Time analysis Based Attacks Simulation in Tor Networks.

Simulazione di Sistemi

July 19, 2015

Davide Berardi 0000712698

Matteo Martelli 0000702472

Università di Bologna.

Table of contents

Introduction

Standard *shapes* of information security:

- Confidentiality
- Integrity
- Availability

There is a new security that we want to obtain: **Anonymity**

> *Anonymity [...] means that the personal identity, or personally identifiable information of that person is not known.*

Introduction

anonymity methods

There are a lot of anonymity driven software online, like *i2p*, *freenet* or *Tor*, we will talk about the last one because is the most used and expanded in the real world (2 million of client per day!).

Onion Routing

The onion routing model is a way to gain anonymity on the net:

- Provides anonymity
- Protects from sniffing

Introduced by David Goldshlag, Paul Syverson and Michael Reed in the 1999.

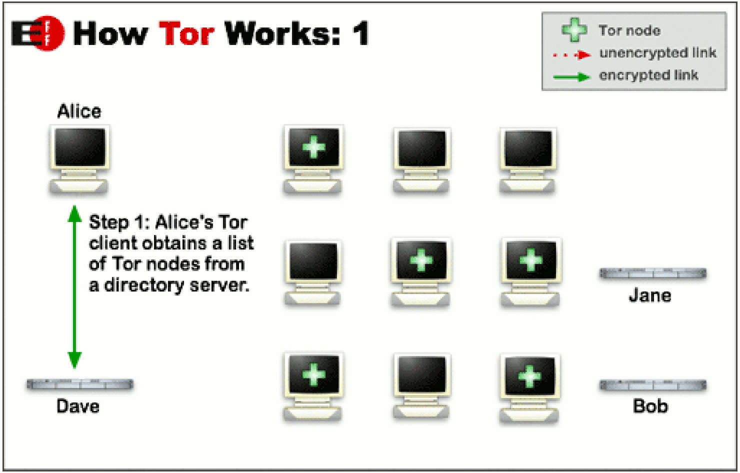
It recalls an onion because every step **peel** a layer.

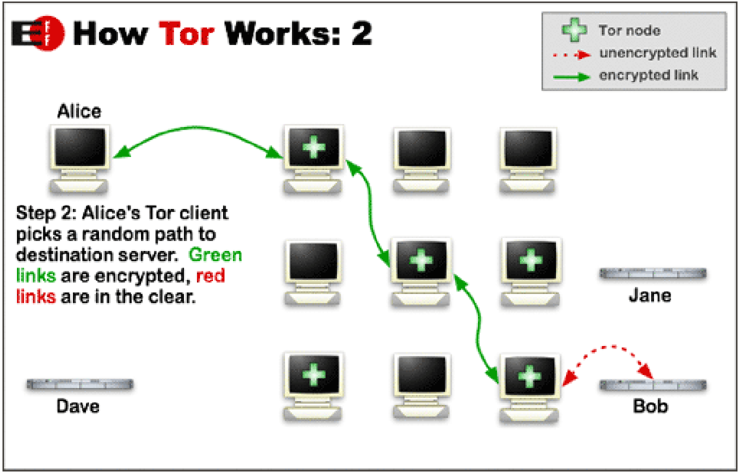
Let us see an implementation.

Overview

Tor is a group of volunteers that operates to defend anonymity online.
The system is based on an interconnection of machines, called **routers**.
It operates over the network level 4.

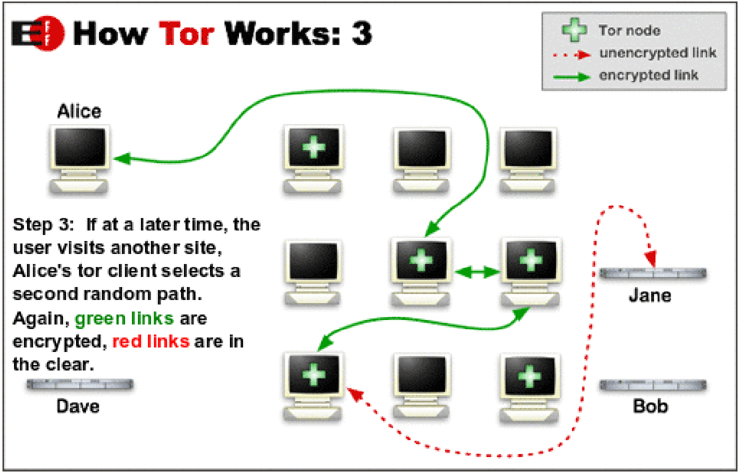
It operates as follow:

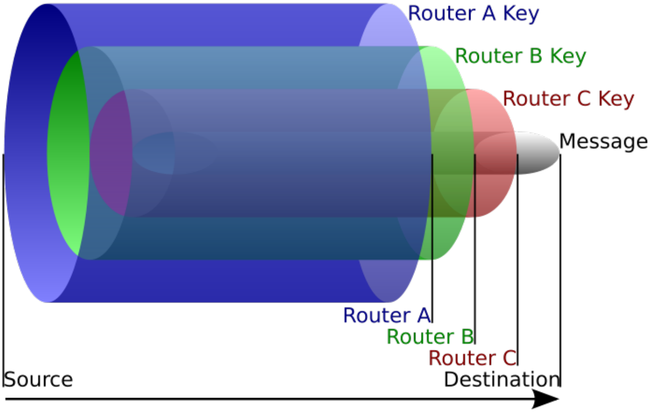

How Tor Works: 1
Tor node
unencrypted link
encrypted link
Alice
Step 1: Alice's Tor client obtains a list of Tor nodes from a directory server.
Jane
Dave
Bob


How Tor Works: 2
Tor node
unencrypted link
encrypted link
Alice
Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.
Jane
Dave
Bob

Tor

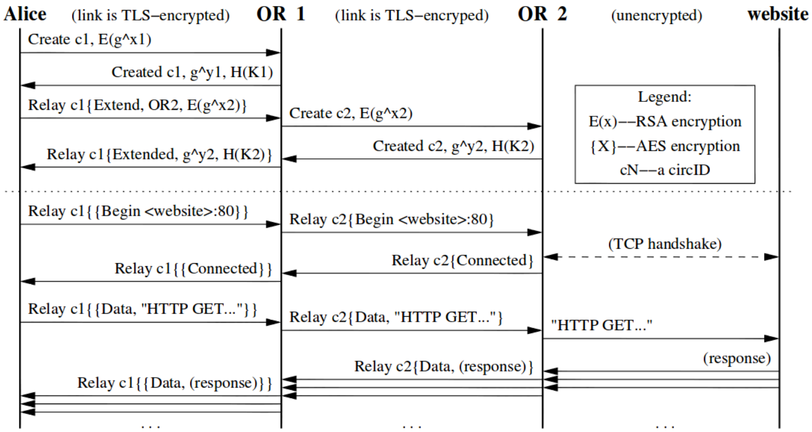
Tor workings




Router A Key
Router B Key
Router C Key
Message
Router A
Router B
Router C
Source
Destination

Tor

Tor encryption



Attacks

A lot of attacks and vulnerabilities has been discovered for the system.

- Bad apple attack.
- Side channel attacks (tor bundle).
- Cypher attacks (Tor changed the cryptosystem a lot of time).
- *Time analysis based attacks*
- Sniper attack.
- Sybil attack.

Time analysis based attacks

12

> *"Tor does not provide protection against end-to-end timing attacks[...]"*

We can place a tracker after the client node and another before the server node and check for the connection time to profile users and nodes (and later associate IP to users.)

Simulation

- From this idea we started our simulation work.
- But **OmNet++** doesn't have a reliable simulation model of Tor[1] and so **NS2/3**.
- We needed a simulation model for Tor.

[1] And Tor is fully implemented in User-Space (over level 4)

Shadow

Introduction

- We used the **Shadow** simulator
- Developed by **Rob Jansen** (U.S. Naval Research Lab).

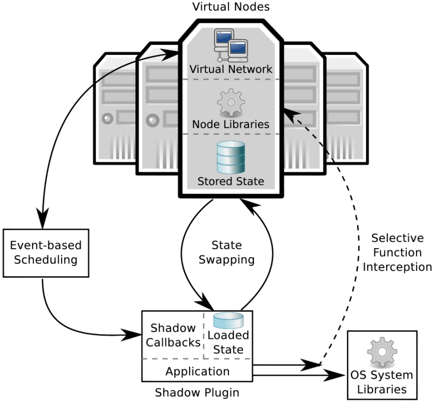
Users

Naval Research Laboratory, Washington, DC
University of Minnesota
Georgetown University
KU Leuven
Tor
University of Cambridge
Illinois, University of Illinois at Urbana-Champaign
Technische Universität Darmstadt
Wesleyan University

Shadow
Simulator internals

The main feature of **shadow** is the capability of running real applications (like tor).

Shadow combines virtualization with simulation, it virtualize network stacks and act as an micro system hypervisor (partial virtualization).



Plug-ins

Shadow plugins

So we needed:

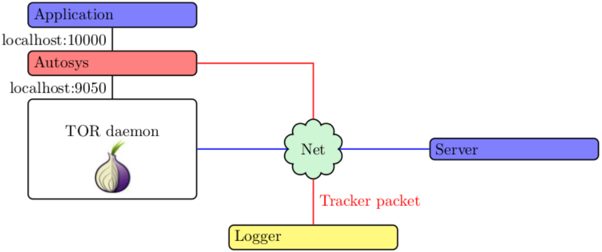
1. A client tracer shadow plug-in (proxy).
2. A server tracer shadow plug-in (proxy).
3. A logger plug-in.
4. A client plug-in (HTTP browser?).
5. A server plug-in (HTTP web-server?).

1,2 and 3 was not implemented by the shadow research team.

Plug-ins

Autosys plug-in

- Trace the SYN flag that pass trough Tor (on both ways)[2].
- Send a packet to the logger $< type(Tracked\_node); Hostname(Tracked\_node); timestamp >$.



[2] A future work could be the trace of the SYN-ACK flag, to get the corresponding gap in the analysis part.

Plug-ins

Autosys plug-in

Can be implemented in a lot of different ways:

- As a sniffer installed on the routers which listen for every TCP SYN flag (the autonomous system).
  - But **Shadow** does not support Raw Sockets.
- We decided to implement that as a simple malware-like connection proxy installed on the client virtual node[3].
- Otherwise the tracer can be installed on the guard relay (but we need to deal with re-association between traced clients and real clients because the path changes every 10 minutes).

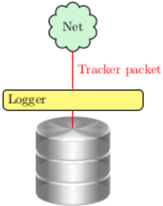
FBI Spent $775,000 on Hacking Team's Spying Tools

[3] A similar solution to the Hacking team one.

Plug-ins

Analyzer/Logger plug-in

- After being captured by the sniffers the data must be stocked for late-processing.
- We used a public logger service that logs this informations.
- Based on UDP for lightness.
- (In a real-world scenario this entity would have some form of security and could be replicated/load balanced).



Plug-ins

Analyzer/Logger plug-in format

This plug-in so will save the data that it receives from the sniffers with a common format:

$$host\_type; hostname; timestamp$$

- *host_type*: C or S if the tracked is a client or a server (the communication is going in or it exiting from Tor?).
- *hostname*: The hostname of the tracked (got by autosys).
- *timestamp*: The temporal reference of the connection (This will be used to compute distances and gaps).

This will be processed in the phase 2 to get the matches.

Plug-ins

SimpleTCP plug-in (Client)

We needed a simple client that sends his hostname on the network to compute the **matching accuracy** later.

- Do some connections to a fixed server.
- A future work should make it capable of multiple connections to multiple serves.
- This plug-in must have SOCKS5 capability to run over Tor.

Plug-ins

SimpleTCP plug-in (Server)

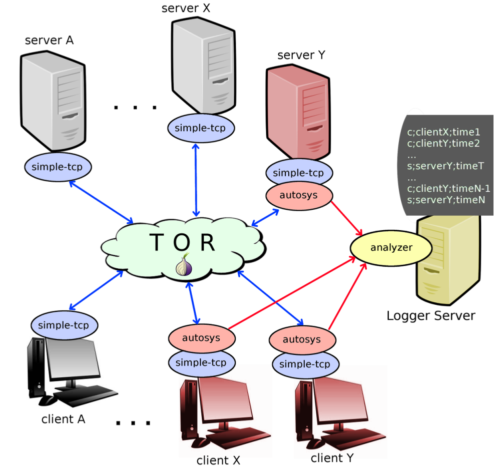
The server part, by opposite:

- Listen for some connections from the clients.
- Add a time stamp to the current received packet (correspondent host name).
- Save this data to a common file (per server).

This data will be used in the phase 2 to compute the **matching accuracy**.

Plug-ins

The big picture



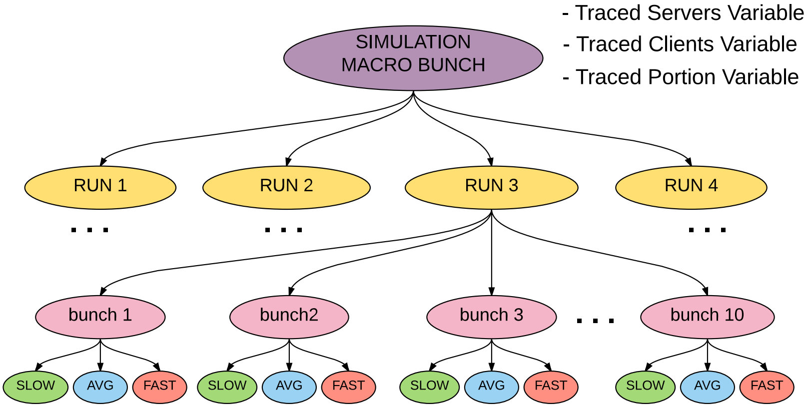
Data Analysis

- Simulation Bunches
- Simulation Handlers
- Analyzer
- Empirical Results

Simulation Bunches

1. Traced clients fixed to the 100% and increasing traced servers at each macro bunch run (0% $\rightarrow$ 100%).
2. Traced servers fixed to the 100% and increasing traced clients at each macro bunch run (0% $\rightarrow$ 100%).
3. Increasing both traced clients and traced servers (traced portion) at each simulation (0% $\rightarrow$ 100%).

Simulation Bunches

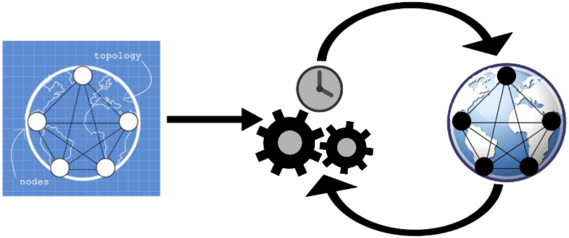


Simulation Handlers

- Netbuilder
- Launcher

Netbuilder

Genereates an XML file that describes the network



Netbuilder

Configuration

Allow the network configuration through:

- The number of TOR exit nodes in the simulation.
- The number of TOR 4authorities[4] nodes in the simulation.
- The number of clients (simpletcp) of the simulation.
- The number of servers (simpletcp) of the simulation.
- The percentage of clients tracked by an autosys plug-in.
- The percentage of servers tracked by an autosys plug-in.
- The density of the network-requests.

[4] A 4 Authority node is simply the database that keep track of the state of the TOR network and the list of the TOR relays/exit-nodes

Netbuilder

Densities

The connection densities are the sleep time thresholds between each client connection request:

- Slow: 800 (mean) - 2000 (mean) milliseconds
- Average: 80 (mean) - 1000 (mean) milliseconds
- Fast: 20 (mean) - 100 (mean) milliseconds

Launcher

Algorithm 2 Launcher script

```
for (simulation_run ← 1; simulation_run <= steps; simulation_run++) do
    for (sim_id ← 1; sim_id <= simulations_per_step; sim_id++) do
        for all density in (slow, fast, average) do
            if The client trace percentage is not fixed then
                client_trace_value ← sim_id/simulations_per_step
            end if
            if The server trace percentage is not fixed then
                server_trace_value ← sim_id/simulations_per_step
            end if
            if A configuration is present for < sim_id, density > And the percentages are fixed then
                Use the previous configuration
            else
                Generate a new configuration with net-builder
            end if
            Launch the Shadow Simulator with the appropriate configuration.
        end for
    end for
end for
```

Analyzer
Log file

...	(∗)
c;client10;1420000000	s;server7;1421023888
s;server7;1420008031	s;server2;1421156205
c;client6;1420005867	c;client8;1421160529
s;server9;1420146660	s;server3;1421318345
s;server6;1420205384	s;server0;1421332488
s;server8;1420252482	c;client7;1421487295
c;client0;1420680882	c;client4;1421634744
c;client1;1421017740	s;server9;1421726485
(∗)	...

Analyzer

Scan

- For each client connection request *creq*, it looks for candidate server acceptances
- Nested loop "temporally" limited between thr_{MIN} (100ms) thr_{MAX} (6sec)

Time distance

Let $\Delta_t(creq, s)$ be the time distance between a *creq* time-stamp and a server candidate acceptance *s* time-stamp.

Analyzer

Scan

	...	$(*)$	
	c;client10;1420000000	s;server7;1421023888	
$\Delta_t < thr_{MIN} \rightarrow$	s;server7;1420008031	s;server2;1421156205	
	c;client6;1420005867	c;client8;1421160529	
	s;server9;1420146660	s;server3;1421318345	
	s;server6;1420205384	s;server0;1421332488	
	s;server8;1420252482	c;client7;1421487295	
	c;client0;1420680882	c;client4;1421634744	
	c;client1;1421017740	s;server9;1421726485	$\leftarrow$ already considered
	$(*)$	...	

Analyzer

Matching Probability

The likelihood for a server acceptance to be related to a client request can be related to their time distance.

pmatch

$$pmatch(creq, s) = 1 - \frac{\Delta_t(creq, s) - thr_{min}}{thr_{max} - thr_{min}} \tag{1}$$

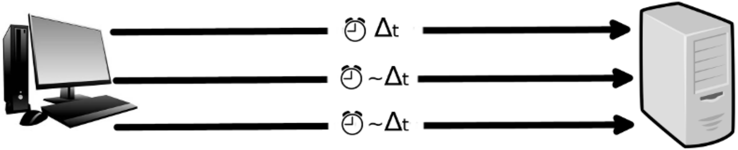
Analyzer

Matching Probability

candidate	pmatch
server9	0.992
server6	0.982
server8	0.975
server7	0.846
server2	0.823
server3	0.769
server0	0.794
...	...

The *pmatch* is higher when the server connection is closer to thr_{min}.

Analyzer
Time Gap



Acceptance Delay Correlation

If a server receives a connection request from a client after a certain time Δ_t, that server will likely receive again another connection from the same client after a time that is close to Δ_t if the Tor communication path is the same as before

Analyzer

Time Gap Average

As the *pmatch* is defined as the Δ_t normalization, let us define the *gap* average of a server s marked as candidate for a client c

$$gap_{AVG}(c, s) = \frac{\sum_{i=0}^{N(c,s)} |pmatch(creq_{i+1}, s) - pmatch(creq_i, s)|}{N(c, s)} \tag{2}$$

where $N(c, s)$ is the number of c connection requests that have been likely accepted from s.

Analyzer

Score

The *score* gained by a server s marked as candidate for a client c

$$score(c, s) = \frac{\sum_{i=0}^{N(c,s)} pmatch(creq_i, s)}{gap_{AVG}(c, s) + 1} \tag{3}$$

Analyzer
Best Candidate

client633		client637		client349	
candidate	**score**	**candidate**	**score**	**candidate**	**score**
server8	9.44	server3	59.17	server0	14.86
server0	7.01	server2	15.14	server1	13.81
server2	6.88	server8	13.96	server5	11.94
server5	6.83	server5	8.33	server2	11.20
...	...	...	...	...	...

Best Candidate

The server candidate with the **highest score** is the best candidate for a certain client.

Analyzer
Real Data

- How much are the estimated results close to the real ones?
- Use of real connections logged by the simple-tcp applications.
- Matched accuracy estimation
- Matched portion estimation

Analyzer

Matched Accuracy

For each client check if the best candidate is the real server that accepted its connections and mark it as **matched**.

If so calculate the client accuracy as the distance between the number of estimated connections N and the number of real connections M:

$$accuracy_c \leftarrow \frac{MIN(M, N)}{MAX(M, N)} \quad (4)$$

The *matched accuracy* is the average of matched client accuracies.

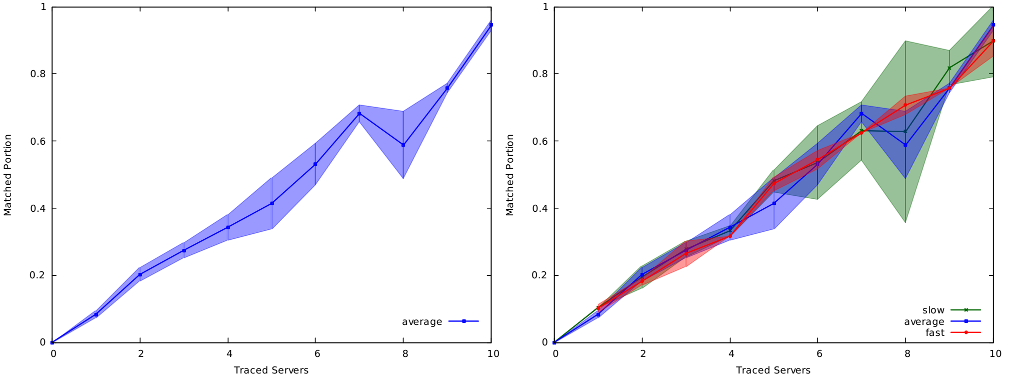
Analyzer
Matched Portion

The matched portion indicates how many traced clients found their real server:

$$matched\_portion = \frac{matched\_clients}{traced\_clients} \tag{5}$$

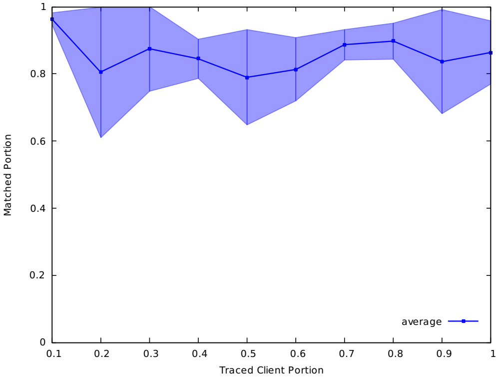
Matched Portion

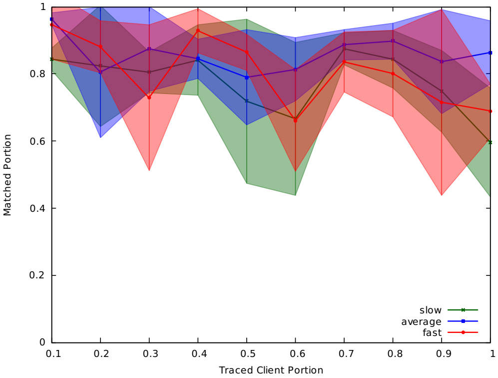
Servers traced portion augmenting



Matched Portion

Clients traced portion augmenting

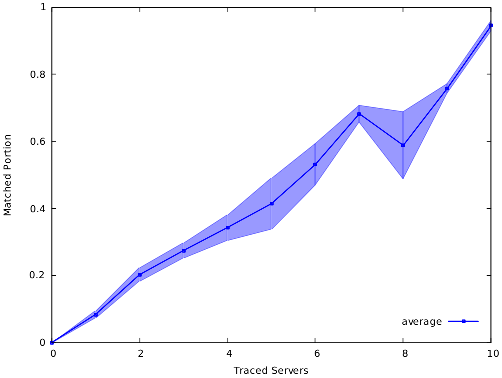


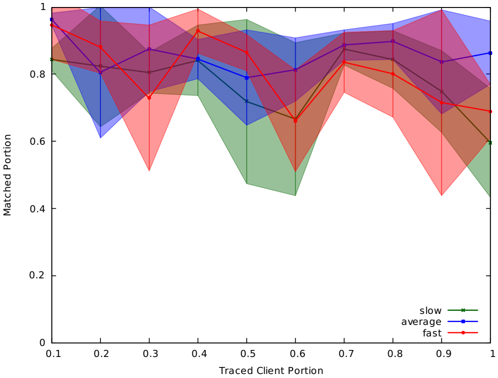


Matched Portion

Details

We can see that the matched portion tends to be constant around 80% (as with an high portion of traced servers).





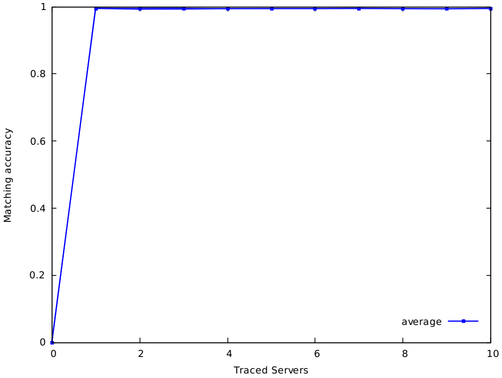
Matched Portion

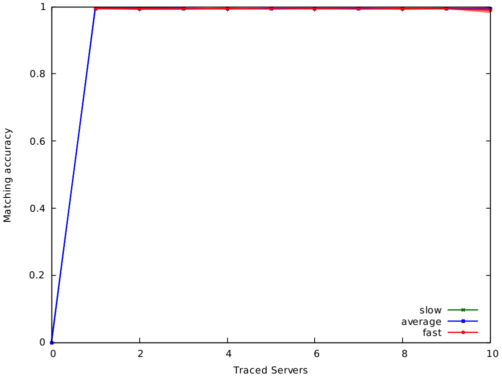
Considerations

- This behaviour was expected because the analysis is conducted from the clients side.
- Increasing the connection density the function trend seems to be more precise.
- Linearly dependent by the number of traced servers.

Matched Accuracy

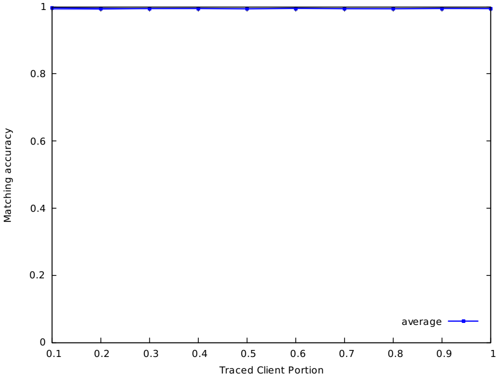
Servers traced portion augmenting

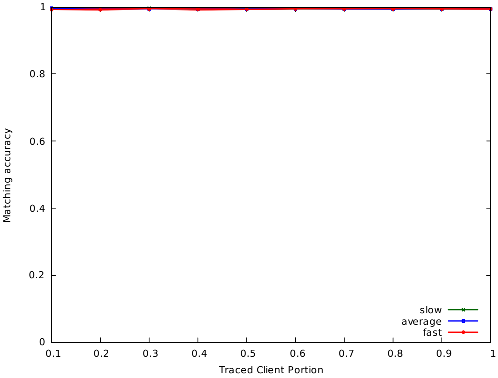




Matched Accuracy

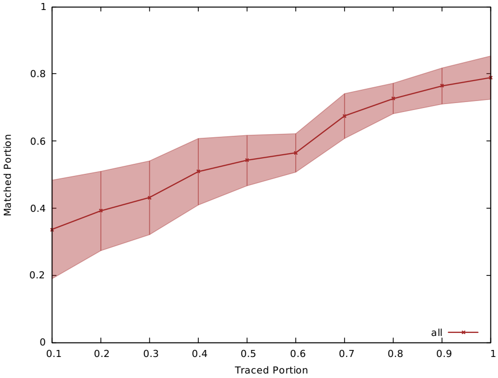
Clients traced portion

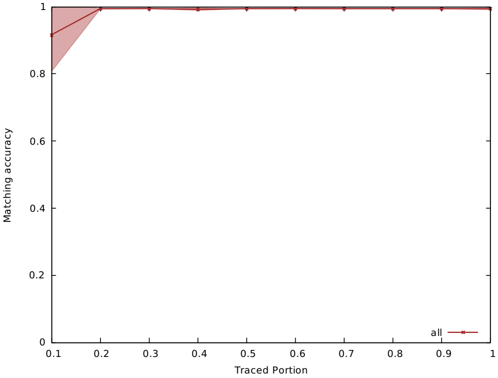




Considered values

Both portions





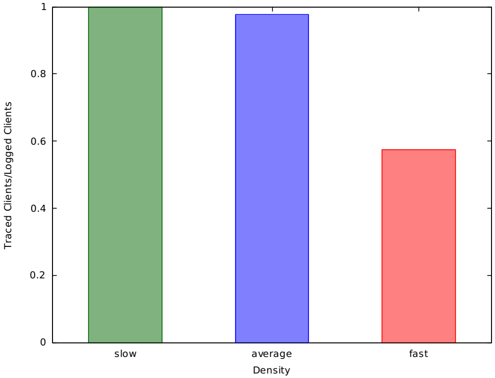
Considered values

Considerations

- Most realistic scenario
- Respect the avg sum of the other two experiments
- An attacker should be interested in trace as much Tor network nodes as possible.

Communication density

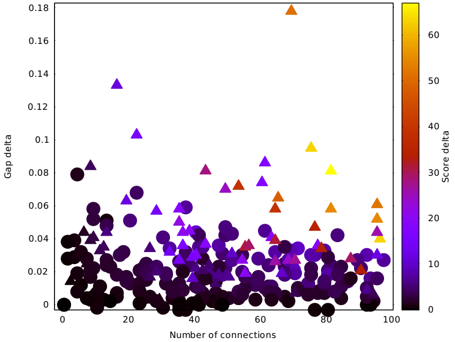
Does not seems to highly interfere with the matched portion.



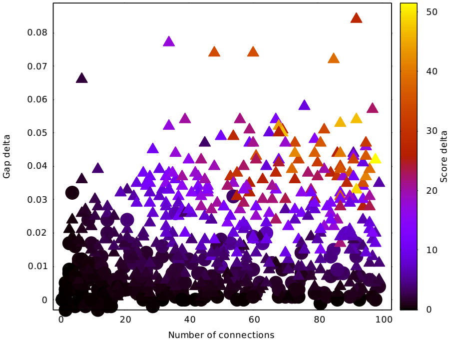
How to distinguish correct guesses?

- An attacker can, so far, get some maps between servers and clients.
- Let us see a 4 dimensional graph.

Correct guesses spatial distribution



40% of traced portion



90% of traced portion

Correct guesses spatial distribution

Considerations

- As we can see the correctly guessed servers (triangles) take place in the upper-right section in a yellowish color.
- We can choose some parameters to get the "Trusted matching set".
- An attacker so can blindly select some matchings.

Future works

- A simulation with the alternative new-born Tor client "Astoria".
- An analysis of the i2p network model and the freenet network model.
- An analysis for some modification based on the paper "Mix network model".
- The modification to the **simple-tcp** plug-in to make it capable of connecting to multiple servers in single instance.
- Go on with the score delta driven analysis.
- Repeat the experiment with a bigger Tor network.

Thank you for your attention.