



BITCOIN NETWORK SIMULATOR DATA EXPLOTATION

MISTIC: MÀSTER INTERUNIVERSITARI EN SEGURETAT DE LES TECNOLOGIES
DE LA INFORMACIÓ I DE LES COMUNICACIONS

Master thesis report for the studies of Màster Interuniversitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions presented by Marti Berini Sarrias and directed by Jordi Herrera Joancomartí.

Abstract

This project starts with a brief introduction to the concepts of Bitcoin and blockchain, followed by the description of the different known attacks to the Bitcoin network. Once reached this point, the basic structure of the Bitcoin Network Simulator is presented. The main objective of this project is to help in the security assessment of the Bitcoin network. To accomplish that, we try to identify useful metrics, explain them and implement them in the corresponding simulator modules, aiming to extract useful information from the simulations and display it on the GUI.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	State of The Art	2
1.3	Goals	4
1.4	Outline	4
2	Basic concepts	5
2.1	Bitcoin cryptocurrency	5
2.1.1	Blockchain	5
2.1.2	Transactions	6
2.1.3	Mining	7
2.2	The Bitcoin P2P network	8
2.3	Bitcoin network attacks	9
2.3.1	51% attack	9
2.3.2	Race attack	10
2.3.3	Sybil attack	10
2.3.4	DoS attacks	10
2.3.5	Eavesdropping	11
2.3.6	Selfish attack	11

3	The Bitcoin network simulator	13
3.1	Overview	13
3.2	Infrastructure	14
3.2.1	Simulator core	14
3.2.2	Simulator database	15
3.2.3	Visualization and analysis module	16
4	Simulation metrics	17
4.1	Simulation Information	17
4.1.1	Basic network configuration	17
4.1.2	Simulation detail level	18
4.1.3	Simulation time-frame	19
4.2	Network topology	20
4.2.1	Node Metrics	20
4.2.2	Network Metrics	20
4.3	Network communications	22
4.3.1	Sent messages distribution by node	22
4.3.2	Network discovery time	23
4.3.3	Reception of duplicated data by origin	24
4.3.4	Most used connections	24
4.3.5	Forwarding index by node	24
4.4	Information related metrics	24
4.4.1	Transactions related information	24
4.4.2	Block related information	25
5	Tools and implementation	27
5.1	Tools	27
5.1.1	MySQL	27
5.1.2	Python	27
5.1.3	Gephi	28
5.2	Implementation design	29
5.2.1	Changes in the database	29
5.2.2	Post Processing database queries	29
5.2.3	Metric calculations	30
5.2.4	Storing data in the database	30
5.2.5	Visualization database queries	32

6	Data Visualization	33
6.1	Web visualization	33
6.1.1	Summary	34
6.1.2	Node metrics	34
6.1.3	Network metrics	35
6.1.4	Network communications	35
6.1.5	Block and Tx metrics	36
6.1.6	Download gexf	36
6.1.7	Mining	37
6.2	Gephi visualization	38
7	Conclusion	41
7.1	Future work	41
	Bibliography	43

List of Figures

1.1	The price of a Bitcoin (semi logarithmic plot).	2
1.2	net.cpp source code comparison (left Dogecoin, right Bitcoin).	3
2.1	Simplified chain of ownership signatures.	6
2.2	Simplified chain of transactions.	7
2.3	Bitcoin double spend diagram.	11
3.1	Bitcoin Network Simulator general overview.	14
3.2	Database diagram of the Bitcoin Network Simulator.	16
4.1	Node clustering.	21
4.2	Graph bridge.	21
4.3	Graphs: Eccentricity, radius and diameter.	22
5.1	Database tables for analytics.	29
6.1	Web GUI simulation listing.	33
6.2	Web GUI analytics Summary tab.	35
6.3	Web GUI analytics Node metrics tab.	35
6.4	Web GUI analytics Network metrics tab.	36
6.5	Web GUI analytics Network Communications tab.	36
6.6	Web GUI analytics Download gexf tab.	37

6.7	Web GUI Mining tab.	38
6.8	Gephi overview.	38
6.9	Gephi node data.	39
6.10	Gephi edge data.	39
6.11	Gephi enhanced network graph.	40

CHAPTER 1

Introduction

Bitcoin is a form of digital currency, created and held electronically by a network of users. Which means that rather than relying on central authorities it makes use of a system that does not rely on trust. Bitcoins aren't printed like conventional currency, they're produced by the computing power of the connected users all around the world. This is achieved through software that solves mathematical problems.

Bitcoin can be used to buy things electronically, in that sense it's like conventional currency, which is also traded digitally. However, Bitcoin's most important feature that makes it different to conventional money is that it is decentralized, meaning that no single institution controls the Bitcoin network.

This provides new features to all the users, such certain degree of anonymity, transactions free of charge, fast international payments, easy mobile payments, accessibility 24/7 and money exchange at low rates. It is widely known that the provided degree of anonymity and the lack of a control authority on the Bitcoin make easier illegal activities. This has propitiated a fast growth and a high exchange rate as we can see in Figure 1.1.

Lately a lot of business have been appearing around the Bitcoin. Some of which are focused on the use of Bitcoin as a currency, for example shops, casinos and exchange business. There are other business that are trying to generate (Mining) or provide means (specialized hardware) to create new Bitcoins.

In any case, these business trade with a lot of money [8] without the endorsement or protection of any government, bank or entity officially recognized. This makes Bitcoin an attractive target, as the attackers are facing very low risks.

Since the Bitcoin proliferation several cases of attacks have been reported [19]. On the one hand vulnerabilities have been discovered on some of the most common wallets [22], furthermore there



Figure 1.1: The price of a Bitcoin (semi logarithmic plot).

have been known attacks against big exchange business [33]. Regardless of not being directed against the Blockchain or the Bitcoin protocols, they affected several users. On the other side there are some theories about possible attacks to the Bitcoin network [35], but because of the system characteristics they can not be tested or proved.

1.1 Motivation

As the Bitcoin is a subject of interest for a lot of people that invested time and money in it, we think that it is important to be able to assess if its network has the sufficient security guarantees. There is few complete documentation about Bitcoin and its scattered all around, furthermore we found that there aren't enough studies about the security of the Bitcoin network.

This makes us think that there is a need for investigation and development in this area. It also has to be noted that several cryptocurrencies share the infrastructure model with Bitcoin, adding value to any investigation as it can be applied to the other cryptocurrencies. For example: Litecoin, Dogecoin, or Quark.

To advance in the research on security of the Bitcoin network, we think that a simulator could be an interesting tool for researchers, developers and users. In fact, a simulator could help to reproduce any case scenario to test the availability, the integrity and the confidentiality of the network and the users information.

1.2 State of The Art

Now a days there are few works regarding the Bitcoin network, some of them are [10][11]. Studying the content we can see that the Bitcoin network is a complex scenario with several agents, interests and a very particular method to spread the information through all the users. From the attackers point of view there is a thin balance between the intention of stealing and cheating in this network and the need of security to keep the users confidence high, and therefore the Bitcoin's price.

It has to be noted that some other cryptocurrencies use the same source code for the networking and P2P communications as Bitcoin. This fact helps to expand the reach of interest, since any

network security flaw in the Bitcoin system affects all other cryptocurrencies that share the code with it.

We identified that the following cryptocurrencies share a very similar source code for P2P networking with Bitcoin:

- **Litecoin** <https://github.com/litecoin-project/litecoin>
- **Dogecoin** <https://github.com/dogecoin/dogecoin>
- **Quark** <https://github.com/MaxGuevara/quark>
- **Peercoin** <https://github.com/Peerunity/Peerunity>
- **Megacoin** <https://github.com/megacoin/megacoin>
- **Digitalcoin** <https://github.com/DGCDev/digitalcoin>

We observed that some of this cryptocurrencies have been developed by making a branch of the newest version of the Bitcoin source code and applying some changes to use it as the new cryptocurrency. Nevertheless, the basic code structure remains as the original Bitcoin source code.

A good example of this is Dogecoin. As we can see in Figure 1.2 they used the bitcoin source code as a base to apply some changes and develop their cryptocurrency as needed. Maybe with the changes applied, an hypothetical vulnerability in the Bitcoin code wouldn't exist in the new cryptocurrency. Nevertheless it is important to find any possible issue in the Bitcoin networking either to correct it or to check if other cryptocurrencies are affected.

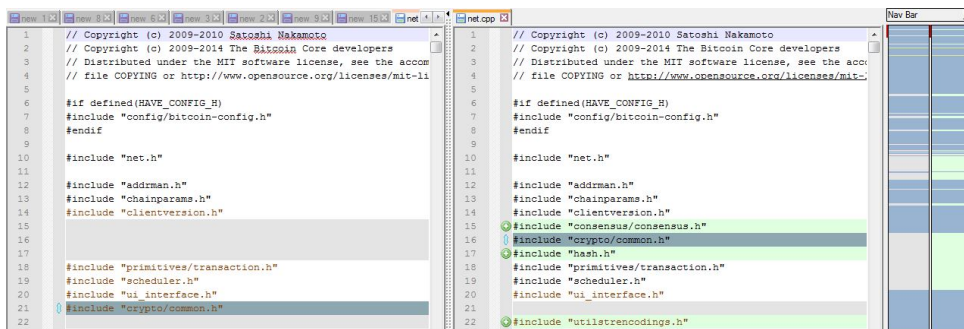


Figure 1.2: net.cpp source code comparison (left Dogecoin, right Bitcoin).

There are very few complete solutions in the area of Bitcoin simulation. The most relevant works found in that area are: a Python [29] simulator and Simbit [12]. The Python simulator is specifically designed for the Bitcoin protocol, but it lacks the ability to show detailed data about the simulation. On the other hand Simbit is a generic network simulator which includes a library to simulate a Bitcoin client, but it is still in development.

With that being said, we think that the best approach for the development of this project is to use the Bitcoin Network Simulator [25] designed by Victor Mora as a Master Final Project for the MISTIC. The simulator has been developed by the UAB team [15] with the support of the

Bitcoin Foundation. In the last months the UAB team have been developing different modules of the simulation software, and with this project we will add the necessary functionality to the simulator to extract the relevant data, process it and show it in a human-friendly way.

1.3 Goals

As the Bitcoin technology is generating a lot of interest, even among the biggest banks [14], there is a need to analyze the system accurately and check its security. Considering that performing experiments on the real Bitcoin network is difficult and is not feasible to modify it, an open-source Bitcoin Network Simulator [15] was created. This project's main objective is to retrieve information and relevant metrics at network security level from the Bitcoin Network Simulator, and at the same time to study how this information can help to understand this complex system.

As we can see below, we thought it would be interesting to include a list of the specific objectives of the project for a better tracking of the work done:

1. Analyze the structure and operation of the Bitcoin P2P network.
2. Study the communication protocols and security systems that this network has.
3. Study the vulnerabilities of the P2P network to make a network security assessment.
4. Study which metrics can be useful to assess the vulnerabilities and security of the Bitcoin P2P network.
5. Implement the metrics to extract relevant information from the Bitcoin Simulator.
6. Search and/or develop visualization methods for the extracted information.

1.4 Outline

The content of this document is organized by chapters in the following way:

- Chapter 1: Introduces the problem and the project goals.
- Chapter 2: Presents some basic concepts related to Bitcoin.
- Chapter 4: Defines the metrics that we will implement in the Bitcoin Simulator.
- Chapter 5: Enumerates the tools used to develop this project and explains the implementation process.
- Chapter 6: The data extracted from the simulator and the results of the tests are shown.
- Chapter 7: States the conclusions and the lines for future work.

CHAPTER 2

Basic concepts

This chapter provides some basic definitions that are used through out the rest of this document regarding Bitcoin, followed by the known attacks to its network.

2.1 Bitcoin cryptocurrency

Bitcoin (cryptocurrency) is a peer to peer electronic payment system invented by Satoshi Nakamoto [26] around 2008 [36]. On 2009 the first Bitcoin software client was released [7] and the first coins created. Bitcoin is designed to use cryptography to control the creation and transference of money, rather than relying on central authorities. In this system, transactions are verified by the network nodes and recorded in a public ledger called the blockchain. This system is distributed across all the users, so anyone can use it for free and it has no central authority or regulatory entity.

Bitcoins are created as a reward for adding new blocks to the blockchain, these new blocks contain user's payment transactions. To perform this task the users use their computing power either to create the blocks or to verify them. This activity is called mining and miners are rewarded with transaction fees and newly created Bitcoins. Besides being obtained by mining, Bitcoins can be exchanged for other currencies, products, and services. Users can send and receive Bitcoins for an optional transaction fee.

More detailed information can be found in the book Mastering Bitcoin [2]. Other useful resources to obtain information on the Bitcoin are [6][5][4].

2.1.1 Blockchain

The blockchain is a public record containing Bitcoin transactions and it is generated without any trusted central authority. In other words maintenance of the block chain is performed by

a network of nodes running a Bitcoin software client. For example, transactions take the form “payer X sends n Bitcoins to payee Y” and are broadcasted to the network so that any node can validate and add them to their private copy of the blockchain. Once the node has completed these tasks it broadcasts the new additions to other nodes [2]. In other words, the blockchain is a distributed database. To verify the ownership of every Bitcoin without confiding in third parties, each network node can store its own copy of the blockchain. Approximately every ten minutes, a new block is created using a group of accepted transactions and immediately its added to the blockchain, and simultaneously published to all nodes. This procedure allows Bitcoin software to determine when a particular Bitcoin amount has been spent, preventing double-spending. It could be said that the blockchain is the only place that Bitcoins can exist in the form of unspent outputs of transactions.

2.1.2 Transactions

A transaction is the method used to deliver or acquire Bitcoins. To do so, a payer must digitally sign the transaction using the corresponding private key (see Figure 2.1). Afterwards the rest of the network can verify the signature using the public key of the payer. Without knowledge of the private key the transaction cannot be signed, therefore Bitcoins cannot be spent. This means that if the private key is lost, the Bitcoin network will not recognize as valid any other evidence of ownership [9]. This implies that the coins are then unusable, therefore lost. The value of the transaction can be specified as an arbitrary multiple of Satoshi. A Satoshi is the smallest amount of bitcoin, represented by 0.00000001 Bitcoin.

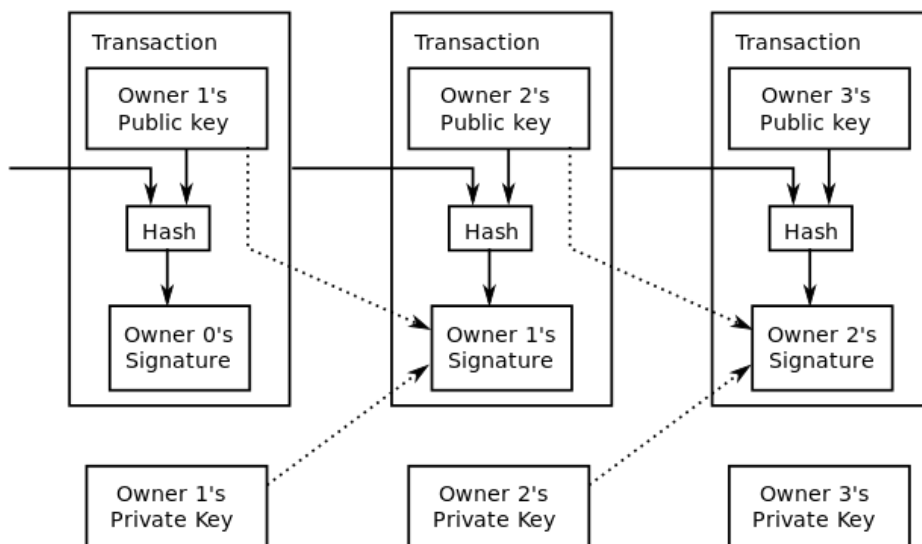


Figure 2.1: Simplified chain of ownership signatures.

A valid transaction shall have one or more inputs which must be an unspent output of a previous transaction (see Figure 2.2) and must be digitally signed to guarantee its validity. The use of multiple inputs can be considered similar to the use of multiple coins in a cash transaction. Equivalently a transaction can also have multiple outputs, allowing the user to make multiple

payments in one go. Like in a cash transaction, the sum of inputs (coins used to pay) can exceed the intended sum of payments, in such a case, an additional output is used to return the remaining Satoshis back to the payer. Note that any input not accounted for in the transaction outputs, becomes the transaction fee collected by the miner.

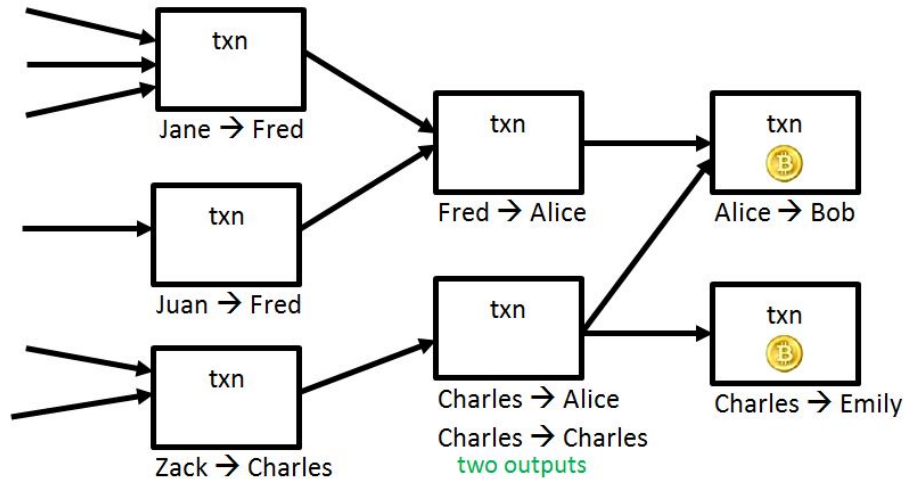


Figure 2.2: Simplified chain of transactions.

2.1.3 Mining

Mining is the method used in the Bitcoin system to record and keep consistent the performed transactions. Any node of the network can perform as a miner to keep the blockchain consistent, complete and unalterable by repeatedly verifying transactions and collecting them into a group called a block. All existing blocks contain information that “chains” them to the previous block, thus giving the blockchain its name. This information is a cryptographic hash of the previous block, using the SHA-256 hashing algorithm [2].

In order to be accepted by the rest of the network, a new block must contain a so-called proof-of-work. This proof-of-work requires miners to find a number called a nonce, so that when the block content is hashed along with the nonce, the result is numerically smaller than the network’s difficulty target [2]. As a result of the characteristics of a secure cryptographic hash, the proof-of-work is easy to verify for any node in the network, but extremely time-consuming to generate. Therefore, miners must try many different nonce values before meeting the difficulty target.

The nodes always consider as valid the longest branch of the blockchain, hence the proof-of-work system makes modifications of the blockchain extremely hard. To do so, an attacker must modify all subsequent blocks in order for the modifications of one block to be accepted. As new blocks are mined all the time, the difficulty of modifying a block increases as time passes and the number of subsequent blocks (also called confirmations of the given block) increases [28].

2.2 The Bitcoin P2P network

The Bitcoin network is composed by several nodes that may have different capabilities (mining, send, receive, verify, ...) and are connected in a P2P structure by using an open source protocol. This structure causes that the nodes only have a limited visibility of the whole network. Anyhow nodes can discover new nodes by asking to the peers.

The users send Bitcoins by broadcasting digitally signed messages to the network using a software wallet. This transactions are recorded into the blockchain through the process of mining. Any node can join or leave the network at will and upon reconnection the node will download and verify new blocks from other nodes to complete its local copy of the blockchain.

Bitcoin uses the P2P network and simple broadcasts to propagate transactions and blocks over TCP. To perform the message broadcasts in a controlled way the nodes use an internal list to send it to known nodes, which is modified dynamically by asking for new nodes within the P2P network. To do so, each node can broadcast `GetAddr` messages to their peers to request their list of known nodes, this way new nodes can discover the network and connect with the nodes that are active at that moment.

The possible messages that are used in the Bitcoin P2P network are the following:

- `version` - Information about the program's version and its block count. Exchanged when first connected.
- `verack` - Sent in response to a version message to acknowledge that we are willing to connect.
- `addr` - List of one or more IP addresses and ports.
- `inv` - List with the blocks and transactions node's inventory.
- `getdata` - Requests a single block or transaction by hash.
- `getblocks` - Requests an `inv` of all blocks in a range.
- `getheaders` - Requests a headers message containing all block headers in a range.
- `tx` - Sends a transaction. This is sent only in response to a `getdata` request.
- `block` - Sends a block. This is sent only in response to a `getdata` request.
- `headers` - Sends up to 2,000 block headers. Nodes that don't mine can download the headers of blocks instead of entire blocks.
- `getaddr` - Requests an `addr` message containing a bunch of known active peers (for bootstrapping).
- `submitorder`, `checkorder`, and `reply` - Used when performing an IP transaction.
- `alert` - Sends a network alert.

- **ping** - Used to check that the connection is still online. A TCP error will occur if the connection has died.

To connect to a peer, the node sends a version message containing its version number, block count, and current time. The remote peer will send back a **verack** message and his own version message if he is accepting connections from that node's version. The first node will respond with its own **verack** if it is accepting connections from the remote peer's version. The node then exchanges **getaddr** and **addr** messages, storing all addresses that he doesn't know about. These **addr** messages, at the beginning of an exchange, can contain up to 1000 addresses.

The time data from all peers is collected, and the median is used by Bitcoin clients for all network tasks that use the time (except for other version messages).

When a node wants to send a transaction, it sends an **inv** message to all of their peers. Their peers will request the full transaction with a **getdata** message. If they consider the transaction valid after receiving it, they will also broadcast the transaction to all of their peers with an **inv** message. Peers ask for or relay transactions only if they don't already have them, that means that a peer will never rebroadcast a transaction that it already knows about, though transactions will eventually be forgotten if they don't get into a block after a while. In that case, the sender and receiver of the transaction will rebroadcast it.

Anyone who is mining will collect valid received transactions and work on including them in a block. When someone finds a block, they send an **inv** containing it to all of their peers. The rest of the communication process works the same way as transactions, as explained above.

All nodes broadcast an **addr** message containing their own IP address every 24 hours. Nodes relay these messages to a couple of their peers and store the address if it's new to them. Through this system, everyone has a reasonably clear picture of which IPs are connected to the network at that moment. After connecting to the network, the node gets added to everyone's address database almost instantly because of an initial **addr**.

Network alerts are broadcasted with alert messages which contain the entire alert and no **inv**-like system is used. If a received alert is valid it is relayed to all peers, additionally for as long as an alert is still in effect, it is rebroadcast at the start of every new connection.

2.3 Bitcoin network attacks

In this section we describe the most significant attacks designed against the Bitcoin network and its users.

2.3.1 51% attack

This is maybe the most well known vulnerability of the Bitcoin network, and it's inherent to the design. The Bitcoin stores all the transactions data in the blockchain, however it may suffer a fork. In that case only the longest chain will be considered valid.

Using this idea an attacker can generate a fork at some point of the blockchain to rewrite all posterior transactions. This can be achieved with guaranties if the attacker controls more than half of the computing power of the network.

For example, an attacker can create two different transactions spending the same coins, one of them to buy a product from an online shop and the other sending the money to itself as we can see in Figure 2.3. At that point the attacker broadcasts the first transactions and simultaneously tries secretly to create a fork with the other transaction. When the first transaction has enough confirmations from the network the shop will send the product. At this point the attacker releases its secret fork, if he has more than half of the computing power of the network the fork will be longer than the main chain, so it will be accepted as the correct chain. The consequence of such action will be that the payment transaction to the shop won't be on the chain, instead there will be a transaction between wallets in possession of the attacker. That render the transaction with the shop invalid, as the money is already spent.

This kind of attack can also be used to block transactions or to block nodes from adding blocks to the blockchain.

2.3.2 Race attack

By design Bitcoin needs some time to confirm every transaction, at least 10 minutes are needed for it to be included in a block and 10 minutes extra for every additional confirmation. Some type of business can't wait so much time, so they accept a payment as valid without any confirmation. That means that they don't wait for the transaction to be added in a block, in other words, when the transaction is detected as broadcast in the network is considered as valid. This behavior is not a vulnerability of the Bitcoin protocol, but a misuse from the users.

It is also important to know that an attacker can exploit this by sending a transaction which moves the money from one wallet to another and immediately after sends the same money to the business. This will make that the second transaction will be invalidated in a short time, but when this happens the business service or product will be already provided. In spite of this fact, the affectation of this attacks can be monitored through the data propagation times of the network.

2.3.3 Sybil attack

This attack consists on isolating a node from the network to control its communication, with the intention of blocking transactions, blocks, discover the users identity or generating a double spend of the money. To perform this the attacker needs to be in control of enough nodes to be sure that the victim is only connected to those nodes, this way the attacker controls what messages the victim can send or receive from the network and the reception time. In the following source we can get hold of some information about the Eclipse [21] attack on Bitcoin.

2.3.4 DoS attacks

The objective of this attack is to cause a Denial of Service by sending big amounts of data to a node. This means that the node would be busy processing the transactions from the attacker

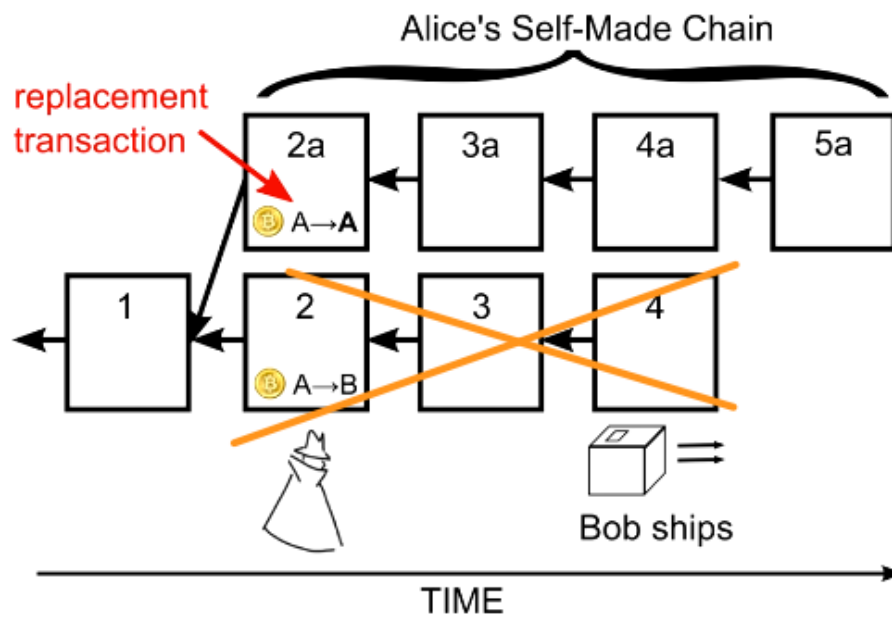


Figure 2.3: Bitcoin double spend diagram.

and the normal transactions would be discarded or kept on hold. For example an attacker could send repeatedly a block or a transaction to force a node to check if is valid or not, however the Bitcoin clients have several countermeasures implemented. This kind of attack has a variant called DDoS (Distributed Denial of Service). Here the principle is the same but the attack is performed by several agents to increase the chances of success and reduce the possibility of blocking or detecting the attack.

2.3.5 Eavesdropping

The nodes are communicating constantly across the network, so it can be detected when a node transmits a new transaction or forwards an existing one by observing the traffic from that node. Therefore an attacker observing this communications could be able to link bitcoin addresses and public IP addresses [27], this may cause anonymity issues in the network if security measures are not implemented.

2.3.6 Selfish attack

This attack [13] is an evolution of the attack seen in 2.3.1, since theoretically the 51% of the computing power of the network is not needed anymore.

In this kind of offense the attacker publishes the detected blocks in a selective way and sometimes simultaneously, in order to make other nodes discard blocks and lose money. Eventually some nodes will join the attacker to avoid suffering the consequences of the attack and increase their benefits.

The Bitcoin network simulator

In this chapter we will introduce the Bitcoin network simulator and perform a general description about its structure and how does it work.

3.1 Overview

The Bitcoin Network Simulator is an event based simulator of the Bitcoin network implemented in Python. Python was chosen to facilitate the development of this very complex system, furthermore there are several libraries in python about the Bitcoin protocols [24][34]. To address the event handling for the simulator, the Simpy [32] framework was chosen for its simplicity and the already developed features.

The main goal of the Bitcoin Network Simulator (BTC-NS) is to be able to create instances of Bitcoin nodes from different profiles, configure the network, simulate the interaction of the nodes during a period of time, log all the events and analyze the obtained data.

The Bitcoin Network Simulator can simulate the behavior of different Bitcoin nodes and their interactions through the P2P network. As in the real Bitcoin network system, many events occur in a simulation during a period of time, those events range from a wide variety of options, for example block and transaction creation, new nodes joining the network, already existing nodes exchanging information or network connectivity problems. However, the Bitcoin simulator is aware of each and every one of the events that happen in the network. This provides valuable knowledge regarding the Bitcoin network that allow researchers and developers to study and improve it. The Bitcoin network simulator not only does it provides the possibility of monitoring the whole network, it also allows the user to fully configure it for study and testing purposes.

With these features, the simulator may be used by developers and researchers to see how small changes in the behavior of the Bitcoin client affect the whole network, to study how the network

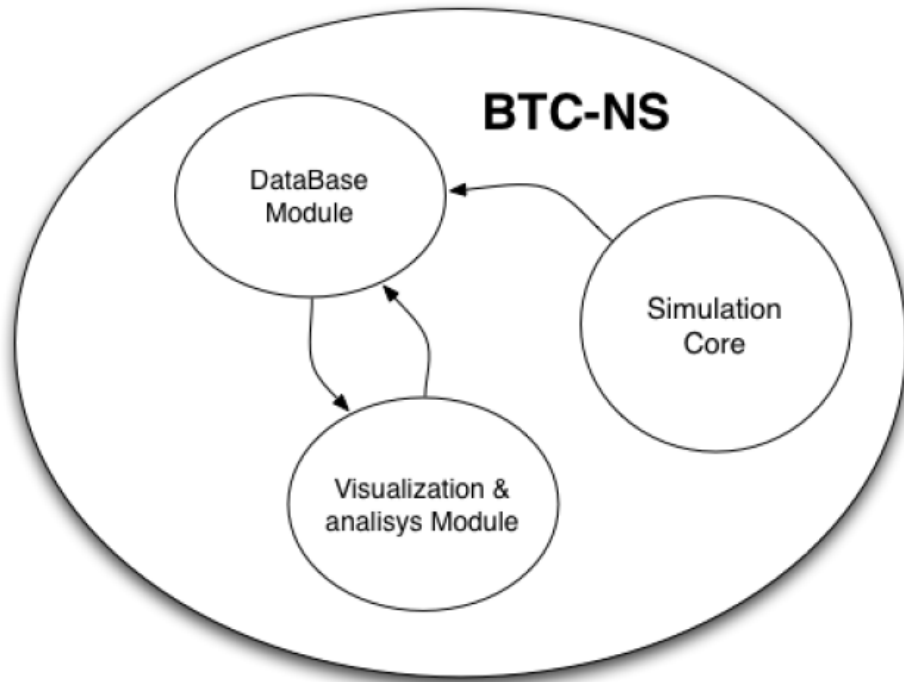


Figure 3.1: Bitcoin Network Simulator general overview.

reacts to misbehavior of a subset of its nodes, to analyze how system or connectivity failures affect the overall functionality of Bitcoin, to study attacks and their ability to success and so forth.

3.2 Infrastructure

The simulator structure has three different components (see Figure 3.1): the simulator core, a database module and a visualization and analysis module. The simulator core is in charge of the simulation process itself, it runs the different Bitcoin nodes following their specified behavior and it handles the network communication between all the simulated nodes. The simulator core will record all the events happening during the simulation in a database so that they can be retrieved after the simulation has ended. This is where the results visualization and analysis module comes into play. The visualization and analysis module will read and process the events that had happened during the simulation and had been stored in a database by the simulator core. This will allow the user to visualize a summary of what has occurred and will offer a graphical interface to further explore them.

3.2.1 Simulator core

This module is the central part of the code and it implements all the necessary methods to represent nodes, connections and the protocols required for the communications. This module stores all the events created during the simulation to the database.

A simulation is the result of running this BTC-NS module with specific parameters. Those parameters are the initial scenario, a defined level of detail and a time interval.

The initial scenario defines the nodes that form the P2P Bitcoin network, including their behavior and the network topology, in other words, how do these nodes connect between each other.

The nodes behavior in the system is defined by their profile by their specific parameters. Examples of node profile could be a specific version of the standard Bitcoin client or a node implementing the selfish behavior as described in [13]. Some parameters in the nodes are shared between all node profiles, while others are specific for a given profile. An example of a configuration parameter that all nodes share is their hashrate.

Although some BTC-NS parameters (such as level of detail and time interval) are straight forward to set, defining the initial scenario is more complex. For that reason, the initial scenario can be provided using two different scenario configuration modes: normal mode and advanced mode. The advanced scenario configuration mode allows a user to upload a GML file that contains all the nodes with their profiles and associated configuration parameters together with all edges describing the network topology, including the existence of a specific connection, its latency and its bandwidth.

On the other hand, the normal scenario configuration mode allows to describe the initial scenario in a more general way. The normal configuration mode allows a user to define the total number of nodes of the network (instead of defining them individually) and establish which percentage of nodes of the network will have each of the profiles (and parameters) of interest. For example, one possible configuration will be to create a network with 100 nodes, 10% of them running an standard Bitcoin client and with a hashrate of 500Mhs, 70% of them running also a standard Bitcoin client but with a hashrate of 50Mhs, and the 20% left running a selfish node with a 300Mhs hashrate. Note that in this mode the user does not control the profile or parameters of each individual node, just the overall number of nodes with each profile.

As a part of this normal configuration mode the network topology is defined by using network models. The usage of network models allows a user to describe high-level properties of the network without having to individually describe every specific connection that exists in the network. In this case a scenario description uses three different models: the network connections model, the latency model and the bandwidth model. The network connections model is used to decide whether two nodes are connected or not, additionally a special network model allows nodes to discover peers following the behavior described by their profile. The latency and bandwidth models are used to assign a latency and bandwidth value to each connection.

3.2.2 Simulator database

The database is created using MySQL. It contains all the necessary tables to store all the simulation data as we can see in Figure 3.2. A specific user has been created for the application so it can access the simulation data and write new entries on the tables.

This database is a critical part of the simulator, so we have to find a tradeoff between the amount of data stored in the database and the data that can be computed on real time by using a simple query. This is important to be taken into account during the development to guarantee that the simulator can be used in machines with limited resources.

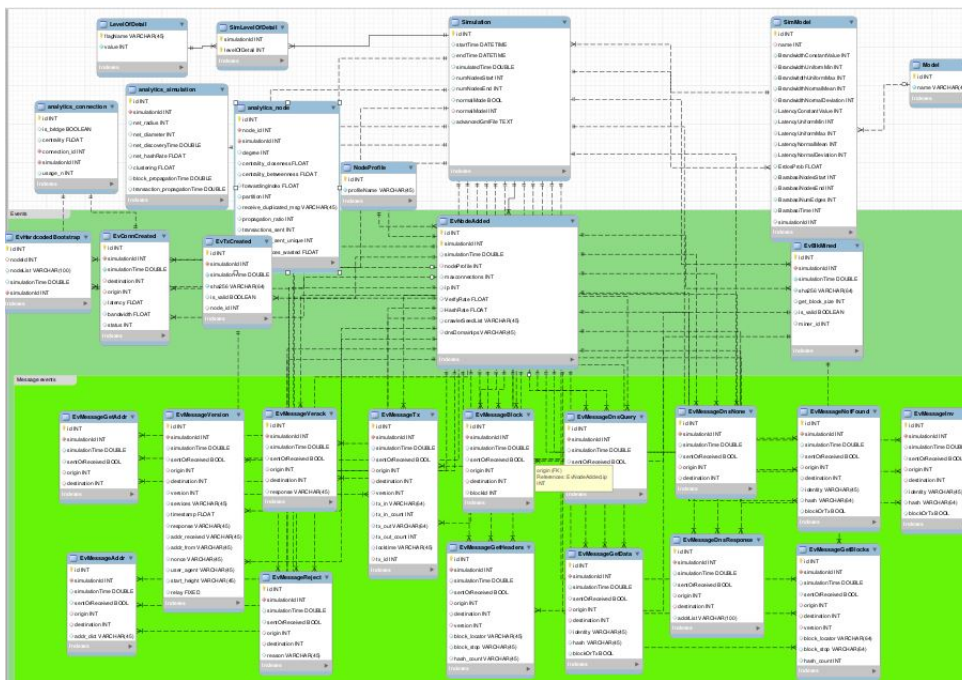


Figure 3.2: Database diagram of the Bitcoin Network Simulator.

3.2.3 Visualization and analysis module

As described before, the visualization module will interact with the database to show the simulation information to the user in a meaningful and understandable way. This module is needed to make the simulator a complete and usable solution. Without it, the user can't interact with the simulation results easily, on top the raw data has very little meaning if it is not processed and showed correctly. That's the reason why we focused this project on the visualization and analysis module development.

Up until now this module wasn't developed, therefore this project will focus on the creation of this module by extracting information from the database, processing it and showing the resulting metrics to the user. In some cases the result of the processing will be stored on the database, however in other cases the metrics need minor computation so they will be calculated on demand, avoiding increasing the size of the database unnecessarily. Taking advantage of this opportunity we will try to add extra value to the module by implementing metrics related to the security of the network.

In the following chapter we provide a detailed description of the different metrics that we will try to develop in the Bitcoin network simulator, and how they can be useful from a security point of view.

Simulation metrics

This chapter provides different metric definitions that have been identified as useful to simplify the analysis of the simulations from the Bitcoin Network Simulator. Those metrics will be calculated with the data extracted from the simulator's database. Additionally, when it applies, we explain briefly how this metrics can be used to help in security analysis tasks.

4.1 Simulation Information

For each execution of the simulator there is a set of metrics that are basic to describe and characterize the environment. In this section we describe those metrics aimed to provide information about the configuration parameters used in the simulation.

Despite this metrics may not be directly related to the security of the network, it is very important to know the environment that we are going to analyze in order to make a good assessment. For that reason we will try to carefully select the metrics to get a good description of any simulation.

4.1.1 Basic network configuration

Node number: This metric indicates the number of nodes in the simulation of the network, regardless of the functionality or profile that they may have.

Connexion number: This metric represents the total number of connections between nodes in the simulated network. Comparing this metric with the node number, we can have an idea of how strongly is the network connected.

Connexion Model: This metric describes how the connections between nodes are created to build the simulated network. There are two main methods:

- Connections file: A gml file containing all the connections between nodes is provided.
- Connections model: A random distribution is selected to create the network graph on execution time.

For the creation of the connection model using a random model we can differentiate between the following generation methods:

- Erdős-Rényi model [30].
- Barbas-Albert model [1].
- Network discovery model: This model consists on simulating the discovery process that the nodes perform in the real network.

Bandwidth model: This parameter defines the method that will determine how the bandwidth of each connection is selected. There are three configuration options:

- Constant: A fixed value is defined for the connection.
- Uniform random: A random value is generated within a range.
- Normal: A random value is generated with a mean and a normal deviation.

These settings are build on the Connection Model parameters, so it increases the flexibility of the simulator.

Latency model: This parameter is similar to the Bandwidth model metric but it is used for the latency on the network connections. Moreover there are the same configuration options: Constant, Uniform random and Normal.

4.1.2 Simulation detail level

The detail level of the simulation determines how accurate the simulation will be. That means that the block mining or the validity checks can be real or simulated. This feature is convenient to find a balance between accuracy and performance.

A set of flags have been defined to control the level of detail of both blocks and transactions. If any property is configured as dummy (or simulated) another flag is set to determine if that property is simulated as valid or invalid.

In the following list we can see the flags defined to describe the level of detail for the blocks in the simulation:

- BLOCK_DUMMY: The block is completely simulated.
- BLOCK_DUMMY_PREVIOUS: The simulation doesn't check the previous block hash.

- `BLOCK_DUMMY_NONCE`: The simulation doesn't check block's nonce.
- `BLOCK_DUMMY_MERKLE`: The simulation doesn't check the Merkle tree.
- `BLOCK_DUMMY_TIMESTAMP`: The simulation ignores the timestamp.
- `BLOCK_DUMMY_DIFFICULTY`: The simulation ignores the difficulty.
- `BLOCK_DUMMY_TRANSACTION`: The simulation ignores the transactions.
- `BLOCK_DUMMY_NUM_TX`: The simulation ignores the number of transactions in the block.
- `BLOCK_DUMMY_SIZE`: The simulation ignores the block size.

Also, there is the list of flags for the level of detail in the transactions:

- `TRANS_DUMMY_CONTENT`: The transaction content is ignored.
- `TRANS_DUMMY_SIGNATURES`: The simulation doesn't check the signatures.
- `TRANS_DUMMY_PREVIOUS`: The simulation doesn't check if the previous transaction exists.
- `TRANS_DUMMY_BITCOINS`: The simulation doesn't check if the previous transaction has enough BTCs.
- `TRANS_DUMMY_PK_ADDR`: The simulation doesn't check if the public key hashes match.
- `TRANS_DUMMY_KEYS`: The simulation doesn't generate public keys.

The level of detail can be set on each simulation to adjust it to our needs. It is a good practice to keep the level of detail at the minimum possible, to get the information we want with the best performance.

4.1.3 Simulation time-frame

Time intervals: This metric represents the amount of time to be simulated, in other words, the seconds that the network simulation is allowed to run. Note that this is not the same as the execution time, which is the real temporal length that the software is running, this one could be lower or bigger than the time interval, depending on the detail level of the simulation.

Start and Finish date: This metric shows the execution time, particularly the time when the simulation starts and the time when it stops. This is an important parameter to monitor the performance of the simulator.

4.2 Network topology

As the whole Bitcoin system is sustained by a P2P distributed network, it is very important to know how the nodes are related to each other and its topology in detail in order to detect possible threats. For that purpose, we describe the following metrics related to nodes and the corresponding connections of the network.

4.2.1 Node Metrics

Node Degree: The degree of a node is the number of connections it has with other nodes. This can give us an indication upon the vast interconnection of the network, which nodes are more widely known, and if there are any isolated nodes or nodes that rely on few sources.

Node Centrality: The centrality is a measurement to find the most important nodes in the network. The centrality can be calculated in different ways depending on what is considered to give importance to a node. In this project we are going to focus on Betweenness centrality and Closeness centrality. The closeness centrality aims to identify most central nodes in the network by calculating which node has the shortest paths compared to all the other nodes of the network. The Betweenness centrality is based on quantifying the number of times a node acts as a bridge along the shortest path between two other nodes. This measures can help to identify nodes that have more influence than others in the network, this influence can be exploited by nodes to perform specific attacks to the Bitcoins network.

Node clustering coefficient: This metric is an indicator of the degree to which nodes in a graph tend to cluster together. This means that we can identify if there are strongly connected groups of nodes that are poorly connected to other groups of nodes. A high clustering coefficient may indicate that there are groups of connected nodes that can be isolated easily from the rest of the network, as we can see in Figure 4.1. Such configuration can weaken the security of the network since different attacks can be performed.

4.2.2 Network Metrics

Connexion Centrality: This metric will be used to calculate the Betweenness centrality, as is described in Section 4.2.1, but focusing on the connections instead of the nodes. This can add some value to the simulator's module by showing which flows of data in the network are more critical.

Bridges and communities: A bridge is a connection which deletion would segment the network. Equivalently a community is a group of connected nodes that would be separated from the network if some connections were deleted, as we can see in Figure 4.2. In order to have more detail this measures are an ampliation to the node clustering coefficient. This indicators will help to detect the best distribution of communities and bridges in the network. This is an important feature as it highlights connections that may be critical to keep the whole network united.

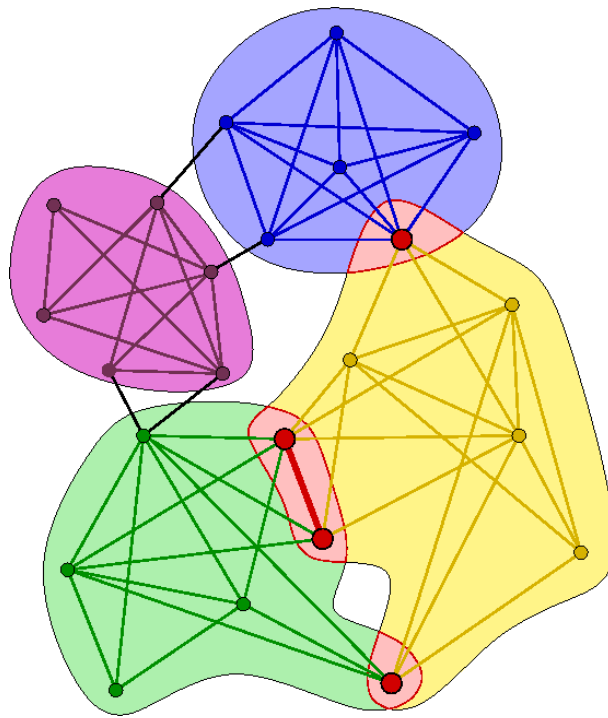


Figure 4.1: Node clustering.

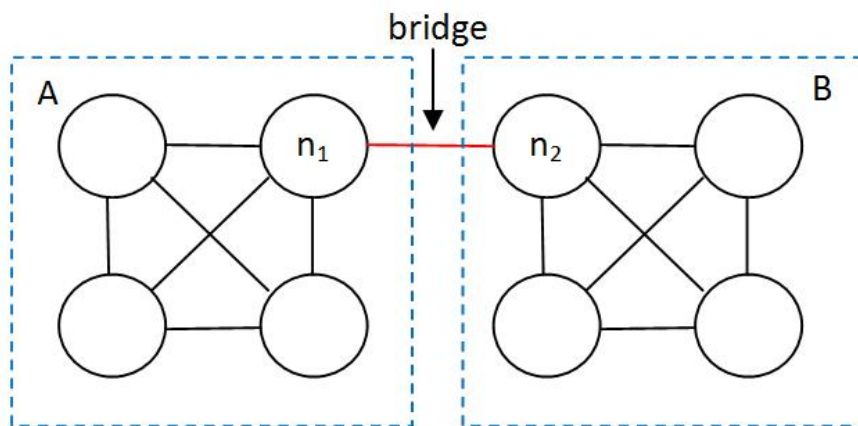


Figure 4.2: Graph bridge.

Network diameter and radius: Network diameter and radius are metrics related to eccentricity. Eccentricity is the distance of the shortest path from any node to the most farthest node. Subsequently the radius is the minimum eccentricity of all the nodes in the network and the diameter is the maximum eccentricity of the nodes in the network. These measurements can give us an idea of the relative distance between nodes, as we can observe these measurements in Figure 4.3. This is very important to know because with a greater distance there are more possibilities of a disruption in the network between them or any attacker could be placed along the path.

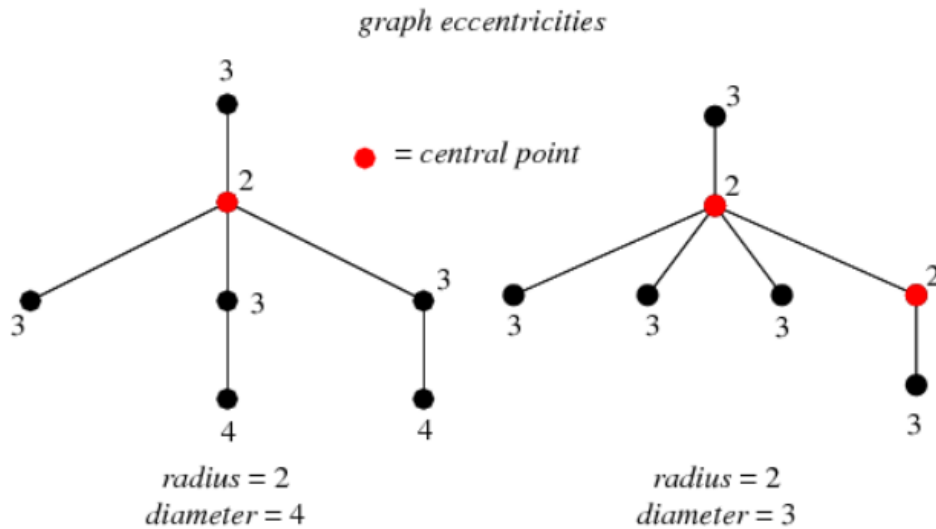


Figure 4.3: Graphs: Eccentricity, radius and diameter.

4.3 Network communications

In this section we describe the metrics derived from the communications protocol in the Bitcoin P2P network. In the previous section we saw how nodes are interconnected and which ones are more influential. Here we will see how this interconnections are used, and what kind of information these nodes transmit. This can assist in detecting strange or malicious behaviors.

4.3.1 Sent messages distribution by node

With this metric retrieve statistical data about what kind of messages are sent by each node and it shows the percentage of messages grouped by type sent to the P2P network. The different types of message that the nodes send in to the simulator are implemented in the following classes:

Message: Is a class that creates the header of a message. This object is used to create all the messages used in the communications.

MessageAddr: Contains a list of addresses of nodes known by the source node. This message is a response to a `getaddr` message.

MessageBlock: Is sent in response to a `getdata` message which requests transaction information from a block hash.

MessageDnsNone: Is sent in response to a failed `dnsquery` message, witch requests a dns resolution.

MessageDnsQuery: Contains a domain and is used to request a dns resolution from the destination.

MessageDnsResponse: Is sent in response to a `dnsquery` message, witch requests a dns resolution.

MessageDummy: This class implements a dummy message for simulations with a low detail level.

MessageGetAddr: It is a request for addresses of known nodes. It sends a request to a node asking for information about known active peers to help with finding potential nodes in the network. The response to receiving this message is to transmit one or more **addr** messages with one or more peers from a database of known active peers.

MessageGetBlocks: Is a request for an **inv** message containing the list of blocks, starting right after the last known hash in the block locator object (up to **hash_stop** or 500 blocks), whichever comes first.

MessageGetData: Is used in response to **inv**, to retrieve the content of a specific object and is usually sent after receiving an **inv** packet. After filtering known elements. It can be used to retrieve transactions, but only if they are in the memory pool or relay set.

MessageGetHeaders: Returns a header packet containing the headers of blocks starting right after the last known hash in the block locator object (up to **hash_stop** or 2000 blocks), whichever comes first.

MessageHeaders: Returns block headers in response to a **getheaders** packet.

MessageInv: Allows a node to advertise its knowledge of one or more objects. It can be received unsolicited or in reply to **getblocks**.

MessageNotFound: Is a response to a **getdata** sent if any requested data items could not be relayed, for example if the requested transaction was not in the memory pool or relay set.

MessageReject: Is sent when messages are rejected.

MessageTx: Describes a bitcoin transaction, in reply to **getdata**.

MessageVerack: Is sent in reply to **version**. This message consists of only a message header with the command string **verack**.

MessageVersion: When a node creates an outgoing connection it will immediately advertise its version. The remote node will respond with its version. No further communication is possible until both peers have exchanged their version. A **verack** packet shall be sent if the version packet was accepted.

4.3.2 Network discovery time

We can consider that the network discovery time is the time it takes for a node to be known to the rest of the network. That means that the address of this new node is in the known nodes list from the rest of the nodes. As it is very unlikely for a node to be known by all the other nodes in the network, we add a threshold to that metric. This threshold is the percentage of network nodes that need to know the node to consider it as discovered.

This metric can be useful to detect anomalies in the propagation of information about nodes. If somewhere in the network there is a node or a group of nodes blocking addresses information about other nodes, we will detect an increment of the time or even the threshold % won't be reached.

4.3.3 Reception of duplicated data by origin

As the messages are usually broadcast, it is possible to receive the same transaction or block message multiple times from different origins, but there are cases where an abnormal behavior can indicate a malicious threat. For example, if there is a block or transaction being sent multiple times by the same node it can be either trying to perform a DoS or a race attack. In any case, if a node sends a message repeatedly in a short period of time its not a normal behavior and it should be studied.

4.3.4 Most used connections

This metric shows which connections had more messages passing through them during the simulation. That means it will be able to see which connections have the highest load, allowing the user to identify critical connections and possible breakpoints.

4.3.5 Forwarding index by node

This metric shows the relation between the messages originated in the node and the messages that this node forwards, and it can be an indicator about the function of this node. For example, if the node has a high rate of forwarded messages we can think of it as a node that has high influence in the network, otherwise we can assume that the node is a miner.

4.4 Information related metrics

In this section we describe some metrics related to blocks and transactions.

4.4.1 Transactions related information

Transactions sent: This metric gives a list with detailed information about the transactions that each node has sent to the network. Here we don't consider if the transactions have been included in a block. The measurement computes all the transactions sent by a node during the simulation.

Transaction propagation time: This metric is similar to Network discovery time but is transaction centered. It gives the time needed for a Transaction to be known by a certain % of the network nodes. Monitoring the propagation of the transactions can show us if there are nodes trying to hide blocks and transactions or if the network is well connected.

Number of unique transactions sent: This metric shows the number of unique transactions broadcast by the nodes to the network. This is accomplished so that it is possible to add the transactions to the blockchain through the miners and it also can be an indicator of the network usage.

Number of total transactions broadcast: This metric shows the total number of transactions broadcasted by the nodes to the network. By comparing this value to the *Number of unique transactions sent* metric we can determine the efficiency of the network.

4.4.2 Block related information

Blocks sent: This metric gives a list with detailed information about the blocks that a node has sent to the network. Here we consider all the block messages sent regardless if they are integrated to the blockchain or not.

Block propagation time: This metric is similar to the Network discovery time but is block centered, it gives the time needed for a Block to be known by a certain % of the network nodes. Monitoring the propagation of blocks can show us if there are nodes that are trying to hide blocks or if the network is well connected.

Number of mined blocks: This metric shows the blocks that are correctly mined as a result of the simulation, regardless of the detail level of the simulation. This metric can be useful to know if the discovery of nodes is following the expected times or not.

Average Time between blocks: The time between blocks is a measurement to control the difficulty used by the network in the block mining. The average time between blocks can show us if the network is still being correctly auto-regulated or there is some kind of deviation.

Average transactions per block: The miners create blocks for the reward, but this blocks are a confirmation that the contained transactions are validated. Furthermore, the blockchain grows with each block, making it more problematic to handle. The number of transactions per block can indicate if the blockchain usage is optimal or not, on top, bad or abnormal behavior can be detected from blocks without transactions.

Mined blocks by user: This metric will show a list of mined blocks, grouped by the node that found them. With this information we can identify the most lucrative nodes and compare the results with other parameters, like the node hashrate power compared to the network's total power. Another parameter to compare to, could be the nodes that have some kind of abnormal behavior.

Network Hashrate: This metric shows the network's total mining Hashrate by adding the capacity of all the network's nodes. This measurement can be useful to study the Hashrate power needed to perform some attacks.

Spent resources: This metric shows the resources spent by all the nodes mining a block discovered by another node. This is represented by the number of hashes a node performed trying to discover a particular block, and they are counted only if this block is discovered by another node of the network.

Tools and implementation

In the following chapter provides some basic definitions about the tools used to develop the objectives proposed in this project and a description of the corresponding implementation.

5.1 Tools

5.1.1 MySQL

This project makes use of a MySQL database to store the information from the simulations, so we had to work with this database to integrate the new metrics and calculations, either to collect the information or to store the new results.

MySQL Workbench: To store the new information we needed to add new tables to the database. To modify the database model we used the MySQL Workbench, in order to manage the tables and relations in an efficient way. The details of the new tables created can be seen in [Section 5.2.1](#)

5.1.2 Python

This project is based on Python, so it is the main programming language used through all the development. To make the development of the project easier we used some additional libraries that we are going to describe in the following sections.

NetworkX: NetworkX [20] is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. This package

has several methods and algorithms to analyze graphs. We used this utilities to calculate some of the metrics.

python-louvain: Python-louvain [3] is a python library that contains the necessary methods to compute the Louvain algorithm for community detection on a NetworkX graph.

pygexf: The pygexf [17] library is the one we used to facilitate the exportation of the data in a standardized format. The GEXF is the xml like format used by the visualization tool Gephi.

This library can either be installed from python's `easy_install` or embedded in our projects as a single file package [18].

This library has the following methods that we will use for the network export:

```
# Create the file
gexf_file = gexf.Gexf("Marti Berini", "A BitcoinSimulator export")

# Create the graph
graph = gexf_file.addGraph("undirected", "static", "Btc Network")

# Add Node Attributes
at1 = graph.addNodeAttribute(title, defaultValue, type, mode, id)

# Add Edge Attributes
at2 = graph.addEdgeAttribute(title, defaultValue, type, mode, id)

# Add Nodes with attributes
n = graph.addNode(id, label)
n.addAttribute(at1, "value")

# Add Edges with attributes
e = graph.addEdge(id, source, target)
e.addAttribute(id, value)

# Print the XML
output_file = open("helloworld.gexf", "w")
gexf_file.write("output file")
```

Flask: Flask [31] is a micro-framework for Python based on Werkzeug and Jinja 2. We use it as a web server for the BTC-NS GUI, so we can easily integrate the python code from the simulator to the web.

5.1.3 Gephi

Gephi [16] is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. We can use it to visualize and analyze the simulated network and to achieve this we have to export it to a gexf file.

This software is able to run on Windows, Linux and Mac OS X and it is open-source and free.

5.2 Implementation design

5.2.1 Changes in the database

The database of the simulator is complex but very complete, as we have shown in Chapter 3. On this database structure for the Bitcoin Simulator we added three new tables to store all the processed data and avoid interfering with the raw simulation data.

The first one is `analytics_node`, it contains the post processed metrics for each node in the simulation. The second one is `analytics_connection`, it contains the post processed metrics for each connection in the simulation. The third one is `analytics_simulation`, it contains the post processed metrics related to the simulation that are not limited to a single node or connection.

In Figure 5.1, we can see the fields defined for each one of the newly created tables.

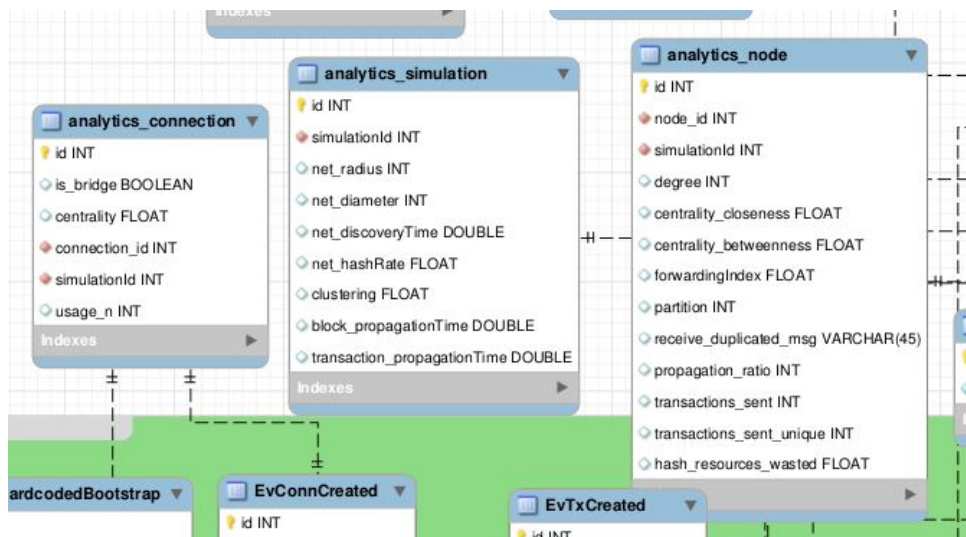


Figure 5.1: Database tables for analytics.

5.2.2 Post Processing database queries

The metrics extracted from the simulation are calculated right after the simulation takes place with the method `post_processing()`. This process can be broken down into three phases:

1. Retrieve the existing simulation information from the Database and from the live simulation.
2. Calculate the corresponding metrics with the information.
3. Store the results in the Database for future display in the web GUI.

After getting the simulation ID from the environment, we proceed to retrieve the simulation data stored into the database regarding the network infrastructure. In this case the information is

Library	Method
NetworkX	degree It gets the network's graph and returns the degree of each node.
NetworkX	betweenness centrality It gets the network's graph and returns the betweenness centrality of each node.
NetworkX	closeness centrality It gets the network's graph and returns the closeness centrality of each node.
NetworkX	edge_betweenness centrality It gets the network's graph and returns the betweenness centrality of each connection.
community	best_partition It gets the network's graph and returns a list mapping each node to a cluster.
NetworkX	average_clustering This function gets the network's graph and finds an approximate average clustering coefficient for G by repeating 1000 times the following experiment: choose a node at random, choose two of its neighbors at random and check if they are connected. The approximate coefficient is the fraction of triangles found over the number of trials.
NetworkX	radius It gets the network's graph and returns the radius of the network, which is the minimum eccentricity.
NetworkX	diameter It gets the network's graph and returns the diameter of the network, which is the maximum eccentricity.

Table 5.1: NetworkX methods used in metrics calculations.

about connections and nodes and is collected using the methods `get_sim_connections(simId)` and `get_sim_nodes(simId)`. With this information we can reconstruct a graph of the simulation with the NetworkX [20] library.

In the following section we will see how metric calculations are performed and, in some cases, simple database queries performed to complete the data needed to make the calculations.

5.2.3 Metric calculations

The Tables 5.1 and 5.2 show the methods that are used to calculate some simulation metrics during the post processing phase and they will be stored in the database. It also shows the library used to get this information, and when not specified it means that the method is developed from scratch.

5.2.4 Storing data in the database

Previously, we defined the methods to compute the information during the post processing phase. In the following paragraph we will describe the methods to store the obtained information to the corresponding database tables.

Library	Method
-	metrics_find_bridges It gets the network's graph and returns a list of connections that are bridges.
-	get_pp_dupMsg_node It gets the block and transaction messages from the database and counts the messages that are sent repeatedly from the same origin.
-	get_pp_node_forwardIndex It gets all the messages from the database and computes for each node the ratio between messages forwarded and total messages sent.
-	get_pp_node_sentTrans It gets all the Tx messages and counts how many are sent by each node.
-	get_pp_node_wastedResources It gets the blocks mined and the nodes information, computing the number of hashes wasted searching for a block that finally is found by another user.
-	get_pp_con_mostUsedCons It gets the network's connections and all the messages, and computes how many messages passed through each connection.
-	get_pp_sim_discoveryTime It gets the Addr messages from the database and computes the average time it takes for each message to stop propagating.
-	get_pp_sim_HashRate It gets the nodes Hashrate information and returns the total sum.
-	get_pp_sim_propagationTime_blocks It gets the block messages from the database and computes the average time it takes for each message to stop propagating.
-	get_pp_sim_propagationTime_transactions It gets the Tx messages from the database and computes the average time it takes for each message to stop propagating.

Table 5.2: Custom methods used in metrics calculations.

`save_metrics_simulation()`: This method stores the metric values in the `analytics_simulation` table. The correct simulation ID has to be provided to avoid errors.

`save_metrics_node()`: This method stores the metrics corresponding to one node to the `analytics_node` table. An existing node IP and a valid simulation ID has to be provided.

`save_metrics_connection()`: This method stores the metrics corresponding to one connection to the `analytics_connection` table. An existing connection ID and simulation ID have to be provided.

5.2.5 Visualization database queries

The methods listed below are used to retrieve data from the database with the objective to show it in the Web GUI. Some of these methods may be redundant or contain duplicated information, this is done on purpose to avoid performing any extra operation during the rendering of the flask HTML template and keep a good performance.

`get_sim_list`: Returns the contents of the `Simulation` table.

`get_sim_model`: Returns the contents of the table `SimModel` for a given simulation ID.

`get_model_name`: Returns the contents of the column `name` from the table `Model` for a given simulation ID.

`get_detail_lvl`: Returns the contents of the table `SimLevelOfDetail` for a given simulation ID.

`get_analytics_messages`: Returns the count of every type of message sent grouped by node.

`get_sim_nodes`: Returns the contents of the `EvNodeAdded` table for a given simulation ID.

`get_sim_connections`: Returns the contents of the table `EvConnCreated` for a given simulation ID.

`get_analytics_nodes`: Returns the contents of the table `analytics_node` for a given simulation ID.

`get_analytics_connections`: Returns the contents of the table `analytics_connection` for a given simulation ID.

`get_analytics_simulation`: Returns the contents of the table `analytics_simulation` for a given simulation ID.

`get_blocNtrans_data`: Gets the mined blocks and registered transactions from the database and returns the average time between blocks, the mined blocks by user and the average transactions per block.

`get_msg_blocks`: Returns the contents of the table `EvMessageBlock` for a given simulation ID.

`get_msg_tx`: Returns the contents of the table `EvMessageTx` for a given simulation ID.

Data Visualization

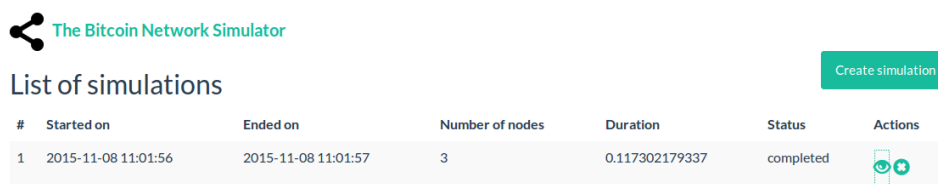
This chapter shows the resulting outputs obtained in the Bitcoin Simulator after applying the visualization module to the source code. Here we will see how this new module shows the data in an human friendly way.

6.1 Web visualization

In this section we describe the web graphical user interface and it has different pages for each task needed to use the simulator. The code for the web is made in flask [31] and is contained in the `routing.py` file. The relevant pages within the web are three: `list`, `view/id` and `download_network(id)`.

The `list` page shows a list of all the simulations stored in the database through the method `list_simulations()` in `routing.py` and the template `list.html`. To rapidly identify the simulations each line shows very basic information as we can see in Figure 6.1.

In the actions column from Figure 6.1 there are two buttons, an eye icon and a cross icon, that allows us to view the simulation details or to delete the simulation information respectively.



The screenshot shows the web interface for 'The Bitcoin Network Simulator'. At the top left is the logo and name. Below it is the title 'List of simulations' and a 'Create simulation' button. A table lists simulation data with columns for ID, Start time, End time, Number of nodes, Duration, Status, and Actions. The first row shows a simulation with ID 1, started on 2015-11-08 11:01:56, ended on 2015-11-08 11:01:57, 3 nodes, duration 0.117302179337, and status 'completed'. The actions column contains an eye icon and a cross icon.



#	Started on	Ended on	Number of nodes	Duration	Status	Actions
1	2015-11-08 11:01:56	2015-11-08 11:01:57	3	0.117302179337	completed	 

Figure 6.1: Web GUI simulation listing.

When the user clicks the eye icon the `view/id` page is loaded, it shows all the details of the selected simulation. This page contains a navigation menu to show all the metrics classified in different levels of the simulation as we can see in Figure 6.2. The method used to retrieve the information from the database is `view_simulation(id)` in `routing.py` and the template `analytics.html` is used to display all the data.

In the following subsections we describe the details of each group of metrics displayed in the web.

6.1.1 Summary

The summary tab is shown by default when the user opens the details of a simulation in a table format and the information shown is generic for all the simulation, as we can see in the Figure 6.2.

Apart from the ones we can see on the previous Figure, the additional metrics displayed in the table are:

- Average Clustering
- Network Radius
- Network Diameter
- Detail Level
- Network discovery time (AVG)
- Blocks mined
- Time between blocks (AVG)
- Tx per Block (AVG)
- Network HashRate
- Network wasted HashRate
- Block propagation time
- Transaction propagation time

6.1.2 Node metrics

The Node metrics tab displays metrics results grouped by node and the representation is made with a table in which, on each entry we can find a node of the simulation represented by its identifier as we can see in Figure 6.3. In the future this identifier will be substituted by the IP address of the node to simplify the identification.

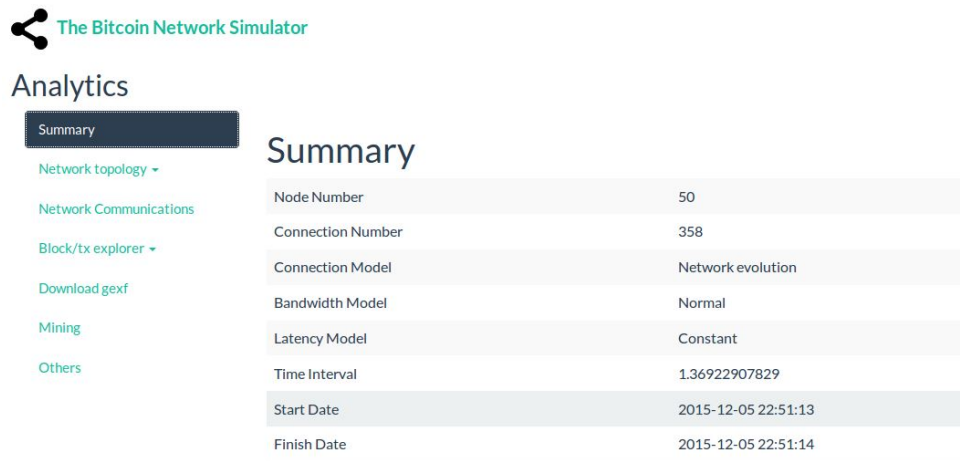


Figure 6.2: Web GUI analytics Summary tab.

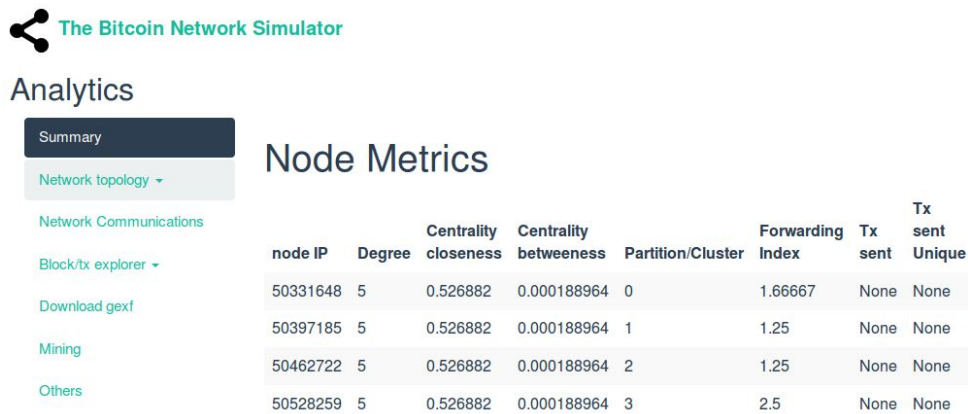


Figure 6.3: Web GUI analytics Node metrics tab.

6.1.3 Network metrics

The Network metrics tab displays metrics results grouped by connection and the representation is made with a table in which, on each entry we can find a connection of the simulation represented by the two nodes it connects as we can see in Figure 6.4.

6.1.4 Network communications

The Network Communications tab displays two types of metrics results, grouped by connection and grouped by node respectively, the corresponding representation is made with a table for each one. For the nodes metrics we can find a table which in each entry a node is represented by its identifier as we can see in Figure 6.5. The columns display the number of messages sent by that node and the distribution by type of message.

For the connections metrics we can find a table which in each entry a connection is represented by the two nodes that it connects. The contents of the table display the number of times a message about a particular transaction is sent from node A to node B. The columns of the table are named as follows:

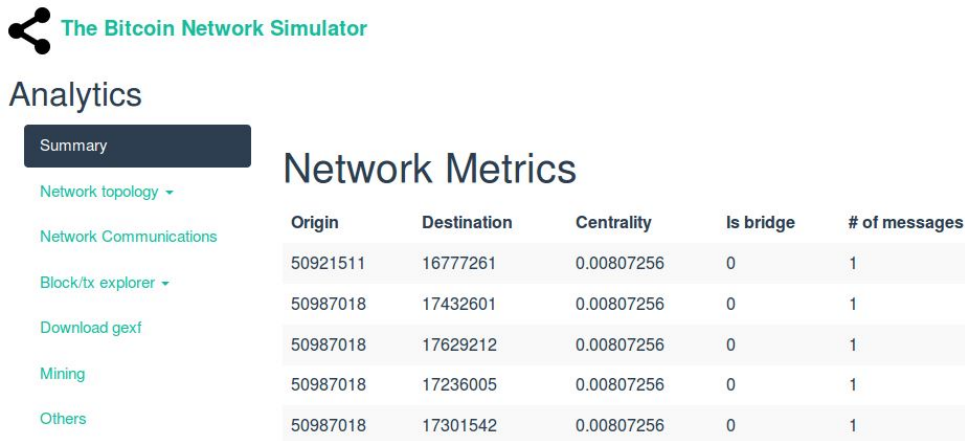


Figure 6.4: Web GUI analytics Network metrics tab.

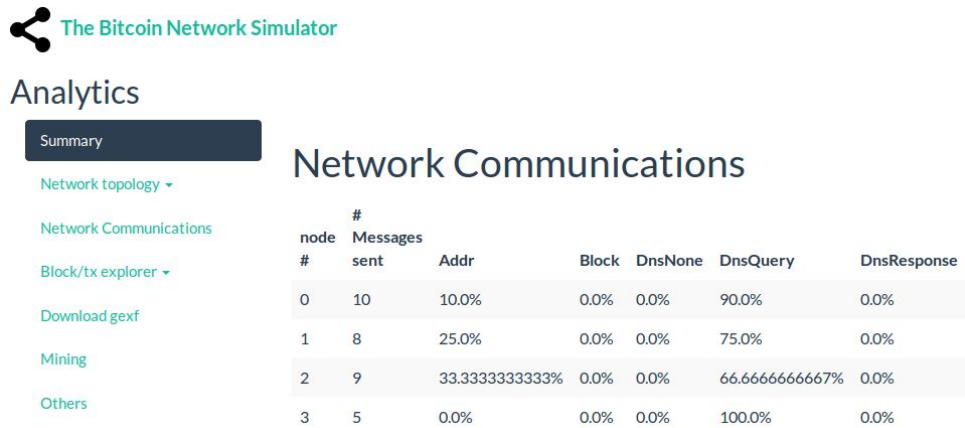


Figure 6.5: Web GUI analytics Network Communications tab.

- Origin
- Destination
- Transaction Id
- # msg sent

6.1.5 Block and Tx metrics

There are two tabs called Block metrics and Tx metrics, that contain a list, and those lists contain all the information in the database regarding every block and transaction respectively. That is, a table with the contents of the database tables `EvBlkMined` and `EvTxCreated`.

6.1.6 Download gexf

The Download gexf tab allows the user to export the simulation's network in gexf format as we can see in Figure 6.6. The export method is developed using the `pygexf` library for Python and

it creates a gexf file on the fly with the simulation data found in the database. The file can be loaded on the Gephi software for independent analysis.

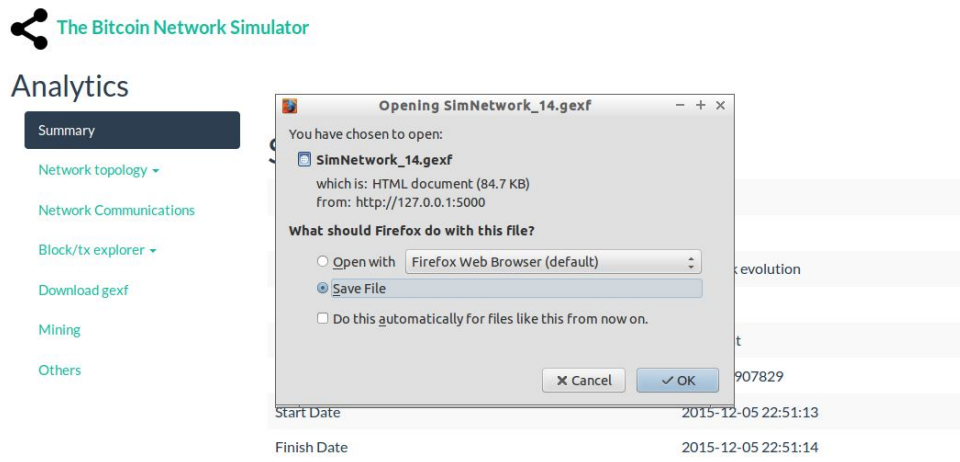


Figure 6.6: Web GUI analytics Download gexf tab.

In the following example we can see the format of a gexf file:

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gexf.net</creator>
    <description>A hello world! file</description>
  </meta>
  <graph mode="static" defaultedgetype="directed">
    <nodes>
      <node id="0" label="Hello" />
      <node id="1" label="Word" />
    </nodes>
    <edges>
      <edge id="0" source="0" target="1" />
    </edges>
  </graph>
</gexf>
```

6.1.7 Mining

The Mining tab allows the user to see a ranking, in a tabular format of the number of blocks mined by node (see Figure 6.7). This allows the user to easily detect the most successful nodes to study how they achieve it, either through a good strategy or by performing a series of attacks.



Figure 6.7: Web GUI Mining tab.

6.2 Gephi visualization

To visualize the information with Gephi we use the network export from a finished simulation. This network export is a gexf file as described in Section 6.1.6. Once we obtain the gexf file we can open it with the Gephi software.

The first thing that we see when we open the file is an overview tab. This tab contains a graph with all the nodes and connections in the network as we can see in Figure 6.8, furthermore there are several tools to interact with the graph.

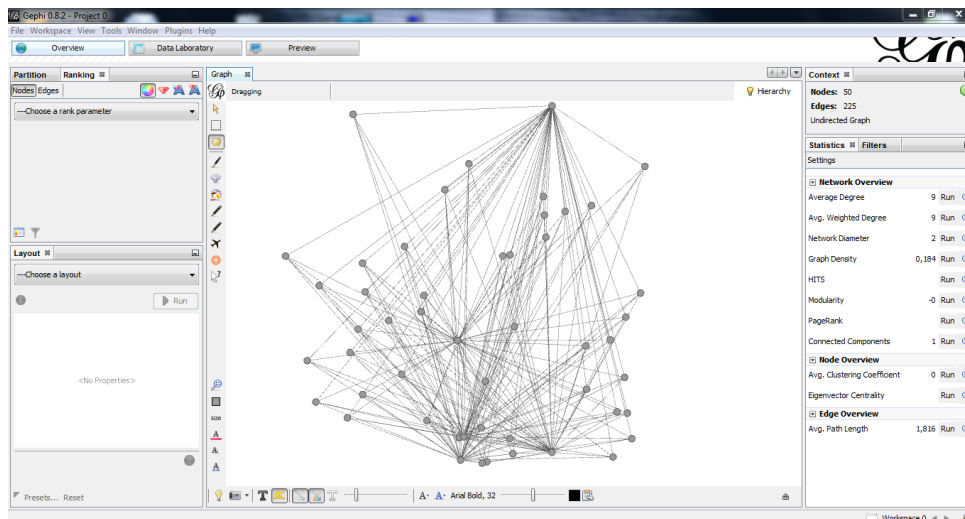


Figure 6.8: Gephi overview.

If we navigate to the Data Laboratory tab, we can find all the information computed during the post processing phase and stored in the nodes (see Figure 6.9) and edges (see Figure 6.10) of the network. For example, we could find the information about the hashrate, the degree and the bandwidth. All this data can be used by Gephi to enhance the graphical representation of

the graph. As an example, in the Figure 6.11 we can see a graph which represents with colors the different clusters of nodes and with the size of each node the weight it has in the network.

Nodes	Id	Label	Profile	Hash Rate	Max C...	Verify Rate	Degree	Weigh...	Ecten...	Closenes...	Between...	Authority	Hub	Modul...	PageRank	Comp...	Cluste...	Num...	Egevec...
0	50331648	0	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	0	0,012	0	0	0	0,333
1	50397185	1	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	4	0,012	0	0	0	0,333
2	50462722	2	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	2	0,012	0	0	0	0,333
3	50528259	3	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	1	0,012	0	0	0	0,333
4	17039364	4	9	0.0	8	10.0	45	45	2	1,082	198	0,092	0,092	0	0,093	0	0	0	1
5	50659333	5	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	3	0,012	0	0	0	0,333
6	50724870	6	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	0	0,012	0	0	0	0,333
7	50790407	7	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	4	0,012	0	0	0	0,333
8	50859444	8	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	2	0,012	0	0	0	0,333
9	17367049	9	9	0.0	8	10.0	45	45	2	1,082	198	0,092	0,092	1	0,093	0	0	0	1
10	50987018	10	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	1	0,012	0	0	0	0,333
11	51052555	11	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	3	0,012	0	0	0	0,333
12	51118992	12	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	0	0,012	0	0	0	0,333
13	51183629	13	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	4	0,012	0	0	0	0,333
14	51249166	14	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	2	0,012	0	0	0	0,333
15	50331663	15	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	1	0,012	0	0	0	0,333
16	50397200	16	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	3	0,012	0	0	0	0,333
17	50462737	17	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	0	0,012	0	0	0	0,333
18	50528274	18	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	4	0,012	0	0	0	0,333
19	50593811	19	0	0.0	8	10.0	5	5	2	1,898	0,222	0,012	0,012	2	0,012	0	0	0	0,333

Figure 6.9: Gephi node data.

Source	Target	Type	Id	Label	Weight	Latency	Bandwidth
50331648	17629227	Undirected	182			13.0	0.0114536
50331648	16777261	Undirected	185			13.0	0.286475
50397185	17367049	Undirected	116			13.0	0.856014
50462722	17039364	Undirected	28			13.0	1.47579
50528259	17629227	Undirected	162			13.0	0.33121
17039364	50331648	Undirected	320			13.0	2.82917
17039364	50397185	Undirected	241			13.0	0.018648
17039364	50528259	Undirected	268			13.0	0.520693
17039364	50659333	Undirected	271			13.0	2.5557
17039364	50724870	Undirected	249			13.0	3.38614
17039364	50790407	Undirected	360			13.0	4.20075
17039364	50987018	Undirected	344			13.0	2.46692
17039364	51118992	Undirected	304			13.0	2.16547
17039364	51249166	Undirected	297			13.0	0.220528
17039364	50331663	Undirected	313			13.0	0.37008
17039364	50397200	Undirected	334			13.0	0.794952
17039364	50462737	Undirected	342			13.0	0.841577
17039364	50528274	Undirected	341			13.0	3.53704
17039364	50659348	Undirected	292			13.0	2.67886
17039364	50790422	Undirected	246			13.0	3.03585
17039364	50855959	Undirected	276			13.0	2.01301
17039364	50921496	Undirected	367			13.0	2.31311
17039364	50987033	Undirected	330			13.0	1.11707
17039364	51118107	Undirected	307			13.0	2.731
17039364	51183644	Undirected	329			13.0	4.26744

Figure 6.10: Gephi edge data.

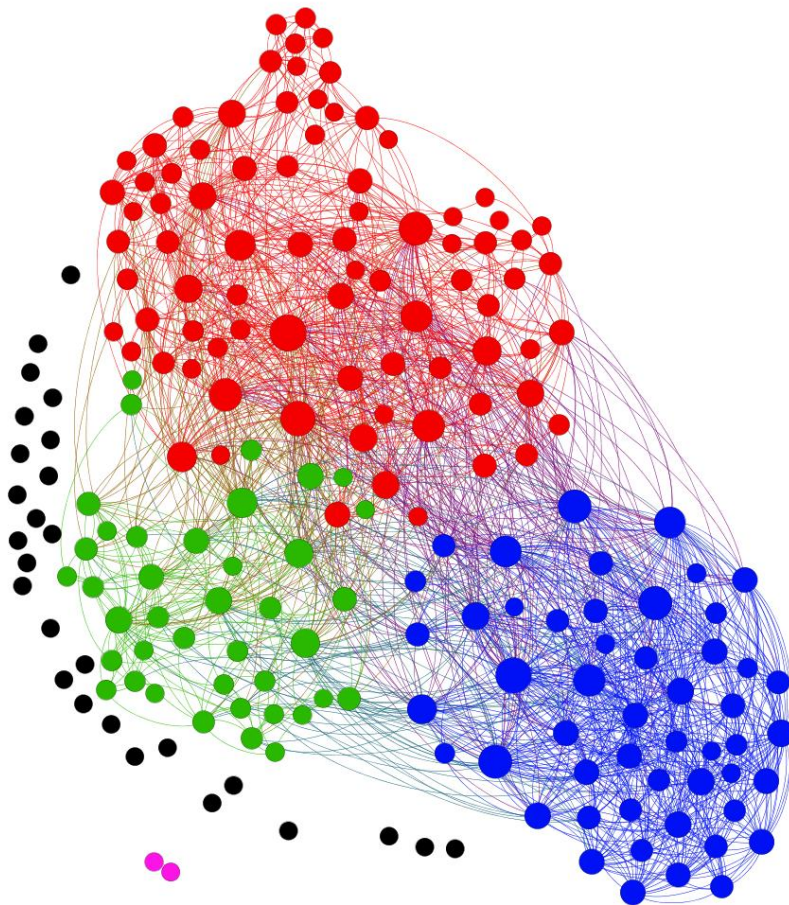


Figure 6.11: Gephi enhanced network graph.

CHAPTER 7

Conclusion

During this project, we have successfully implemented the visualization module for the Bitcoin Network Simulator. Hereby the simulator has the added feature to compute and display several metrics and useful information to the user. There are still several points of improvement as well as new features that can be added, however the Bitcoin Network Simulator is now a complete product and it can be used by the general public.

This project has been a great opportunity to learn about the Bitcoin and its underlying network, as well as a good practice on software engineering. Consequently after implementing the Bitcoin Network Simulator module we tried to summarize all the newly acquired knowledge into this document, aiming to make easier the basic understanding of the Bitcoin system and to facilitate future studies and developments.

As the Bitcoin have a notable impact in society and currently are a topic of interest, we hope that this project helps in the understanding of the Bitcoin network and incentives new studies on the security and use of cryptocurrencies.

7.1 Future work

During the development of this project, we encountered some problems and found extra features that would be interesting to develop in the future:

1. The gexf library to create the network export only allows to add attributes as strings, when one tries to use integers or floats it generates an error. So temporally all the numerical metrics on the gexf export are set as strings.

2. The rendering of the simulation information using flask is slow. Apparently this is caused by too much information inserted to create and show the different tables. A possible solution would be to optimize and reduce the information provided to the html template or the segmentation of the analytics html template in several files loaded on demand.
3. All the metrics are based on static data stored on the database. A useful feature would be to add metrics for dynamic scenarios, using data that changes over time.
4. Another feature could be to add a custom query constructor in the GUI. This would allow users to create their own query in the database in a controlled and safe way.
5. Also a block browser could be added to enable the users to navigate through the blockchain freely, simplifying the navigation through nodes, transactions or Bitcoin flows in a similar way to the online block explorers (for example see [23]).
6. Finally, it would be useful for the user to add a functionality that would only show metrics applicable to the detail level of the simulation set. This would avoid to show metrics that don't contain any results keeping the GUI clean and simple.

Bibliography

- [1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- [2] Andreas M Antonopoulos. *Mastering Bitcoin unlocking digital cryptocurrencies*. ” O’Reilly Media, Inc.”, 2014.
- [3] Thomas Aynaud. Python-louvain. <https://pypi.python.org/pypi/python-louvain>. Accessed November 18, 2015.
- [4] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better how to make bitcoin a better currency. In *Financial cryptography and data security*, pages 399–414. Springer, 2012.
- [5] bitcoin.it. Bitcoin wiki. https://en.bitcoin.it/wiki/Main_Page. Accessed October 18, 2015.
- [6] bitcoin.org. Bitcoin dinero p2p de codigo abierto. <https://bitcoin.org/es/>. Accessed October 18, 2015.
- [7] Bitcoin.org. Bitcoin core. <https://bitcoin.org/en/download>. Accessed October 18, 2015.
- [8] Blockchain.org. Blockchain dashboard. <https://blockchain.info/es/stats>. Accessed October 8, 2015.
- [9] Jerry Brito and Andrea Castillo. *Bitcoin: A primer for policymakers*. Mercatus Center at George Mason University, 2013.
- [10] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Proceedings of the IEEE Internation Conference on Peer-to-Peer Computing (P2P)*, Trento, Italy, 2013., 2013.
- [11] Joan Antoni Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin p2p network. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, volume 8438 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin Heidelberg, 2014.
- [12] ebfll. simbit (alpha). <https://github.com/ebfull/simbit>. Accessed October 16, 2015.
- [13] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.
- [14] Mobey Forum. Mobeydays 2015. <http://www.mobeyforum.org/events/mobey-day-2015/>. Accessed October 8, 2015.
- [15] Bitcoin Foundation. Bitcoin foundation grant. <https://bitcoinfoundation.org/bitcoin/q1-2014-grant-recipient-development-of-an-open-source-bitcoin-network-simulator/>. Accessed October 8, 2015.
- [16] Gephi.org. Gephi. <http://gephi.github.io/>. Accessed October 24, 2015.
- [17] Paul Girard. pygexf. <https://github.com/paulgirard/pygexf>. Accessed October 24, 2015.

- [18] Paul Girard. Pygexf instalation. <https://pythonhosted.org/pygexf/users.html>. Accessed October 24, 2015.
- [19] The Guardian. A history of bitcoin hacks. <http://www.theguardian.com/technology/2014/mar/18/history-of-bitcoin-hacks-alternative-currency>. Accessed October 16, 2015.
- [20] A Hagberg, Daniel A Schult, and Pieter J Swart. Networkx. <http://networkx.github.io/index.html>, 2013.
- [21] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. <http://cs-people.bu.edu/heilman/eclipse/>. Accessed November 8, 2015.
- [22] Business Insider. A critical bug left a bitcoin wallet app's users completely vulnerable to cybertheft. <http://uk.businessinsider.com/bitcoin-app-blockchain-issues-critical-update-random-number-bug-security-breakdown-android-2015-6>. Accessed October 8, 2015.
- [23] insight. Bitcoin explorer. <https://blockexplorer.com/>. Accessed December 22, 2015.
- [24] jgarzik. bitcoinlib. <https://github.com/jgarzik/python-bitcoinlib>. Accessed October 21, 2015.
- [25] Victor Mora Afonso. Design and implementation of a bitcoin simulator. <http://openaccess.uoc.edu/webapps/o2/handle/10609/37001>. Accessed October 05, 2015.
- [26] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2008. Accessed October 8, 2015.
- [27] Ivan Pustogarov. Eavesdropping attack. <http://www.coindesk.com/eavesdropping-attack-can-unmask-60-bitcoin-clients/>. Accessed December 28, 2015.
- [28] Zulfikar Ramzan. Bitcoin: What is it? <https://www.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/bitcoin/v/bitcoin-what-is-it>, 2015.
- [29] rbrune. Bitcoin network simulator. <https://github.com/rbrune/btcsim>. Accessed October 16, 2015.
- [30] A Renyi and P Erdos. On random graphs. *Publicationes Mathematicae*, 6(290-297):5, 1959.
- [31] Armin Ronacher. Flask (a python microframework). <http://flask.pocoo.org/>. Accessed November 9, 2015.
- [32] SimPy team. Simpy. <http://simpy.readthedocs.org/en/latest/index.html>. Accessed October 21, 2015.
- [33] The Telegraph. Bitcoin exchange mtgox faced 150,000 hack attacks every second: <http://www.telegraph.co.uk/finance/currency/10686698/Bitcoin-exchange-MtGox-faced-150000-hack-attacks-every-second.html>. Accessed October 8, 2015.
- [34] vivictormora. Pynode. <https://github.com/vivictormora/pynode>. Accessed October 21, 2015.
- [35] Bitcoin Wiki. Bitcoin weaknesses. http://en.bitcoinwiki.org/Bitcoin_weaknesses. Accessed October 8, 2015.
- [36] Wikipedia. History of bitcoin. https://en.wikipedia.org/wiki/History_of_Bitcoin. Accessed October 18, 2015.

Marti Berini Sarrias
Barcelona, 2015