to-end throughput and the capacity of the Tor network. We tested the switching circuit approach when uploading the same file size of 12.3 MB to the web server.

Figure 9 shows the topology for circuit switching method in a live Tor network. We selected 50 ORs from the Tor directory server, based on the advertised bandwidths and fast, stable performance flags. From the ORs lists, we allocated the middle and

exit ORs. The middle ORs relay traffic within Tor network and the exit ORs have the exit policy to relay traffic outside of Tor.

In this paper, we run our own setup entry OR and connected it to the middle and exit ORs from the live Tor network. The objective is to easily control and observe the circuit switching algorithm that monitors the four OPs circuits that multiplex at a single entry OR, up to time when the congestion occurs. In addition, to analyze any improvements in throughput after the bulk circuit is switched. We ensure that the entry OR located in our testbed must have the guard flag at the entry point to confirm the high uptime, stable performance and have higher than the median of advertised bandwidths of all other ORs [*8].

We analyzed the performance when distributing the traffics across 25 middle ORs and 25 exit ORs when the entry OR_1 switches the circuit to different TCP connections. The circuit switching protocols applied in the live Tor are same with the steps explained in Sections 4.3.1 and 4.3.2. As shown in Fig. 9, the entry OR_1 switches the bulk circuit one-by-one from the current list of running ORs in the Tor network to distribute the traffics.

We measured the default 3-hop circuits to understand the network capacity, end-to-end throughput and latency delays faced by cells into various types of constituent delays that lead to higher congestion and packet drops.

Since we only use our web server as the destination and the middle and exit ORs are located in the public Tor network, we can measure both end-to-end latencies and throughput from OPs to the web server. However, we do not have access to either node delay information or public Tor ORs capacity (i.e., the actual capacity used on public ORs when transferring data) to reach our web server. Therefore, to measure the end-to-end throughput and latency in the live Tor network, we make use of the feedback (HTTP response time) from the web server beginning from when we push the bandwidth probe packets through the TorCtl port from the client OPs. In addition, we confirmed the OR capacity by directly measuring the setup entry OR on our testbed. Furthermore, we applied the Tor bandwidth monitor to estimate the overall network capacity and throughput rates from OPs to the web server during upload. We repeated the measurements for 60 times after 10 minutes intervals during uploading files.

5.4.1 OR Capacities After and Before Switching the Circuit

Since our evaluation is focused on reducing the congestion by increasing the network capacity when applying the circuit switching method, we measured the entry OR_1 capacity by calculating the capacities of after switching the circuit and before switching the circuit $C_{after} - C_{before}$. Note we can only monitor our entry OR capacity that change over time, because we have full control over it.

Figure 10 shows the distribution of entry OR_1 capacities which increase after switching the circuit to another TCP connection. We observed that 50% of capacity measured at the entry OR_1 is less than 1.7 Mbps. This increase of OR_1 capacity shows the improvement of flow rates after entry OR_1 switches the bulk circuits to other higher ORs capacity in the public Tor network. However, in some cases we realized that some public Tor routers are highly

[*8] Note that we selected and worked with the public ORs in the live Tor network, but we only monitored our own circuit traffics.

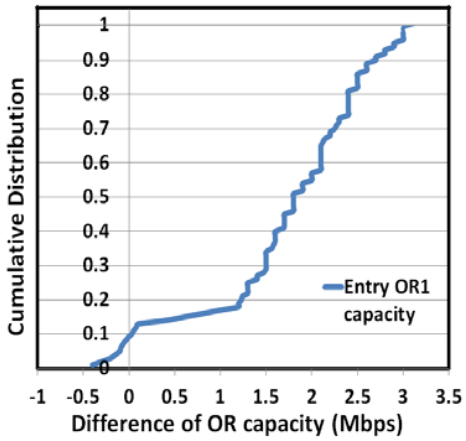

Fig. 10 Cumulative distribution of capacities in entry OR_1 for after and before switching the circuit $C_{after} - C_{before}$. Before switching is when all the OPs circuits are multiplexed to one TCP connection.

Table 3 Average end-to-end throughputs measured from OPs to the web server. We compared the Tor multiplex circuit and the circuit switching approach for all OPs.

Configuration	Average end-to-end throughput (KB/s)	Effective drop rate
Tor multiplex circuits	250 ± 78	0.03 %
All OP circuits after bulk circuit switched	380 ± 110	0.08 %

congested with other cross-circuit traffics in the live Tor network. Therefore when we switched the bulk circuit to another TCP connection after congestion occurs at the entry OR_1, we observed no improvement in the OR capacities. 10% of the measurements show the capacity changes at the entry OR_1 is below 0 Mbps. In addition, most of the ORs selected in the live Tor have the capacity of 500 KB/s to 10 Mbps. Therefore the bottleneck that was experienced along the middle and exit ORs still affects the variances of network capacity.

In overall measurements, we observed that 90% of increase capacity in the entry OR_1 is less than 3 Mbps. This small improvement in capacity increase shows the throughput entry OR_1 is able to transfer to a circuit is increased as well, after connected to higher ORs capacities in the live Tor network.

5.4.2 End-to-end Throughput

Table 3 shows the average end-to-end throughput measured for all four OP circuits to the web server. **Figure 11** shows the distribution of end-to-end throughput.

We observed in Table 3 that the average end-to-end throughput degrades for Tor multiplex circuits when packet drop occurs at the rate of 0.03%. The results in the live Tor network confirm the results in Table 2, which we tested on our setup testbed environment.

From our observations in the testbed environment (Table 2) and the live Tor network (Table 3), the main reason which causes the increase of effective drop rates of the proposed method is that when the entry ORs send the SWITCH_CONN control cell to other ORs for circuit switching and no SWITCH_ACK is sent from either middle ORs or exit ORs. The entry ORs tears down the attempted switch circuit and does not quickly attach the re-

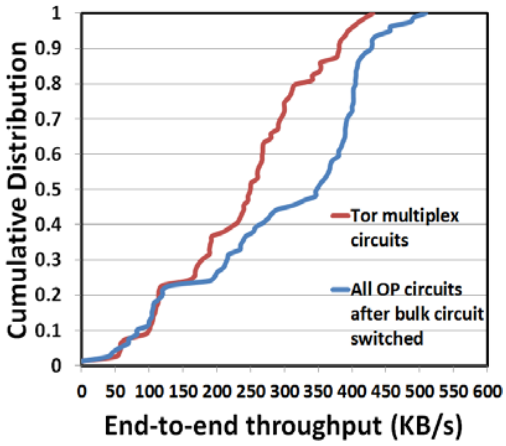

Fig. 11 Cumulative distribution of end-to-end throughputs measured from OPs to the web server. We compared the Tor multiplex circuit and the circuit switching approach for all OPs.

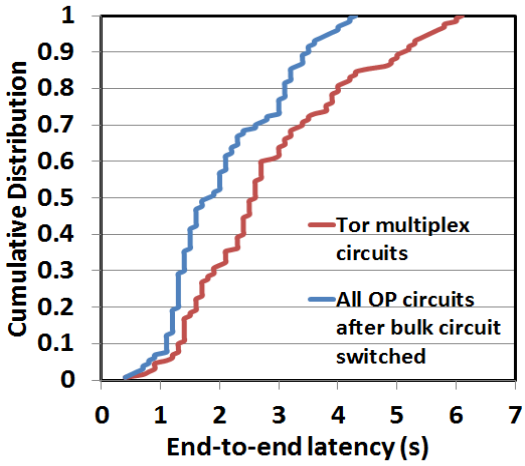

Fig. 12 Cumulative distribution of the end-to-end latency from the client OPs to the web server.

quest's stream to different TCP connection. In that case, Tor considers the reason to send the CLOSE relay cell and drops off all the remaining cells. The client OPs have to restart a new circuit and retransmit the whole data again at the beginning. This problem occasionally happens when switching the bulk circuit to ORs which are highly congested with many incoming circuits and has very lower capacity.

However, though Tor multiplexes circuits to one TCP connections gain lower overall drop rates compared to switching method, the behavior of one circuit drops can adversely affect all the other incoming circuits' and reduce the average end-to-end throughput. Considering the average end-to-end throughput for all the circuits after the bulk circuits switched to different TCP connection in the live Tor network. The average end-to-end throughput for all the circuits is higher compared to when Tor multiplexed circuits to one TCP connection. In overall distribution measurements, we observed in Fig. 11 that 50% of end-to-end throughput for circuit switching is less than 350 KB/s, whereas 50% of end-to-end throughput for Tor multiplex circuits is less than 250 KB/s.

We concluded that even if the proposed control scheme is only installed in the entry ORs, the communication quality is still improved according to our measurement results. This improvement means better transfer rates and lower congestion on all the selected ORs. The problem to our circuit switching method is the effect of dropping cells along the circuit as shown in Table 3. Switching of active TCP connection to different connection took an average delay of 1 second to be successfully completed and continues on with the data transmission. We observed that the average delay of 1 second is caused by the cells which are dropped.

5.4.3 End-to-end Latency

Figure 12 shows the end-to-end latency results for Tor multiplex circuits (before circuit switch) compared to all OP circuits after switching the bulk circuit traffic to different TCP connection. The result shows significant improvements when applying the switching method in the live Tor network. 50% of end-to-end latency measured for all the OPs is less than 1.7 seconds; whereas 50% of Tor multiplexed circuits is less than 2.6 seconds. The delay time of 2.6 seconds is too high which can cause the client OPs to detect congestion very late. In addition, Tor users can felt the slowness of transferring data to the web server.

From the result, as compared with the conventional approach where Tor multiplexes circuits to one TCP connection, the end-to-end latency of almost all OPs circuit can be reduced by switching the bulk circuit to a different TCP connection. This result shows that by separating the connection-sharing among bulk and light circuits, we can reduce the end-to-end latency in Tor and improve the flow rates of cells.

6. Conclusion and Future Work

In this work, we improved the end-to-end throughput of data transfer through the Tor network by applying our circuit switching method. We discovered and showed that the use of different TCP connections on some hops does increase the network capacity and reduce the overall end-to-end latency. We used the two metrics to assess the network congestion, the Tor output buffer occupancy and the TCP socket un-writable events. We observed that packet losses are a problem for our control scheme because it contributes to additional delay when switching the active TCP connections.

For our future work, the main focus is to improve the problems we faced in our current work and the limitation concerning the dynamic selection of circuit when switching the active bulk traffics. In addition, we wish to improve the congestion detection method in the current Tor algorithm.

References

[1] Dingledine, R., Mathewson, N. and Syverson, P.: Tor: The second-generation onion router, *Proc. 13th USENIX Security Symposium*, p.21, USENIX Association (2004).
[2] Mccoy, D., Bauer, K., Grunwald, D., Kohno, T. and Sicker, D.: Shining Light in Dark Places: Understanding the Tor Network, *Proc. 8th Privacy Enhancing Technologies Symposium*, pp.63–76 (2008).
[3] Jansen, R., Hopper, N. and Kim, Y.: Recruiting New Tor Relays with BRAIDS, *Proc. 17th ACM Conference on Computer and Communications Security* (*CCS*), pp.319–328, ACM (2010).
[4] J Ngan, T.-W., Dingledine, R. and Wallach, D.S.: Building Incentives into Tor, *Proc. Financial Cryptography*, pp.238–256 (Jan. 2010).
[5] Dingledine, R. and Murdoch, S.: Performance improvements on Tor or, why Tor is slow and what we're going to do about it (Mar. 2009), available from ⟨http://www.torproject.org/press/presskit/2009-03-11-performance.pdf⟩.
[6] Wang, T., Bauer, K., Forero, C. and Goldberg, I.: Congestion aware Path Selection for Tor, *Financial Cryptography and Data Security* (*FC*) (2012).

[7] Alsabah, M., Bauer, K., Goldberg, I., Grunwald, D., Mccoy, D., Savage, S. and Voelker, G.M.: DefenestraTor: Throwing out Windows in Tor, *11th Privacy Enhancing Technologies Symposium*, pp.134–154 (July 2011).

[8] Reardon, J. and Goldberg, I.: Improving Tor Using a TCP-over-DTLS Tunnel, *Proc. 18th USENIX Security Symposium* (Aug. 2009).

[9] Tang, C. and Goldberg, I.: An Improved Algorithm for Tor Circuit Scheduling, *Proc. 17th ACM Conference on Computer and Communications Security* (*CCS*), pp.329–339 (Oct. 2010).

[10] Moore, W.B., Wacek, C. and Sherr, M.: Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise, *Proc. 27th Annual Computer Security Applications Conference* (*ACSAC*), pp.207–216 (Dec. 2011).

[11] Dingledine, R.: Research problem: Adaptive throttling of Tor clients by entry guards, available from ⟨https://blog.torproject.org/blog/research-problem-adaptive-throttling-tor-clients-entry-guards⟩.

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

[13] Gopal, D. and Heninger, N.: Torchestra: Reducing Interactive Traffic Delays over Tor, *Proc. 2012 ACM Workshop on Privacy in the Electronic Society, WPES 2012*, pp.31–42, ACM (2012).

[14] Goldschlag, D., Reed, M.G. and Syverson, P.F.: Hiding routing information, *Proc. Information Hiding: First International Workshop*, pp.137–150, Springer-Verlag (May 1996).

[15] Reardon, J.: Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo, Waterloo, ON (Sep. 2008).

[16] Dhungel, P., Steiner, M., Rimac, I., Hilt, V. and Ross, K.W.: Waiting for anonymity: Understanding delays in the tor overlay, *P2P*, pp.1–4, IEEE (2010).

[17] Snader, R. and Borisov, N.: A tune-up for Tor: Improving security and performance in the Tor network, *Proc. Network and Distributed Security Symposium – NDSS '08*, Internet Society (Feb. 2008), available from ⟨http://www.freehaven.net/anonbib/cache/snader08.pdf⟩.

[18] Kale, T.G., Ohzahata, S., Wu, C. and Kato, T.: Analyzing the Drawbacks of Node-Based Delays in Tor, *Proc. IEEE CQR 2014*, pp.1–6 (2014).

[19] Dingledine, R. and Mathewson, N.: Tor Protocol Specification, available from ⟨https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt⟩ (accessed 2014-04).

[20] Torprojects, available from ⟨https://www.thesprawl.org/projects/tor-autocircuit/⟩ (accessed 2014-04).

[21] Dingledine, R. and Mathewson, N.: Tor Path Specification, available from ⟨http://tor.hermetix.org/svn/trunk/doc/spec/path-spec.txt⟩ (accessed 2014-04).

[22] Reardon, J.: libspe (2009), available from ⟨http://crysp.uwaterloo.ca/software/⟩ (accessed 2014-04).

[23] Edman, M. and Syverson, P.: As-awareness in Tor path selection, *Proc. 16th ACM Conference on Computer and Communications Security, CCS'09*, pp.380–389, ACM, New York, NY, USA (2009).

[24] Chen, F. and Pasquale, J.: Toward Improving Path Selection in Tor, *Global Telecommunications Conference*, pp.1–6, IEEE (2010).

[25] Salowey, J., Zhou, H., Eronen, P. and Tschofenig, H.: Stateless TLS Session Resumption, Standards Track (Jan. 2008), available from ⟨https://tools.ietf.org/html/rfc5077⟩.

Timothy Girry Kale received his M.E. degree from the Graduate School of Information Systems, the University of Electro-Communications, Tokyo, Japan, in 2012. He is currently a Ph.D. candidate at the same university. His current research interests include overlay network, networking architectures and protocols.

Satoshi Ohzahata received his M.E., and D.E. degrees from University of Tsukuba in 1998, 2000 and 2003, respectively. He was a research associate, Department of Computer, Information & Communication Sciences at Tokyo University Agriculture and Technology from 2003 to 2007, and was an assistant professor of the same university from 2007 to 2009. Since 2009, he has been an associate professor at Graduate School of Information Systems, the University of Electro Communication. His interests are mobile ad hoc networks, the Internet architecture in mobile environments and Internet traffic measurement. He is a member of IEEE, ACM and IEICE.

Celimuge Wu received his M.E. degree from Beijing Institute of Technology, Beijing, China, in 2006, and Ph.D. degree from the University of Electro-Communications, Tokyo, Japan, in 2010. He is currently an assistant professor of the Graduate School of Information Systems, the University of Electro-Communications. His current research interests include mobile ad hoc networks, networking architectures and protocols.

Toshihiko Kato received his B.E., M.E., and Dr.Eng. degrees in electrical engineering from the University of Tokyo, in 1978, 1980 and 1983, respectively. He joined KDD in 1983 and worked in the field of communication protocols of OSI and Internet until 2002. From 1987 to 1988, he was a visiting scientist at Carnegie Mellon University. He is now a professor of the Graduate School of Information Systems in the University of Electro-Communications in Tokyo, Japan. His current research interests include protocol for mobile Internet, high speed Internet and ad hoc network.