Bitcoin Throughput Analysis

Presented by:

Tal Tzafrir – 200944007

Itzhak Solomon – 201522315

Supervisor:

Alexander Manuskin

NSSL Lab

Winter 2018/9

# Table of Contents

[Table of Contents 2](#_Toc5660880)

[Introduction 4](#_Toc5660881)

[Main objectives 5](#_Toc5660882)

[Side objectives 5](#_Toc5660883)

[Existing solutions 5](#_Toc5660884)

[Project tools 6](#_Toc5660885)

[Programming languages 6](#_Toc5660886)

[Primary project stages 6](#_Toc5660887)

[Blockchain 7](#_Toc5660888)

[Bitcoin 8](#_Toc5660889)

[Block 8](#_Toc5660890)

[UTXO 9](#_Toc5660891)

[Bitcoin Core 10](#_Toc5660892)

[System overview 11](#_Toc5660893)

[Parameters of interest 12](#_Toc5660894)

[UTXO set 12](#_Toc5660895)

[Block size 12](#_Toc5660896)

[Network topology 12](#_Toc5660897)

[Script – make**\_**setup**\_**test 13](#_Toc5660898)

[Creating data directories 13](#_Toc5660899)

[Setting up AWS 14](#_Toc5660900)

[Copying data to instances 15](#_Toc5660901)

[Creating configuration file 16](#_Toc5660902)

[Mining and dissemination timing 17](#_Toc5660903)

[Script – create**\_**starting**\_**blockchain 19](#_Toc5660904)

[Script – run**\_**multiple**\_**tests 20](#_Toc5660905)

[Setup and how to run 21](#_Toc5660906)

[Setup 21](#_Toc5660907)

[How to run 23](#_Toc5660908)

[Block dissemination time graphs 27](#_Toc5660909)

[Block size varied 27](#_Toc5660910)

[UTXO set size varied 28](#_Toc5660911)

[Dissemination process of different topologies 29](#_Toc5660912)

[Future improvements 31](#_Toc5660913)

[Conclusions 32](#_Toc5660914)

[Bibliography 33](#_Toc5660915)

# Introduction

Bitcoin has been a growing phenomenon ever since it first appeared in 2009. One of the main problems with bitcoin, is how to optimize its usage and how to receive greater throughput in a more efficient manner. We have long been interested in the subject but never really understood much about it. There are a lot of disagreements on several subjects regarding the bitcoin Blockchain, and we set out to explore the subject to try and find ways to improve the manner in which bitcoin works today.

# Main objectives

To build a platform that will enable its user to change parameters of the bitcoin blockchain and measure their effect on the throughput of the system, as represented by the amount of time it takes for a single miner to send a block to its peers and receive confirmation of the verification of the block.

# Side objectives

* Learn more about bitcoin.
* Learn about blockchain technologies in general and in bitcoin specifically.
* Learn about and utilize AWS.
* Improve our skills in python development.

# Existing solutions

None to our knowledge, but there is academic material on the subject.

# Project tools

* “Bitcoin core” bitcoin client.
* AWS – EC2 services.

# Programming languages

* Python 3.7.
* C++.

# Primary project stages

1. Research and learn about bitcoin and blockchain.
2. Download and experiment with bitcoin core client.
3. Writing python script to set up a number of bitcoin core nodes locally, and allow changing of blockchain parameters.
4. Modifying the python script to set up nodes in remote instances in AWS.
5. Change parameters of blockchain and time block dissemination.
6. Analyze results.

# Blockchain

In order to understand what bitcoin is and how it works, we need an understanding of blockchain technology in general, and specifically some of its core concepts.

A blockchain is a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree).

By design, a blockchain is resistant to modification of the data. It is "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way". For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks, which requires consensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been claimed with a blockchain.

Blockchain was invented by a person using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin. The identity of Satoshi Nakamoto is unknown. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server.

# Bitcoin

Bitcoin (₿) is a cryptocurrency, a form of electronic cash. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries.

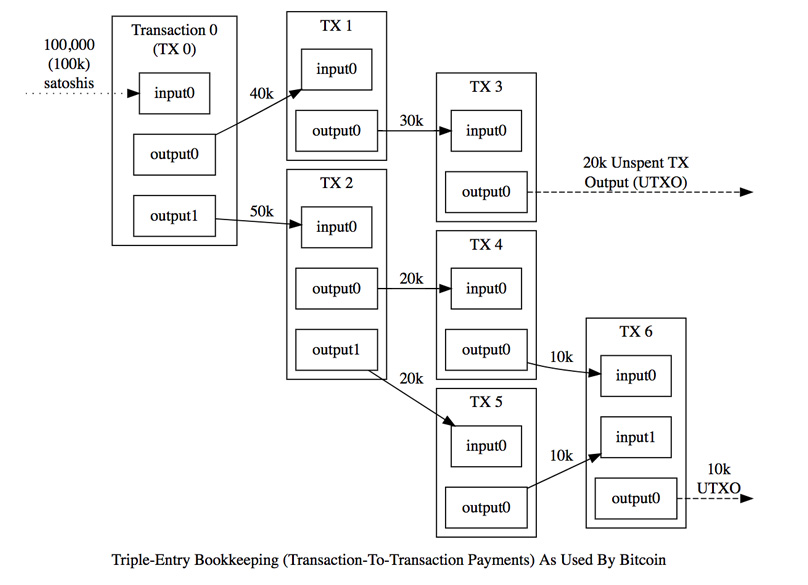
Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain. Bitcoin was invented by an unknown person or group of people using the name Satoshi Nakamoto and released as open-source software in 2009. Bitcoins are created as a reward for a process known as mining. They can be exchanged for other currencies, products, and services. Research produced by the University of Cambridge estimates that in 2017, there were 2.9 to 5.8 million unique users using a cryptocurrency wallet, most of them using bitcoin.

# Block

Transaction data is permanently recorded in files called blocks. They can be thought of as the individual pages of a city recorder's recordbook (where changes to title to real estate are recorded) or a stock transaction ledger. Blocks are organized into a linear sequence over time (also known as the block chain). New transactions are constantly being processed by miners into new blocks which are added to the end of the chain. As blocks are buried deeper and deeper into the blockchain they become harder and harder to change or remove, this gives rise of bitcoin's Irreversible Transactions.

# UTXO

The fundamental building block of a bitcoin transaction is a transaction output. Transaction outputs are indivisible chunks of bitcoin currency, recorded on the blockchain, and recognized as valid by the entire network. Bitcoin full nodes track all available and spendable outputs, known as unspent transaction outputs, or UTXO. The collection of all UTXOs is known as the UTXO set. The UTXO set grows as new UTXO is created and shrinks when UTXO is consumed. Every transaction represents a change (state transition) in the UTXO set.



# Bitcoin Core

Bitcoin Core is free and open-source software that serves as a bitcoin node (the set of which form the bitcoin network) and provides a bitcoin wallet which fully verifies payments. It is considered to be bitcoin's reference implementation and is the most used implementation by a large margin. Initially, the software was published by Satoshi Nakamoto under the name "Bitcoin", and later renamed to "Bitcoin Core" to distinguish it from the network. For this reason, it is also known as the Satoshi client. We ran or bitcoin network using regtest mode. For situations where interaction with random peers and blocks is unnecessary or unwanted, Bitcoin Core’s regression test mode (regtest mode) lets you instantly create a brand-new private block chain with the same basic rules as testnet—but one major difference: you choose when to create new blocks, so you have complete control over the environment.

# System overview

The main parameters who’s effect on block dissemination time that we are testing in our project are (and which can be passed as arguments to the script):

* How much of the UTXO set will be stored in the cache.
* Block size.
* Network topology.

All of which can be tested on different amounts of nodes (can be passed as parameter).

In our project we used one main python script named “make\_setup\_test”, which uses one other auxiliary script named “create\_starting\_blockchain”. For the timing of block dissemination we use two more scripts named “block” and “server”. We will go into detail on each one of them. Running the system will be explained in the “setup and how to run” section.

# Parameters of interest

## UTXO set

The first parameter that we wanted to check is the size of the UTXO set that will be saved in the cache. We controlled this parameter by using the -dbcache flag when running a bitcoin core node. This flag determines how much of the memory will be allocated for the use of the UTXO set. Along with this flag, when running the main script, one of the parameters passed is the UTXO set size in percentage, out of the fixed 4 MB (actually 1.8). For example, passing 50 as the parameter will set the UTXO set size to 0.9 MB.

## Block size

The second parameter we checked was the size of each block in the block chain. This parameter was controlled by setting the block size in the bitcoin core source code to 16 times the original block size (original is 1MB), and then controlling the actual block size in our code when constructing the blockchain. We control it by receiving the desired block size as an argument for the script (of course up to 16MB).

## Network topology

The third parameter that we checked was the topology of the network, or how all of the nodes are connected to each other. This too is decided by passing an argument to the main script, and we can choose between three choices:

* Fully connected graph (clique).
* Statically random connections: nodes are connected randomly to 4 other nodes but using the same seed every time (meaning we will get the same topology for each run of the same parameters).
* Dynamically random connections: nodes are connected randomly to 4 other nodes.

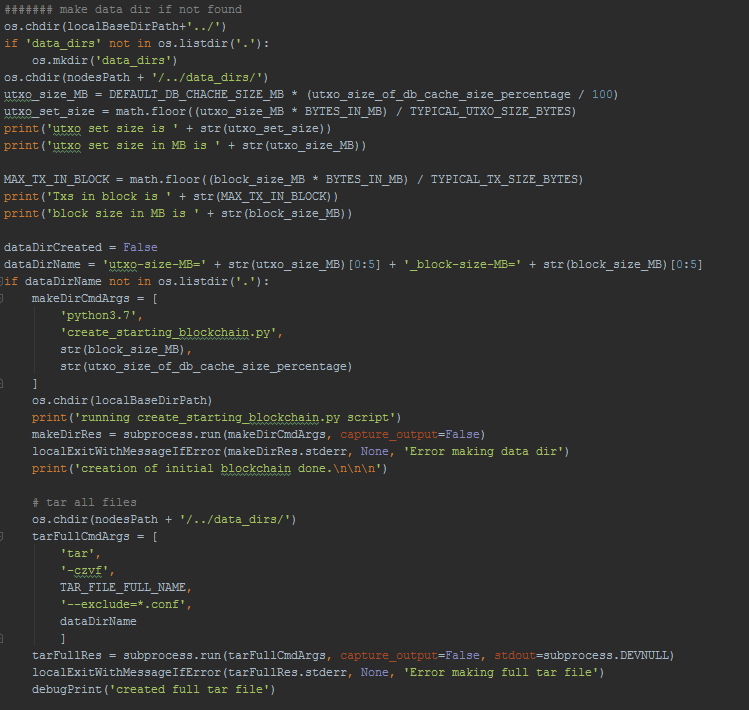
In both random options we have taken measures to make sure that the graph is connected (connected each node to its numbered successor and the 3 other nodes are chosen randomly).

# Script – make**\_**setup**\_**test

This is our main script and is in charge of setting up and running our system and testing the block dissemination time.

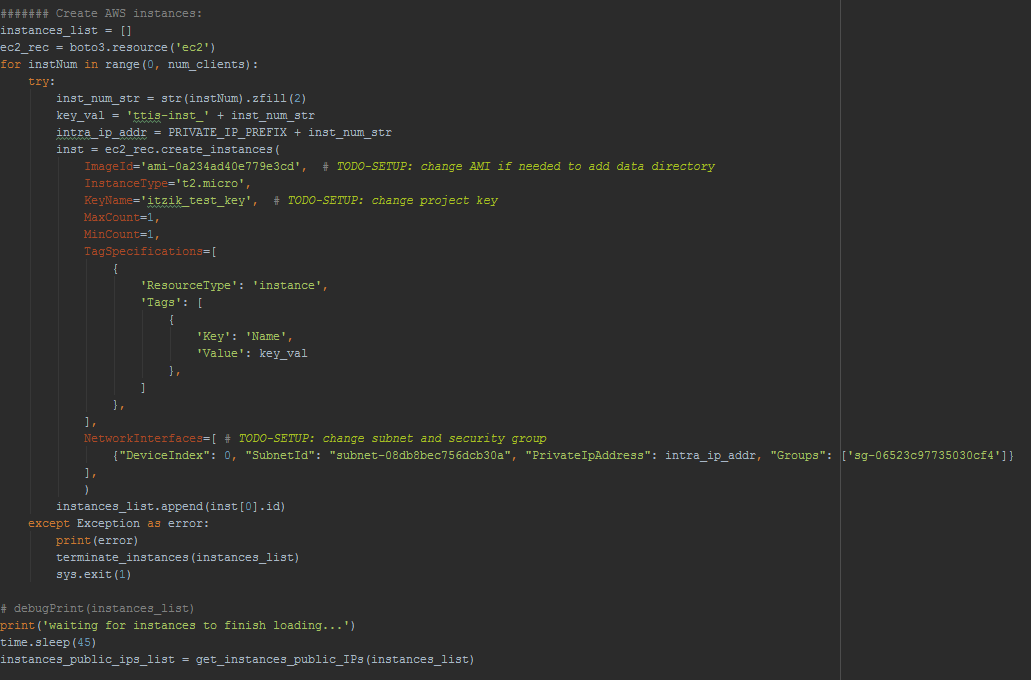
## Creating data directories

The first thing that the script does is create a data directory (the main working directory for each node), in case a suitable directory isn’t available on the AMI. It does so by calling the script “create\_starting\_blockchain” which we will cover later. The data is then zipped and ready to be sent to remote AWS instances.



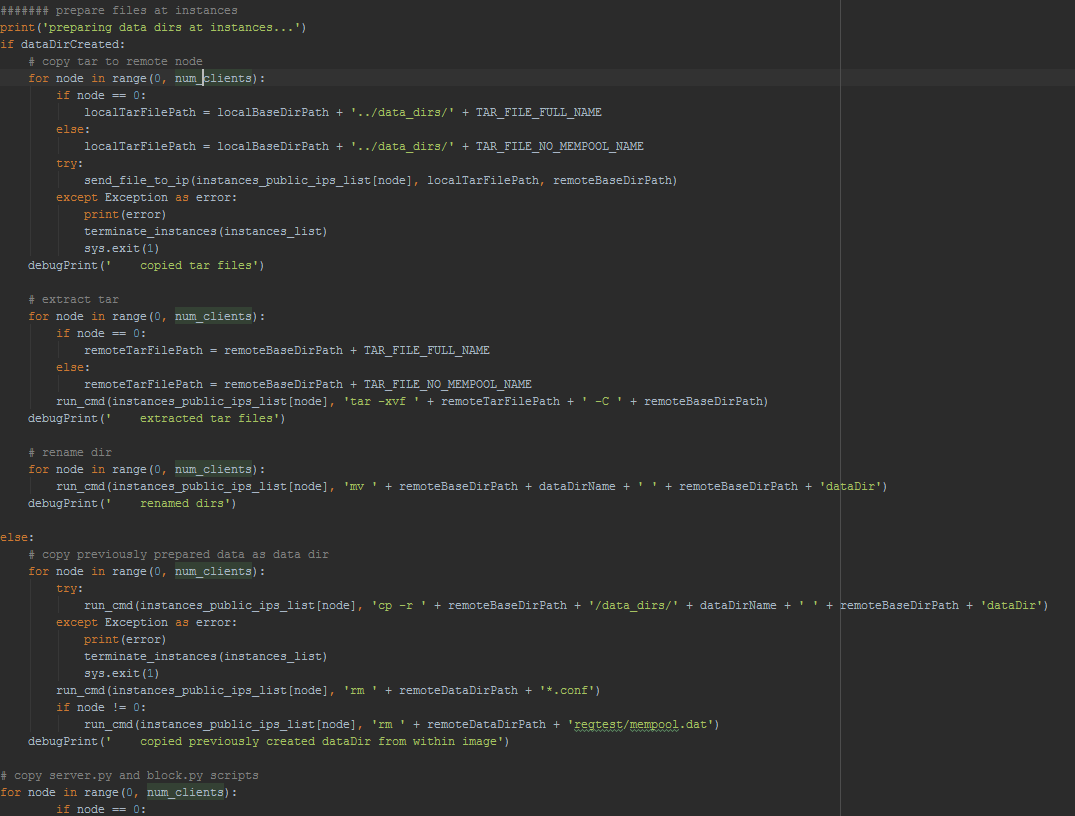
## Setting up AWS

Next, the script sets up and runs a number of AWS instances. The instances are configured to use a pre-built AMI as an installation image so that everything needed by the script will already be set up. The user enters his credentials in specific configuration files so as to run the instances on their AWS account (how to do so will be covered in the “how to run” section).



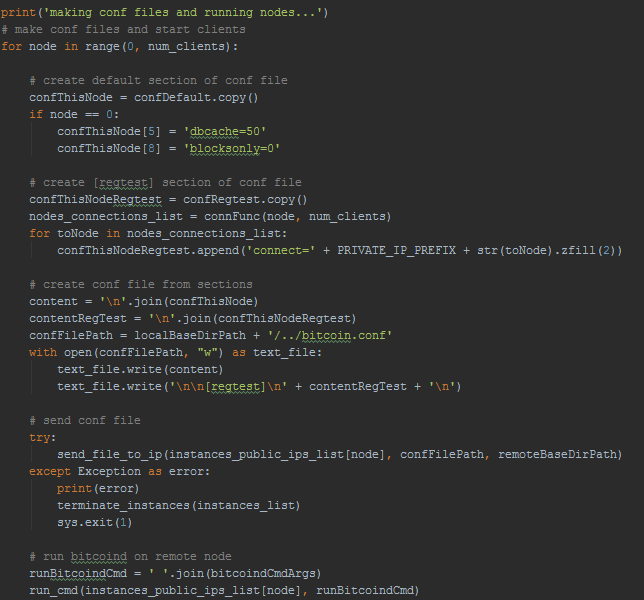
## Copying data to instances

After creating the instances, the script copies to them the data directories it created before. The data directories are then unzipped on the instances and are ready for use. In addition, the two scripts “block” and “server” are also copied to the instances (more on those scripts later on).  
One difference between the miner node and the rest of the nodes is that we copy mempool.dat file only to the miner node. This file contains transactions that have yet to be included in a block (transactions that the miner will include when generating a block). We want the other nodes to validate the block from scratch so excluding mempool file prevents the nodes from loading the transactions and preprocess them.



## Creating configuration file

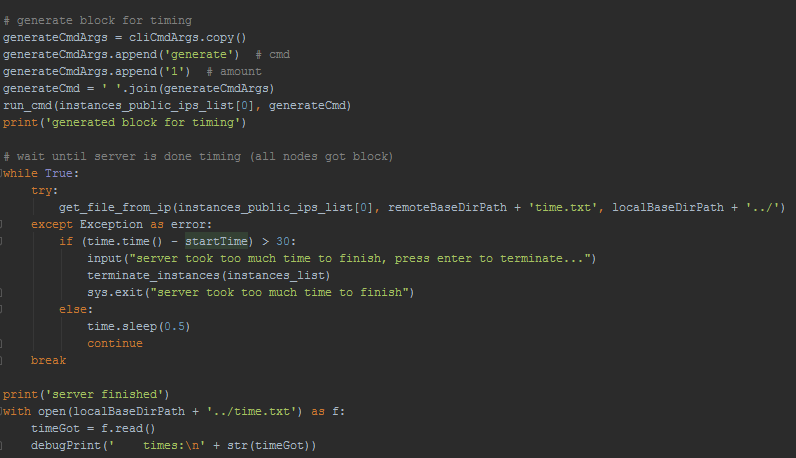
Now, after the instances are set up with all the data they need, we need to create a “.conf” file that will be used when running the bitcoin core client. The “.conf” file contains certain parameters that the bitcoin server will use when starting up, some of which are pre-determined by us as they should be the same for every run, while others are determined by the arguments passed to the script (as explained previously). one exception in the “blocksonly” option, which is different for the node that mines the block and times the dissemination than it is for the other nodes. The reason for this is similar to the exclusion of mempool.dat file, “blocksonly” option prevents the nodes for getting information about transactions and UTXOs from other nodes and only listen to new blocks. After the “.conf” is created, it is copied to the AWS instances and we run the bitcoin core client.



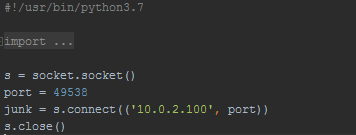
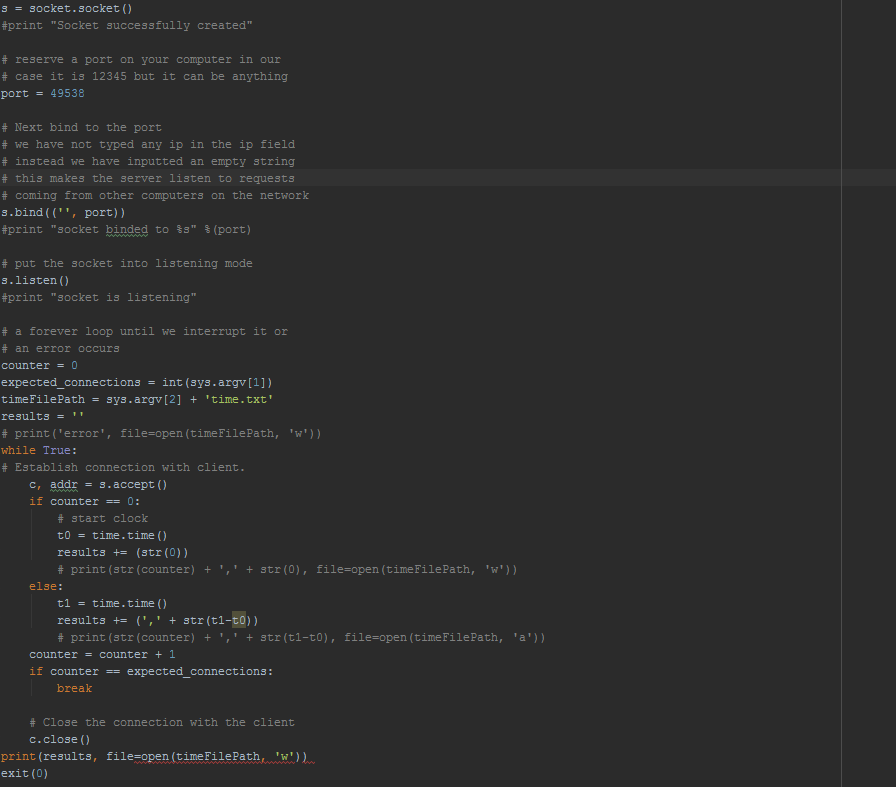
## Mining and dissemination timing

In this final stage, tell our mining node to mine a new block, send it to its peers and measure the time it takes the block to reach all of them. This is done with the help of two scripts: “block.py” and “server.py”. The mining node starts timing and sends the block to all other nodes. The server script runs on the mining node and listens on a certain socket and waits. Meanwhile, when the other nodes receive the block, they run the block script, which attempts to connect to the miner node on the socket it opened. Upon opening the connections, the miner considers that node as having received the block. Upon receiving a connection from the final node, the miner stops timing and writes the result to a file.

Mining the block and timing its dissemination



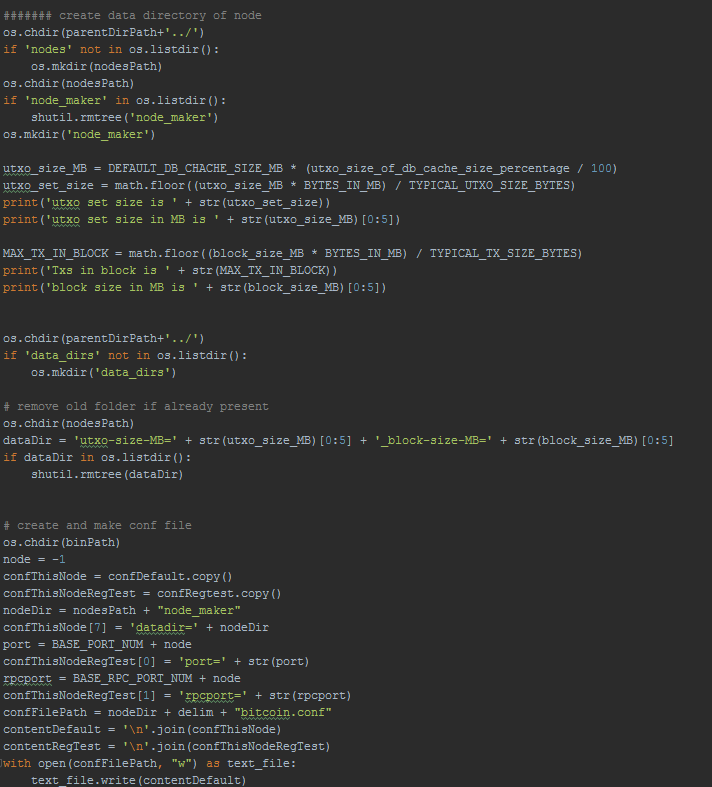
server.py



block.py

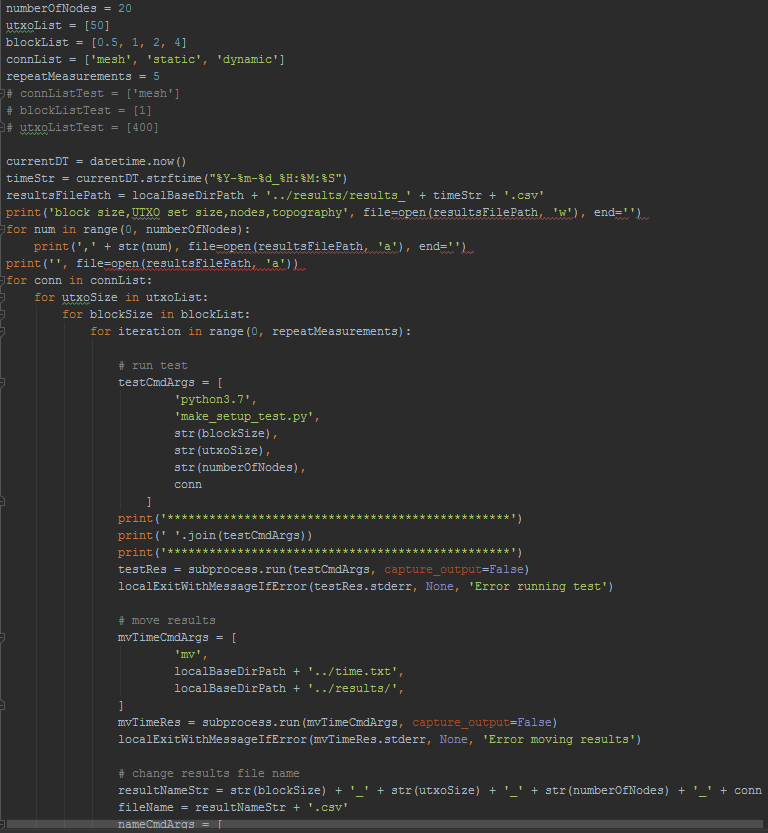
# Script – create**\_**starting**\_**blockchain

This script is called by the make\_setup\_test script to create the starting blockchain for all the nodes. The running of this script is deterministic, so as to create the same starting point for every test that is run. The starting blockchain is constructed using the parameters of block size and UTXO set size that were passed to the main script. The blockchain is created locally and will be copied to the AWS instances later (as explained in the previous part about the main script). The starting blockchain is of course identical for each node. The process consists of creating transactions to validate and put into blocks, which are then inserted into the blockchain.



# Script – run**\_**multiple**\_**tests

For convenience, another script is supplied, called run\_multiple\_tests.py, which can run multiple subsequent tests according to lists of parameters inside it. The script will go over every combination of parameters from all lists and can do multiple iterations for each combination (the parameters can be changed in the code). this script is intended only for parameter combinations that already have a corresponding data directory in the AMI, or at least will work for only one iteration (subsequent iterations will result in an error, because the main scrip will think the folder created on the first iteration is at the AMI too). Results file (generated by main script) is renamed to reflect the parameters used as a CSV file, and the content is appended to an aggregated results file.



# Setup and how to run

## Setup

On a linux computer, inside a dedicated folder, run the following commands:

|  |
| --- |
| git clone -b 0.17 --single-branch https://github.com/FoXPeeD/bitcoin.git  sudo apt-get update  sudo apt-get install build-essential libtool autotools-dev automake pkg-config libssl-dev libevent-dev bsdmainutils python3 libboost-system-dev libboost-filesystem-dev libboost-chrono-dev libboost-test-dev libboost-thread-dev  sudo apt-get install software-properties-common  sudo add-apt-repository ppa:bitcoin/bitcoin  sudo apt-get install libdb4.8-dev libdb4.8++-dev  cd bitcoin/  ./autogen.sh  ./configure --without-gui --without-miniupnpc  make |

This will pull all files of project under the directory “bitcoin”.

Next, install python and projects required packages:

|  |
| --- |
| sudo apt update  sudo apt install software-properties-common sudo add-apt-repository ppa:deadsnakes/ppa  sudo apt install python3.7  sudo apt-get install python3-pip sudo python3.7 -m pip install paramiko  sudo python3.7 -m pip install boto3  sudo python3.7 -m pip install scp |

Three files of AWS needed:

* ~/config

|  |
| --- |
| [default]  region=us-east-2 |

* ~/credentials

|  |
| --- |
| [default]  aws\_access\_key\_id = <PLACE\_YOUR\_KEY\_HERE>  aws\_secret\_access\_key = <PLACE\_YOUR\_SECRET\_HERE> |

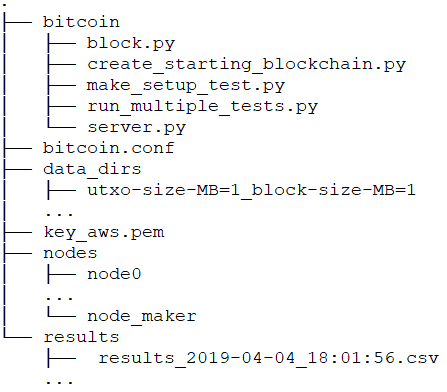
* Authorization file (pem), placed at the base folder.

You also need to create ssh keys (ssh-keygen -t rsa) which will be generated under ~/.ssh folder. After that use chmod 755 cmd on these new files.

Folder structure

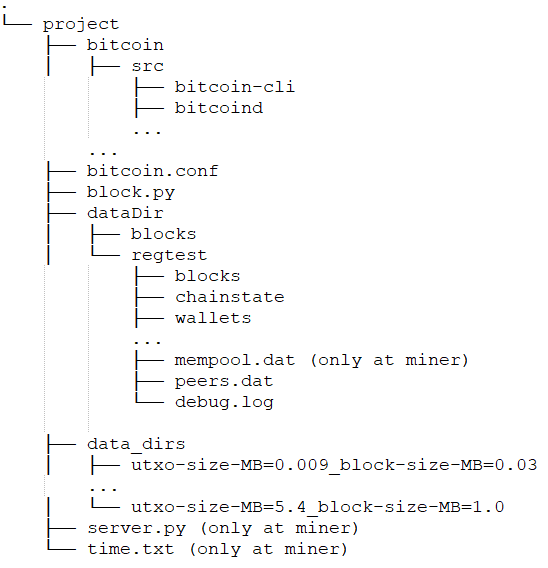
Local computer:

Under main folder of project:



Remote AWS servers:

Under root directory:



## How to run

The main script is called make\_setup\_test.py.

It receives 4 parameters:

1. Maximum size of blocks in MB.   
   This parameter will also be used as the size of the block which will be timed when mined.  
   This parameter essentially replaces the hard limit of block size, set in original source code.
2. UTXO set size in percentage, out of the 1.8 MB cache size.   
   (e.g. the value 50 will set the UTXO set size to 0.9 MB)  
   Clarification: in the script the dbcache is set to 4 MB, but only 1.8 MB out of this memory allocation is reserved for UTXO set in memory.
3. Number of bitcoin nodes to participate in the experiment (each in its own AWS instance),  
   including the node that acts as the miner.
4. Topology, namely:  
   mesh – all nodes are connected to all nodes (a clique).  
   static – every node is connected to 3 nodes randomly and another connection to the node with the next index (to ensure full connectivity). Random is based on hashes, which means its deterministic and thus consistent between runs.

dynamic – every node is connected to 3 nodes randomly and another connection to the node with the next index (to ensure full connectivity). Not consistent between runs.

Running exemple:

python3.7 make\_setup\_test.py 2 200 20 mesh

The command will initiate a test with block size of 2MB, 3.6MB of UTXO set (200% out of the 1.8MB cache), 20 instances running in AWS and a topology of a clique.

If a folder corresponding to the (first two) parameters mentioned above does not exist on the server (and thus not in the AMI as well), another script will be called to generate the folder required.  
note that generating a new folder is good for only one use. after that, the folder should either be erased or added to an AMI.  
after the script is done, the results will be printed to screen and written to time.txt file at the base folder.  
the measurements are the difference from when the first block notified (of the miner) to the time when some other node notified about a new block. results are comma separated.

For convenience, another script is supplied, called run\_multiple\_tests.py, that can run multiple subsequent tests according to lists of parameters inside it. The script will go over every combination of parameters from all lists (nested for loops for each parameter list) and can do multiple iterations for each combination.   
this script is intended only for parameter combinations that already have a corresponding data directory in the AMI, or at least will work for only one iteration(subsequent iterations will result in an error, because the main scrip will think the folder created on the first iteration is at the AMI too).   
results file (generated by main script) is renamed to reflect the parameters used as a CSV file, and the content is appended to an aggregated results file.

AWS setup requirements

* VPC –needs to assign a public IP.
* Security group – needs to allow ssh connection and all tcp ports in both directions.
* Subnets – needs to assign private IP addresses in the 10.0.2.100 - 10.0.2.199 range.

Parameters to change in script:  
parameters that need to change when adapting the scripts to a new AWS account are marked in main script by “TODO-SETUP” tag.

* Range of private IP addresses, if different from AWS setup requirements.
* pem file name.
* name associated with pem file.
* AMI ID
* Subnet ID
* Security group (sg) ID

Important bitcoind/conf file arguments:  
Mempoolexpiry – by default, when a node launches it loads transactions to mempool from file, it refers to any txs older then two weeks as expired. In order for pre-generated data directories to not expire after a while, this parameter (in hours) is set to a calculated value such that no data directory will expire.  
blocksonly – rising this flag prevents nodes from getting information about transactions and UTXOs from other nodes. This is applied to all nodes except the miner, in order for them to process the blocks “from scratch” (without any preparation or prepared calculations)  
reindex-chainstate – sometimes, when copying the data directories, some meta data or indexes of the blockchains state gets corrupted. This flag recalculates all this meta data at launch from the blocks themselves.

dbcache-specifies how much RAM will be allocated for block index, chain state and UTXO set.  
we are interested in how much UTXO set which gets 1.8MB out of 4MB provided in cache.  
note that this allocated space is only the in-memory part of the UTXO set, remaining space needed will be saved on secondary drive.

# Block dissemination time graphs

## Block size varied

UTXO set size is 0.9MB, clique topology

We can clearly see that there is a linear trend to the graph, which tells us there is a linear relationship between the block size and the dissemination time. This is an expected result, as the bigger the block is, the longer it should take to process it at each node. This is of course also due to the fact that there are no dependencies between transactions, so we would indeed expect that adding more of them would increase the dissemination time linearly.

## UTXO set size varied

Block size is 1MB, clique topology

This measurement gave us some interesting results. Up until 100% the behavior was expected and almost linear. We expected that once the UTXO set size surpassed the cache size, we would see a spike, and then continue to see a steeper linear advancement, but to our surprise there, was a dip in the next data point and then a gentler linear trend. Further research is needed to ascertain the cause of this.

## Dissemination process of different topologies

Block size is 1MB, UTXO set size is 0.9MB

The first graph describes the time it takes for a block to reach a different amount of nodes with different network topologies. The line representing the clique topology shows us that there is a very gentle slope for the process. This is to be expected as the block is sent directly to all other nodes (as the miner node is connected to all of them). In the two random topologies we see that the time to reach first four nodes converges (except for the fourth data point of the static topology which we can attribute to network fluctuations or differences in instance locations on AWS servers) with the time for the clique, as they each have four direct connections. From 5 nodes and onwards, they both follow a relatively identical trend. In the second graph we can see the total time it takes for full dissemination for each topology. As expected, the clique topology is much faster that the random topologies, who are very close to one another.

# Future improvements

There are many more parameters and to research and cross reference. Our project provides a good platform for many more tests to be run, such as testing different topologies or different combinations of the various attributes, and also provides the option of adding and testing many more parameters of the bitcoin network to research more into.

# Conclusions

As with many intricate system, here too we can see that there are many tradeoffs. There are many ways to improve latency in a bitcoin network, but they come at a price. We can improve our throughput by increasing block size and thus increasing the amount of transactions per block, but we pay for it with increased processing time at each node. Different topologies can also give us lower latency, but in turn we can get much higher network traffic overhead (as with the clique topology). The platform we built lets us track and measure the effects of these changes on the latency, and allows us to see what parameters affect it the most and in what way. Our platform also allows for other parameters to be added and studied, which could lead to many interesting results, as the bitcoin network has a seemingly endless list of different attributes that can be altered and cross referenced with each other.

# Bibliography

Websites

<https://bitcoin.org/en/> - main bitcoin website.

<https://en.bitcoin.it/> - bitcoin wiki.

<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html> - boto3 python package documentation.

<https://docs.python.org/2/library/subprocess.html> - Subprocess python package documentation.

<http://www.paramiko.org/> - paramiko python package website and documentation.

<https://en.wikipedia.org/wiki/Main_Page> - general knowledge and research.

<https://stackoverflow.com/> - general debugging and code assistance.

<https://bitcointalk.org/index.php?topic=2196785.0> – bitcoin core source code assistance (block size limit).

<https://github.com/bitcoin/bitcoin> - bitcoin core project on GitHub.

<https://github.com/FoXPeeD/bitcoin> - our project on GitHub.

Books

Mastering Bitcoin: Programming the Open Blockchain 2nd Edition - by Andreas M. Antonopoulos – bitcoin and blockchain theoretical and terminology research.