|  |
| --- |
|  |
| Blockchain Network |
|  |

|  |
| --- |
|  |

Table des matières

[Presentation review (31/05 /2024): 1](#_Toc173412225)

[- Statistics node available / unavailable percentage graph 2](#_Toc173412226)

[- location to be added (see useful online tool for this) 2](#_Toc173412227)

[- Graph for connections between nodes (next week) 2](#_Toc173412228)

[Database study: 2](#_Toc173412229)

[Project implementation 2](#_Toc173412230)

[Postgres local database information (installed on the full node) 3](#_Toc173412231)

[Access DB step: 4](#_Toc173412232)

[Data visualization result: 5](#_Toc173412233)

[Hourly and daily update of the database (Python Script) 5](#_Toc173412234)

[Full Nodes availability analyze 6](#_Toc173412235)

[Running script in the background: 7](#_Toc173412236)

[Connection between nodes analysis 7](#_Toc173412237)

[WRAPPING EVERYTHING: 14](#_Toc173412238)

[Piece of code that describes how connections are made in Bitcoin 14](#_Toc173412239)

[Convergence script 15](#_Toc173412240)

[Analysis of the geographical decentralization of nodes 15](#_Toc173412241)

## Presentation review (31/05 /2024):

- Create a python script that store data in a SQL database about node availability and existing full nodes (retrieve data directly from our full node)

- Present the algorithm used by nodes to discover their peers (read about how it works)

- The aim is to have a database with information about full nodes 🡪 Create a graph using this data.

- The program should check whether full-nodes are available every hour and update the database accordingly.

- It is also necessary (if possible) to store in the DB all the nodes known by each full-node 🡪 Update this result once a day.

- You also need to keep track of the date so that you can plot it against time.

(10 / 06 / 2024)

- Timeline by day

- Populate Connected node table

(17 / 06 / 2024)

## Statistics node available / unavailable percentage graph

## location to be added (see useful online tool for this)

## Graph for connections between nodes (next week)

## Database study:

* SQL 🡪 Structured data (SQL) 🡪 PostgreSQL
* NoSQL 🡪 Unstructured data (NoSQL) 🡪 Graph database look good for this project.

Full node 🡪 Has an IP address

Full-nodes 🡪 Know other ip-address (of other full-node) 🡪 Look difficult to retrieve this information, for privacy reasons no command exist to do that in the bitcore software (bitcoin-cli).

* Solution: We can use a python library that allow Low-level communication in the network to forge a get-address messages (GitHub: https://github.com/petertodd/python-bitcoinlib)

# Project implementation

**Table : full\_nodes**

Column name Data type Details

node\_id INTEGER PRIMARY KEY, AUTO\_INCREMENT

ip\_address VARCHAR (50) NOT NULL, UNIQUE

status BOOLEAN NOT NULL, DEFAULT FALSE

last\_seen DATE

Port Integer

Network VARCHAR(50)

Last\_ping DATE

Location VARCHAR(100)

**Table: node\_connections**

Column name Data type Details

connection\_id INTEGER PRIMARY KEY, AUTO\_INCREMENT

source\_node\_id INTEGER NOT NULL, FOREIGN KEY REFERENCES full\_nodes(node\_id)

destination\_node\_id INTEGER NOT NULL, FOREIGN KEY REFERENCES full\_nodes(node\_id)

Last\_analyse DATE

**Table: node\_status\_history**

CREATE TABLE node\_status\_history (

history\_id SERIAL PRIMARY KEY,

node\_id INTEGER NOT NULL,

ip\_address VARCHAR(50) NOT NULL,

status BOOLEAN NOT NULL,

last\_ping TIMESTAMP,

record\_time TIMESTAMP NOT NULL,

FOREIGN KEY (node\_id) REFERENCES full\_nodes (node\_id)

);

## Postgres local database information (installed on the full node)

Postgres 🡪 Password : ‘root’

bitcoin\_user (utilisateur postgre) 🡪 Password : ‘root ‘

postgres (default BDD user)

Database name: Bitcoin\_Full\_nodes

Step to create a DB:

1. sudo su postgres (to connect to the postgres user)
2. psql (to enter in configuration mode)
3. CREATE DATABASE dbname; (to create the database)

SQL request to create all the tables of the database (need to be connected to the database on which we want to create the tables):

Table - Full\_nodes :

CREATE TABLE full\_nodes (

node\_id SERIAL PRIMARY KEY,

ip\_address VARCHAR(50) NOT NULL UNIQUE,

status BOOLEAN NOT NULL DEFAULT FALSE,

last\_seen DATE,

port INTEGER,

network VARCHAR(50),

last\_ping DATE,

location VARCHAR(100)

);

Table – node\_connections:

CREATE TABLE node\_connections (

connection\_id SERIAL PRIMARY KEY,

source\_node\_id INTEGER NOT NULL,

destination\_node\_id INTEGER NOT NULL,

last\_analyse DATE,

FOREIGN KEY (source\_node\_id) REFERENCES full\_nodes(node\_id),

FOREIGN KEY (destination\_node\_id) REFERENCES full\_nodes(node\_id)

);

Table – node\_status\_history:

CREATE TABLE node\_status\_history (

history\_id SERIAL PRIMARY KEY,

node\_id INTEGER NOT NULL,

ip\_address VARCHAR(50) NOT NULL,

status BOOLEAN NOT NULL,

last\_ping TIMESTAMP,

record\_time TIMESTAMP NOT NULL,

FOREIGN KEY (node\_id) REFERENCES full\_nodes(node\_id)

);

## Access DB step:

1) sudo su – postgres

2) psql

3) \c Bitcoin\_Full\_nodes

Once connected we can use \dt+ full-nodes or \d full-nodes to see information about our table.

We can also execute sql request from there: Example:

* SELECT COUNT (ip\_address) FROM full\_nodes;
* SELECT \* FROM full\_nodes ORDER BY node\_id DESC FETCH FIRST 5 ROWS ONLY;

Une image contenant texte, capture d’écran, Police, noir

Description générée automatiquement

Files transfer (by SSH and SCP): (Note 🡪 SSH don’t work when connected to the Wifi network of the university)

scp -J [kaci@eidf-gateway.epcc.ed.ac.uk](mailto:kaci@eidf-gateway.epcc.ed.ac.uk) [bitcoin@10.24.2.46:/home/bitcoin/EDI\_Network\_Project/Graph.png /home/kaci/](mailto:bitcoin@10.24.2.46:/home/bitcoin/EDI_Network_Project/Graph.png%20/home/kaci/)

scp -J kaci@eidf-gateway.epcc.ed.ac.uk kaci@10.24.2.46:/home/eidf086/eidf086/kaci/EDI\_Project/degree\_histogram.jpg /home/kaci/

## Data visualization result:

Number of nodes: ~ 58000 🡪 This is the nodes discover by our full-node not the nodes our full-nodes is connected to.

Node connected to our full node: 10

Une image contenant texte, diagramme, Tracé, ligne

Description générée automatiquement

## Hourly and daily update of the database (Python Script)

We have three different script that are run constantly to maintain the database updated:

* Update\_Check\_Status\_Node.py: This script updates the status of the nodes in the database by sending them a ping to see if they are available or not.
* Populate\_DB\_Getnodeaddress.py: This script uses the Bitcoin protocol's "Getnodeaddress" command to see whether our full nodes have discovered any new nodes and updates the database accordingly.
* Populate\_DB\_connections\_Automated.py: This script sends a "Getaddress" message to all the nodes in the database to retrieve the nodes to which they are connected and populate the database's node\_connections table.
* Track\_ip2.py: This script updates the location of the nodes

For that we use CRON on the server (full node) so that we can execute out script each X times:

1. Crontab -e
2. The following tasks have been added in the file:

* 0 \* \* \* \* /usr/bin/python3 /home/bitcoin/EDI\_Network\_Project/Update\_Check\_Status\_Node.py
* 0 1 \* \* \* /usr/bin/python3 /home/bitcoin/EDI\_Network\_Project/Populate\_DB\_Getnodeaddress.py
* 0 1 \* \* \* /usr/bin/python3 /home/bitcoin/EDI\_Network\_Project/Populate\_DB\_connections\_Automated.py

0 1 \* \* \* 🡪 Script is executed one time a day at 1 PM.

0 \* \* \* \* 🡪 Script is executed each hour.

# Full Nodes availability analyze

Une image contenant diagramme, texte, ligne, Tracé

Description générée automatiquement

SQL Query simplified :

query = """

SELECT

    DATE\_TRUNC('day', record\_time) AS day,

    (SUM(CASE WHEN status = true THEN 1 ELSE 0 END) \* 100.0 / COUNT(\*)) AS percentage

FROM

    node\_status\_history

GROUP BY

    day

ORDER BY

    day ASC;

"""

Query that takes duplicates in the table into account:

query = """

WITH latest\_status AS (

    SELECT

        node\_id,

        DATE\_TRUNC('day', record\_time) AS day,

        status,

        ROW\_NUMBER() OVER (PARTITION BY node\_id, DATE\_TRUNC('day', record\_time) ORDER BY record\_time DESC) AS rn

    FROM

        node\_status\_history

)

SELECT

    day,

    (SUM(CASE WHEN status = true THEN 1 ELSE 0 END) \* 100.0 / COUNT(\*)) AS percentage

FROM

    latest\_status

WHERE

    rn = 1

GROUP BY

    day

ORDER BY

    day ASC;

"""

## Running script in the background:

To run the script that track the location of the full nodes in our database in the background we use the command:

**- nohup python3 your\_script.py &**

## Connection between nodes analysis

Date: 11 / 07 / 2024 🡪 Sparse graph

- Number of connections fetched: 7505497

- Number of nodes in graph: 58888

- Number of edges in graph: 7482390

Check if this changes over-time (the number of nodes know by each node) 🡪 The connection can change over time as a peer can disconnect or connect to another node.

Peer-to-peer Network – Decentralization:

* Fully Decentralized network 🡪 No singular computing unit controlling any process
* Copy of the exact same data of distributed ledger
* Not everyone in the network knows each other
* All nodes have equal authority

Paper about network decentralization: <https://arxiv.org/pdf/1801.03998> (1)

<https://www.sciencedirect.com/science/article/pii/S2096720922000501> (2)

Key points from these two papers:

* Localization of the nodes: “A geographically clustered network can quickly propagate a new block to many other nodes. This makes it more difficult for a malicious miner to propagate conflicting blocks/transactions quicker than honest nodes. However, a less clustered network may mean that full nodes are being run by a wider variety of users which is also good for decentralization.” (from paper 1)
* Neighbors: “In both Bitcoin and Ethereum, peers do not reveal their neighbors. Hiding the network structure boosts privacy and security [45, 56], but also makes it harder to infer properties about the network.” (1) 🡪 The protocol doesn’t simplify that but it’s possible with forged packet (this is what I did for the database postgre – but as it’s a forged packet it’s difficult to justify that the data we receive are 100 % okay)

Statistical analysis of the sparse graph:

* Each edges represent one connection between two nodes.
* The degree indicates the number of nodes a specific node is connected to.

Factor to evaluate the decentralization of a blockchain network:

(Try to answer: What does the decentralization mean in a peer-to-peer network?)

* Number of full nodes (participants).
* Node geographical distribution.
* Diversity of node clients (to run the full nodes) 🡪 For bitcoin
* Where are the nodes operating? (Cloud provider, data centers, TOR, Local server…)   
  🡪 Purpose of this question is to see if the nodes are not all running in a specific cloud provider or in a specific data center.

For example, if a lot of nodes are running in a specific cloud provider, then does that mean that this cloud provider centralized can control these nodes (for example shutdown this nodes) meaning that this cloud provider has impact on the blockchain network.

* Developper wich maintains the peer-to-peer protocol of the blockchain? (For this one I’m not sure at all 🡪 Need to be validate).
* Software and protocol version? 🡪 If not, all nodes have the same version, does it improve the decentralization?
* Degree distribution (to analyze the topology of the P2P network)?

Analysis of connections on 09/07/2024:

The graph is sparse:

Number of nodes (n): 58888

Number of edges (m): 8812080

Average degree: 299.2827061540552 🡪 Impossible - because maximum of 125 connections by nodes. 🡪 It’s because the degree refers to the “known” relationship between the nodes and not the “connection” relationship between the nodes.

Graph density: 0.00508232217898781

Complete graph would have 1733868828.0 edges.

Complete graph if:

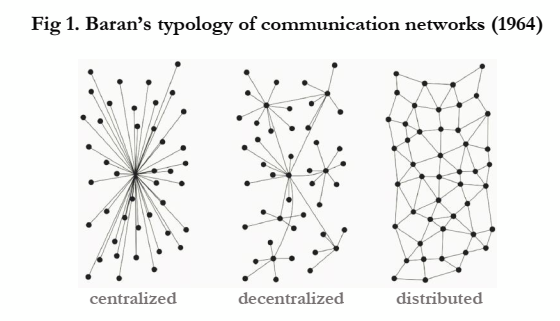
Process:

* The data used is the one from the bitcoin database from our full node.
* Each Edges represent one connection between two nodes.
* One Vertice in the graph represent one bitcoin node.
* We verify if the graph is complete using
* We verify if the graph is dense or sparse: (for a complete graph density = 1)
* From the result we can conclude that the graph is sparse.
* From the statistics conclude about what the topology of the network looks like.

Sparse graph Correlation with decentralization:

* Main objective: to determine a metric for assessing the degree of decentralization of a peer-to-peer network 🡪 How sparse graph can help?
* Sparse graph 🡪 So not all nodes are connected to all nodes in the network
* The density (~0.005) and the degree (~300) of the full nodes in the peer-to-peer network prove that the topology of the peer-to-peer network is not centralized.
* With the information of the Sparse (degree) we can study the topology of the network.

3 different topology of network that exist:



Source:

Book : HANDBOOK OF PEER-TO-PEER NETWORKING (PDF LINK - <https://link.springer.com/chapter/10.1007/978-0-387-09751-0_5>)

Blog : <https://blog.acolyer.org/2015/01/29/on-distributed-communications-networks/>

Blog (good explanation about decentralized vs distributed): <https://www.hivenet.com/post/decentralized-or-distributed-whats-the-big-difference>

Paper : <https://policyreview-info.translate.goog/open-abstracts/decentralisation-blockchain-space?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=sc>

How can we verify that the topology is decentralized and not distributed? 🡪

* How can we determine the degree of distribution and decentralization of the network?
* Maybe not that useful because a peer-to-peer network is basically decentralized (it’s better to study the different structures possible of a peer-to-peer network.

Different topology and structure possible for a P2P network:

The two Classes of P2P:

* Unstructured (This is what is used by Bitcoin P2P)
* Structured (Kademlia, CAN…)

P2P overlay network:

* Mesh network
* Random graph Network (Erdos Reny)
* Scale-free Network

Some interesting definition (specific to a peer-to-peer network):

1. Decentralization: The behavior of the P2P system is determined by the collective actions of peer nodes, and there is no central control point. Some systems however secure the P2P system using a central login server. The ability to manage the overlay [24] and monetize its operation may require centralized elements. (Source: HANDBOOK OF PEER-TO-PEER NETWORKING - <https://link.springer.com/chapter/10.1007/978-0-387-09751-0_5> )
2. Survivability: Baran defines survivability for a network as “the percentage of stations surviving a physical attack and remaining in electrical connection with the largest single group of surviving stations.” (source: <https://blog.acolyer.org/2015/01/29/on-distributed-communications-networks/>)

Study of P2P network topology specific to Blockchain system:

P2P network (very complete paper about different type of P2P) : <https://snap.stanford.edu/class/cs224w-readings/lua04p2p.pdf> (A Survey and Comparison of Peer-to-Peer Overlay Network Schemes – 2004)

Paper for P2P in Ethereum: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9426434> (very interesting)

Paper for P2P Bitcoin:

Bitcoin's dynamic peer‐to‐peer topology: (by far the best paper I found about bitcoin P2P topology network 🡪 Year of the paper: 2019)

<https://inria.hal.science/hal-03380595/file/brains2021_Bitcoin_P2P_network_study.pdf> (more basic but very clear)

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8703385> (measurement of the bitcoin network) 🡪 match the results from the database (for the number of nodes using getnodeaddress-cli)

Bitcoin:

* Software client : Bitcoin Core (official implementation) – Unofficial implementations (btcd, bitcoinj, bcoin)
* A Bitcoin peer can select up to 8 (default value) maximum outbound connections (to other peers having the full node role) and subsequently, maintain them. The bandwidth of each outbound connection is limited to 160 kb/s to prevent excessive bandwidth occupation by Bitcoin traffic. Also, each peer can accept at maximum 117 (default value) inbound neighbours or can refuse entirely any inbound connections. (Paper: BTCmap-Mapping-Bitcoin-Peer-to-Peer-Network-Topology.pdf 🡪 Date of this information: 2019)
* 🡪 So, by default a node has a degree of 8.
* 2 % of all nodes represent 75 % of the mining power (In paper above for bitcoin)
* Bitcoin nodes (miners) are mostly centralized by cloud provider (Paper above)
* Topology study with some useful info: Paper (Bitcoin's dynamic peer‐to‐peer topology – Chapter 3.5)
* They found an average degree of 2,364 in the paper: Bitcoin's dynamic peer‐to‐peer topology and compare the result to different P2P overlay network 🡪 Une image contenant texte, capture d’écran, Police, nombre

  Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* The average degree is not representative because some Peers have a very high degree and other a very low degree (maybe the answer why I didn’t find the same average degree) 🡪 **Using the Median may be better**

Ethereum:

* Kademlia DHT structure (Ethereum, DHT = Distributed Hash Table)
* In Ethereum nodes do not exchange information of their neighbor for security reason 🡪 In bitcoin also but it’s possible by using forged packet (with python) 🡪 But by using forged packet the reliability is less than using the official protocol of the blockchain.
* To measure the degree distribution of the nodes you can use the block or transaction (this is what they do in the paper above)

Some interesting metrics and information (from the paper mentioned above):

* They say that the nodes in the Ethereum P2P have degrees less than 50.
* Delay distribution
* Geographical distribution
* Latency

Analyze conclusion about the topology (for Bitcoin Network) :

* The number of nodes a peer is connected to varies greatly
* Measuring the network with Median or other mor representative metric may be better (all the paper about measuring the bitcoin network use the average degree)
* A lot of churn nodes: 97% leave and rejoin the network multiple times over a time span of about two months. In fact, the average churn rate in the Bitcoin network exceeds 4 churns per node per day.

(Source: Boston university ( <https://people.bu.edu/staro/Churn_extended_final.pdf> ) - Paper about Churn peers

Information: Can we get the node (Ip address) which create a specific block? 🡪 In theory it should be possible (but the protocol doesn’t look to allow to see or even store the Ip address of the miner that has generated a block).

Variance:

Between two nodes A and B to analyze if the nodes they know change over time 🡪 The answer is always 1000 nodes (Address message protocol) randomly selected between the nodes each nodes know. 🡪 Over time a node will discover new nodes automatically (for example our full node knows approximately 58 000 Ip addresses of nodes (alive or not).

Process : I just run the “[Get\_address\_between\_2\_nodes.py](https://github.com/Blockchain-Technology-Lab/network-decentralization/blob/main/Get_address_between_2_nodes.py)” script (In the GitHub) and then analyze the result at different day.

Distribution:

Graph that shows the distribution of how many nodes each nodes know. (it was on little less data because I clean the data and repopulate my table from zero)

Une image contenant texte, capture d’écran, Tracé, ligne

Description générée automatiquement

Wireshark Ethereum:

<https://github.com/Consensys/ethereum-dissectors> --> We can use this GitHub to add to Wireshark the feature that will allow us to inspect traffic from Ethereum.

Code analyses (for the most used version):

Found some piece of code that can be useful to analyze (for example for “get address” to see if there is some good information like the first 10 results are the node I’m connected to and the other only node I know?)

Interesting function 🡪 Peers Misbehavior:   
File: src/net\_processing.cpp

Functions:

* Misbehaving (to mark a peer for bad action)
* MaybePunishNodeForBlock or MaybePunishNodeForTx (to detect misbehavior and then call the Misbehaving function)

**Detect Misbehavior**: When an issue with a block or transaction is detected (via MaybePunishNodeForBlock or MaybePunishNodeForTx), they call Misbehaving to mark the problematic peer.

**Punish peer**: MaybeDiscourageAndDisconnect checks the peer’s misbehavior state and determines if action (like disconnecting and discouraging) is necessary.

Protocol message - degree distribution -

Ethereum note VM:

* Geth & Prysm (clients)

# WRAPPING EVERYTHING:

* Confirm degree distribution
  + how quickly does it converge? after how many queries (Just for one node)
  + Scipt that create the database

All the query that allows to do that are above (to create the DB)

* + Plot histogram for each query or use mark and recapture technique
  + once we are confident about the graph, test for graph connectivity
    - minimum cut algorithms, eg [https://en.wikipedia.org/wiki/Karger's\_algorithm](https://en.wikipedia.org/wiki/Karger%27s_algorithm)
    - clustering coefficient: <https://en.wikipedia.org/wiki/Clustering_coefficient>

## Piece of code that describes how connections are made in Bitcoin

Bitcoin core implementation of “Addr” message: **bitcoin/src/addrman.cpp**

Part of code that manage how the 1000 Ip addresses are selected:

Function:

std::vector<CAddress> AddrManImpl::GetAddr\_(size\_t max\_addresses, size\_t max\_pct, std::optional<Network> network, const bool filtered) const {

// gather a list of random nodes, skipping those of low quality

const auto now{Now<NodeSeconds>()};

std::vector<CAddress> addresses;

for (unsigned int n = 0; n < vRandom.size(); n++) {

if (addresses.size() >= nNodes)

break;

* Figure out how to get node version directly
  + look into protocol messages / packets 🡪 We can do it the same way as for the IP addresses by parsing the message result in python.

# Convergence script

There are two scripts for that:

1. Get\_address\_Export.py

This script will send “Get address” to a specific node and collect the results of the “Addr” message in a .txt file. We need to execute this script as much as possible to collect the more data possible. we can do that in ubuntu with the bash command:

**# for ((i=1; i<=5; i++)); do python3 Get\_address\_Export.py; echo "Finished iteration $i"; if [ $i -lt 10 ]; then sleep 20; fi; done**

**(This will execute automatically the script X times 🡪 In this case 5)**

1. Compare\_Convergence\_V3.py

This script will compare all the .txt files returned from the “Get\_address\_Export” script to evaluate the convergence and the number of known IP addresses of the node.

# Analysis of the geographical decentralization of nodes

The data used to generate this graph are the one from the Postgres Database.

Une image contenant texte, diagramme, cercle, Caractère coloré

Description générée automatiquement