The Onion Routing Performance using Shadow-plugin-TOR

Hartanto Kusuma Wardana, Liauw Frediczen Handianto, Banu Wirawan Yohanes*
Faculty of Electronic and Computer Engineering
Universitas Kristen Satya Wacana
Salatiga, Indonesia
Corresponding author's email: *bonayo@gmail.com

Abstract—**Anonymous network provides user privacy to protect identity. The onion routing (TOR) project is one kind of Internet anonymous networks which attracts many researchers and clients nowadays, because of its simplicity and scalability. However, there are some difficulties to analyze TOR performance within live TOR networks since it is distributed and its security nature. This paper presents a TOR network simulation using shadow-plugin-TOR to configure a small scale virtual computer network within a host, running TOR, and then to analyze TOR performance in several measurements. One main advantage is that the shadow-plugin-TOR application can be implemented within a host in which used to build a client-server architecture of TOR network. Various number of relays and clients have been set to examine TOR network performance. The small scale simulations conducted using maximal CPU capacity between 60%–75% and memory (RAM) usage between 320 MiB–1.5 GiB for each configuration. The simulation results show that average transmission time is shortened and bandwidth is faster along with increasing number of relay rather than adding more client.**

Keywords—anonymous network, TOR, shadow-plugin-TOR

I. Introduction

In recent years there have been increasing number of electronic transactions all over Internet, mainly using web e-commerce. The phenomena supported with the ubiquitous of mobile digital devices, e.g., smartphone and laptop. Two main advantages of electronic transactions are simplified process and shorten time response, almost no queueing, compared to conventional transactions. In contrast, there are several risks of using electronic transactions such as virus, malware, lost data packet, and eavesdropping by unauthorized third party.

Internet, particularly TCP/IP, was not designed with anonymity at first. A solution for providing anonymity is to create overlay network which runs over TCP/IP network. Then the overlay network provides message routing control, hence concealing hosts' IP addresses. This control brings through IP address obfuscation, and thus enables anonymity.

Anonymous network is one kind of data privacy protection schemes where both parties of a transaction are unable to resolve each other. In other words, the sender is undetected or unknown. Two main criteria of an anonymous network are user access cannot be traced and user activities are hidden during network activity. However, data sent still can be traced back to its sender.

One of widely used anonymous network applications is the onion routing (TOR) browser developed by TOR project [1], [2]. TOR is a distributed system using low latency network, adding extra encryption layer per network hop, and creating random network paths for each transaction. The client and server paths cannot be traced without traffic analysis. There is not exist a node on the communication path which can resolve messages sent by client to those received by server. Still many researchers find it is complicated to examine how TOR works, because of its security features enabled.

Several researches has been conducted to modeling TOR network. Since a live TOR experimentation is difficult, because it is not a predictable and controllable environment. There are variety of network conditions which may result in bias, hence it is troubled to repeat the experiments. In addition, collecting client data is inconvenient because it can expose privacy risks. Alternative approaches then emerged, such as utilization of virtual networks, e.g., using emulation [3] and simulation [4].

This paper proposes a complete model of TOR simulation and another novel TOR network measurement. This article is structured as follow, section I provides background literature in anonymous networks, specifically TOR. Section II explains how TOR works in details. Section III describes experiment setup and running a TOR simulation in virtual network using shadow-plugin-TOR to evaluate TOR process and performance. Section IV presents experimental results with varying number of relays and clients within TOR network simulations. Section V gives discussion and analysis of experimental results. Last, section VI concludes the results.

II. The Onion Routing

A. Onion Routing

Onion routing is a kind of anonymous network with several encryption layers stacked. The layers are peeled one by one to get the original data. Each layer consists information about only one single next destination address in networks hop. Generally, onion routing has three stages [1], i.e. connection setup, data movement, and connection tear-down.

TOR is responsible for creating communication paths between sender and receiver. The first stage of onion routing is connection setup in which information distributed to each relay within server coverage. Each relay acquired decryption keys for each onion routing layer. The second stage is data movement where data sent from both client and server using algorithms and keys defined earlier. The third stage is connection tear-down to close down the onion routing network among relays or between both relay endpoints from data transmission when needed.

As an application of TOR, generally TOR browser is used to resolve three main issues of privacy protection within a computer network, i.e. to prevent user location tracking from web sites, servers, or other services; to avoid data transmission tracking or eavesdropping by unauthorized third party or internet service provider (ISP); and to stave off each relay from extracting information about sender and receiver, except for its hop network only. Privacy protection scheme using TOR is illustrated in Fig. 1.

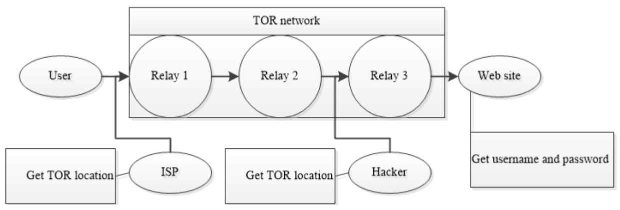


Fig. 1. Privacy protection in TOR networks.

Each relay in TOR networks has two keys, a long-term key, named identity key, and a short-term key, called onion key. Identity key is used to sign digital certificate made by an authority, relay descripTOR documents, and direcTORy. Meanwhile, onion key is used to decrypt network path finding within client request [1], [4].

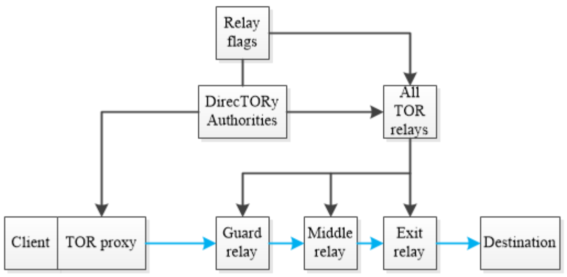


Fig. 2. The guard, middle, and exit relays (nodes) within TOR networks.

TOR user installs an onion proxy (OP) application to handle connection setup and routing through TOR network. OP retrieves a consensus from a directory server. Consensus consists of a list of available TOR nodes, also known as relays or onion routers, which created hourly. OP then randomly selects three node, called a circuit as shown in Fig. 2, i.e. entry or guard node, middle node and exit node.

Similarly in TOR simulations, DirecTORy authorities send consensus documents to clients. The documents created and agreed by all direcTORy authorities. There are nine direcTORy authorities in TOR project, listed in TOR project web site [2]. Each TOR relay connecting to direcTORy authority to receive flags that used to create consensus documents. Then direcTORy authority sends the documents to each client. Clients build a network routing consists of three relays as displayed in Fig. 2, i.e., guard relay, middle relay, and exit relay. The relays selected according to flags sent by direcTORy authorities.

B. Shadow

Shadow is a discrete-event simulator built upon the distributed virtual network (DVN) simulator used to simulate TOR Project using shadow-plugin-TOR [4]. It can run on a host with average hardware requirements. TOR encapsulated in a plugin composed of application code and other functions to interact with TOR network. Every TOR condition loaded once in memory, and plugin registers all memory addresses for all TOR variables. Then Shadow manages each copy of memory area for each node in the simulations. Shadow loads plugin dynamically and run virtual node specified in simulation scripts. The communication between Shadow and plugin takes place through callback interface implemented by plugin. When it is executed, plugin running a non-blocking application. Hence events transferred within schedule using system calls intercepted by Shadow and directed to functions in node library. It integrates TOR to a simulation environment without TOR source code modifications. Every simulation run composed of processing stage which allows user access to Shadow commands to create plugins, to build and connect to networks, and to make nodes. Every event in simulation scripts created, and it starts until finished according to the schedule defined or timeout.

III. Experiment Setup

One main criterion in this simulation is to provide insight about various combinations in size/number of relays and clients within a TOR network. Modelling a TOR network with different number of relays and clients is important, because small networks may not perform equally as large networks. Moreover, a diverse number of relays and clients can simulate privacy adjustment within a TOR network.

After setting up TOR virtual networks within Shadow using shadow-plugin-TOR, twelve simulation configurations run with variety number of relays and clients. They are classified into three categories, i.e. first group for constant number of relay or client with configurations include 10r40c (10 relays and 40 clients), 10r80c, 10r120c, 40r10c, 80r10c, and 120r10c, second group for increasing number of relay or client gradually, i.e. 10r40c, 20r80c, 30r120c, 40r10c, 80r20c and 120r30c, and third group with minimal number of relay or client configured with maximal number of relay or client, i.e.

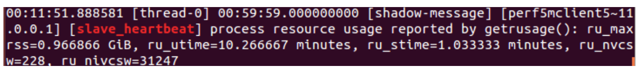
10r40c, 40r10c, 40r120c, 120r40c. Each configuration run using "`shadow-tor -w -y 3`" command.

A log created, named `shadow.log`, for each simulation run. It prints messages from all simulation processes. This log is used to check simulation result and TOR virtual networks performance. There are three main message functions in a log file, i.e. slave-heartbeat, torctl-message, and transfer-complete [4].

Slave-heartbeat function measures computer main memory (RAM) usage while running TOR virtual networks simulation. `getrusage()` function is used to get RAM usage in every second simulation. An example of a slave-heartbeat message is displayed in Fig. 3.

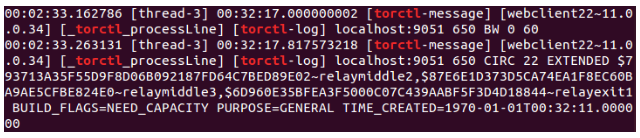
Torctl-message in TOR library used to monitor and to connect TOR relays. It composed of bandwidth and connection scheme at network routing from each relay and client within TOR networks simulation. An example of a torctl-message is displayed in Fig. 4.

Transfer-complete message is a traffic generaTOR (tgen) function used to monitor each connection built and measure data collection times from successful connections. Successful clients connected to a server have several data types. They are total-bytes-read as amount data downloaded, total-bytes-write as amount data uploaded, payload-bytes-read as total amount data collected from download and upload combination, msecs-to-command as how long connection command is starting in millisecond, msecs-to-response as how long is connection response in millisecond, msecs-to-first-byte as how long is to get first byte data in millisecond, msecs-to-last-byte as how long is to get last byte data in millisecond, msecs-to-checksum as how long is to validate data acquired in millisecond. An example of a transfer-complete message is shown in Fig. 5.



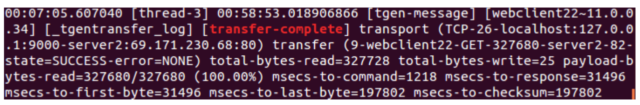
```
00:11:51.888581 [thread-0] 00:59:59.000000000 [shadow-message] [perf5mclient5~11
.0.0.1] [slave_heartbeat] process resource usage reported by getrusage(): ru_max
rss=0.966866 GiB, ru_utime=10.266667 minutes, ru_stime=1.033333 minutes, ru_nvcs
w=228, ru_nivcsw=31247
```

Fig. 3. A slave-heartbeat message in `shadow.log`.



```
00:02:33.162786 [thread-3] 00:32:17.000000002 [torctl-message] [webclient22~11.0
.0.34] [_torctl_processLine] [torctl-log] localhost:9051 650 BW 0 60
00:02:33.263131 [thread-3] 00:32:17.817573218 [torctl-message] [webclient22~11.0
.0.34] [_torctl_processLine] [torctl-log] localhost:9051 650 CIRC 22 EXTENDED $7
93713A35F55D9F8D06B092187FD64C7BED89E02~relaymiddle2,$87E6E1D373D5CA74EA1F8EC60B
A9AE5CFBE824E0~relaymiddle3,$6D960E35BFEA3F5000C07C439AABF5F3D4D18844~relayexit1
 BUILD_FLAGS=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=1970-01-01T00:32:11.0000
00
```

Fig. 4. A torctl-message in `shadow.log`.



```
00:07:05.607040 [thread-3] 00:58:53.018906866 [tgen-message] [webclient22~11.0.0
.34] [_tgentransfer_log] [transfer-complete] transport (TCP-26-localhost:127.0.0
.1:9000-server2:69.171.230.68:80) transfer (9-webclient22-GET-327680-server2-82-
state=SUCCESS-error=NONE) total-bytes-read=327728 total-bytes-write=25 payload-b
ytes-read=327680/327680 (100.00%) msecs-to-command=1218 msecs-to-response=31496
msecs-to-first-byte=31496 msecs-to-last-byte=197802 msecs-to-checksum=197802
```

Fig. 5. A transfer-complete message in `shadow.log`.

IV. Experiment Result

All simulation results recorded in a `shadow.log` file are displayed in Table I, II, and III. Table I presents average and maximal CPU speed, and memory (RAM) usage from both `dstat` and `getrusage()` commands used in the simulations. Table II presents average transmission time for web-client, bulk-client, perf50kb, perf1mb, perf5mb. At last, Table III lists average bandwidth for relay, web-client, and bulk-client.

Number of relays and client combinations chosen in these experiments are adjusted with computer resources available in the computer networks laboratory within faculty, i.e., Intel Dual Core CPU @ 2.4 GHz and 4 GB memory (RAM). From 4 GB available in host, 2 GB used to set up a Linux Ubuntu guest where TOR network simulations can be run.

TABLE I. CPU Speed And RAM Usage

Number of relay	**Number of client**	**Average CPU speed (%)**	**Max. CPU speed (%)**	**dstat (GiB)**	**getrusage (GiB)**
10	40	44.02	59.6	0.286	0.321
10	80	43.53	74.5	0.56	0.53
10	120	45.76	74.25	0.786	0.734
20	80	39.71	74.31	0.762	0.708
30	120	39.47	74.44	0.881	0.9
40	10	34.9	60.55	0.333	0.326
40	120	40.7	73.87	0.975	0.909
80	10	42.46	66.33	0.669	0.695
80	20	38.5	74.25	0.75	0.728
120	10	44.08	73.94	1.091	1.054
120	30	40.92	74	1.469	1.493
120	40	46.67	74.5	1.262	1.326

The smallest configuration of 10 relays and 40 clients using 60% from maximal CPU speed and approximately 350 MiB RAM. Meanwhile, for largest configuration with 120 relays and 30 clients using 74% from maximal CPU speed and approximately 1.5 GiB RAM. For configurations with fixed 10 relays and increasing number of clients, i.e. 40, 80, and 120 the transmission time of web-clients are also increasing 181 s, 322 s, and 438 s respectively. Meanwhile, for configurations with fixed 10 clients and increasing number of relays, i.e. 40, 80, and 120 the transmission time of web-clients are constant approximately 38 s. Bandwidth measurements for fixed 10 relays and increasing number of clients are as follow, for 40 clients bandwidth is 1.8 KB/s, for 80 clients bandwidth is 1 KB/s, and for 120 clients bandwidth is 0.75 KB/s. Meanwhile, bandwidth measurements for fixed 10 clients and increasing number of relays are as follow, for 40 relays bandwidth is 8.5 KB/s, for 80 relays bandwidth is 8.4 KB/s, and for 120 relays bandwidth is 8.9 KB/s.

Total simulation time and maximal download time for three groups are presented in Fig. 6 and Fig. 7 respectively. For the first group, more number of relays decreases download time, but increases simulation time. Meanwhile for the second and third groups, the largest number of relays and clients produces smallest download time, but largest simulation time. Besides, moving average read time for three groups are illustrated in Fig. 8.

TABLE II. AVERAGE TRANSMISSION TIME (S)

Number of relay	Number of client	Web-client (327680 byte)	Bulk-client (5242880 byte)	Perf-50kb (51200 byte)	Perf-1mb (1048576 byte)	Perf-5mb (5242880 byte)
10	40	181.3	3045.9	116.3	487.4	1491
10	80	322.2	4268.2	182.4	674.5	2632.1
10	120	438.6	8433.3	294.6	1546.8	3425.6
20	80	51.2	217.9	65.2	123.1	346.9
30	120	50	253.2	71.6	163.1	335.3
40	10	38.2	83.9	55.3	94.7	187.8
40	120	46.5	98.9	61.7	114.2	248.8
80	10	38.8	85.4	54.5	87.1	159.2
80	20	38.6	165.4	56.9	91	212.4
120	10	36.5	53.1	53.7	85.5	153.3
120	30	41.3	106	55.9	89.2	241.1
120	40	39.5	67.1	55.9	87.6	210.5

TABLE III. AVERAGE BANDWIDTH (KB/S)

Number of relay	Number of client	Relay	Web-client	Bulk-client
10	40	720.3	1.8	1.7
10	80	720.3	1	1.2
10	120	720.3	0.8	0.6
20	80	1021.9	6.4	24.1
30	120	1111.1	6.6	20.7
40	10	1233.8	8.6	62.5
40	120	1233.8	7.1	53
80	10	1460.6	8.4	61.4
80	20	1460.6	8.5	31.7
120	10	1116.8	9	98.8
120	30	1116.8	7.9	49.4
120	40	1116.8	8.3	78.1

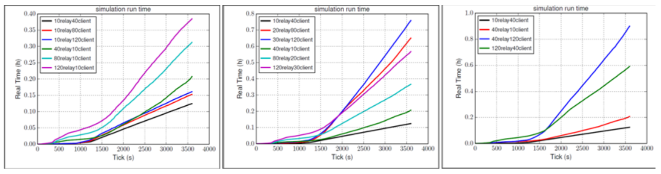


Fig. 6. Total simulation time for three groups.

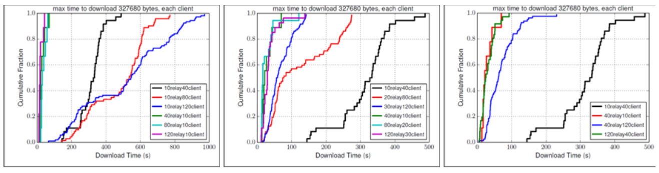


Fig. 7. Maximal download time for three groups.

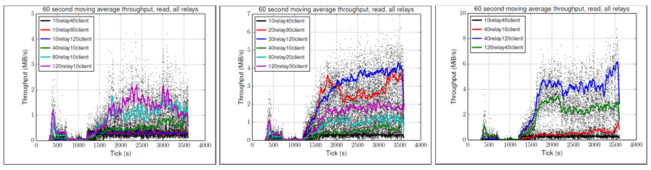


Fig. 8. Moving average read time for three groups.

V. DISCUSSION AND ANALYSIS

From simulation results, variety number of relays and clients slightly affects maximum CPU speed. From Table I, the lowest max. CPU speed is 59.6%, while highest max. CPU speed is 74.5%. Hence, there is only 20% increasing max. CPU speed from all configurations tested. From Table I, same number of relays and increasing number of clients will add more memory usage, from 0.17 GiB to 0.5 GiB. Hence, number of clients more affecting memory usage rather than number of relays.

From Table II, it can be seen that adding more relays may increase privacy or security level of data transmitted within TOR. It is caused by of more options of communication paths available for TOR. Moreover, Table II also presents faster packet transmission time with increasing number of relays.

From Table III, it can be seen that more number of relays will increase average bandwidth. However, maximum average bandwidth is achieved when number of relays reaches 80. Then average bandwidth is slower again when number of relays reaches 120 equal to 30 relays used. Increasing number of client will also add average bandwidth, except an anomaly when number of relays reaches 120 and number of clients reaches 40.

VI. CONCLUSIONS

The scalability of Shadow reduces TOR network complexity by running on a single moderate host with several assumptions. Small scale simulations with variety number of relays and clients have been conducted to examine TOR networks performance. This paper contributes in standardizing TOR measurements with two metrics, i.e., number of relays (nodes) and clients (users).

More works required to compare this result with other TOR experimentation testbeds, e.g., ExperimenTor, SNEAC, TorPS, or COGS. In addition, several relevant metrics i.e., routing approach, topology, congestion, and adversaries can be used in performance comparison from those simulations.

ACKNOWLEDGMENT

The authors thank to Computer Systems Department for providing access to the computer network laboratory where all simulations take place, and to Universitas Kristen Satya Wacana, Salatiga for supporting this article publication.

REFERENCES

[1] R. Dingledine, N. Mathewson, P. Syverson, "TOR: The second-generation onion router," The Free Haven Project and Naval Research Lab, 2011.

[2] ________, "TOR (The Onion Routing)," 2016. Accessed online at https://www.TORproject.org

[3] K. Bauer, M. Sherr, D. McCoy, D. Grunwald, "ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation," Proc. of the 4th Workshop on Cyber Security Experimentation and Test, 2011.

[4] R. Jansen and N. Hopper, "Shadow: Running TOR in a box for accurate and efficient experimentation," U.S. Naval Research LaboraTORy and University of Minnesota, 2012.

[5] S.J. Murdoch and R.N.M. Watson, "Metrics for Security and Performance in Low-Latency Anonymity Systems," Proc. of the 8th Privacy Enhancing Technology Symposium, 2008.

[6] ________, "Shadow real application, simulated network," 2016. Accessed online at https://shadow.github.io