



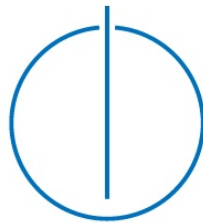
**Technische Universität
München**

Fakultät für Information Systems

Master's Thesis in Informatik

Bitcoin-like Blockchain Simulation System

Fabian Schüssler





**Technische Universität
München**

Fakultät für Informatik

Master's Thesis in Information Systems

Bitcoin-like Blockchain Simulation System

Bitcoin-ähnliches Blockchain Simulationssystem

Author: Fabian Schüssler

Supervisor: Prof. Dr. Hans-Arno Jacobsen

Advisor: Pezhman Nasirifard

Submission: xx.xx.20xx

I confirm that this master's thesis is my own work and I have documented all sources and material used.

München, xx.xx.20xx

(Fabian Schüssler)

Abstract

English Abstract

Inhaltsangabe

German Abstract

Acknowledgment

Acknowledgement

Contents

List of Figures	4
List of Tables	5
Abbreviations	6
1 Introduction	7
1.1 Motivation	8
1.2 Problem Statement	8
1.3 Approach	8
1.4 Organization	9
2 Background	10
2.1 Bitcoin	10
2.1.1 Node/ Miner	10
2.1.2 Block	10
2.1.3 Block size	11
2.1.4 SegWit	11
2.2 Alternative History Attack	12
2.3 VIBES: Fast Blockchain Simulations for Large-scale Peer-to-Peer Networks	12
2.3.1 Prerequisites	12
2.3.2 The Actor Model	12
2.3.3 Executables and Work Requests	12
2.3.4 Best Guess	12
2.3.5 Fast-forward	12
2.3.6 Priority Queue	12
2.3.7 Votes	12
2.3.8 Executable Types	12
2.3.9 Configuration parameters	12
2.3.10 Summary of existing classes	12
2.4 Analysis of hash rate-based double-spending	12
3 Related Work	13

4	Approach	14
4.1	Bitcoin-like Blockchain Simulation	14
4.2	Time-outs and Configuration	15
4.3	Lazy Logging	15
4.4	Time between Blocks	15
4.5	Block size limit	16
4.6	SegWit	18
4.7	Transaction per second	19
4.8	Processed and pending transactions per block	20
4.9	Transaction incentives and confirmation status	22
4.10	Alternative history attack	24
4.10.1	Prerequisites	24
4.10.2	Design and Architecture	25
4.10.3	Implementation	27
4.11	Transaction Spam	30
4.12	Technology Choice / Backend	30
4.13	Frontend	30
5	Evaluation	33
5.1	Correctness	33
5.1.1	Expected and calculated probability of a successful double spend	33
5.1.2	Expected and calculated maximal safe transaction value	36
5.1.3	Expected and simulated success probability of double spending	36
5.1.4	Expected and simulated transactions per seconds	37
5.1.5	Simulated and expected transaction incentives	41
5.2	Speed	41
5.3	Scalability	41
5.3.1	Varying Nodes	41
5.3.2	Varying Neighbours	41
5.3.3	Varying Transactions	41
5.4	Flexibility	41
5.5	Extensibility	41
5.6	Powerful Visuals	41
5.7	Case Studies	41
5.7.1	Optimising transactions per second	41
5.7.2	Securing a blockchain merchant	41
5.7.3	Choosing transaction fees	41
5.7.4	Flood attack	41
6	Summary	42
6.1	Status	42
6.2	Conclusions	42
6.3	Future Work	42

<i>CONTENTS</i>	3
Appendices	43
A Appendix	44

List of Figures

2.1	Byte map of transaction template.	11
2.2	Weight map of transaction template.	12
4.1	Screenshot Time between Blocks	16
4.2	Screenshot Transaction Summary	20
4.3	Screenshot Pending Transactions per Block	21
4.4	Screenshot Processed Transactions Per Block	22
4.5	Screenshot Transaction Confirmation Status per Transaction Fee	23
4.6	Screenshot Average Transaction Confirmation Time per Transaction Fee	23
4.7	Screenshot Block Tree, Branch Selection and Attack Summary - Attacker wins	29
4.8	Screenshot Block Tree, Branch Selection and Attack Summary - Attacker loses	29
4.9	Screenshot Tweet from Vitalik Buterin about Transaction Spam	30
4.10	Screenshot Configuration	31
4.11	Screenshot Simulation	32
5.1	Screenshots Success Probability of Double-Spending	35
5.2	Screenshot Transactions per Second - 6h Simulation Duration	38
5.3	Screenshot Transactions per Second - 48h Simulation Duration	40
A.1	The probability of a successful double spend, as a function of the attacker's hashrate q and the number of confirmations n . The abscissa shows the confirmations n and the ordinate shows the attacker's hashrate q	45
A.2	The maximal safe transaction value, in BTC, as a function of the attacker's hashrate q and the number of confirmations n . The abscissa shows the confirmations n and the ordinate shows the attacker's hashrate q	46

List of Tables

5.1	Double-spending outcomes and their simulated and expected probabilities .	37
5.2	Simulated and expected results of transactions per second: Sample with 6h simulation duration	39
5.3	Simulated and expected results of transactions per second: Sample with 48h simulation duration	40

Abbreviations

SegWit Segregated Witness.

tps transactions per block.

tps transactions per seconds.

VIBES Visualizations of Interactive, Blockchain, Extended Simulations.

Chapter 1

Introduction

Currently and in the last years Blockchain technologies such as Bitcoin and Ethereum are a very hot topic. According to the Gartner Hype Cycle [1] Blockchain technology is undergoing the peak of inflated expectation in 2017. Price and market capitalization changes of cryptocurrencies are covered by the media.

Blockchain technology enables decentralized consensus and can be used for record keeping. Bitcoin is the first digital currency to solve the double-spending problem without the need of a trusted authority. One of the main problems of Bitcoin or in general of Blockchains is the low maximum amount of possible processed transactions per seconds. Additionally, the Bitcoin community disagrees about how to solve this scalability problem, already split about different approaches and created Bitcoin forks.

VIBES (Visualizations of Interactive, Blockchain, Extended Simulations) is a blockchain simulator, which allows fast, scalable and configurable network simulations on a single computer without any additional resources. It was developed in a master thesis by Lyubomir Stoykov [2] and which is the foundation for this master thesis proposal. The goal of my master thesis is to improve VIBES to make more realistic simulations possible. In the future VIBES could be used by developers or heavy blockchain users to simulate changes to different Blockchains. So maybe in the future it can help the Bitcoin community to agree on change proposals.

There are other Simulators like Bitcoin-Simulator, but VIBES is the first simulator designed to be extended to blockchain systems beyond bitcoin and the first of its kind to be able to simulate transactions in the network. Bitcoin-Simulator only simulates the network at block level and therefore does not consider transactions [3].

1.1 Motivation

Motivation of Thesis.

1.2 Problem Statement

There are lots of possibilities to improve VIBES, which were also already outlined in the master thesis of Stoykov. The portability to different kinds of Blockchains and the ease of use is very important.

Here are some possibilities to extend the current version of VIBES, that I would like to work on in my master thesis:

- Add maximal block size and transaction incentives
- Break propagation delay down into three components: network bandwidth, block size and distance between Nodes.
- Improve speed by testing mutable variables
- Differentiate between miners and full nodes
- Calculate resources used by network: CPU, electricity and bandwidth.
- Change transaction generation from coordinator to nodes.
- Other improvements: code improvements, analysis and visualizing of information that is already captured by VIBES (probability of forks), ...

These extensions can improve the quality of the simulations and the use cases of VIBES. For example, the implications of Segwit2x could be analyzed. Maybe these extensions could also make it possible to realistically simulate the current Bitcoin blockchain.

1.3 Approach

The goal to make simulations more realistic. The approach from the original VIBES paper is taken over. Some designs and parts of the architecture have to be adjusted.

The configuration parameters (4.1.8) must be extended to break down propagation delay, to add maximal block size and transaction incentives.

Change transaction generation from coordinator to nodes (4.3.1).

Differentiate between miners and full nodes (4.3.2).

1.4 Organization

Organization of Thesis.

Chapter 2

Background

Background concepts required for understanding the thesis.

2.1 Bitcoin

100.000.000 Satoshi = 1 Bitcoin

2.1.1 Node/ Miner

in this work node = miner

2.1.2 Block

genesis block

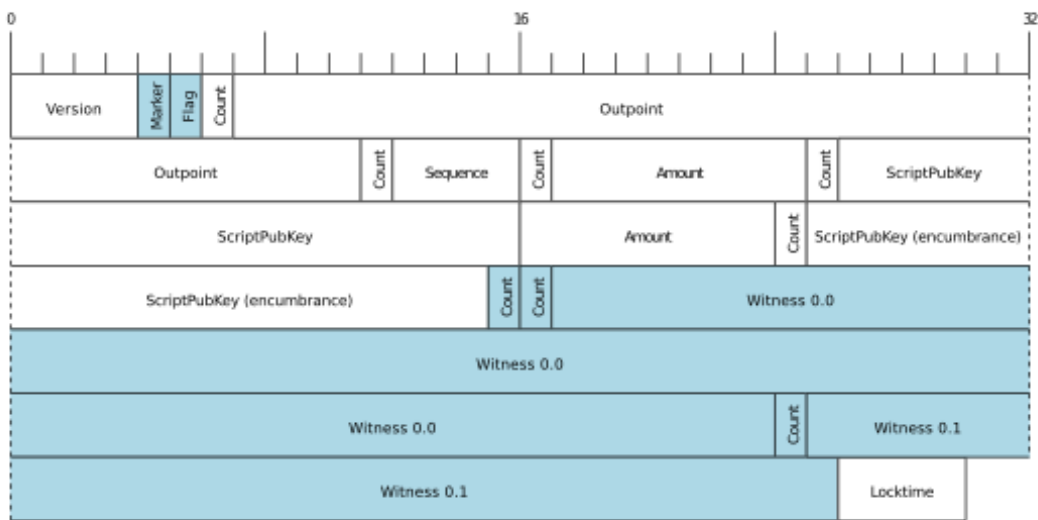


Figure 2.1: Byte map of transaction template.

Stale Blocks

Orphan Blocks

2.1.3 Block size

2.1.4 SegWit

[4]

[5]

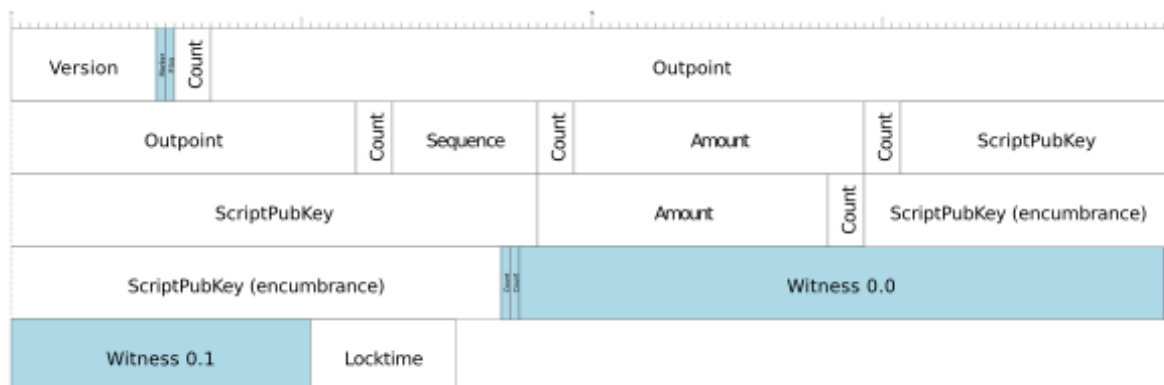


Figure 2.2: Weight map of transaction template.

2.2 Alternative History Attack

2.3 VIBES: Fast Blockchain Simulations for Large-scale Peer-to-Peer Networks

2.3.1 Prerequisites

2.3.2 The Actor Model

2.3.3 Executables and Work Requests

2.3.4 Best Guess

2.3.5 Fast-forward

2.3.6 Priority Queue

2.3.7 Votes

2.3.8 Executable Types

2.3.9 Configuration parameters

Chapter 3

Related Work

Related work materials

Chapter 4

Approach

In this chapter the changes to the existing VIBES framework are presented one by one. First the reasons for each improvement are described. Then the prerequisites for every change and the design and architecture are shown. Finally the implementation is explained. The implementation is split into frontend and backend. Since the frontend is only displaying the information from the backend the focus is on the backend, where the changes to the actual behaviour of the simulation are done. The frontend, the console output and the log file show the results of the backend and can therefore be used for the evaluation.

4.1 Bitcoin-like Blockchain Simulation

Extending the previous *Generic Simulation* to *Bitcoin-like Blockchain Simulation* made lots of changes necessary, especially in the frontend. More abstract ways to implement different strategies in the backend in Scala were researched. The backend differentiates between the currently only two strategies mainly with If-clauses. This seemed to be the best option, which avoids creating unnecessary complexity. It was also recommended by the author of VIBES. Bloated methods due to having multiple strategies in one method can be avoided by outsourcing strategy-specific parts to their own method(s).

4.2 Time-outs and Configuration

Previously the frontend could only display the information from the backend, if the simulation results were sent within 60 seconds. After checking the existing time-outs in the project and researching the default time-outs of the used frameworks, the problem was found in the `akka.http.server.idle-timeout` default setting. This default setting of 60 seconds was changed in `\vibes\server\src\main\resources\application.conf` to infinite.

Listing 4.1: application.conf

```
akka.http {  
  
  server {  
    idle-timeout = infinite  
  }  
}
```

Currently the time-out for providing the information to the frontend is set to 24 hours in `Main.scala`.

4.3 Lazy Logging

Previously logging only occurred in the console. This made debugging of long simulations difficult. Especially for the evaluation of any implementations a log file is necessary. For this reason the Scala modules `logback` and `scala-logging` were integrated into the project. Every important event is logged into `/logfile.log`.

4.4 Time between Blocks

The time between the blocks or also called block time is a very important metric. It can be used to check the system health. Due to the nature of the algorithm, a new block can be immediately found after the last one or it can take longer than usual. This can lead to unusual behaviour, as nodes might not have enough time to send transactions, synchronise their transactions pools or the blockchain. Therefore the block time can also be used to explain unusual behaviour. For the implementation of the Time between Blocks Figure

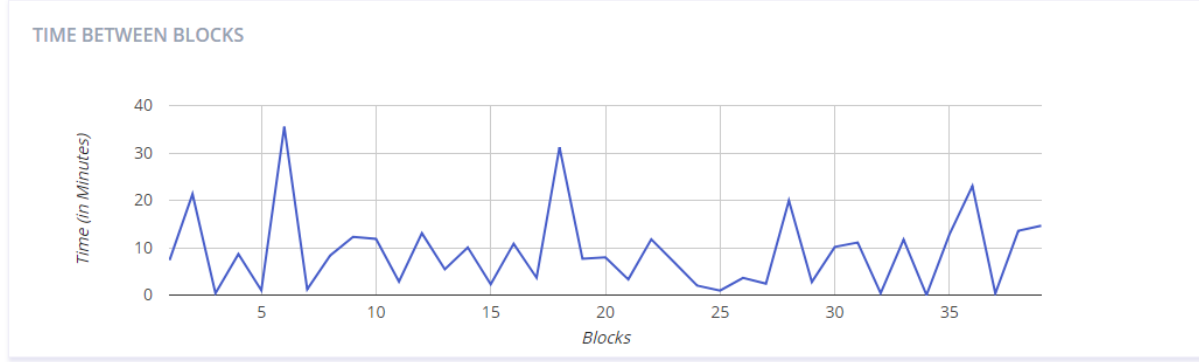


Figure 4.1: Screenshot Time between Blocks

4.1 next to the event timestamps only the beginning of the Simulation was additional necessary to calculate the time of the first block.

4.5 Block size limit

One of the biggest unresolved issues of Bitcoin-like Blockchains is scalability. The main metric to measure scalability is transactions per seconds *tps*. One drawback of this metric is that it contains no information about the transaction size or the usefulness of the transaction. Previously VIBES had no block size limit. This means infinite transactions can be processed and changing input parameters has no effect on the scalability. To be able to investigate the effects of different input parameters on the scalability, the introduction of a block size limit is necessary. This allows a more accurate simulation of Bitcoin. The block size limit is for example necessary to analyse the implications of Segwit2x according to VIBES [2].

The only prerequisites is to add one additional **configuration parameter** *maxBlockSize*: the maximal block size in KB, the current default value is 1.000 KB.

The design and architecture changes of the backend mainly happen in the model VBlock. All generated blocks obey the block size limit depending on if the simulation is a Bitcoin-like Blockchain Simulation.

Listing 4.2: VBlock with focus on block size limit

```

object VBlock extends LazyLogging {
  def createWinnerBlock(node: VNode, timestamp: DateTime): VBlock = {
    var maxTransactionsPerBlock : Int = 0
    var processedTransactionsInBlock: Set[VTransaction] = Set.empty

    if (VConf.strategy == "BITCOIN_LIKE_BLOCKCHAIN") {
      maxTransactionsPerBlock = Math.floor(VConf.maxBlockSize /
        ↳ VConf.transactionSize).toInt

      // takes the amount of maxTransactionsPerBlock out of the transaction
      ↳ pool into the winner block
      processedTransactionsInBlock =
        ↳ node.transactionPool.toSeq.take(maxTransactionsPerBlock).toSet
    } else {
      maxTransactionsPerBlock = node.transactionPool.size
      processedTransactionsInBlock = node.transactionPool
    }

    VBlock(
      id = UUID.randomUUID().toString,
      origin = node,
      transactions = processedTransactionsInBlock,
      level = node.blockchain.size,
      timestamp = timestamp,
      recipients = ListBuffer.empty,
      transactionPoolSize = node.transactionPool.size
    )
  }
}

```

The Listing 4.2 shows only the essential lines of code. First, the maximal transactions per block is calculated via the maximal block size and the transaction size. For simplicity the transaction size is constant, therefore this calculation is simple. Due to rounding down the actual block size can't be bigger than the limit. Finally the transaction are taken out of the transaction pool and later this variable is used in the creation of the winner block.

4.6 SegWit

In Chapter 4.5 the block size limit was implemented to make the simulation more accurate and similar to the actual Bitcoin Network. In this subsection one step further is taken. To accurately simulate the Bitcoin Network the block size limit needs to be replaced by a block weight limit like the actual Bitcoin Network did with the soft fork SegWit on 24th August 2017.

For a very simple implementation of SegWit, one could maybe just introduce a boolean `segWitEnabled` and replace the already existing block size limit with a block weight limit and transaction size with transaction weight. Since the comparison between non-SegWit and SegWit figures could be a very interesting use-case, new **configuration parameters** are introduced instead.

- *blockWeightLimit*: maximal block weight limit in weight unit
- *transactionWeight*: witness data per transaction in weight unit

For the simulation part of the backend a new condition is added to the VBlock object.

Listing 4.3: VBlock with focus on block weight limit

```
if (VConf.maxBlockWeight != 0 && VConf.transactionWeight != 0) {
    // SegWit is enabled
    maxTransactionsPerBlock = Math.floor(VConf.maxBlockWeight /
        ↪ VConf.transactionWeight).toInt
} else {
    // SegWit is disabled
    // multiplies by 1000 because maxBlockSize is in KB and transaction
    ↪ size is in B
    maxTransactionsPerBlock = Math.floor(VConf.maxBlockSize * 1000 /
        ↪ VConf.transactionSize).toInt
}
```

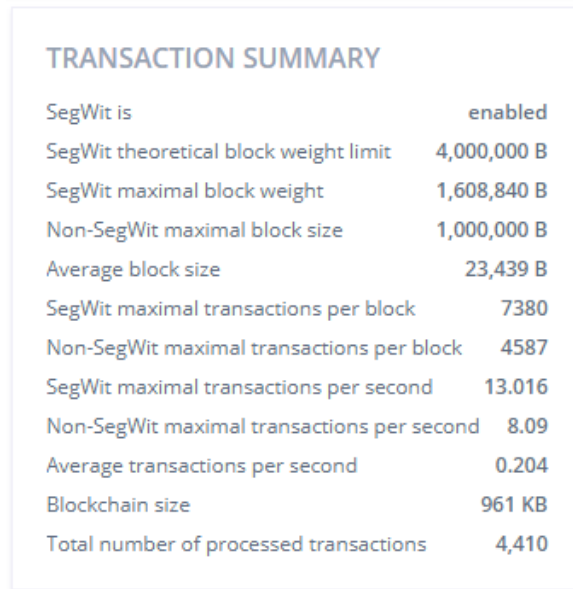
Numbers for the SegWit vs Non-SegWit analysis are provided in the frontend. These are calculated in the ReducerActor as can be seen in Listing 4.4. The SegWit theoretical block weight limit and the Non-SegWit maximal block size are given input values. The SegWit maximal block weight considers the transaction size and is more realistic than the theoretical block weight limit. Additionally both maximal transactions per block and transactions per second values show the differences between SegWit and Non-SegWit while the actual simulation values are also shown in Figure 4.2.

Listing 4.4: Calculations for the comparisons in the ReducerActor

```
// works only for constant transaction size and weight, otherwise an array is
  ↳ necessary
var segWitMaxBlockWeight = 0 // nonSegWitMaxBlockSize = VConf.maxBlockSize
var segWitMaxTransactionsPerBlock = 0
var nonSegWitMaxTransactionsPerBlock = 2147483647
var maxTransactionsPerBlock = 0
var segWitMaxTPS: Double = 0
var nonSegWitMaxTPS: Double = 0
if (VConf.transactionSize != 0) {
  // multiplies by 1000 because maxBlockSize is in KB and transaction size
  ↳ is in B
  nonSegWitMaxTransactionsPerBlock = Math.floor(VConf.maxBlockSize * 1000
    ↳ / VConf.transactionSize).toInt
  maxTransactionsPerBlock = nonSegWitMaxTransactionsPerBlock
  nonSegWitMaxTPS = nonSegWitMaxTransactionsPerBlock.toDouble /
    ↳ VConf.blockTime.toDouble
  nonSegWitMaxTPS = (math rint nonSegWitMaxTPS * 1000) / 1000
}
if (VConf.maxBlockWeight != 0 && VConf.transactionWeight != 0) {
  segWitMaxTransactionsPerBlock = Math.floor(VConf.maxBlockWeight /
    ↳ VConf.transactionWeight).toInt
  segWitMaxBlockWeight = segWitMaxTransactionsPerBlock *
    ↳ VConf.transactionSize
  maxTransactionsPerBlock = segWitMaxTransactionsPerBlock
  segWitMaxTPS = segWitMaxTransactionsPerBlock.toDouble /
    ↳ VConf.blockTime.toDouble
  segWitMaxTPS = (math rint segWitMaxTPS * 1000) / 1000
}
```

4.7 Transaction per second

Figure 4.2 also shows the average transaction per second. The *tps* is also called throughput or transaction rate. The average *tps* is - as the Listing 4.5 shows - calculated over the duration of the whole simulation and then given to the frontend.



A screenshot of a 'TRANSACTION SUMMARY' table. The table lists various metrics related to SegWit and Non-SegWit transactions, including block weight limits, block sizes, and transaction rates. The values are presented in a clean, modern font with a light blue header.

TRANSACTION SUMMARY	
SegWit is	enabled
SegWit theoretical block weight limit	4,000,000 B
SegWit maximal block weight	1,608,840 B
Non-SegWit maximal block size	1,000,000 B
Average block size	23,439 B
SegWit maximal transactions per block	7380
Non-SegWit maximal transactions per block	4587
SegWit maximal transactions per second	13.016
Non-SegWit maximal transactions per second	8.09
Average transactions per second	0.204
Blockchain size	961 KB
Total number of processed transactions	4,410

Figure 4.2: Screenshot Transaction Summary

Listing 4.5: Calculations for the tps in the ReducerActor

```

var actualTPS: Double = longestChainNumberTransactions.toDouble /
    ↪ secondsBetween(VConf.simulationStart,
    ↪ VConf.simulateUntil).getSeconds.toDouble
actualTPS = (math rint actualTPS * 1000) / 1000

```

According to Bitcoin-NG the *tps* of Bitcoin pre-SegWit was limited to only 1 to 3.5 *tps* for typical transaction sizes due to the block size at 1 MB [6]. For Bitcoin heavy transaction loads are an obstacle for a more widespread use [7]. A payment processor like VISA handles 4,000 transaction per second on average and has been stress-tested in 2013 to handle 47,000 transactions per second. In comparison, Bitcoin can only handle 7 transactions per second, due to the fact that block sizes are restricted to have a maximum size of 1 MB.

4.8 Processed and pending transactions per block

After introducing a block size limit in Chapter 4.5 and SegWit in Chapter 4.6 for the evaluation of correctness about transactions in general it is necessary to know what actually happens in the blocks. This means visualizations about the pending transactions and processed transactions per block would be very insightful.

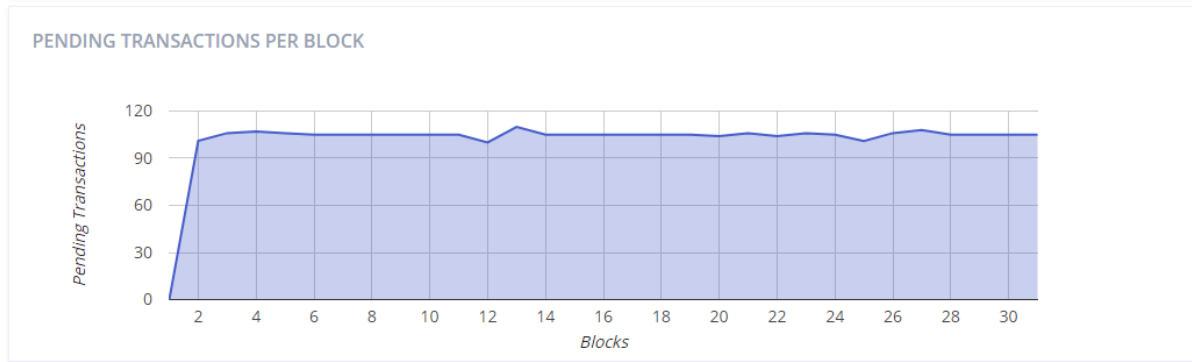


Figure 4.3: Screenshot Pending Transactions per Block

For the pending transactions per block Figure 4.3 changes in the models VBlock and VEventTypes and the ReducterActor were done. Every block has the transaction pool size minus the included amount of transaction at the time of the block creation as a new attribute.

For the processed transactions per block Figure 4.4 changes to the same classes and additionally Main were done. The new attribute processed transactions of a block describes the transactions which were included by the node. This is shown as the blue line, the red line shows the maximum possible transactions depending on if the block size limit or SegWit are enabled. If the block size limit and SegWit are disabled (equal to zero), then the transaction limit per block is unlimited and the red line isn't shown.

Both Figures 4.3 and 4.4 show that the first block contains no pending and no processed transactions. The reason for this is that the nodes do not send transactions before the genesis block is mined. One reason for having an empty genesis block is the fact that no Bitcoin exists before the first block, so no transaction fees can be paid and no Bitcoin can be send. But also transactions with zero transactions fees and zero Bitcoin sent can be valid and included in blocks. These transactions would be considered transaction spam. The miners just have no incentive to include zero fee transactions. Bitcoin's genesis block contains one transaction [8]. In the end, this question about having zero, one or lots of transactions in the genesis block seems to be a question of personal preference. One disadvantage of having zero or one transaction in the genesis block is that it distorts transactions metrics like *tps* for a low number of blocks or transactions.

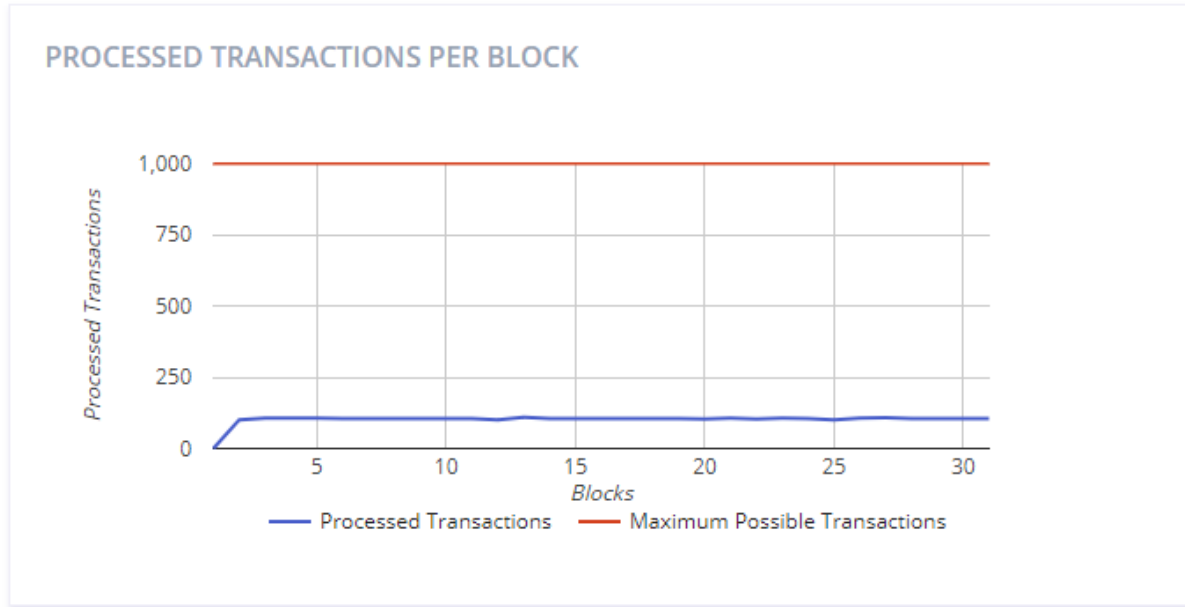


Figure 4.4: Screenshot Processed Transactions Per Block

4.9 Transaction incentives and confirmation status

In Bitcoin there are two types of incentives for miners, block rewards and transaction incentives or also called transaction fees. Previously these mining incentives were not considered. To make a more realistic simulation transaction incentives are added to the simulation. This allows analysis for example for determining the necessary price to include a transaction in a block within a certain time. It can also be used in future work for example about mining pools. Since research questions about transaction incentives are closely linked to the confirmation status new data structures are needed to easily access information about the creation time and confirmation time of transactions.

New variables for the transaction incentives and confirmation status are added to the VTransaction model.

- *transactionFee*: transaction fee in Satoshi
- *confirmation*: confirmation status as a boolean
- *creationLevel*: block level when the transaction was created
- *confirmationLevel*: block level when the transaction was included in a block

Transactions are assigned a random integer between 0 and 124 as transaction fee in Satoshi. This is about the same range as in reality, but the distribution is different. The

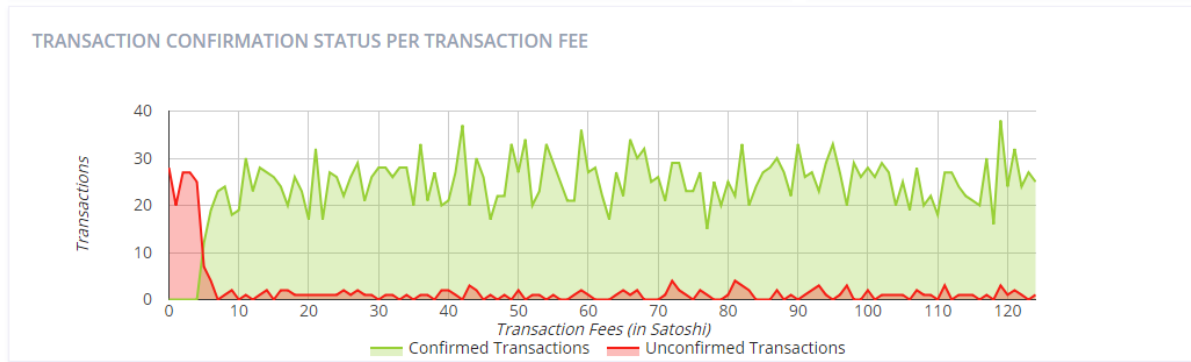


Figure 4.5: Screenshot Transaction Confirmation Status per Transaction Fee

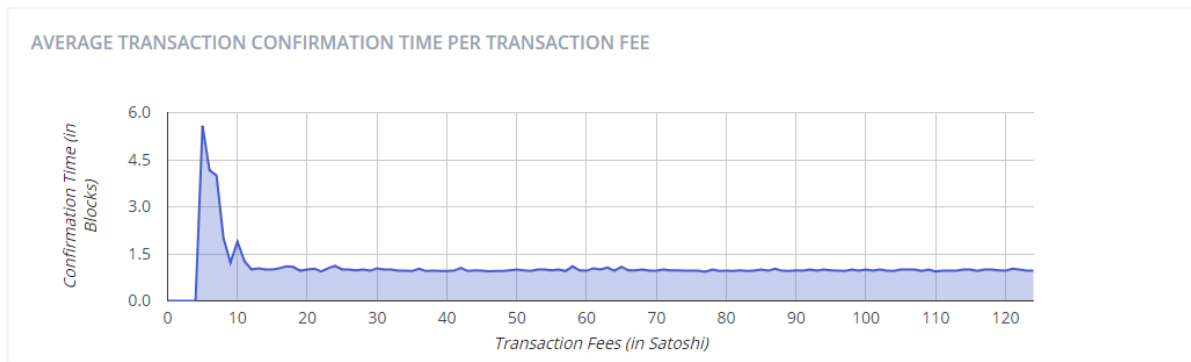


Figure 4.6: Screenshot Average Transaction Confirmation Time per Transaction Fee

real distribution can change from one moment to the next and is difficult to model. When a transaction gets included into a block, its transaction status changes from *false* to *true*.

To show the results of this implementation two charts were created. For these charts a slightly higher transaction throughput than block transaction capacity was chosen to show, because this shows the interesting in which the miners can choose between including transactions with high or low transaction incentives into their block.

Figure 4.5 shows the transaction confirmation status per transaction fee. The abscissa shows the transaction fees from 0 to 124 in Satoshi, the ordinate shows the amount of transactions. The red area shows the unconfirmed and the green area shows the confirmed transactions. It can be clearly seen, that the red area is only really big from 0 to 7 Satoshi. This means most of the pending transactions are the ones with the lowest transaction incentives.

Even in the case of block size limit or SegWit maximal transactions per block smaller than transaction throughput the red area exists. The reason is the nodes are sending transactions even after the last block was mined.

The next Figure 4.6 shows the average transaction confirmation time per transaction fee. The abscissa shows the transaction fee again, the ordinate shows the confirmation time in blocks. Both Figures are from the same simulation. Therefore we can see that the average confirmation time for transaction fees 0 to 3 is zero blocks. The reason for this is that there are no confirmed transactions in this range. Maybe this visualisation is not perfect, since it may lead to the conclusion that transactions with fees from 0 to 3 are instantly included in a block. But infinity blocks is hard to visualise and taking the first non zero confirmation time would also be misleading. The highest confirmation time is the point where the transactions barely get included into blocks. After this bottleneck all transactions get included in about the same time.

For the generation of both charts all created transactions are sent from the backend to the frontend. The frontend then summarizes the transactions per transaction fee, this could also be done by the backend. Sending all transactions ever created in a simulation to the frontend is probably a bottleneck for simulations with a very large number of transactions. The reason for this design decision is the flexibility to create or change charts to analyse different aspects of transactions. During development no issues were found. For simulations with a very large number of blocks parts of the previously created frontend would also be a bottleneck, since also every created block is sent to it.

VIBES offers the possibility to change the parameters sent to the frontend fast and easily. For simulations with a very large number of transactions and/ or a very large number of blocks the frontend could be changed or the sought after data can also be output via log file or console.

4.10 Alternative history attack

A 51% attack is one of the most commonly discussed attacks on the Bitcoin protocol. It belongs to the group of alternative history attacks. Due to the way it works see Chapter 2.2 complex changes are required.

4.10.1 Prerequisites

To simulate an alternative history attack additional **configuration parameters** are necessary. These parameters are used for the actual simulation of the attack, the calculation of the success probability of the attack and the maximum safe transaction value.

- *isAlternativeHistoryAttack*: if an alternative history attack is simulated as a boolean
- *hashRate*: attacker's hashRate as a percentage of the total hashRate of the Bitcoin Network
- *confirmations*: the amount of confirmations the attacked merchants are waiting for to accept a transaction
- *attackDuration*: the attacker gives up after mining a certain amount of blocks and not succeeding or if it is not possible any more to surpass the level of the honest blockchain
- *discountOnStolenGoods*: discount of the stolen goods by the attacker, a value from 0 (= full discount) to 1 (= no discount)
- *amountOfAttackedMerchants*: the attack is carried out against a certain amount of merchants at the same time
- *blockReward*: current block reward in BTC

4.10.2 Design and Architecture

Simulating the attack

In the following the attacker's nodes, blockchain or blocks are interchangeably described as evil, private or malicious and the honest networks' nodes as good or public.

The solution for the simulation of an alternative history attack selects nodes as attacking nodes according to the attacker's hash rate as a percentage of the total Bitcoin Network. The good and the evil nodes both can mine the genesis block. The genesis block is then the first block in both the good and the evil blockchain. For simplicity we assume that the attacker successfully sent the transactions to the attacked merchants in the second block of the honest blockchain. Immediately after the genesis block is mined, the evil nodes start mining together on their own evil blockchain. It is necessary for all nodes to update their neighbour nodes to only have their corresponding nodes as neighbours. The synchronising of the blockchain is only possible for the same type of node. As a counterexample the attacker's hash rate would suffer if the evil nodes could not synchronise their blocks properly if a high percentage of their neighbours are honest nodes.

Finally the success of the simulated attack is decided if the attacker's blockchain level can surpass the honest blockchain's block level after waiting for the Merchants confirmation

and before the attack duration ends. The attack can succeed, fail or be undecided. For example if a huge percentage of the network is malicious, then the honest network is likely to need a long time to reach the needed block level for the Merchants confirmation.

In the case of success or failure the two networks need to merge back together by updating their neighbours, allowing the synchronising of all blocks and taking the winning blockchain.

Calculating the success probability [9]

To be able to validate the results of the Bitcoin-like Blockchain Simulation with an Alternative History Attack a correct reference value for the success probability is required. Therefore the success probability of an Alternative History Attack needs to be calculated.

Before the formula to calculate the success probability of an alternative history attack is shown, the variables need to be explained. q is *hashRate*, the attacker's percentage of the hash rate of the total network. p is $1 - q$ and the percentage of the honest network.

$$p + q = 1 \quad (4.1)$$

It is the goal to calculate the success probability r . If the attacker's hash rate q is equal or bigger than p , then the success probability of the attacker is 100%. Due to the implementation the behaviour of the implementation can deviate from the 100%. For example the variables *attackDuration*, *confirmations* or the simulation duration can have an impact. If $q < p$, then the upper complex formula with binomial coefficients needs to be calculated.

$$r = \begin{cases} 1 - \sum_{m=0}^n \binom{m+n-1}{m} (p^n q^m - p^m q^n), & \text{if } q < p \\ 1, & \text{if } q \geq p \end{cases} \quad (4.2)$$

The formula for $q < p$ is transformed for the implementation. This allows the usage of factorial functions instead of binomial coefficients.

One difference between this formula and the implementation is the attack duration. The formula is not limited by an attack duration, while the implementation has one. We assume the difference is negligible. The default value of the attack duration is 20 blocks. The probability of an attacker winning despite being behind 20 blocks is in most cases

very small.

$$r = \begin{cases} 1 - \sum_{m=0}^n \frac{(m+n-1)!}{m!(n-1)!} (p^n q^m - p^m q^n), & \text{if } q < p \\ 1, & \text{if } q \geq p \end{cases} \quad (4.3)$$

Calculating the maximal safe transaction value [9]

(4.3)

$$\frac{(1-r)oB}{k(\alpha + r - 1)} \quad (4.4)$$

4.10.3 Implementation

The implementation of double-spending is complex due to the need of splitting up the nodes and having two separate blockchains running at the same time and after the double-spending attack the nodes are merging back together to work on one blockchain.

VConf

The VConf implements the new parameters for the alternative history attack which are mentioned in the Prerequisites (Chapter 4.10.1).

Main

The simulation and also the attack starts in Main. Main checks if an alternative history attack happens and sets up the configuration. After the end of the simulation Main also sends the results back to the frontend.

VNode

The VNode Model needs a new parameter *isMalicious* as Option[Boolean].

NodeRepoActor

In case of an attack NodeRepoActor creates/registers nodes with their corresponding type Option[Boolean] in the predefined ratio according to the attacker's hash rate. Malicious nodes are set to Some(true).

DiscoveryActor

The DiscoveryActor updates the neighbours according to the defined Neighbours Discovery Interval. In the case of an active attack the nodes are only allowed to receive neighbours of the same type (malicious/non-malicious).

NodeActor

The node with the smallest timestamp is allowed to add his block in the NodeActor. Here several conditions are checked to make sure only in valid cases the blocks are added. In the case of an attack also the status of the attack is determined in the *addBlockIfAlternativeHistoryAttack* function. In the case of an preRestart the configuration is reset. NodeActor has new property *isEvil*.

VBlock

Robustness increased by considering that in an attack the recipients of a block can be null.

ReducerActor

The ReducerActor calculates the maximal safe transaction value and the success probability of the alternative history attack. It also prepares the other values connected to the attack.

VEventTypes

VEventTypes provides the new figures to the Main.

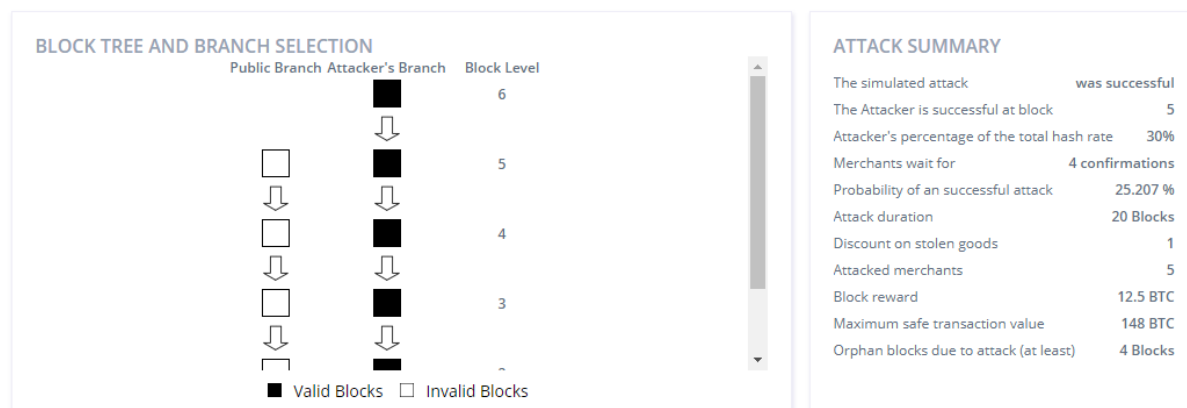


Figure 4.7: Screenshot Block Tree, Branch Selection and Attack Summary - Attacker wins

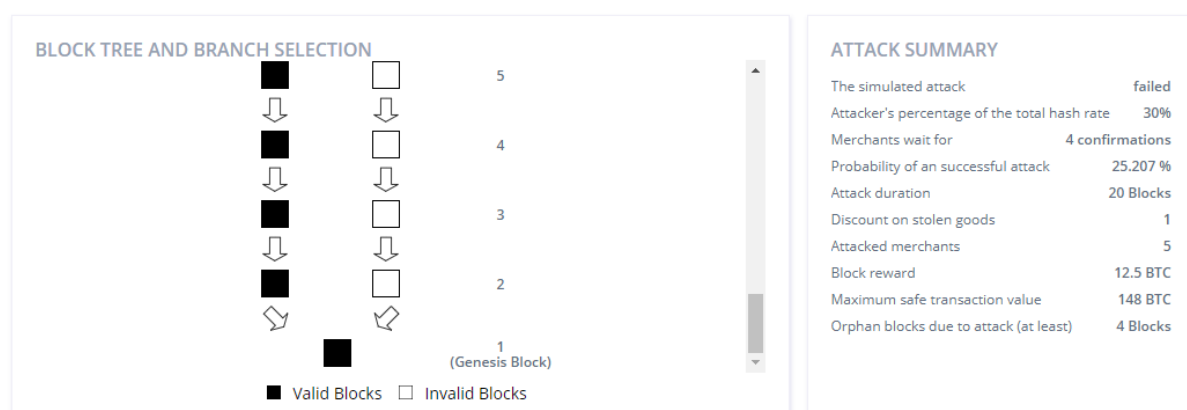


Figure 4.8: Screenshot Block Tree, Branch Selection and Attack Summary - Attacker loses

The block tree and branch selection visualization of the frontend was inspired by the double-spending paper [9]. Figure 4.7 shows a successful attack. In this case the attack was successful immediately after the public branch reached the Merchants confirmation requirement. The blocks of the winning branch are shown as black squares and the losing branch's blocks are white. The public branch is always on the left side, the attacker's branch in the middle and the right column shows the block level. For better visibility in case of the attack succeeding only the necessary valid blocks are shown. todo: update graphic

Figure 4.8 shows a failed attack. As can be seen, the attack failed due to the public branch reaching the end of the attack duration without the attacker's branch overtaking once. todo: better graphic



Figure 4.9: Screenshot Tweet from Vitalik Buterin about Transaction Spam

4.11 Transaction Spam

In July 2018 the Ethereum Network was effected by transaction spam or also called a flood attack. Such an "attacker" in principle trades their own money to increase the transaction costs for everyone. Only intended functionality and valid transactions is used. Even Vitalik Buterin tweeted about it [10]. The tweet can be seen in Figure 4.9. This transaction spam is also possible in the Bitcoin Network. Interesting research questions arise about the costs, which an malicious actor has to pay to make the Bitcoin Network unusable or too uneconomical to use for certain use-cases. In economics this is called crowding out.

4.12 Technology Choice / Backend

4.13 Frontend

talk about tech stack

The React Google Charts package was used for all the charts [11]. During development some error messages appeared about *loader.js*, which seem to be all fixed now [12] [13].

One of the main focuses of this paper is the frontend. Previously not everything captured by VIBES was visualised. This paper added lots of interesting graphics like block tree and branch selection, block time, pending transactions, processed transactions, transaction status and transactions fees and also new figures about double-spending, transactions and orphans to the frontend. But there may still very interesting information that has

VIBES
Fast Blockchain Simulations

CONFIGURE OPTIONS

GENERIC BLOCKCHAIN OPTIONS					
Number of nodes	20	Number of neighbours	4	Block time	567
Transaction size	218	Throughput	105	Latency	900
Neighbours Discovery Interval	3000	Simulate until: 07/21/2018 06:00 PM			
BITCOIN-LIKE BLOCKCHAIN SPECIFIC OPTIONS					
Transaction Propagation Delay	150	Max block size	1000	Network bandwidth	
SEGREGATED WITNESS OPTIONS					
Transaction weight	542	Maximal block weight			
DOUBLE-SPENDING OPTIONS					
Confirmations	4	Attacker's hash rate			
FLOOD ATTACK OPTIONS					
Transaction fee	0				

BACK

NEXT

Figure 4.10: Screenshot Configuration

not yet been analysed or visualized. These graphics and figures were chosen to validate the correctness of the changes to the backend and can also be used for research questions about double-spending, transaction fees, transaction status, tps etc.

minor other stuff: Masteractor: asserts VBlock: assert error messages check if simulation in past



Figure 4.11: Screenshot Simulation

Chapter 5

Evaluation

The evaluation chapter.

5.1 Correctness

The evaluation of the correctness of the Simulator's front-end output is essential. By validating the output of the front-end we also validate the output of the back-end.

Some parts of the evaluations are sample testing for consistency between the input parameters, expected and simulated or calculated output. Other parts are empirical tested.

For the calculation of the probability of a successful double spend and the maximal safe transaction value sample testing is used. For validating the success probability of simulated double spends empirical testing is applied.

5.1.1 Expected and calculated probability of a successful double spend

The probability of a successful double spend is tested with five samples and the results are compared to the figure A.1 from the original paper [9].

The edge cases $q = 2\%$ and $n = 1$, $q = 50\%$ and $n = 1$, $q = 2\%$ and $n = 10$ and $q = 2\%$ and $n = 10$ are tested as well as the common case of $q = 30\%$ and $n = 6$.

All mentioned test cases of figure 5.1 match the expected result.

... table comparing both

ATTACK SUMMARY	
The simulated attack	failed
Attacker's percentage of the total hash rate	2%
Merchants wait for	1 confirmations
Probability of an successful attack	4 %
Attack duration	20 Blocks
Discount on stolen goods	1
Attacked merchants	5
Block reward	12.5 BTC
Maximum safe transaction value	1199 BTC

(a) Sample with input: $q = 2\%$ and $n = 1$

ATTACK SUMMARY	
The simulated attack	failed
Attacker's percentage of the total hash rate	2%
Merchants wait for	10 confirmations
Probability of an successful attack	0 %
Attack duration	20 Blocks
Discount on stolen goods	1
Attacked merchants	5
Block reward	12.5 BTC
Maximum safe transaction value	2147483647 BTC

(b) Sample with input: $q = 2\%$ and $n = 10$

ATTACK SUMMARY	
The simulated attack	was successful
The Attacker is successful at block	2
Attacker's percentage of the total hash rate	50%
Merchants wait for	1 confirmations
Probability of an successful attack	100 %
Attack duration	20 Blocks
Discount on stolen goods	1
Attacked merchants	5
Block reward	12.5 BTC
Maximum safe transaction value	0 BTC

(c) Sample with input: $q = 50\%$ and $n = 1$

ATTACK SUMMARY	
The simulated attack	was successful
The Attacker is successful at block	11
Attacker's percentage of the total hash rate	50%
Merchants wait for	10 confirmations
Probability of an successful attack	100 %
Attack duration	20 Blocks
Discount on stolen goods	1
Attacked merchants	5
Block reward	12.5 BTC
Maximum safe transaction value	0 BTC

(d) Sample with input: $q = 50\%$ and $n = 10$

ATTACK SUMMARY	
The simulated attack	failed
Attacker's percentage of the total hash rate	30%
Merchants wait for	6 confirmations
Probability of an successful attack	15.645 %
Attack duration	20 Blocks
Discount on stolen goods	1
Attacked merchants	5
Block reward	12.5 BTC
Maximum safe transaction value	269 BTC

(e) Sample with input: $q = 30\%$ and $n = 6$ **Figure 5.1:** Screenshots Success Probability of Double-Spending

The probability for the test case (e) with $q = 30\%$ and $n = 6$ is used in Chapter 5.1.3.

5.1.2 Expected and calculated maximal safe transaction value

For testing the maximal safe transaction value the samples from the evaluation of the successful double spend probability are reused and compared to the figure A.2 from the ... paper [9].

The additional input parameters were:

- Attack duration: 20 Blocks
- Discount on stolen goods: 1
- Attacked merchants: 5
- Block reward: 12.5 BTC

Compared to the original paper the block reward was updated from 25 BTC to the current block reward of 12.5 BTC. This means the maximal safe transaction values need to be doubled to compare them with the correct values.

All test cases of figure 5.1 match the expected result, except for (a) and (b). (a) is off by one due to rounding. The correct result of (b) is infinite BTC, the front-end shows the maximal 32-bit signed integer value of 2,147,483,647 BTC.

... table comparing both

5.1.3 Expected and simulated success probability of double spending

For the empirical testing of double-spending the following script was used. It starts the simulation, waits for a certain time to let the simulation finish and repeats this for a total of 100 times.

```

ECHO Start of Loop

FOR /L %%i IN (1,1,100) DO (
    ECHO %%i
    start chrome "http://localhost:8082/vibe?blockTime=567&
        ↪ numberOfNeighbours=4&numberOfNodes=20&simulateUntil
        ↪ =1531411943382&transactionSize=1&throughput=105&latency=900&
        ↪ neighboursDiscoveryInterval=3000&maxBlockSize=100&maxBlockWeight
        ↪ =4000&networkBandwidth=1&strategy=BITCOIN_LIKE_BLOCKCHAIN&
        ↪ transactionPropagationDelay=150&hashRate=30&confirmations=6"
    timeout /t 40
)

```

Information for replication: New features might make additional parameters in the URL necessary.

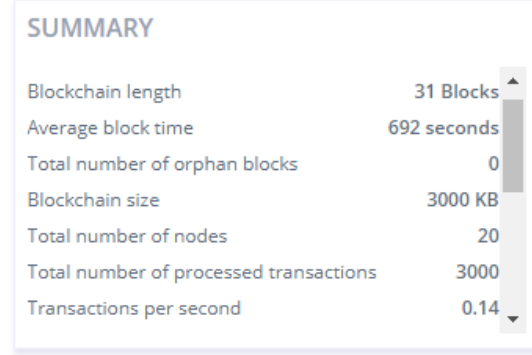
Outcome	Occurrences	Simulated probability	Expected probability
ATTACK NEITHER SUCCESSFUL NOR FAILED	45	-	-
ATTACK SUCCESSFUL	8	15.09%	15.645%
ATTACK FAILED	47	84.91%	84.355%
TOTAL	100	100%	100%

Table 5.1: Double-spending outcomes and their simulated and expected probabilities

After counting the occurrences of the outcomes in the logfile, they were summarized in the table 5.1. All simulations were finished in the specified time. The outcome "ATTACK NEITHER SUCCESSFUL NOR FAILED" can happen if the simulation time was too short. This number is ignored for the calculation of the probabilities. It is assumed that the unfinished simulations have a similar probability distribution like the finished ones. The simulated probability of the double-spending attack is with 15.09% very close to the expected probability of 15.645%, which was calculated in Chapter 5.1.1. Hereby is shown that the simulation of double-spending has the correct success probability.

5.1.4 Expected and simulated transactions per seconds

For the evaluation of correctness of the transaction per seconds (tps) of our simulation the formula for the calculation of *tps* is compared to the implementation.



SUMMARY	
Blockchain length	31 Blocks
Average block time	692 seconds
Total number of orphan blocks	0
Blockchain size	3000 KB
Total number of nodes	20
Total number of processed transactions	3000
Transactions per second	0.14

Figure 5.2: Screenshot Transactions per Second - 6h Simulation Duration

$$tps = \text{transactions per block} / \text{block time} \quad (5.1)$$

Since the *tps* for the whole simulation is an important key metric, the division is not done for one block, but instead for all blocks.

```
var tps: Double = longestChainNumberTransactions.toDouble /
    ↪ secondsBetween(VConf.simulationStart,
    ↪ VConf.simulateUntil).getSeconds.toDouble
```

As can be seen, the formulas are identical and the result should therefore be correct. Sample testing is done to check for implementation errors.

Sample with a simulation duration of six hours

For the first sample a short simulation of six hours is chosen. The chosen block time is 567 seconds and the chosen throughput is 105 transactions per block (*tpb*).

Calculation of the expected total processed transactions *pt*:

$$pt = 6h / 567\text{sec} * 105\text{tpb} = 6 * 60 * 60 / 567 * 105 = 4000 \quad (5.2)$$

Calculation of the expected *tps*:

$$tps = 4000\text{transactions} / 6h = 0.185 \quad (5.3)$$

As can be seen in table 5.2, for such a short simulation the *tps* is off by about 30%. One

Metric	Simulation	Expectation
Block time	692	567
Total number of processed transactions	3000	4000
Transactions per second	0.14	0.185

Table 5.2: Simulated and expected results of transactions per second: Sample with 6h simulation duration

reason for this significant difference is the the difference in the block time between the simulated and expected values. This can be due to variance. To reduce block time variance a longer simulation time is chosen as a second sample.

URL for reproduction (The `simulateUntil` parameter has to be adjusted. Due to new features additional parameters might be necessary.):
`http://localhost:8082/vibe?blockTime=567&numberOfNeighbours=4&numberOfNodes=20&simulateUntil=1531752759474&transactionSize=1&throughput=105&latency=900&neighboursDiscoveryInterval=3000&maxBlockSize=100&maxBlockWeight=4000&networkBandwidth=1&strategy=BITCOIN_LIKE_BLOCKCHAIN&transactionPropagationDelay=150&hashRate=0&confirmations=4`

Sample with a duration of 48 hours

For the second sample a longer simulation of 48 hours is chosen. The chosen block time is still 567 seconds and the chosen throughput is also still 105 transactions per block (*tpb*).

Calculation of the expected total processed transactions:

$$pt = 48h/567sec * 105tpb = 48 * 60 * 60/567 * 105 = 32000 \quad (5.4)$$

Of course the expected tps stays the same, since only the simulation duration parameter has been changed.

In the second comparative table 5.3 it can be observed, that a longer simulation duration leads the simulated block time to be closer to the expected block time. As a result the simulated *tps* is close to the expected value.

URL for reproduction (The `simulateUntil` parameter has to be adjusted. Due to new features additional parameters might be necessary.):
`http://localhost:8082/vibe?blockTime=567&numberOfNeighbours=4&`

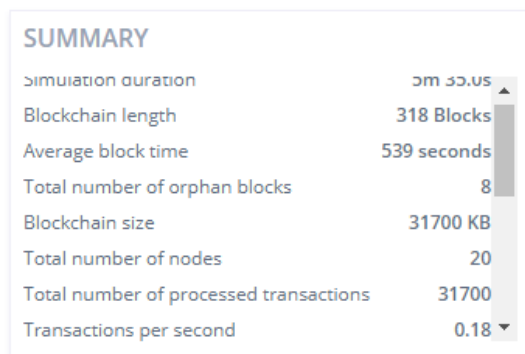


Figure 5.3: Screenshot Transactions per Second - 48h Simulation Duration

Metric	Simulation	Expectation
Block time	539	567
Total number of processed transactions	31700	32000
Transactions per second	0.18	0.185

Table 5.3: Simulated and expected results of transactions per second: Sample with 48h simulation duration

```
numberOfNodes=20&simulateUntil=1531905360000&transactionSize=1&
throughput=105&latency=900&neighboursDiscoveryInterval=3000&maxBlockSize=
100&maxBlockWeight=4000&networkBandwidth=1&strategy=BITCOIN_LIKE_
BLOCKCHAIN&transactionPropagationDelay=150&hashRate=0&confirmations=4
```

5.1.5 Simulated and expected transaction incentives

5.2 Speed

5.3 Scalability

5.3.1 Varying Nodes

5.3.2 Varying Neighbours

5.3.3 Varying Transactions

5.4 Flexibility

5.5 Extensibility

5.6 Powerful Visuals

5.7 Case Studies

5.7.1 Optimising transactions per second

5.7.2 Securing a blockchain merchant

5.7.3 Choosing transaction fees

5.7.4 Flood attack

Chapter 6

Summary

Summary

6.1 Status

Final Status of the Thesis

6.2 Conclusions

Concluding remarks of Thesis

6.3 Future Work

Future Work

for bitcoin related stuff

node != miner selfish mining mining pools

transaction sizes differ

Appendices

Appendix A

Appendix

A.1 First

First Appendix

q	1	2	3	4	5	6	7	8	9	10
2%	4%	0.237%	0.016%	0.001%	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
4%	8%	0.934%	0.120%	0.016%	0.002%	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
6%	12%	2.074%	0.394%	0.078%	0.016%	0.003%	0.001%	≈ 0	≈ 0	≈ 0
8%	16%	3.635%	0.905%	0.235%	0.063%	0.017%	0.005%	0.001%	≈ 0	≈ 0
10%	20%	5.600%	1.712%	0.546%	0.178%	0.059%	0.020%	0.007%	0.002%	0.001%
12%	24%	7.949%	2.864%	1.074%	0.412%	0.161%	0.063%	0.025%	0.010%	0.004%
14%	28%	10.662%	4.400%	1.887%	0.828%	0.369%	0.166%	0.075%	0.034%	0.016%
16%	32%	13.722%	6.352%	3.050%	1.497%	0.745%	0.375%	0.190%	0.097%	0.050%
18%	36%	17.107%	8.741%	4.626%	2.499%	1.369%	0.758%	0.423%	0.237%	0.134%
20%	40%	20.800%	11.584%	6.669%	3.916%	2.331%	1.401%	0.848%	0.516%	0.316%
22%	44%	24.781%	14.887%	9.227%	5.828%	3.729%	2.407%	1.565%	1.023%	0.672%
24%	48%	29.030%	18.650%	12.339%	8.310%	5.664%	3.895%	2.696%	1.876%	1.311%
26%	52%	33.530%	22.868%	16.031%	11.427%	8.238%	5.988%	4.380%	3.220%	2.377%
28%	56%	38.259%	27.530%	20.319%	15.232%	11.539%	8.810%	6.766%	5.221%	4.044%
30%	60%	43.200%	32.616%	25.207%	19.762%	15.645%	12.475%	10.003%	8.055%	6.511%
32%	64%	48.333%	38.105%	30.687%	25.037%	20.611%	17.080%	14.226%	11.897%	9.983%
34%	68%	53.638%	43.970%	36.738%	31.058%	26.470%	22.695%	19.548%	16.900%	14.655%
36%	72%	59.098%	50.179%	43.330%	37.807%	33.226%	29.356%	26.044%	23.182%	20.692%
38%	76%	64.691%	56.698%	50.421%	45.245%	40.854%	37.062%	33.743%	30.811%	28.201%
40%	80%	70.400%	63.488%	57.958%	53.314%	49.300%	45.769%	42.621%	39.787%	37.218%
42%	84%	76.205%	70.508%	65.882%	61.938%	58.480%	55.390%	52.595%	50.042%	47.692%
44%	88%	82.086%	77.715%	74.125%	71.028%	68.282%	65.801%	63.530%	61.431%	59.478%
46%	92%	88.026%	85.064%	82.612%	80.480%	78.573%	76.836%	75.234%	73.742%	72.342%
48%	96%	94.003%	92.508%	91.264%	90.177%	89.201%	88.307%	87.478%	86.703%	85.972%
50%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Figure A.1: The probability of a successful double spend, as a function of the attacker's hashrate q and the number of confirmations n . The abscissa shows the confirmations n and the ordinate shows the attacker's hashrate q .

q	1	2	3	4	5	6	7	8	9	10
2%	2400	42K	644K	9370K	$\approx \infty$	$\approx \infty$	$\approx \infty$	$\approx \infty$	$\approx \infty$	$\approx \infty$
4%	1150	10K	82K	615K	4437K	$\approx \infty$	$\approx \infty$	$\approx \infty$	$\approx \infty$	$\approx \infty$
6%	733	4722	25K	127K	626K	3018K	14M	$\approx \infty$	$\approx \infty$	$\approx \infty$
8%	525	2650	10K	42K	159K	588K	2144K	7749K	$\approx \infty$	$\approx \infty$
10%	400	1685	5741	18K	56K	168K	503K	1486K	4361K	12M
12%	316	1158	3391	9212	24K	62K	157K	396K	990K	2460K
14%	257	837	2172	5200	11K	27K	60K	132K	290K	632K
16%	212	628	1474	3178	6580	13K	26K	52K	102K	200K
18%	177	484	1043	2061	3901	7202	13K	23K	42K	74K
20%	150	380	763	1399	2453	4190	7039	11K	19K	31K
22%	127	303	571	983	1615	2582	4053	6288	9671	14K
24%	108	244	436	710	1103	1665	2467	3608	5229	7525
26%	92	198	337	523	775	1113	1570	2182	3005	4106
28%	78	161	263	392	556	766	1035	1377	1815	2372
30%	66	131	206	296	406	539	701	899	1141	1435
32%	56	106	162	225	299	385	485	602	740	901
34%	47	86	127	172	221	277	340	411	491	582
36%	38	69	99	130	164	200	240	283	331	383
38%	31	54	76	98	121	144	169	196	224	254
40%	25	42	57	72	87	102	118	134	151	168
42%	19	31	41	51	61	70	80	90	99	109
44%	13	21	28	34	40	46	51	57	62	68
46%	8	13	17	21	24	27	30	32	35	38
48%	4	6	8	9	10	12	13	14	15	16
50%	0	0	0	0	0	0	0	0	0	0

Figure A.2: The maximal safe transaction value, in BTC, as a function of the attacker's hashrate q and the number of confirmations n . The abscissa shows the confirmations n and the ordinate shows the attacker's hashrate q .

Bibliography

- [1] K. Panetta, “Top trends in the gartner hype cycle for emerging technologies, 2017.” <https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/>, 2017. Accessed: 2018-04-15.
- [2] L. Stoykov, “Vibes: Fast blockchain simulations for large-scale peer-to-peer networks,” Master’s thesis, Technische Universität München, 2018.
- [3] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the security and performance of proof of work blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016* (E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, eds.), pp. 3–16, ACM, 2016.
- [4] F. Schüssler and Y. Kandalaft, “Bitcoin-like blockchain scalability issues and improvements.” Seminar Paper.
- [5] “Weight units.” https://en.bitcoin.it/wiki/Weight_units. Accessed: 2018-07-16.
- [6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, “Bitcoin-ng: A scalable blockchain protocol,” in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation, NSDI’16, (Berkeley, CA, USA)*, pp. 45–59, USENIX Association, 2016.
- [7] R. Beck, J. S. Czepluch, N. Lollike, and S. Malone, “Blockchain - the gateway to trust-free cryptographic transactions,” in *24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016*, p. Research Paper 153, 2016.
- [8] “Details on genesis block.” <https://bitcoin.stackexchange.com/questions/18454/details-on-genesis-block>. Accessed: 2018-07-23.
- [9] M. Rosenfeld, “Analysis of hashrate-based double spending,” *CoRR*, vol. abs/1402.2009, 2014.

- [10] Vitalik Buterin, “Transaction Spam.” <https://twitter.com/VitalikButerin/status/1018990773042405376>. Accessed: 2018-07-21.
- [11] “React google charts.” <https://github.com/RakanNimer/react-google-charts>. Accessed: 2018-07-21.
- [12] “React google charts - issue 197.” <https://github.com/rakannimer/react-google-charts/issues/197>. Accessed: 2018-07-21.
- [13] “React google charts - issue 202.” <https://github.com/rakannimer/react-google-charts/issues/202>. Accessed: 2018-07-21.