Bazo – A Cryptocurrency from Scratch

Livio Sgier
Zurich, Switzerland
Student ID: 12-918-702

Supervisors: Dr. Thomas Bocek, Bedrija Hamza, Bruno Rodrigues
Date of Submission: August 23, 2017
Abstract


A financial service company based in Zurich developed a bonus program that incentivizes customers to use its credit cards. Customers collect virtual points with every purchase using a card issued by the company which can then be exchanged for gift cards and coupons of registered partners. A downside of this approach is the centralized position of the company. In order to include a new partner into the program, time-consumming and tailored contracts need to be drafted. The scope of this thesis is the development of a new cryptocurrency, called Bazo (Esperanto: base; foundation) that substitutes the traditional system. Bazo coins can be used as a unified payment medium, avoiding the restriction to exclusively crafted gift cards and coupons. The new system also offers the possibility for direct payments between partners and customers and among customers themselves. This leads to a disintermediation of the company, thereby reducing administration overhead. Bazo is a blockchain-based cryptocurrency that supports the creation of new accounts and the transfer of funds on the blockchain. It has been designed for scalability and simplicity. Bazo's use case necessitates central elements to deal with invite-only and money issuance requirements of the system. The side effect of a central element is added flexibility that is leveraged to dynamically change system parameters in case of system failures or unforeseeable circumstances without the need for a hard fork.
Acknowledgments

I would like to thank my supervisor, Dr. Thomas Bocek, for the interesting conversations, his continuous assistance and inputs throughout this work. It has been a great pleasure and inspiration to work with someone as passionate and bright as Thomas.

I would also like to thank Bedrija Hamza for his supervision. He was always very helpful and provided many useful inputs to this thesis.

Besides my supervisors, I would also like to thank Prof. Dr. Burkhard Stiller, the head of the Communication Systems Research Group, for giving me the opportunity to work on such an interesting topic.
# Implementation

4.1 Software Architecture ........................................... 15
4.2 Storage ................................................................. 16
   4.2.1 Key/Value Structure ......................................... 17
   4.2.2 Storage Hierarchy ........................................... 18
4.3 Networking ........................................................... 19
   4.3.1 Peer-to-Peer Network ....................................... 19
   4.3.2 Message Format ................................................ 20
   4.3.3 Connection Types ............................................ 21
   4.3.4 Message Types ................................................ 22
4.4 Consensus ............................................................. 23
   4.4.1 Proof-of-Work ................................................. 23
   4.4.2 Competing Chains ........................................... 24
4.5 Transaction Malleability ........................................... 25
4.6 Miner-Client API ..................................................... 26
   4.6.1 Network .......................................................... 27
   4.6.2 Transactions .................................................. 28
4.7 Concurrency .......................................................... 29
4.8 Testing ................................................................. 30

# Evaluation

5.1 Security Considerations ........................................... 33

# Future Work

6.1 Consensus Mechanism .............................................. 35
6.2 Simplified Payment Verification (SPV) .......................... 35
6.3 Checkpointing ....................................................... 36

# Summary and Conclusions

7. Summary and Conclusions ........................................... 39
CONTENTS

Abbreviations 45

List of Figures 45

A Message Types 49

B Installation Guidelines 51
  B.1 Miner Application .................................................. 51
  B.2 Client Application .................................................. 51
    B.2.1 Key Handling .................................................. 52
    B.2.2 Account Transaction ........................................... 52
    B.2.3 Transferring Funds Transaction .............................. 53
    B.2.4 Changing System Parameters Transaction ................ 53

C Contents of the CD 55
Chapter 1

Introduction

Blockchain-related applications have gained a lot of attention from academia and industry in recent times. Gartner’s Strategic Technology Trends for 2017 ranked blockchain among the top 10 trends [1]. The Guardian news outlet speculates whether blockchain is the most important IT invention of our age [2] and the Swiss media outlet NZZ boldly claims that the invention of blockchain is as significant as the Internet [5]. Blockchain-based cryptocurrencies emerged with the invention of Bitcoin, whose white paper was published in October, 2008 [6]. Since then, cryptocurrencies are on the rise. At the time of writing, 1058 digital assets are listed with a combined market capitalization of 140 $bn\(^1\). New cryptocurrencies emerge on a weekly basis, each with their own set of features and potential use cases. Therefore, it is of no surprise that companies in the finance industry are keen on familiarizing with the potential of this new technology.

One such company is a financial service provider based in Zurich. This thesis emerged out of a collaboration effort between them and the Communication Systems Group of the University of Zurich. This financial service provider developed a bonus program that incentivizes customers to use its credit cards by issuing virtual points for every conducted purchase with these cards. The virtual points can in turn be used to buy gift cards and coupons from registered partners on a centralized marketplace offered by the service provider. The scope of this thesis consists of the development of a cryptocurrency, called Bazo (Esperanto: base; foundation), that substitutes the traditional system.

1.1 Motivation

The traditional bonus system has several drawbacks. Tailored contracts need to be drafted among the company and all included partners. Furthermore, the central role of the company as an intermediary leads to administration work by maintaining a marketplace. The goal of this thesis is to design a system that overcomes these disadvantages. A blockchain-based cryptocurrency brings the advantage to use a unified payment medium that drastically lower the contract complexities between the company and the partners. Additionally,

\(^1\)Source: coinmarketcap.com as of August 15, 2017
the cryptocurrency can be used for direct payments among peers, which leads to the disintermediation of the company, thereby reducing administration and maintenance overhead. Familiarizing with blockchain technology, specifically cryptocurrencies, by witnessing the technology in action, can pave the way for further potential business applications.

1.2 Description of Work

This thesis covers the development of the Bazo cryptocurrency, a private/invite-only blockchain. The technical goals of Bazo as a cryptocurrency are to substitute the traditional system, to simplify the process of including new partners into the ecosystem and to facilitate a concept for value exchange in the form of direct payments. Bazo covers all properties of the traditional system, which are 1) Only registered customers have access to the blockchain and 2) Value (i.e., Bazo coins) can be issued by the company. This necessitates a private/invite-only blockchain.

1.3 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 familiarizes the reader with blockchain technology and explains the traditional bonus system. Chapter 3 is concerned with the design of Bazo, based on the given requirements. In Chapter 4 implementation and practical topics are elaborated. An evaluation of Bazo is conducted in Chapter 5, which is followed by Chapter 6 about future work. A summary and conclusions are drawn in Chapter 7.
Chapter 2

Background and Related Work

The following section introduces blockchain technology and explains the different technical components as well as key differences among well-known instances of cryptocurrencies. It is followed by a section about the traditional bonus system and its workings among the different stakeholders.

2.1 Blockchain-based cryptocurrencies

Blockchain-based cryptocurrencies are a decentralized payment system with a public ledger maintaining a global state. This ledger is maintained by network participants called miners. The following technical components shared by a majority of cryptocurrencies can be identified and examined independently: Transaction system, data model, consensus protocol, peer-to-peer communication network and the openness of the blockchain.

The global state of the public ledger is represented by a series of transactions. Examples thereof include the encoding of funds transferred to a public key (e.g., TX_PUBKEY in Bitcoin [7]) or more complex transactions, such as the creation and interaction with a smart contract (e.g., in Ethereum [8]). Several transactions are combined in blocks, backwardly pointing to one another, resulting in a chain of blocks called blockchain.

The dominating data models are account- and unspent transaction output (UTXO)-based. Account-based data models are intuitively easier to understand because of the similarity to traditional bank accounts. Accounts in this data models are first-class citizens that have an address and a balance with potentially additional information specific to the cryptocurrency. UTXO-based systems, however, do not have a notion of accounts. Because all transactions are stored in the blockchain, the inputs of future transactions can reference the outputs of validated transactions without having to store an intermediary balance state.

The acceptance criteria for a block as well as the consensus of the valid blockchain are specified by the consensus protocol. The most prominent consensus protocol currently powering more than 90% of the total market capitalization of existing digital currencies is Proof-of-Work (PoW), which is employed in the Bitcoin and Ethereum [9] consensus
### Table 2.1: Comparison of Bazo and popular cryptocurrencies and smart contracts

<table>
<thead>
<tr>
<th>Transaction System</th>
<th>Data Model</th>
<th>Consensus</th>
<th>Private vs. Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitcoin</td>
<td>Cryptocurrency</td>
<td>Unspent Transactions</td>
<td>Proof of Work</td>
</tr>
<tr>
<td>Ethereum</td>
<td>Cryptocurrency, Smart Contract</td>
<td>Account</td>
<td>Proof of Work</td>
</tr>
<tr>
<td>Ripple</td>
<td>Cryptocurrency</td>
<td>Unspent Transactions</td>
<td>Ripple Protocol Consensus Algorithm</td>
</tr>
<tr>
<td>Hyperledger Fabric</td>
<td>Smart Contract</td>
<td>Account</td>
<td>Practical Byzantine Fault Tolerance</td>
</tr>
<tr>
<td>Bazo</td>
<td>Cryptocurrency</td>
<td>Account</td>
<td>Proof of Work</td>
</tr>
</tbody>
</table>

protocol. PoW allows a miner to prove to another miner that a certain amount of computational resources have been utilized. One such example is the repeated calculation of a cryptographic hash function with differing inputs until the output is less than a defined value. Other consensus protocols, such as Practical Byzantine Fault Tolerance (PBFT) [10] used in the Hyperledger blockchain [11], Raft [12] or Proof-of-Stake (PoS) remain an active research area [13].

The peer-to-peer communication network has a significant impact on the stability of the system. First, decentralization of network participants is important for the prevention of 51% attacks [6]. Additionally, any latency between block broadcasting and reception leads to potential blockchain forks, making the system vulnerable to double-spending attacks [14]. Furthermore, it is also important to distinguish between public and private blockchains. Public blockchains (e.g., Bitcoin and Ethereum) allow any user to freely join and leave the network, whereas private blockchains have stricter control over who is authorized to participate in the network. Reasons for a shift towards private blockchains include regulatory compliance, enhanced security, privacy guarantees as well as improved efficiency [3].

Bazo is a new cryptocurrency, based on an invite-only private blockchain, giving a considerable amount of control to its central element. Its consensus mechanism is a PoW system similar to the one implemented in Bitcoin. The data model of Bazo is account-based. Table 2.1 compares Bazo with other well-known blockchain-based cryptocurrency and smart contract systems.

### 2.2 Traditional Bonus System

To incentivize usage of issued credit cards, the company launched a system that transfers virtual points to the customer account. These virtual points can be collected and exchanged for exclusive offers on the company’s website or within an app. The traditional system as it is in place now can be seen in Figure 2.1. There are three actors: the company, partners and customers. The relationship among the actors are as follows:
2.2. TRADITIONAL BONUS SYSTEM

Company – Partners In order to include a new partner in the ecosystem, tailored contracts need to be drafted. These contracts specify the describe offers by the partner and the compensation paid by the company. Offers include gift cards and coupons.

Company – Customers The company maintains a marketplace on its website and an app that offers the gift cards and coupons by the partners. Customers can then pay for this offers with virtual points that are collected with every purchase using a credit card issued by the company.

Partners – Customers The customers can then use their purchased gift cards or coupons as a means of payment directly in the partner’s store.
Chapter 3

Design

This chapter outlines the design of Bazo based on the given requirements. The sections discuss the benefits of using a cryptocurrency-based system over the traditional system, the trade-offs regarding different data models and the motivation for an account-based model for Bazo. Furthermore, the design aspects for all Bazo entities, e.g., accounts, transactions and blocks, are explained.

3.1 Cryptocurrency-based System

Chapter 2 described the traditional bonus system in place and the relationships among all stakeholders. The traditional system exposes the following weaknesses:

Administration Overhead The company has to draft tailored contracts with every partner.

Intermediation The intermediary role of the company and the central marketplace leads to more administration and maintenance work.

Limited customer awareness Many customers are not aware of the existence of the bonus program and can therefore not profit.

Using a blockchain-based cryptocurrency allows the system to become decentralized and simplified. Figure 3.1 depicts the new system with Bazo integration. Bazo coins can be treated as a currency and the offers by the partners need not be limited to exclusive gift cards. Customers can pay with Bazo coins directly in the partner’s store, which can then regularly exchange their Bazo coins to a fiat currency (e.g., CHF). This procedure leads to the following two advantages over the traditional bonus system:

Unified payment medium Instead of being restricted to exclusively crafted gift cards and coupons, Bazo coins can be used as a unified payment medium. This reduces the complexities of the individual contracts among the company and the partners.
Figure 3.1: Proposed system with Bazo integration

**Direct payments** Customers do not need to exchange virtual points for gift cards, but can use Bazo coins as a direct payment method. This leads to the disintermediation of the company, followed by less administrative and maintenance work. Customers can also exchange Bazo coins among themselves.

### 3.2 Data Model

There are two dominating data models in blockchain-based cryptocurrencies. On the one hand, there are (UTXO)-based models in which new transactions can be seen as the chaining of past transactions. There is no notion of an account in this data model. On the other end, accounts in account-based models are first-class citizens and the validity of transactions is based on the account state. Updating an account based on transactions can be seen as an intermediate step that offers benefits as well as drawbacks.

#### 3.2.1 Unspent Transaction Output (UTXO)-based Model

The simplified notion of UTXO-based models is as follows: A transaction has one or more inputs and one or more outputs. The input is a list of past transaction outputs, each defining the amount of coins and the address to which the coins were transferred. The output is analogous: It consists of one or more addresses and the amount of coins to transfer to each of them. A user $U$ wishing to send an amount of $X$ coins to a defined address combines past transaction outputs (that listed $U$ as the recipient) with a total sum of at least $X$ coins and lists as output the address(es) of choice. In case the total sum amounts to more than $X$, the user $U$ can add a second output entry with the difference pointing to an address to which $U$ is in possession of the private key. The benefit of this approach is parallel transaction processing. If a user includes several different outputs from
past transactions, they can all be processed in parallel. However, this comes at a cost. In order to validate a transaction, all past transactions have to be saved which prevents memory optimizations\(^1\). Furthermore, the optimal choice of combining past transactions is not trivial \([4]\): A naive approach combines the smallest available transaction outputs until the sum gets equal or larger than the spending amount. However, this can cause the transaction to get very large and expensive if the fees are relative to the transaction size (e.g., in Bitcoin). On the other hand, the benefit of this naive approach is the consolidation of small transaction outputs, which decreases the number of UTXOs a wallet has to handle, which in turn boosts performance. To optimize these trade-offs, several heuristics based on the desired design goals have to be considered which increases the complexity of the implementation logic.

3.2.2 Account-based Model

Accounts are first-class citizens in account-based models. A generalized notion of this model is as follows: Transactions lead to state changes. Compared to the UTXOs-based model, however, these state changes are consolidated in accounts. The union of all accounts make up the system state. The validity of a transaction depends on the available balance on the sender’s account and no prior transactions have to be inspected. This leads to several benefits: The union of all accounts after a block validation can be considered a transactional snapshot of the system. The caused space overhead by introducing accounts is outweighed by the fact, that past transactions can be discarded as they are not referenced by new transactions. However, in case of a blockchain fork, a conservative amount of past transactions should always be kept in memory to simplify rollback operations. On the one hand, the lower memory requirements reduces the barriers of entry for new miners. On the other hand, it is more difficult for new miners to bootstrap if only a subset of all miners possess the whole blockchain. To improve system health, it is therefore advisable to not erase any parts of the blockchain history. Another advantage in account-based models is the reduced implementation complexity because of the aforementioned optimization problem in UTXO-based models. A significant drawback of the account-based model is the resynchronization of the state after a system crash. Techniques to deal with this drawback are elaborated in Chapter 6. After a careful trade-off analysis for the use case at hand, it was decided that the lower implementation complexity of account-based models outweighs the potential benefits of the UTXO-based model. Thus, the data model implemented in Bazo is account-based. An account object has the following fields:

**Address** The address of the account is the public key of an elliptic curve key pair.

**Balance** The balance represents the amount of Bazo coins residing in the account.

**Transaction Counter** The transaction counter is used to prevent replay attacks. This counter is incremented in the sender’s account for every transaction that involves the transfer of coins.

\(^1\)The Bitcoin blockchain amounts to more than 123 GB at the time of writing.
3.3 Transactions

Transactions are the basic building blocks for every state change in the system and the basis for blocks, a higher-level construct. The common denominator for all transaction types is a declaration of a fee and a signature. The fee payment is an incentive for the miner to include the transaction in the next block. The higher the fee payment, the faster a transaction gets validated if rational miners are assumed. The signature serves the purpose of authentication. The creator digitally signs the transaction with his or her private key. Bazo supports three transaction types which are described in the following subsections.

3.3.1 Account Creation

Participants in the system can not join the Bazo system voluntarily. Before a user can engage in the transfer of coins, a registered account is required. The account creation transaction type serves this purpose. A new account is initialized with a balance of 0 coins. This type consists of the following parameters:

**Header** Contains information about the type of account to be created. Additionally to creating user accounts, new root accounts can be created and removed depending on the encoded bitmap.

**Issuer** The hash of a root account’s public key.

**Address** The public key of the newly created account. The corresponding private key needs to be handed over to the user off-chain.

The transaction gets validated if the signature was signed with the private key of a root account. Account creation transactions are referred to as \textit{AccTx} for the rest of the thesis.

3.3.2 Transferring Funds

The transferring funds transaction type transfers Bazo coins from one account to another, more precisely the subtraction of coins from the sender account and the addition to the receiver account. The following fields make up this transaction:

**Header** This field is reserved for later use.

**Amount** The amount of Bazo coins to be transferred between accounts.

**Transaction Counter** The transaction counter of the sender’s account.

**From/To** The hashes of the public keys of both the sender’s and the receiver’s accounts.
3.3. TRANSACTIONS

A transaction of this type is validated if the following conditions are met: The transaction was signed with the private key of the sender, the sender’s account balance is equal or larger than the amount spent and the transaction counter of the transaction and the state match.

Several memory optimizations were discussed during the course of the thesis. In order to minimize the overall size for this transaction, the transaction size can be compressed the following way: Instead of using the full hashes of the sender and receiver accounts, only the first $n$ bytes are used. The remaining $32 - n$ bytes of both hashes are XORed with the first $32 - n$ bytes of the signature. This compression leads to recalculations in the validation of the transaction in case of conflicts in the first $n$ bytes of an address hash, but decreases the overall transaction size relative to $n$. For $n = 8$, the transaction size shrinks from 149 to 101 bytes. While this approach is technically feasible and was successfully tested, it obstructs the modularity of the system because the hash of the transaction can only be calculated by accessing the state. This led to a strong coupling among different packages. In light of the negative effects on the design it was decided to not use this optimization technique.

Transferring funds transactions are referred to as $FundsTx$ for the rest of the thesis.

3.3.3 System Parameters

The rationale behind this transaction type is that system parameters can be changed without a hard fork. Bad design decisions during development and unforeseeable circumstances can thus be corrected. The following system parameters are subject to change:

- **Block Size** This parameter stands for the maximum block size (in bytes). Received blocks that are larger than this parameter get rejected by the miners.

- **Difficulty Interval** For stability reasons the difficulty of the PoW should be relative to the hashrate of the Bazo system. This parameter indicates the amount of blocks that are to be validated before a new target value is calculated.

- **Block Interval** The block interval is a time parameter that describes how much time (in seconds) shall pass between two blocks.

- **Fee Minimum** To incentivize miners for their mining work, the minimum fee that a transaction needs to pay can be set.

- **Block Reward** To further boost miner incentives, an additional block mining reward can be set with this parameter.

As with the account creation transaction type, a transaction of this type is only valid if was signed by a root account. System parameters transactions are referred to as $ConfigTx$ for the rest of the thesis.
3.4 Blocks

Transactions themselves do not change the state of the system directly when they are broadcast and received in the system. Instead, they are combined in a block and the validation of blocks lead to the state changes. The main reason for having blocks instead of just transactions is system stability and consensus. Committing state changes on transactions directly upon their arrival and validation leads to an increased likelihood of chain forks relative to the transaction throughput. As an example, Bitcoin [7] has a fixed block interval of 10 minutes on average, Litecoin [15] (a Bitcoin fork) uses a less conservative measure of 2.5 minutes on average and Ethereum [8] adds new blocks to the blockchain every 15.4 seconds on average. The trade-off can be stated as follows: Increasing the interval for state changes leads to transaction validation delay but decreases the likelihood for chain forks, while decreasing the interval leads to faster transaction validation with the increased chance of potential chain forks.

A block in Bazo consists of the following fields:

**Hash** The block hash acts as a unique identifier of blocks within the blockchain.

**Previous Hash** This value is equal to the identifier of the previous block in the blockchain.

**Nonce** The nonce is set to a value such that the resulting hash fulfills the PoW requirements.

**Timestamp** Refers to the block creation time (seconds elapsed since January 1, 1970 UTC).

**Merkle Root** The value of the merkle tree’s root node.

**Beneficiary** The address hash of the account that receives fee payments and the block reward.

**Number of FundsTxs/AccTxs/ConfigTxs** Corresponds to the number of transactions of each type that are included in the block.

**Hash Data FundsTxs/AccTxs/ConfigTxs** The hashes of all transactions included in this block in sequential order.

The hashes of the transactions within a block are indistinguishable. Therefore, it is necessary to include the number of each transaction type separately. Contrary to Bitcoin, only the transactions hashes are included in the block instead of the transaction payload itself. This is to keep the block size as small as possible.

---

2Source: https://etherscan.io/chart/blocktime
A block is considered valid and appended to the blockchain if the following conditions are met:

- All included transactions must be structurally valid and their sequential execution must lead to legal state changes (according to Section 3.3.1, 3.3.2 and 3.3.3)
- There must be no transaction duplicates within the block.
- There must be no transaction that was already validated in a previous block.
- The block must belong to the longest chain at the time of validation.
- The timestamp of the block must be within a predefined timerange of the validation time.
- The block size must not be larger than the current block size parameter.
- The beneficiary account must exist in the state.
- The merkle root must be correctly calculated.
- The PoW must satisfy the consensus properties.

### 3.4.1 Merkle Tree

A widely-used data structure in blockchains to simplify and scale transaction verification is the merkle tree [16]. As depicted in Figure 3.2, a merkle data structure is a tree, in most cases implemented as a complete binary tree with the leaves representing the transaction hashes. $H_Z$ is the hash function $H$ applied to the input $Z$ with $Z = A, B, ..., P$. If the number of transactions is not a power of 2, the last transaction is repeated multiple times such that the total number of leaves is a power of 2. The parent of two child nodes is the hash function applied to the concatenation of the child nodes: $H_{AB} = H(H_A + H_B)$. This process is repeated until the root hash is calculated.
The root hash of the merkle tree is included in the block itself. This provides two advantages: Block integrity and transaction verification. Because all transactions in the block are part of the merkle root hash calculation, it is neither possible to add or remove transactions to or from the block nor to change the ordering of the transactions inside the block without also changing the merkle root. However, the integrity property can also be accomplished by hashing every transaction without a tree structure. The advantage of the tree structure is faster verification. Consider Figure 3.3 as an example: A user wants to know whether his or her transaction $K$ was included in a block. To verify the inclusion of the transaction as part of the merkle root, the following nodes are requested from other peers in the network in sequential order: $H_L$, $H_{IJ}$, $H_{MNOP}$ and $H_{ABCDEFGH}$. The nodes $H_{KL}$, $H_{IJKL}$, $H_{IJKLMNOP}$ and finally the root hash are recalculated based on the received nodes. If the hash function is pre-image resistant (non-invertible) [18], it is practically impossible to send fake nodes and not changing the root hash of the recalculations. Therefore, if the recalculated root matches the merkle root hash of the block, the transaction is in the block (this process is also referred to as finding a merkle proof for a given transaction hash). The opposite case, however, is not true: If the merkle root hash does not match, it is not certain that the transaction was not in the block because nothing prevents malicious peers from sending bogus tree nodes when being queried for a merkle path.

![Figure 3.3: Merkle Path for the node $H_K$][17]
Chapter 4

Implementation

The following sections cover several implementation details about the Bazo system. First, a high-level overview of the software architecture is presented, followed by the closer examination of the storage and network layer. The consensus protocol is then explained in more detail. Furthermore, it is explained how Bazo deals with the problem of transaction malleability. A miner application programming interface (API) definition for future client applications follows. Then, Bazo’s concurrency system is elaborated before finishing the chapter with explaining the testing approach for the Bazo miner application.

4.1 Software Architecture

The mining software is modular with special emphasis on high cohesion and loose coupling. The responsibilities are logically split according to well-defined components. Figure 4.1 shows the package-level granularity of the architecture and the interconnection among the packages. The following paragraphs describe the high-level design of the packages and their context within the application.

The Protocol package consists of all building blocks for Bazo. It defines a transaction interface, allowing an abstract treatment of transactions. The package also includes the structure, encoding, decoding and hashing of all transactions types, blocks and accounts.

The responsibilities of the Miner package is everything related to mining operations. Tasks include the validation and consolidation of incoming transactions into blocks, the creation and validation of blocks, the calculation of the PoW and the building of the merkle tree. The validation logic of blocks is also handled in the Miner package. Also, the blockchain as a whole is maintained which includes tasks such as rollback operations and state changes. These responsibilities require access to all Bazo models in the Protocol package. In order to broadcast and request transactions and blocks to and from the network and to signal storage-related operations, access to the P2P and the Storage packages is required.

Networking operations are implemented in the P2P package. This includes the connection setup and maintenance to other peers in the Bazo network. Additionally to the
broadcasting, requesting and receiving of transactions and blocks, several services run to maintain system health and enable time synchronizations. The package interacts with the Storage module to read and write Bazo elements that are requested and received by other network participants. It also accesses the Protocol package to encode, decode and hash transactions and blocks.

The Storage package manages memory-related tasks. Bazo was first implemented as an in-memory application but a disk-based key/value database was added to increase scalability. The encoding before writing to the storage and the decoding after reading from storage as well as generating the hash as a key for all Bazo elements requires access to the Protocol package. The lower-level functionality of the storage is implemented with the external BoltDB package.

BoltDB [19] is an embedded key/value database with the goal of providing a simple, fast and reliable database for projects that do not require a full database server such as Postgres or MySQL. It has been designed with simplicity as a top design priority, consisting of an API that is small and focuses on getting and setting values.

4.2 Storage

This section deals with all storage-related properties of the Bazo system. Firstly, the motivation for a key/value storage paradigm is discussed. Secondly, the storage hierarchy and the need for disk-based storage is explained.
4.2. STORAGE

4.2.1 Key/Value Structure

The storage paradigm used in Bazo is a key/value-based system. The key represents the hash and the value is the payload of the Bazo element being stored. A key/value tuple $T$ can be formalized as follows:

$$T = (H_E(E), E)$$  \hspace{1cm} (4.1)

$H_E$ is the hash function defined on a Bazo element $E$, which is calculated by applying the hash function SHA3-256 [20], the newest secure hash algorithm in the SHA series. The motivation for using SHA3 over alternatives as Bazo’s hash function is its usage in Ethereum [8]. The hash function is applied over the sequential concatenation of a predefined set of encoded fields $e_i$ for a Bazo element $E$, ($\circ$ is the string concatenation operation). In case of an account element, the hash function is defined as the SHA3-256 hash over the elliptic curve digital signature algorithm (ECDSA) public key of the account:

$$H_{Account} = SHA3-256(pubKey)$$  \hspace{1cm} (4.2)

Transaction hashes of all types cover all fields of the transaction except the signature, as shown in Equation 4.3. This prevents a bug called transaction malleability (cf. Section 4.5)

$$H_{Transaction} = SHA3-256(e_1 \circ e_2 \circ ... \circ e_{|E|}), e_i \neq signature$$  \hspace{1cm} (4.3)

The calculation of a block hash deviates from Equations 4.2 and 4.3. This is because the PoW is encoded in the block hash itself. Thus, the hash calculation cannot be the hash function applied over the concatenation of all block fields. A formal description of the block hash calculation follows in Section 4.4.1.

The technical reasons for choosing this paradigm are unique identification, scalability and data integrity. The hash function for every Bazo element was chosen that it is practically impossible for two distinct objects to have the same hash value. Transaction types for which equal transactions exist (e.g., FundsTx; Account A sends Account B an amount of $X$ Bazo coins multiple times in a row), an additional Transaction Counter (cf. Section 3.3.2 and 3.3.3) field is included in the hash calculation to achieve a unique hash for all transactions. The uniqueness property prevents against replay attacks and simplifies the software logic, because incoming transactions and blocks that match an already verified and validated transaction hash in storage can simply be rejected.

The storage paradigm improves scalability. This is due to the fact that the transaction themselves are not included in a block, they are only referenced by their hash which is 32 bytes long. The storage savings in case of a AccTx with a fixed size of 169 bytes is substantial with a factor of more than 5. This compression allows more transactions to be included in a block. Another factor is reduced redundancy: Because the payload of transactions are broadcast prior to a block that includes them, it is not necessary to include transaction payloads in blocks as well. In case new miners join the system, all unknown transactions included in received blocks are requested from the network before the block can be validated.

To use cryptographic hashes provides the additional property of data integrity. It is impossible for any attacker to change the contents of a transaction without also changing
its hash. Another desirable property is that a hash is unique for the semantics of a transaction, e.g., there must not be differing transaction hashes for two equivalent state transitions. A bug in the Bitcoin system makes it possible for an attacker to modify the transaction signature and thus creating a different transaction hash without changing its validity. This bug is referred to as Transaction Malleability [21] and is solved in Bazo (cf. Section 4.5).

4.2.2 Storage Hierarchy

Bazo was first developed as an in-memory solution, keeping all data in volatile storage. However, to increase scalability and to protect against system failures, a disk-based storage solution was incorporated. Bottlenecks and time-critical parts of the system were analyzed to decide which data is mapped to which storage medium.

In-Memory

A cache is implemented to store frequently used data to boost performance and make operations critical to mining faster. The state of the system consisting of all accounts is stored in random-access memory (RAM). Thus, verifying blocks with a large amount of FundsTxs becomes significantly faster. In case of bootstrapping the system and downloading the history of potentially thousands of blocks from neighboring peers, incorporating an in-memory storage solution leads to a substantial difference in verification time. Storing the state in RAM is technically feasible, because the memory footprint of a single account is fixed at 76 bytes, e.g., it takes 1 gigabyte of storage to store more than 13 million Bazo accounts. Because Bazo is an invite-only blockchain and account creation needs to be authorized, this number is highly unlikely to be surpassed.

Additionally to keeping all accounts in-memory, all transactions that were received but not validated by a block (open transactions) are kept in RAM as well. There are two reasons for this. Firstly, open transactions will be reused in the near future and will be evicted from RAM as soon as they have been successfully validated. Thus, there is an upper bound on memory used, limited by the block interval (cf. Section 3.3.3) multiplied by the average transaction frequency. Secondly, mining blocks can be seen as a competition among miners. The first miner to broadcast a valid block receives the block reward and the transaction fees of all transactions included in this particular block. Therefore, it is advantageous to increase the throughput of adding open transactions to a new block and start to calculate the PoW. The difference in delay is relative to the number of open transactions.

Disk-based

Whenever a block is successfully validated, all transactions in the block and the block itself are written to disk. This increases scalability and fault-tolerance. All data that is
kept in RAM is bound by system parameters and the properties of Bazo (e.g., private blockchain). On the other hand, the blockchain itself grows in time as new blocks are periodically generated even in the absence of transactions, i.e., there is no upper bound in terms of memory consumption. Thus, storing historical blockchain data on disk-based storage makes Bazo more scalable. It also increases fault tolerance as data stored in RAM is lost in case of power outages or other system failures.

The following disk-based key/value storage systems were evaluated: RocksDB [34], Badger [36], Kyoto Cabinet [35] and BoltDB [19]. Kyoto Cabinet and RocksDB are both written in the C++ programming language, which require Cgo to run within the Go programming language, negatively impacting performance for reasons explained in [36]. There exist bindings for Go, but they have little community support and are not well tested. The decision between Badger and BoltDB was made in favor of BoltDB because of better community support, simplicity and interoperability. BoltDB provides buckets, which are a collection of key/value pairs within a database. Bazo employs different bucket to logically split the data:

Open Blocks Blocks that were received but not validated, i.e. open blocks, are stored in this bucket. The reason to store open blocks on disk and not in RAM is to simplify the bootstrapping process, which will be explained in detail in Section 4.3.1.

Closed Blocks All validated blocks that are part of the valid blockchain from the miner’s perspective, are stored in this bucket.

Closed FundsTx/AccTx/ConfigTx A distinct bucket for all closed transactions of each type exists. The reason to use a different bucket for each type is simplified decoding.

4.3 Networking

All communication channels in Bazo are based on the Transmission Control Protocol (TCP) [22]. Every miner listens on port number 8000 for incoming connections. The implemented communication paradigm is asynchronous, e.g., requests are non-blocking. The following subsections elaborate on the implementation details of the communication package, covering the peer-to-peer network, the syntax and semantics of Bazo messages as well as the different connection and message types.

4.3.1 Peer-to-Peer Network

Communication in Bazo is handled by a decentralized, unstructured ad hoc peer-to-peer (P2P) network. To disseminate transaction and block data in the network, a broadcast service is employed. However, a flooding approach does not scale with the number of connected peers in terms of network traffic. One alternative of using a flood-based dissemination mechanism is to introduce random walks [23] to reduce network traffic in the
system. The mechanism implemented in Bazo is to define a threshold in the amount of
connected miners. As soon as the number of connected miners reaches this threshold,
subsequent incoming connections trying to perform a handshake 4.3 are rejected. This
limits the network traffic and improves scalability.

At startup a miner needs to bootstrap the system. This is accomplished by querying
a list of hard-coded miner IP addresses and request a list of neighboring miners from
each of them. In order to avoid network clustering and centralization the process of
which IP addresses to connect to is randomized. This is achieved by requesting the
full neighbor list of a miner and establishing connections to a randomly chosen subset.
Every connection among two miners is kept open to avoid reconnection overhead. A
connection timeout is avoided by sending periodic timestamp broadcasts to all connected
peers. After bootstrapping, a service is constantly running in the background to assert
that a minimum amount of connections to other miners are established. This is to protect
against connection drops from other miners and to keep the peer well-connected.

4.3.2 Message Format

![Bazo Message Diagram]

Figure 4.2: The structure of a Bazo message: Header and Payload

Every Bazo message that is transmitted over the network includes a header and an optional
payload, as depicted in Figure 4.2. The header has a constant size of 5 bytes and consists
of the following fields:

**Length** The length of a Bazo message (excluding the header itself) is encoded in a 4
bytes integer and the encoded number refers to the amount of bytes of the following
payload. This allows a maximal payload of roughly 4.3 gigabytes in size.

**Msg Type** The message type is the description of the message and its payload. Messages
can generally be split into requests, responses and broadcasts. A total of 256 distinct
message types are possible with an encoding of 1 byte.

The payload contains the actual data of the message. Its contents are Bazo elements such
as blocks and transactions or other administration-related messages (e.g., neighbor lists of
connected miners, timestamps etc.). The full list of messages can be found in Appendix
A.
4.3. NETWORKING

4.3.3 Connection Types

Bazo is split into two applications with different functionalities. The miner application is responsible for verifying and validating transactions and blocks, communicating with other miners in the network to reach consensus and accepting new connections. The system is fed with new transaction data by the client application. The API offered by the miner for client connections is described in detail in Section 4.6.

The network connection types among the two applications are handled differently in Bazo. Connections are possible among miners themselves, between miners and clients. Because the client application does not validate nor broadcast transactions and blocks, it does not listen to incoming connections. Therefore, clients do not connect to other clients. Figure 4.3 shows the connection workflow from the perspective of a miner. Every miner is listening for incoming connections. The signature for a miner to identify himself is to send an initial message with Msg Type = MINER_PING. The receiving miner responds with a message with Msg Type = MINER_PONG and both parties add the connection to their internal miners list, which is a list of all miner connections. However, whether the receiving end concludes the handshake depends on the miner’s internal miners list. If a threshold for the maximum amount of miner connections is reached, the connection is dropped.

There is no branching from the client’s perspective: As soon as the connection is established, the message consisting of the payload is sent from client to miner and upon receiving the response, the connection is closed.

![Figure 4.3: Miner perspective: Connection workflow](image-url)
4.3.4 Message Types

Message types in Bazo can be distinguished between broadcasts and requests/responses. For the latter type, the process is asynchronous from the perspective of the request issuer. This way all incoming messages have the same entry point and the messages can be distinguished by the `Msg Type`. From the responding miner’s perspective, the process is synchronous, e.g., the request is forwarded to the corresponding routine which generates the payload and responds immediately.

The workflow of the broadcast service is depicted in Figure 4.4. There exist four broadcast messages which refer to distinct `Msg Types`. These are block and transaction broadcasts for all transaction types. Whenever a miner receives a broadcast message, it checks whether the message can be properly decoded according to the block and transaction decoding rules. The message is discarded if this process fails. If it succeeds, however, both open and closed storage modules are queried whether the same Bazo element was already received. This is to avoid that a messages does not get broadcast indefinitely but exactly once. In case the message was not previously received, it is broadcast to all connected miners.
4.4 Consensus

Informally, the goal of a consensus protocol is to achieve agreement among a majority of participants on the state of the system. Translated to blockchain technology, it means that a majority of network participants agree on which chain is the valid chain. This requires that a majority of miners follow the same protocol. This is achieved by incentives in the form of transaction fees and block rewards. It is therefore in a rational miner’s best interest to contribute his mining effort to the valid chain, assuming there is no threat of a 51% attack [6].

4.4.1 Proof-of-Work

The Bazo consensus protocol is PoW-based. For a new block to be accepted by the network, it needs to fulfill certain requirements, which have been elaborated in Section 3.4. One requirement is a valid PoW, which acts as proof that miners utilized a certain amount of computational power on average in the generation of the block. The PoW employed in Bazo is to compare the $SHA3$-$256$ hash function applied to the concatenation of $g(block)$ and the nonce field against a defined difficulty level. The function $g(block)$ is defined in Equation 4.4: It is the $SHA3$-$256$ hash function applied to a subset of the blocks’ fields. It excludes the nonce field, which is needed for the calculation of the PoW, it also omits all transaction data because they are represented in the merkle root.

$$g(block) = SHA3$-$256(PrevHash \circ Header \circ Timestamp \circ MerkleRoot \circ Beneficiary)$$  \hspace{1cm} (4.4)

Equation 4.5 evaluates to boolean true or false, depending on whether the inequality is fulfilled or not. The parameter target refers to the amount of leading zero bits in the resulting hash.

$$f_{target}(g(block), nonce) = SHA3$-$256(g(block) \circ nonce) < 2^{256-\text{target}}$$  \hspace{1cm} (4.5)

The final block hash is defined in Equation 4.6 as the hash function applied to the concatenation of $g(block)$ and the nonce. The block hash is valid if it passes the inequality test of Equation 4.5.

$$H_{block} = SHA3$-$256(g(block) \circ nonce)$$  \hspace{1cm} (4.6)

The target value is dynamically recalculated based on the system parameters difficulty interval and block interval. It is assumed that the hash function $SHA3$-$256$ provides the necessary randomness that makes it impossible to guess whether a position within a hash is a binary zero or one before calculating the hash. The probability that a randomly calculated hash $X$ has target leading zeros is thus:
This can be modeled as a Geometric Distribution with the success probability \( p = 2^{-\text{target}} \). Thus, the expected amount of failures before a success is \( E(X) = (1 - p)/p \).

After the number of blocks specified in the difficulty interval parameter have been added to the blockchain, the target is recalculated, based on the ratio of the measured average block interval and the parameter block interval.

### 4.4.2 Competing Chains

The PoW requirement described in the previous section states a condition a block must fulfill to be accepted as a valid block. However, this is not sufficient to achieve consensus because it is not yet determined which chain is valid in case of forks. There are two options to deterministically encode rules for the valid chain:

1. **Number of blocks**: A miner takes the first received block that fulfills all conditions as the valid blockchain and rejects all blocks belonging to a chain that is not longer in terms of total amount of blocks.

2. **Total amount of work**: If a block is received that belongs to a chain which is not longer in terms of total amount of blocks but the chain utilized more computational work to produce all blocks, this chain is used as the valid chain. This is evaluated by comparing the numerical representation of the block hashes and taking the smaller number.

Using the first heuristic leads to the fact that transactions in the last block are less likely to be invalidated by a future block, because miners stop mining operations on a competing block as soon as they receive a valid block that belongs to a chain which is longer than the current chain. Assume for example that the average time between two blocks is 5 minutes. If the total amount of work is used as a heuristic, a miner has an incentive to continue mining after receiving a block and the chances of finding a block with slightly larger difficulty in the next 5 minutes is high. The miner can set the same target difficulty in its PoW calculation and the chances that the calculated hash shows a higher level of difficulty is 50%. To increase this likelihood further, the miner can use a slightly larger difficulty parameter in its PoW calculation. On the flip side, however, implementing the heuristic of the total amount of blocks has the disadvantage of a temporary chain fork in the case when two or more blocks were mined and broadcast in different parts of the network at around the same time. The higher likelihood that a transaction in a valid block remains valid outweighs the risk of a temporary chain fork, which is the reason that the total amount of blocks heuristic is implemented in Bazo.

In case a peer missed one or more blocks (the whole blockchain in case of a bootstrap), he or she needs to be able to synchronize and request missing blocks and their contents from
the network. The pseudocode in Algorithm 1 takes as input a new block and determines
the longest chain while handling the case of missing blocks.

**Algorithm 1:** Determination of the longest chain

<table>
<thead>
<tr>
<th>Input: newBlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. tmpBlock ← newBlock</td>
</tr>
<tr>
<td>2. currentChain ← (genesisBlock, ..., currentBlock)</td>
</tr>
<tr>
<td>3. while tmpBlock ∉ currentChain do</td>
</tr>
<tr>
<td>4. if ¬ prev(tmpBlock) then</td>
</tr>
<tr>
<td>5. return</td>
</tr>
<tr>
<td>6. else</td>
</tr>
<tr>
<td>7. tmpBlock ← prev(tmpBlock)</td>
</tr>
<tr>
<td>8. end</td>
</tr>
<tr>
<td>9. end</td>
</tr>
<tr>
<td>10. ancestor ← tmpBlock</td>
</tr>
<tr>
<td>11. currentBranch ← (ancestor, ..., currentBlock)</td>
</tr>
<tr>
<td>12. newBranch ← (ancestor, ..., newBlock)</td>
</tr>
<tr>
<td>13. if length(currentBranch) ≥ length(newBranch) then</td>
</tr>
<tr>
<td>14. return</td>
</tr>
<tr>
<td>15. end</td>
</tr>
<tr>
<td>16. reverseBranch ← reverse(currentBranch)</td>
</tr>
<tr>
<td>17. chainRollback(reverseBranch)</td>
</tr>
<tr>
<td>18. chainValidate(newBranch)</td>
</tr>
</tbody>
</table>

The current chain is initialized to the list of all blocks starting from the genesis block to
the last and current block. The while loop (lines 2–8) iterates through the chain backwards
to find the common ancestor. In case the block is missing from the internal storage, it is
requested from the network. If a previous block cannot be requested it is assumed to be
inexistent and the algorithm stops. If, however, a common ancestor is found (in case of
a rollback, the genesis block), a current branch consisting of the list of blocks that range
from the ancestor to the current block and a new branch that consists of all blocks that
range from the ancestor to the new block are built (lines 10–11). If the current branch
is larger or equal than the new branch, the algorithm stops (lines 12–14). Otherwise, all
blocks are rolled back in the opposite direction (lines 15–16) and the new chain is built,
validating block by block from the new branch (line 17). One possible case of execution
is pictured in Figure 4.5 in which a new block is part of a chain that counts more blocks
than the current chain and gets thus accepted as the new valid chain.

### 4.5 Transaction Malleability

Transaction malleability, specifically signature malleability, refers to a specific bug in
popular cryptocurrencies in which the transaction hash can be changed by modifying the
signature without invalidating the transaction. This leads to several attack vectors [25]
and the famous Japanese-based bitcoin exchange MtGox claimed that transaction malle-
ability was the reason for the loss of over 850,000 bitcoins [24]. However, this was refuted
by a careful analysis of the involved transactions [21].

Bitcoin’s reference implementation uses OpenSSL for signature validation. OpenSSL accepts several signature formats. But because the signature of a transaction is part of the hash calculation, different signature representations lead to different transaction hashes. Several attempts to fix the issue emerged: Bitcoin Improvement Proposal (BIP) 62 [26]: *Dealing with malleability* introduced canonically encoded ECDSA signatures (cf. BIP 66 [27]) among other rules. However, it was rejected due to the implementation complexity. Another approach is *Segregated Witness* (BIP 141 [28]). It decouples signature data from actual transaction data and, thus, makes unintentional signature malleability impossible. *Segregated Witness* has been successfully implemented in Litecoin [15]. Preventing transaction malleability in Bazo is crucial because the blocks consist of transaction hashes instead of the payload. Transaction malleability can thus be exploited by miners to include two versions of the same transaction with differing transaction hashes within the same block. Transaction malleability has been solved in Bazo by decoupling transaction and signature data. As seen in Equation 4.3, the signature is not part of the hash calculation. This implies that the signature format can be changed without changing the hash of the transaction. All signatures in Bazo use the ECDSA curve $P_{-256}$ (FIPS 186-3, section D.2.3 [29]) and the signature pair $(r,s)$ is saved in a fixed byte array of size 64 bytes.

### 4.6 Miner-Client API

The Bazo system consists of two applications that interact with each other, namely the *miner* application which validates transactions and blocks and the *client* application that interacts with the Bazo system by creating transactions. A clearly defined API definition is thus needed that can be implemented by future client applications.
4.6. MINER-CLIENT API

4.6.1 Network

Before sending the transaction into the network, connections with miners need to be established. Theoretically, it is enough to connect to just a single miner to send the transaction to. The miners themselves have an incentive to broadcast all incoming transactions to the network because all transactions need to be known by all miners if they are included in a block that needs to be validated by the network. However, to increase the likelihood that transactions get disseminated and eventually end up in a block, connections to more miners can be established.

Two following two IP addresses are bootstrapping servers that act as an entry point for all miner and client applications:

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.41.136.199</td>
<td>Primary IP Address</td>
</tr>
<tr>
<td>13.80.151.135</td>
<td>Secondary IP Address</td>
</tr>
</tbody>
</table>

Bootstrapping servers listen to incoming TCP connections on port number 8000. If more miner IP addresses are needed, a request is made to either of the IP addresses with the syntax depicted in Figure 4.6. The encoding for all messages is Big Endian. $\text{Msg Type} = 0x1F$ corresponds to the $\text{NEIGHBOR_REQ}$ type (cf. Appendix A).

![Figure 4.6: Requesting neighbor IP addresses from a miner](image)

The subsequent response is a list of IP:Port tuples formatted according to Figure 4.7. At this point only IPv4 addresses are supported. However, this is extendable by prepending the number of each version type (e.g., IPv4 and IPv6) before listing the addresses of each type.
Upon receiving the response to the IP neighbor request (\textit{Msg Type = 0x29}), the IP addresses can be extracted in the format which are encoded as 6 bytes. The IP address is encoded in the first 4 bytes as \textit{IPA.IPB.IPC.IPD}. The port number is 2 bytes in size and encoded as \textit{PortA.PortB}. Subsequent connections can then be established to the received IP:Port tuples.

![Figure 4.7: Response to a IP neighbor request](image)

### 4.6.2 Transactions

Regular users can send exactly one transaction type, namely \textit{FundsTx}, which consists of the following fields:

```go
// type FundsTx struct {
    Header byte
    Amount uint64
    Fee uint64
    TxCnt uint32 // Transaction Counter
    From [32] byte
    To [32] byte // Signature
    Sig [64] byte
}
```

The semantics of the fields were explained in Section 3.3.2. The \textit{signature} is calculated by concatenating all fields until \textit{To} (the hash of the receiver account) and hashing it with the \textit{SHA3-256} hash function. The first element of the signature pair is stored in the first 32 bytes and the second element is stored in the last 32 bytes the \textit{Sig} array. All fields are then sequentially encoded in \textit{Big Endian} order and packed into a Bazo message depicted in Figure 4.8. The length of a \textit{FundsTx} is fixed at 149 (0x95) bytes and the message type for broadcasting a \textit{FundsTx} is 0x01.

Transaction verification by means of a merkle path (cf. Section 3.4.1) and specific implementation details are mentioned in Section 6.
4.7 Concurrency

The Bazo miner application is inherently concurrent. This section introduces the concurrent structure that is important to understand the underlying workflows. The Go programming language has a rich support for concurrency using goroutines and channels. On a high-level, a goroutine is a function that is capable of running concurrently with other functions. The first routine is implicit and consists of the main function itself. A new goroutine is called by prepending the function name with the go keyword.

Channels in the Go programming language are the connections between two or multiple goroutines and provide means of synchronization. The concurrency model in which values are passed between independent goroutines is called Communicating Sequential Processes, this is in contrast to the memory sharing in the Shared Memory Multithreading model which is implemented with threads in mainstream programming languages [30].

In Bazo, communication and synchronization among different packages is handled with channels, whereas memory sharing is used in the networking package (cf. Section 4.3). Figure 4.9 depicts the different goroutines in a running application and the packages they originate from:
Calculating PoW This is the initial goroutine that is implicitly started with the main function. Calculating a valid PoW is done constantly.

Miner-Networking Interface There is bidirectional communication line between the miner and the networking package. The networking package uses different channels to transfer blocks and transactions from the network to the miner and the miner clears blocks and transactions for broadcast which are channeled from the miner to the networking package.

Broadcast Service Whenever a miner clears blocks and transactions for broadcast, a service iterates through all connected neighbors and channels the data to the different goroutines.

Network Health Service Connections might drop due to various reasons. This goroutine observes all maintained connections and establishes new connections when the number of neighbors fall below a predefined threshold.

Time Service This service periodically broadcasts the local time to each connected miner. Time synchronization is needed to reach consensus on the validity of recorded timestamps in new blocks.

Listener This goroutine is the entry point for all miners and clients. It listens to all incoming TCP connections on port number 8000. As soon as a new connection is established, a new goroutine is spawned.

New Connection Each new connection is handled by its own goroutine. As explained in Section 4.3, connections to other miners are kept open (up to a predefined threshold) while connections to clients are aborted after the reception of a message.

4.8 Testing

The Bazo miner application was thoroughly tested across all packages, leading to a good overall code coverage. The chosen testing approach differs between packages, the following list explains the specific approaches for each package.

Protocol Tests in this package covered the serialization and deserialization of all Bazo entities, e.g., accounts, transactions and blocks. Simple variable assignments, trivial if conditions and the functions concerned with a textual representation of the respective entities are not tested. The test coverage of this package is 86.1%.

P2P The P2P package handles all networking-related tasks of Bazo. Code that interfaces with other packages cannot be properly tested with unit tests. The same is true for dynamic networking tests, leading to a coverage of 42.1%. Interconnection among different packages as well as the parts involving dynamic networking are to be tested with integration tests.
4.8. TESTING

Miner The unit tests in this package cover the core functionality of Bazo, e.g., the verification and validation of transactions and blocks, the formation of a chain and its state changes, rollbacks in case of chain forks, difficulty adaptations relative to the networking hash power, PoW logic and merkle tree calculation. Trivial cases and code that interfaces with the networking package, e.g., fetching unknown transactions or blocks from the network, were not tested. Test coverage for this package is 75.1%.

Storage The unit tests written for this package cover both in-memory and disk-based storage operations for all read, write and delete operations of accounts, transactions and blocks. Testing low-level read and write operations for disk-based storage has been done extensively by BoltDB itself [19] and is not repeated. Test coverage for this package is 87.5%.
Chapter 5

Evaluation

Security considerations directly influenced the design of the Bazo protocol and a range of attacks are mitigated by the employed countermeasures. The following section describes the attack scenarios and what countermeasures are installed in Bazo to mitigate them, specifically: Sybil attacks [37], replay attacks [38] and denial of service attacks [39]. Furthermore, the implications of quantum computers and the potential weaknesses in the cryptographic hash function SHA3 are discussed.

5.1 Security Considerations

In computer security, a sybil attack [37] refers to the forging of identities in a peer-to-peer network with the aim of gaining an advantage or causing system malfunctioning. The design of Bazo’s consensus protocol, specifically its PoW, renders multiple identities useless, because the average hash rate of a miner is the hash rate of a single system divided by the amount of hashing operations or identities that are being used. However, multiple identities can be used to occupy the connection list of other miners. Attackers can then stop broadcasting transactions and blocks to those miners which can lead to peer isolation. Malicious peers of this kind are not actively detected, but the fact that joining peers connect to random peers in the network and that connection lists of miners have an upper limit in the number of connections makes it more difficult to force peer isolation. A related attack, called 51% attack, takes place, when malicious peers control more than 50% of the network hash rate. They can then rewrite history which leads to double spending attacks and a financial gain for the attackers. Because the network hash rate is not directly controllable and an inherent part of a PoW consensus protocol, protection against a 51% attack is difficult.

Replay attacks [38] are a form of network attacks with the goal of maliciously repeating data transmissions. For example, assume that Alice wants to send 10 Bazo coins to user Bob. Bob can then just repeatedly rebroadcast the same transaction, effectively draining the account balance of user Alice. In Bazo, this is protected by rejecting duplicate transaction hashes. A block is not validated if it consists of two identical transactions or
a transaction that has been validated in a previous block. In the case that Alice wants to replay this transaction, she has to include the transaction counter (cf. Section 3.3.2) of her account in the transaction, leading to a different hash value for every subsequent transaction.

Denial of Service (DoS) [39] attacks aim to make a peer or the network as a whole unavailable by disrupting it. The most common example is to flood the victim or the system with a large amount of traffic. Bazo mitigates this by only broadcasting valid transactions and blocks at most once and limiting the number of connected peers.

Another rising threat is the development of quantum computers. Many asymmetric cryptosystems (including ECDSA used in Bazo) are not quantum-resistant, which means that the encryption can be broken in a reasonable amount of time by computing the private key from the public key [40]. It would then be possible to crack the private keys of Bazo’s root accounts, thereby allowing the attacker to issue an unbounded amount of money, which would pose unsurmountable challenges. Designing quantum-proof blockchains is an ongoing research topic [41, 42].

Furthermore, attacks on Bazo’s cryptographic hash function, SHA3-256 have to be considered. Vulnerabilities exploited by cryptanalytic attacks on cryptographic hash functions can be generalized into three distinct categories [43]. Pre-image resistance: Targets the one-way function property, e.g., given hash function $H$, applied on the message $m$ ($H(m)$), it is infeasible to calculate $m$. Second pre-image resistance: Given message $m_1$ and the hash function $H$, it is infeasible to find $m_2$, such that $H(m_1) = H(m_2)$. Collision resistance: Given hash function $H$, it is infeasible to find two messages $m_1$ and $m_2$ such that $H(m_1) = H(m_2)$. If $SHA3-256$ is not pre-image resistant, the input of a hash with desirable difficulty can be determined. If this input additionally adheres to the structural properties of a block, the PoW is broken because the nonce does not need to be exhaustively calculated. If collision resistance, in the specific case of second pre-image resistance, is broken, it is possible for an attacker to create a block with the hash of an already existing block. If this new block adheres to the structural properties of a block but differs from the block with an equal hash, state consistencies are the result. Replacing the vulnerable hash function with a secure hash function requires a hard fork.
Chapter 6

Future Work

6.1 Consensus Mechanism

The consensus mechanism employed in Bazo is a classical PoW which is very similar to Bitcoin’s consensus mechanism. PoWs that are based on the exhaustive enumeration of a target hash value promote the employment of application-specific integrated circuits (ASICs) which can be cheaply produced to achieve very high hash rates, easily outpowering general-purpose graphics processing units. This leads to mining centralization, which is counter to the core philosophy of decentralization. Ethereum developed its own PoW consensus model, called EthHash [32], whose goal is to be ASIC-resistant. However, a big drawback in every PoW consensus model is energy consumption. Proof-of-Stake (PoS) aims to overcome the high energy consumption disadvantage that is inherent to PoW consensus models. In PoS systems, the creator of the next block is chosen in a way that the chances that an account’s block is chosen depends to its funds (i.e., stake). A larger stake leads to statistically more valid block creations [31].

Replacing the existing PoW in Bazo with PoS has the advantage of reduced electricity. Also, due to the private nature of the Bazo network, limiting accounts to less than 50% of the total amount of Bazo coins (which makes a 51% attack in PoS feasible) can be controlled and implemented whereas control over the hashrate of the miner is lacking.

6.2 Simplified Payment Verification (SPV)

In order for a miner to validate transactions and blocks, state access is needed, e.g., all accounts in the Bazo system need to be accessible. To respond to transactions and block requests, the blockchain history needs to be stored as well. Section 3.2.2 introduced the possibility to remove past transactions. This allows a miner to still participate in the validation of transactions and blocks with significantly less storage requirements. Another optimization in terms of memory consumption is the simplified payment verification (SPV), a term coined by Bitcoin [7]. It allows clients to verify whether transactions were successfully included in a block without knowing the blockchain history. Section 3.4.1
introduced the concept of a merkle proof. In order to be certain that a client transaction is recorded in the blockchain, two conditions have to be met:

1. Valid merkle proof
2. Merkle root hash is part of the blockchain

To implement SPV in Bazo, the block structure needs to be refined according to Figure 6.1. All transaction-specific data is removed and the new structure is called *SPV Block*. It is fixed in size at 146 bytes. For the miner to support Merkle path requests, the storage structure has to be adapted to save the transaction hash alongside the merkle root hash of the block it was validated in to reduce search costs. The same optimization technique of erasing blockchain history can also be applied to have an upper bound on storage consumption. The introduction of bloom filters [33] for privacy-enhancing futures are a logical next step.

![Figure 6.1: SPV Block structure](image)

6.3 Checkpointing

The technique of checkpointing adds fault-tolerance to a system by saving snapshots at certain time intervals to faster recover in the case of system failure. To increase the fault tolerance of the Bazo system, two distinct forms of checkpointing can be implemented, namely state and block checkpointing. Because the state of the system (e.g., the union of all Bazo accounts) is stored in volatile memory, system failures causes this data to get
6.3. CHECKPOINTING

lost. Even though resyncing is fast because blocks and transactions are already stored on disk, the time to finish the process increases correspondingly with the blockchain length. Block checkpointing assures that block validation and rollbacks are atomic operations. This protects the system in the event of a system failure during multiple state updates. A similar point discussed in Section 3.2.2 is the upper bound of blockchain memory consumption to lower the barriers of entry for miners. However, there must be miners that store all transactions since the genesis block to respond to requests from new miners that validate the blockchain from the beginning. An interesting research question is whether and how snapshots can be issued as transactions on the blockchain such that old transactions can be effectively erased from all miners.
Chapter 7

Summary and Conclusions

The purpose of this thesis was the development of a private/invite-only blockchain-based cryptocurrency with the goal of substituting a traditional bonus system employed by a financial service company to incentivize the usage of its issued credit cards. Using a cryptocurrency instead of the centralized traditional system simplifies the process of value exchange, lowers the administration overhead, leads to the disintermediation of the company and allows direct payments between customers and partners as well as among customers themselves. Also, valuable insights can be drawn from the experimentation with blockchain-based applications, potentially paving the way for new business opportunities.

After a motivation of a cryptocurrency-based system as a replacement for the traditional system, the corresponding technical requirements were derived. The design for Bazo was then outlined based on these requirements, explaining the data model and the different entities in the Bazo system. The following chapter tackled implementation topics, such as the high-level software architecture for the miner application, the storage and network layer, the consensus protocol, the API for client applications, concurrency in Bazo, transaction malleability and testing. The evaluation chapter discussed security considerations for the Bazo system. Several topics for future work were then evaluated, which revealed interesting and novel research questions.

Bazo can be seen as a suitable system to substitute the traditional bonus system in place. The built-in modularity of the system offers many interesting ways to extend it by including other incentivization programs by other companies, each offering an exchange rate to Bazo coins. However, given the implications of successful exploitation, rigorous reviews need to be conducted before running Bazo in production. Also, resources should be invested in formally verifying security-related parts of Bazo.
Bibliography


## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>Practical Byzantine Fault Tolerance</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof-of-Work</td>
</tr>
<tr>
<td>PoS</td>
<td>Proof-of-Stake</td>
</tr>
<tr>
<td>UTXO</td>
<td>Unspent Transaction Output</td>
</tr>
<tr>
<td>FundsTx</td>
<td>Transferring Funds Transaction</td>
</tr>
<tr>
<td>AccTx</td>
<td>Creating Account Transaction</td>
</tr>
<tr>
<td>ConfigTx</td>
<td>Changing Configuration/System Parameter Transaction</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signature Algorithm</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>BIP</td>
<td>Bitcoin Improvement Proposal</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>SPV</td>
<td>Simplified Payment Verification</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Relationship of stakeholders in the traditional bonus system ................ 5
3.1 Proposed system with Bazo integration .......................................... 8
3.2 Merkle Tree consisting of 16 transaction hashes [17] ......................... 13
3.3 Merkle Path for the node $H_K$ [17] ............................................. 14
4.1 UML Package Diagram for the Miner Application ............................ 16
4.2 The structure of a Bazo message: Header and Payload ....................... 20
4.3 Miner perspective: Connection workflow ....................................... 21
4.4 Workflow after receiving a broadcast message .................................. 22
4.5 Choosing the longest chain in case of a chain fork .......................... 26
4.6 Requesting neighbor IP addresses from a miner ............................... 27
4.7 Response to a IP neighbor request .............................................. 28
4.8 Structure of $FundsTx$ broadcast .................................................. 29
4.9 Concurrent goroutines in the Bazo miner application ......................... 29
6.1 SPV Block structure ................................................................. 36
Appendix A

Message Types

Every message sent among peers consists of a header, as shown in Figure 4.6, and an optional payload. The header of a message consists of the length and the message type of the Bazo message. The message type is parsed by the receiver and processed accordingly. Table A.1 shows all available message types.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Numeric Representation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNDSTX_BRDCST</td>
<td>1</td>
<td>Broadcast FundsTx</td>
</tr>
<tr>
<td>ACCTX_BRDCST</td>
<td>2</td>
<td>Broadcast AccTx</td>
</tr>
<tr>
<td>CONFIGTX_BRDCST</td>
<td>3</td>
<td>Broadcast ConfigTx</td>
</tr>
<tr>
<td>BLOCK_BRDCST</td>
<td>4</td>
<td>Broadcast block</td>
</tr>
<tr>
<td>FUNDSTX_REQ</td>
<td>10</td>
<td>Request a FundsTx from a miner</td>
</tr>
<tr>
<td>ACCTX_REQ</td>
<td>11</td>
<td>Request an AccTx from a miner</td>
</tr>
<tr>
<td>CONFIGTX_REQ</td>
<td>12</td>
<td>Request a ConfigTx from a miner</td>
</tr>
<tr>
<td>BLOCKTX_REQ</td>
<td>13</td>
<td>Request a block from a miner</td>
</tr>
<tr>
<td>ACC_REQ</td>
<td>14</td>
<td>Request an account from a miner</td>
</tr>
<tr>
<td>FUNDSTX_RES</td>
<td>20</td>
<td>Respond with a FundsTx to the requester</td>
</tr>
<tr>
<td>ACCTX_RES</td>
<td>21</td>
<td>Respond with an AccTx to the requester</td>
</tr>
<tr>
<td>CONFIGTX_RES</td>
<td>22</td>
<td>Respond with a ConfigTx to the requester</td>
</tr>
<tr>
<td>BLOCK_RES</td>
<td>23</td>
<td>Respond with a block to the requester</td>
</tr>
<tr>
<td>ACC_RES</td>
<td>24</td>
<td>Respond with an account to the requester</td>
</tr>
<tr>
<td>NEIGHBOR_REQ</td>
<td>30</td>
<td>Request a list of neighbors from a miner</td>
</tr>
<tr>
<td>NEIGHBOR_RES</td>
<td>40</td>
<td>Respond a list of neighbors to the requester</td>
</tr>
<tr>
<td>TIME_BRDCST</td>
<td>50</td>
<td>Broadcast local time</td>
</tr>
<tr>
<td>MINER_PING</td>
<td>100</td>
<td>Initiate miner connection handshake</td>
</tr>
<tr>
<td>MINGER_PONG</td>
<td>101</td>
<td>Respond to miner connection handshake</td>
</tr>
<tr>
<td>NOT_FOUND</td>
<td>110</td>
<td>Respond with error to the requester</td>
</tr>
</tbody>
</table>

Table A.1: Message types and their corresponding descriptions
Appendix B

Installation Guidelines

B.1 Miner Application

In order to start the miner application, Go (version 1.8.3 or higher) needs to be installed. The newest version can always be fetched from the following website:

https://golang.org/dl/

After downloading Go, the environment variables GOROOT and GOPATH need to be set to the corresponding paths.

To get the miner application and all necessary libraries, the following command needs to be executed on the command line:

```bash
go get github.com/lisgie/bazo_miner
```

Upon successfully downloading the miner application and all additional libraries, the miner application can be started with the following command:

```bash
bazo_miner <database_file.db> <listening_port>
```

The application consists of two arguments: `database_file.db` points to the storage location of the disk-based key/value store. `listening port` indicates the port the application listens on for incoming miner and client connections.

Several log files are created at startup of the application which are dynamically written to and are stored in the logs folder.

B.2 Client Application

To download the client application, execute the following command on the command line:

```bash
go get github.com/lisgie/bazo_client
```
The Bazo client lets a user issue transactions by supplying the necessary arguments for each transaction type on the command line. For a transaction to be successfully validated, it needs to be signed with a private key (this is true for both root and regular user accounts). The keys are stored in regular files and the filename is supplied as an argument.

### B.2.1 Key Handling

All signatures in Bazo are based on the elliptic curve digital signature algorithm (ECDSA), a digital signature algorithm based on elliptic curve cryptography. An ECDSA key pair consists of a public key and a private key. The private key is used to sign a transaction and the public key is used for signature verification. Listing B.1 shows an example of a key file:

```
Listing B.1: Bazo Key File
4dc418348a5c77263a70544b49ed07d78714ed4df0efe277f4df20cc0a0583717
cec17514ae732cd02e0af7f6d5125d202fe650b912c489da6ddea0f0a153b904
806372a5e4766bd75e9c234280f4879832629f3030b1ebc2cfe12d3b50d6aeac
```

The first two lines make up the public key, the last line is the private key. For transferring funds, only the public key is necessary, the third line can thus be omitted when creating a transaction of this type.

### B.2.2 Account Transaction

An account transaction is launched with the following command:

```
bazo_client accTx <header> <fee> <privKey> <keyOutput>
```

The meaning of the arguments is as follows:

- **header** Value = 1 creates a new root account, value = 2 removes a root account.
- **fee** The fee to be paid for the transaction. Must be larger or equal than the *Minimum Fee* system parameter.
- **privKey** Points to the key file of a root account (the transaction is invalid if it is signed by a non-root account).
- **keyOutput** File location that stores the keypair belonging to the newly created Bazo account.
B.2.3 Transferring Funds Transaction

A transferring funds transaction is launched with the following command:

```
bazo_client fundsTx <header> <amount> <fee> <txCnt> <fromHash> <toHash> <privKey>
```

The meaning of the arguments is as follows:

- **header** Reserved for future use.
- **amount** Amount to be sent from the sender to the receiver account.
- **fee** The fee to be paid for the transaction. Must be larger or equal than the Minimum Fee system parameter.
- **txCnt** This integer parameter is linked to the sender account and needs to be increased with every newly created funds transfer transaction (starting at 0).
- **fromHash** The key file of the sender account (only public key needed).
- **toHash** The key file of the receiver account (only public key needed).
- **privKey** Key file of the sender account (will most likely point to the same location as <fromHash>, private key needed).

B.2.4 Changing System Parameters Transaction

A system parameter change transaction is launched with the following command:

```
bazo_client configTx <header> <id> <payload> <fee> <txCnt> <privKey>
```

The meaning of the arguments is as follows:

- **header** Reserved for future use.
- **id** The ID of the parameter to change:
  - ID = 1: Block Size [bytes]
  - ID = 2: Difficulty Interval [#blocks]
  - ID = 3: Minimum Fee [Bazo coins]
  - ID = 4: Block Interval [seconds]
  - ID = 5: Block Reward [Bazo coins]
- **payload** The new value to be set.
fee The fee to be paid for the transaction. Must be larger or equal than the Minimum Fee system parameter.

txCnt The transaction counter. Won’t be checked, this is just to force a new hash for the same transaction.

privKey Points to the key file of a root account (regular users are not allowed).
Appendix C

Contents of the CD

- Miner Application Source Code
- Client Command Line Application Source Code
- The \LaTeX\ Source Code
- Final Thesis (.pdf)
- Intermediate Presentation
- Final Presentation