# LINFO2345 - Projet
# Random Peer Sampling

2022

*Author:*
Matthieu Pigaglio

# 1 Introduction

In Client-Server architecture, such as the UCLouvain web server, the client only needs the server address to initiate a connection and access all the services provided by it. On the other hand, we have decentralized systems or distributed systems where a node needs to know the path to all other nodes in the system. How can we discover all the nodes in the system? A basic approach consists of a file located on a remote server or locally. This file can contain the path to specific data, such as in Torrent, or the list of all nodes and their address. For a small network, this is a good solution but the major inconvenience is the lack of scalability for large-scale systems and low adaptability to churn. The churn represents the flow of node entering or exiting it. The more the churn is high, the more you need an adaptable discovery system. Random Peer Sampling [6, 7, 4], aggregate RPS, is a scalable and adaptable solution to the problem based on a static size table containing known nodes and updated periodically.

The goal of this project for you is to implement an RPS protocol with Erlang and analyze the advantage and inconveniences of the family of node discovering protocol.

# 2 Random Peer Sampling

Random Peer Sampling protocols are nodes discovering protocols. They are a discovery stack of a vast amount of applications from network routing to large-scale video streaming. The first advantage of these protocols is their ability to handle churn. As explained previously, churn is a metric reflecting the proportion of exiting and entering nodes in the system. RPS protocols can rapidly detect entering node and exit nodes.

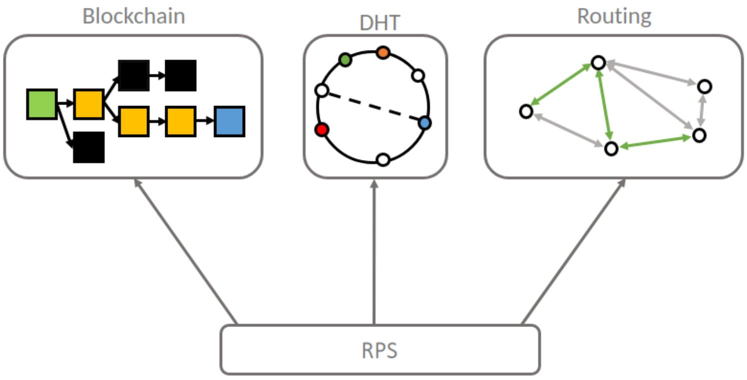

Figure 1: RPS Stack

RPS protocols are based on a local turn system. There is no global synchronization for the turn, each node is independent except for the duration of a turn which is a global parameter. In the vast majority of cases, a turn duration is between 10 and 30 seconds, but for a particular case, like streaming, this can be less than 5 seconds.

A node maintains a fixed-size view populated by its neighbor's address. This view is updated each turn by the protocol. The way the view is updated depends on the protocol. Because of the fixed size of the view, a node doesn't know all the other nodes of the system but a subset that is representative of the system. The RPS protocol creates a mesh grid between nodes which combined with a routing protocol provides a path to other nodes of the system. The static view size also provides scalability to the system.

During the project, you will implement Cyclon, one of the state-of-the-art RPS protocols, which provides fast and efficient sampling.

# 3 Project Summary

Your contribution to this project consists of 3 parts:

- Implement in Erlang the Cyclon Random Peer Sampling protocol and analyze its metrics
- Implement byzantine nodes for Cyclon and study their impact on the system reliability
- Suggest and implement two mechanisms to reduce the impact of byzantine nodes on Cyclon

# 4 Part I: Random Peer Sampling Implementation

As explained previously, you will implement the Cyclon protocol [7] in Erlang.

## 4.1 The view of a node

In Erlang, a node is a process that communicates with messages. In Cyclon, each node possesses a static size view. This view is a table where each entry is the address of a distant node and its age in the table. This age represents the number of turns this distance has been present in the table. When a node appears in the table its age is zero and each turn until it is removed from the view, the age is incremented by one.

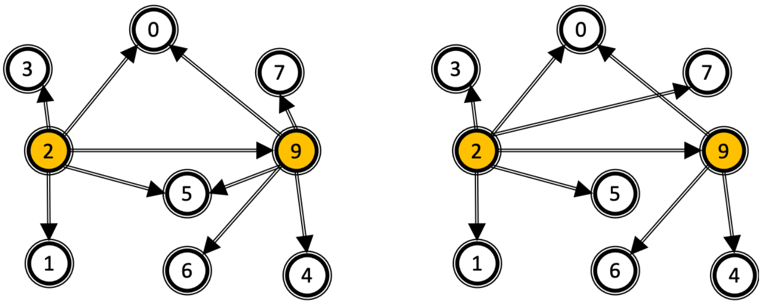

Figure 2: Links before and after a turn

## 4.2 View updating

To update its view each turn, a node P will launch the same process at the beginning of a turn:

- Increase by one the age of all nodes.
- Select the older node, note Q, and l-1 other random entries of the table. l is a global parameter.
- Reset to zero the age of the node Q in the view.
- Send the l-1 entries to Q.
- Q sends back a subset of the l-1 entry of its table.
- Remove entries pointing to P or already present entries in P's view from the subset send by Q.
- Update P's view with the remaining entries commencing by empty entries of P and after by replacing entries sent to Q. Each new entry is set with an age of zero.

If a node doesn't respond, the entry is discarded and the node waits until the next turn.
At the bootstrap of the system, each node is given a view populated with random nodes of the system. To choose the size of the view and the size of the subset, it is common to take 10% of the number of nodes in the system for the view size and half of this size for the subset. But feel free to experiment with other configurations to see the impact on the system.

## 4.3 Metrics analyse

For the rest of the project, time is in terms of turn.

Two metrics are useful to analyze when evaluating RPS protocols, the discovery proportion and the churn resilience.

The discovery proportion represents the proportion of nodes that have been at least one turn in the view of our node. We want an RPS protocol to take the lowest number of turns to discover at least 75% of all nodes in the system.

The churn resilience represents the ability of the RPS protocol to handle churn. There are two metrics, resilience against the exiting node and resilience against the entering node. The first one is the average number of turns to see an exiting node be removed from at least 75% of all node's views and the second one is the time for the node to be discovered by at least 75% of all nodes.

### 4.4 Your task

- Implement a node that sends and receives messages and runs a random peer sampling function.
- Implement a control server that launches all nodes and gets data of the experiment
- Print a graph of the discovery proportion and churns resilience for at least 3 sizes of systems (100, 500, 1000), 3 sizes of view and 3 sizes of the subset.
- Do some insight on the perfect configuration for view size and subset size, depending on the size of the system.

*Hints:*

- *Generate a CSV of your data obtained on Erlang*
- *Use python or Gnuplot to print the graph*
- *Try several duration of an RPS turn to find the smallest one to reduce the time of experiments*

## 5 Part II: The Byzantines Nodes Problem

Cyclon is considered one of the fastest RPS protocols. However, it is not suitable for an open distributed system because of its lack of security. The two types of permission for distributed systems: closed systems and open systems. In a closed system, the security is handled by the mechanism used to accept a new node which is considered to be honest. Open system or permissionless system, no one can be trusted and the system need to integrate security on each stack of it. The major threat to discovering protocols is Byzantine nodes which can conduct eclipse attacks [3] on the system to isolate parts for future attacks [2]. To conduct the attack, a byzantine node will reply to other nodes with a list of corrupted information or a list of known byzantine nodes.

To measure the impact of Byzantine nodes, we use a graph representing the average proportion of Byzantines in the views of nodes depending on the proportion of Byzantine nodes in the system at the convergence time.

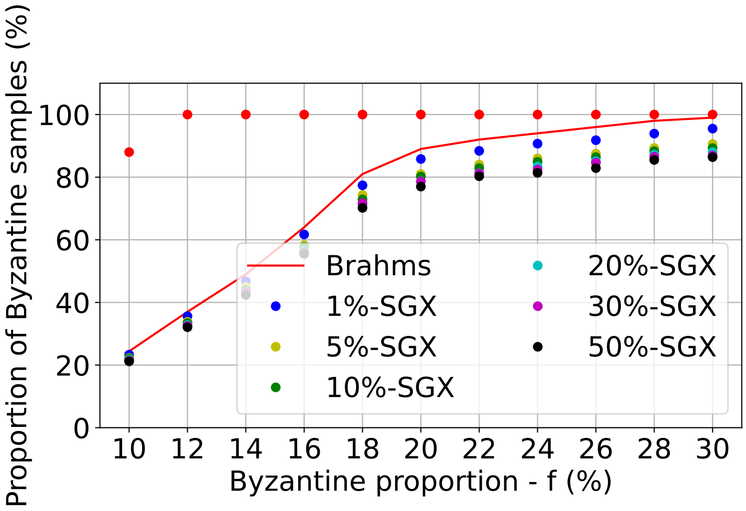

Figure 3: Exemple of a resilience plot.

Your task is to implement a byzantine node which will reply with a list of byzantine nodes during the RPS. After this, print graph of discovering proportion and convergence time for the best configuration of variables found previously with 10% of Byzantines nodes. Explain the difference with an RPS with no Byzantine nodes. Finally, print the graph of Byzantines in the views depending on Byzantines in the system for a proportion of byzantine from 0% to 50%. Explain what you see and conclude on the security against Byzantine nodes of Cyclon.

### 5.1 Your task

- Propose at least 2 mechanisms to improve cyclon's security against byzantines nodes
- If possible, implement them and test their efficiency with the three metrics used in the previous tasks.

# 6 Part III: Countermeasure to Byzantines Nodes

As you see previously, Cyclon is not protected against eclipse attacks from Byzantine nodes. There are two types of RPS protocols, Byzantine fault tolerant protocols like Brahms [1] or Raptee [5] and non-Byzantine fault tolerant like Cylon [7] protocols. However, BFT RPS protocols are generally slower than NBFT RPS.

## 6.1 Your task

- Propose at least 2 mechanisms to improve cyclon's security against byzantines nodes
- If possible, implement them and test their efficiency with the three metrics used in the previous tasks.

# 7 Quotation

The project can be done by a group of two or alone.
You will find the weight of each part below:

- Random Peer Sampling Implementation: 60%
- The Byzantine Nodes Problem: 20%
- Countermeasure to Byzantines Nodes: 20%

The project will count for 5 points on your final grade for the course.
You will provide a zip file containing 3 directories, one for each part of the project, and a report in PDF format. In each part directory, you will provide your erlang code and graph print code with comments in it and a readme on how to compile your code and how to use it. The report will contain all the results of your experiment and analysis. You need to complete all the previous parts of the project before beginning a new one. The project is on 20 points and counts for 5 points of the final mark. The majority of the point is on the RPS protocol and its analysis. You have until Friday 16/12 at 6 pm to submit your project on moodle.

Good luck and feel free to send me an email or contact me on teams, if you have any questions.

# References

[1] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, 2009.

[2] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer, 2002, pp. 251–260.

[3] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *24th USENIX Security Symposium*, 2015, pp. 129–144.

[4] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, pp. 8–es, Aug. 2007.

[5] M. Pigaglio, J. Bruneau-Queyreix, Y.-D. Bromberg, D. Frey, E. Rivière, and L. Réveillère, "Raptee: Leveraging trusted execution environments for byzantine-tolerant peer sampling services," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 603–613.

[6] E. Riviere and S. Voulgaris, "Gossip-based networking for internet-scale distributed systems," in *International Conference on E-Technologies.* Springer, 2011, pp. 253–284.

[7] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and systems Management*, vol. 13, no. 2, pp. 197–217, 2005.