NETWORK ANONYMITY AND PRIVACY

TOR Circuit Creation Steps and Algorithm

Assignment - 2

Mayank Mohindra
2015056
mayank15056@iiitd.ac.in

Assignment Problem Statement:

TOR (The Onion Router) has many relays running in the network (we have used a list of 6,367 relays). These relays have various configurations and not all relays can be used everywhere. There are some which have been identified by the TOR network as “invalid” or “bad”, but there is still need of these relays by smartly placing them in the circuit that we plan to create. The problem statement involves the use of these relays to create legitimate TOR circuits based mostly on default path-configuration and also making some suitable assumption, so that we know how these relays are selected and what effect does compromising relays can have on the circuit creation.

Approach used:

To understand the process of circuit creation, we need to answer the two important question: when do we need to build the circuit and how are the circuits built.

To answer the first question in brief, which is when to build the circuits, we keep the time the requests arrives from the client as the origin (t=0) and the time previous to it as t=-1 and later to it as t=+1.

At t=-1: the circuit creation is called preemptive

At t=0: the circuit used is the one already built so cannibalizing.

At t=+1:the circuit creation is on demand, which adds little latency but is better than building circuits for each request every time.

Even before this TOR must be ready to build circuits and needs to gather some much needed information:

1. **Gathering enough information about the current network**

TOR network is very wide and secure and to keep it this way, the network must communicate within itself through some means so that if any router state is troubled, then the clients and the relays are well informed about the risks involving the use of that router. TOR majorly achieves this using two documents: server descriptors and network consensus (for the purposes of this assignment).

The network consensus is a broad signed information document that is made available to all the clients and contains information on all the relays existing and operating in the network. To account for the viability of the consensus, this document is voted by various directory authorities and is released once per hour. The first task of the path building stage is to check for

the presence of a viable consensus document that is not more than an hour old and to download a fresh copy in case it has expired.
The server descriptors, on the other hand had a different purpose. Clients initially downloaded the server descriptors for the required relay set, and were signed by directory authorities. However we must realise that a colluded/compromised directory authority can cause much damage to an extent of changing the course of the packet traversal. To address this issue the concept of network status documents or consensus was introduced. Now server descriptors are used to gather information of only the relays that are not mentioned in the consensus or to check for the advertised information that that relay want to share like family, or exit policies etc. Rest of the information is combined in the consensus and voted upon by multiple directory authorities so that the collusion can have no effect on the circuit creation.

2. **Preemptive and on demand circuit building:**

Preemptive involves the concept of building the circuits even before making requests. The circuit building process is time consuming and TOR aims to reduce latency to the maximum with minimal loss to anonymity. Thus TOR keeps a set of circuits developed and connected waiting to serve even when client has not started making requests. TOR scans through the list of used ports in the recent past, and tries predict the requests that the client might be making upon the TOR circuit creation. The circuits are chosen with flags like “fast”, “stable”, “valid” etc. enabled and one can use these circuits at their disposal.
If requests that were not intended to be handled preemptively by the TOR circuits, can be done either by making new circuits for the client, which is called circuit creation on-demand and is done in the decreasing order of pending requests, or by cannibalizing circuits, which means that if an already created circuit which was built to handle e.g PORT 80 is idle, but can also handle requests to PORT 22, then upon receiving a PORT 22 requests will be employed to serve it.

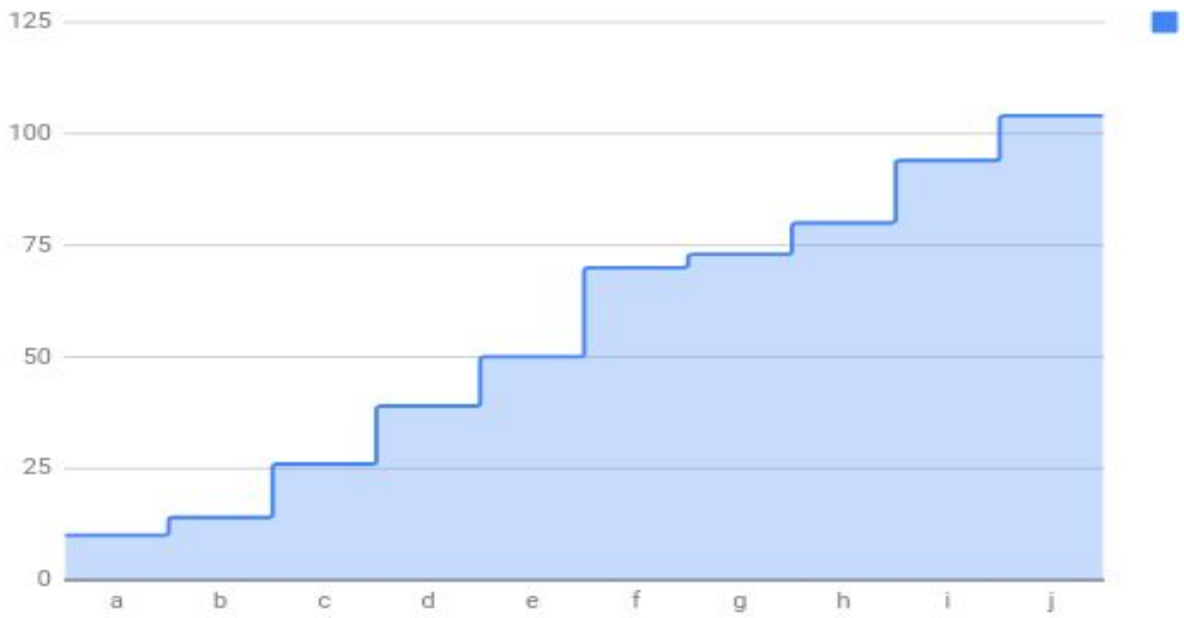
Now we move to the second question which is how to create a circuit? This step is simple and involves some algorithmic understanding. We know that the primary aim of TOR is to reduce latency to an anonymity network. Thus each client at all times would be attracted to relays that are advertising high bandwidths. This incorporates two major issues:

1. Colluding/compromised relays will start advertising high bandwidths and our paths will all be compromised
2. Most of the clients will start choosing the same set of relays for their information exchange thus reducing the available bandwidth and load carrying capacity and slowing down the network

But bandwidth must play some role in circuit creation otherwise its capacity will be wasted. TOR views its relays bandwidths from a larger perspective. It utilises the concepts of the bandwidth of the network. This means that the relays are chosen randomly but with bandwidth playing a major probabilistic role.
Consider the following problem explanation:
Let there be 10 test relays and their respective bandwidths mentioned, calculative cumulative bandwidths and plot them on a number line:

Relay	Bandwidth	Cumulative Bandwidth
a	10	10
b	4	14
c	12	26
d	13	39
e	11	50
f	20	70
g	3	73
h	7	80
i	14	94
j	10	104

As mentioned, TOR views this small subnetwork with consensus bandwidth as mentioned, in terms of the network bandwidth of 104. Now our aim is to randomly select relays but also make sure that their selection is weighted by the respective bandwidth, which means that a relay with a higher bandwidth is that has a higher probability of being selected.



Now we choose a number from 0-104 randomly and since we can see that the probability of it falling on ‘f’ with the highest bandwidth is also maximum since it occupies more region along the y direction from the range 0-104 in range 50-70. This is the weighted bandwidth selection algorithm.
In all further explanation, we use symbol & for set intersection.
We take the following assumption:
The default configuration for any TOR circuit on the client side has been set to select a “valid”, “running”, “stable”, “fast” circuit.

$$config\_set = running\_set \ \& \ stable\_set \ \& \ valid\_set \ \& \ fast\_set$$

Now we must address the choosing of each of the nodes individually. There are by default 3 relays that are selected in the TOR network and each relay has its own order of being chosen:

1. **EXIT-RELAY**

Exit relays chosen are the ones that are not marked as Bad Exit by the consensus, and are in accordance with the mentioned exit policies of the circuit requirements

$$possible\_exit = exit\_set \ \& \ goodExit\_set \ \& \ config\_set$$

Then we apply the bandwidth selection algorithm on this set of possible exits.

2. **GUARD-RELAY**

Guard relays are again the one in accordance with the configuration rules and with guard flag true. These guard relays must also be compliant with the constraints to the chosen exit node.
The constraints checked here are as follows:

a. No two routers chosen must be same.
b. No two routers chosen must be of the same family.
c. No two routers should be in the same /16 subnet.

$$possible\_guard = guard\_set \ \& \ config\_set$$
$$apply\_constraints(possible\_guards)$$

3. **MIDDLE-RELAY**

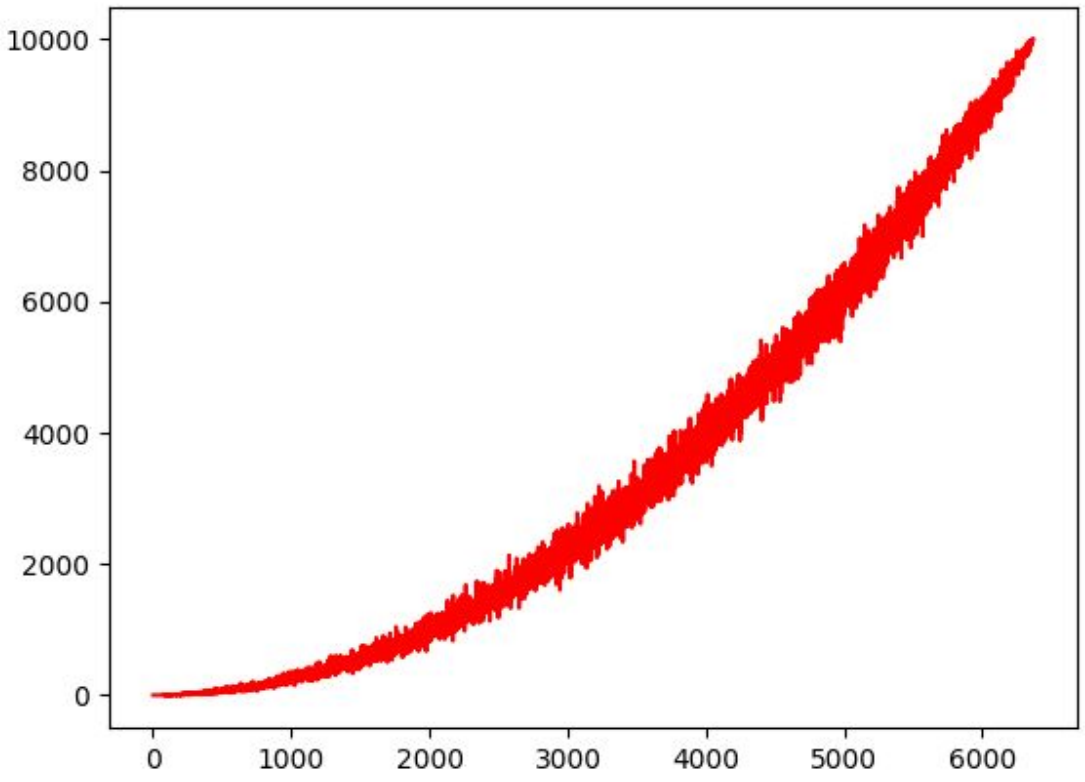
Middle relays need not follow the configurations as such, which means that they might be invalid as well:

$$possible\_middle = all\_relay\_set \ \& \ fast\_set \ \& \ running\_set$$
$$apply\_constraints(possible\_middle)$$

Plots

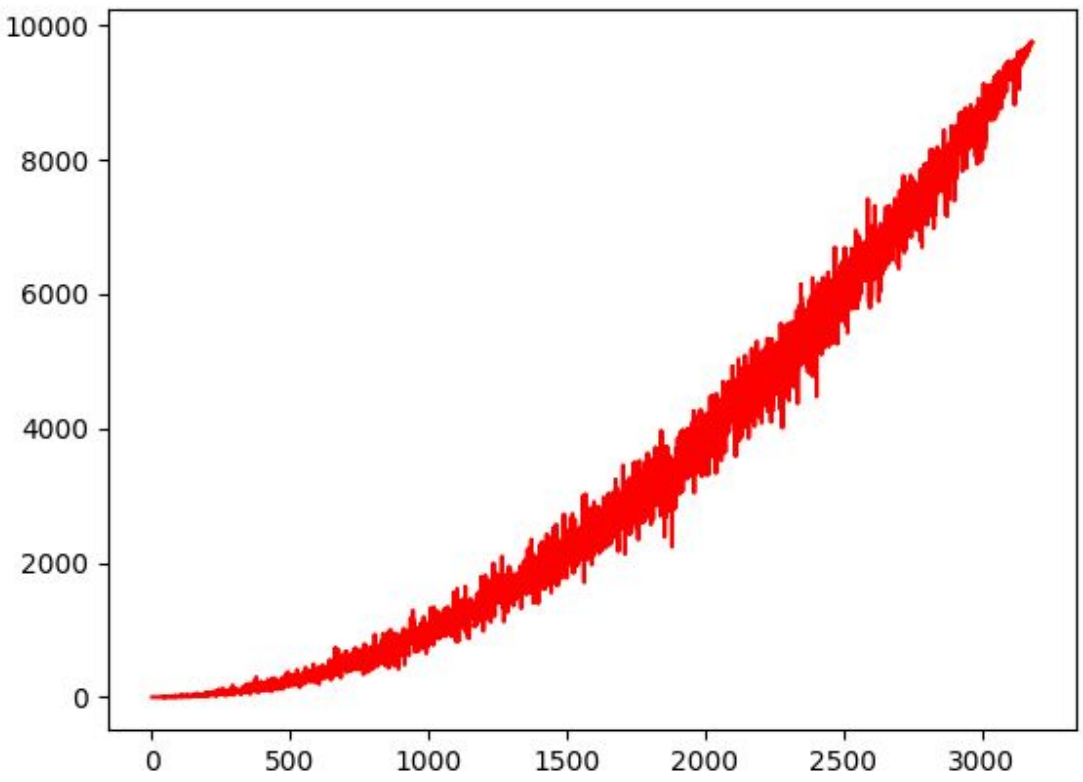
The plots for this assignment have been constructed on the 10,000 created circuits and random nodes are chosen to be compromised

1. All the relays taken into consideration and the variation between the number of nodes compromised and the number of circuits compromised is shown:



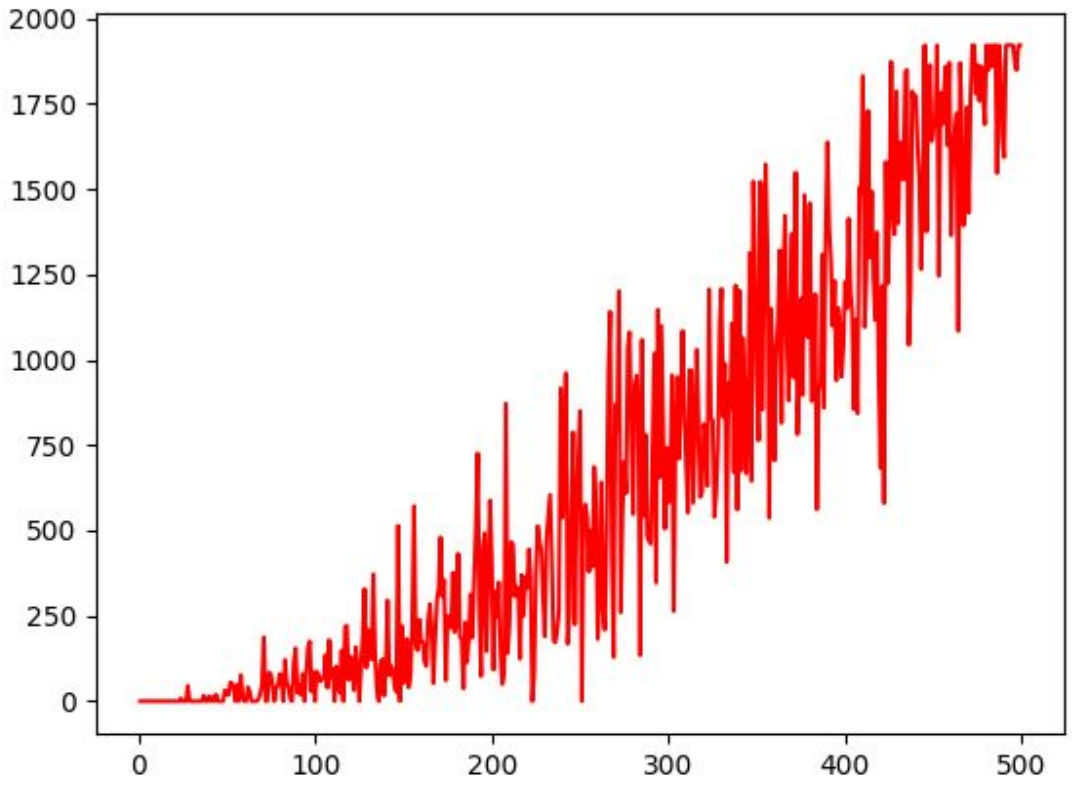
X-axis: number of relays
Y-axis: number of compromised circuits (from total 10,000 circuits)

2. Relay set has been reduced to half and the entire process is repeated:

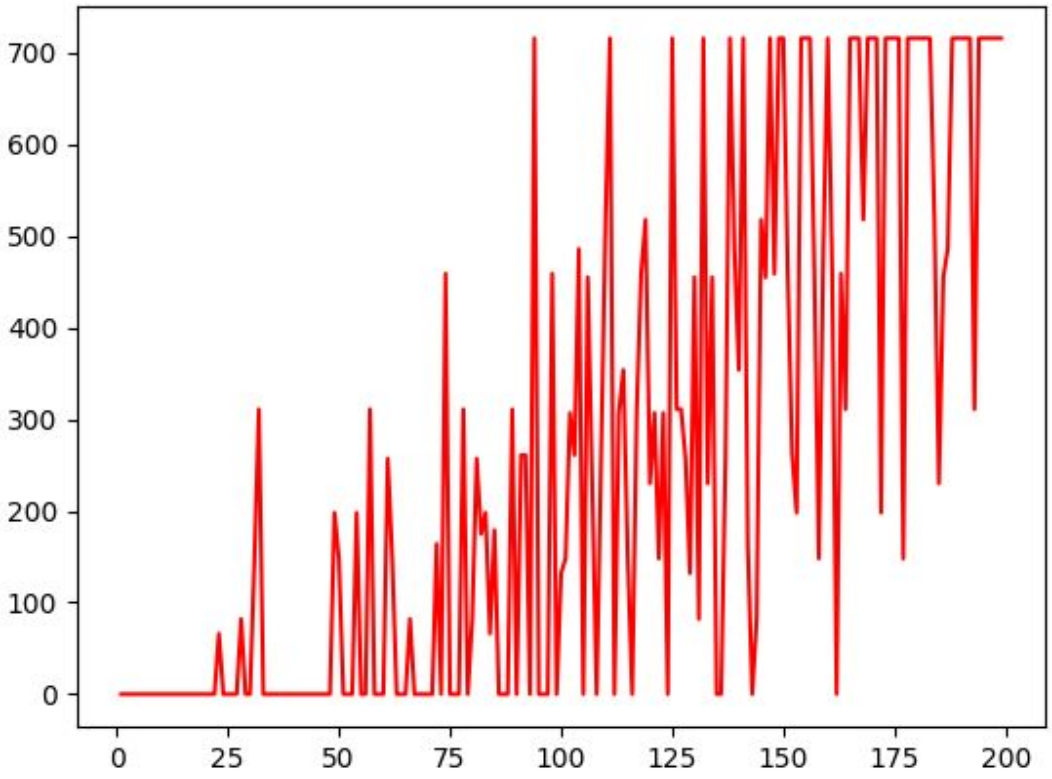


X-axis: number of relays
Y-axis: number of compromised circuits (from total 10,000 circuits)

3. Relay set is reduced to as low as 500 relays. Variability in the graph is visible but the exponential shape is maintained:

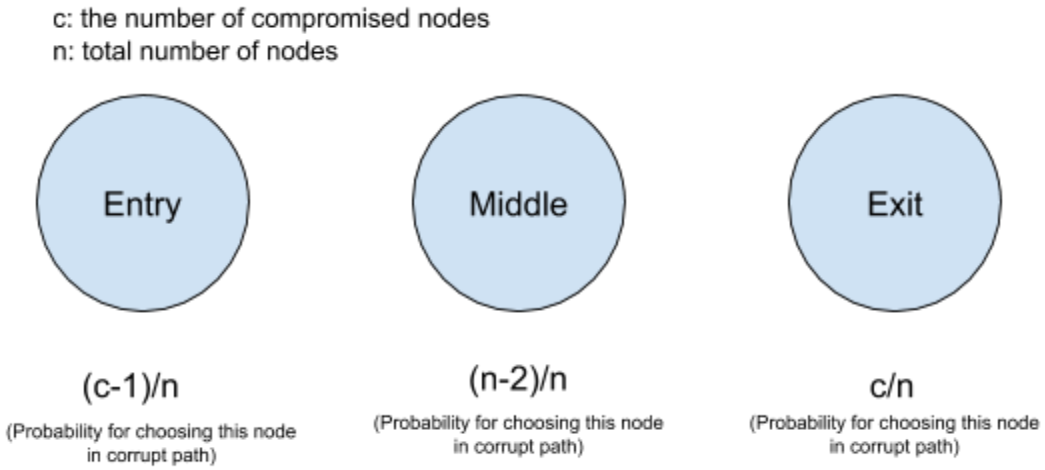


4. Relay set has been reduced to as low as 100 and the exponential shape appears to be distorted:



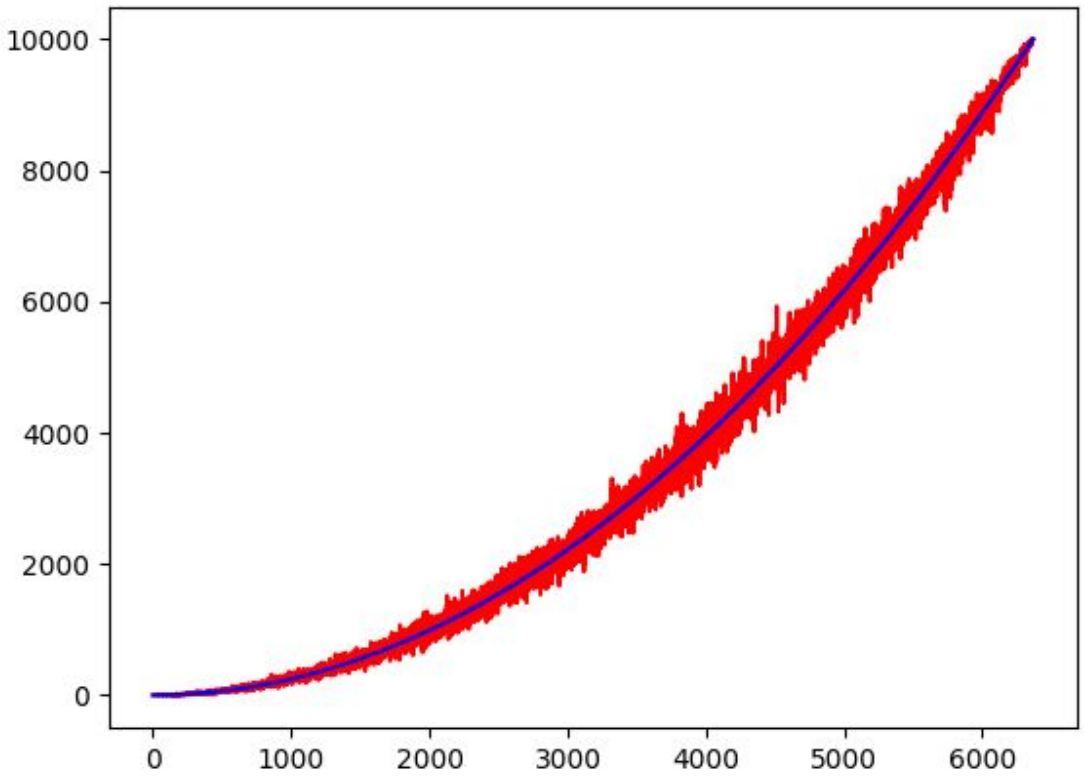
Proof of the exponential nature of the curve:

The plots even after reducing the number of relays shows similar shape. The shape of the graph clearly displays an exponential growth. To prove the exponential growth we take the help of the following diagram:



Probability of choosing a single corrupt path	=	*(c-1)/n * (n-2)/n * c/n*
	= *(approx.)*	*(c/n) ^ 2*
Number of corrupt paths from 10,000 total paths	=	*10,000 * (c/n) ^ 2*

Plotting the curve with the extracted function, we get:



Red-curve is our inference plot and the blue is of the function:
10,000 * (number of compromised nodes/6367) ^ 2