University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Scalable IoT-based Blockchain

Kürsat Aydinli
Zurich, Switzerland
Student ID: 13-926-910

Supervisor: Sina Rafati
Date of Submission: March 1, 2019

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

0.1 Einleitung

Blockchains und das Internet der Dinge (engl. Internet of Things, IoT) haben das Potential, ein Konstrukt zu bilden, um grosse Datenmengen dezentralisiert und sicher zu speichern. Um dies zu erreichen, müssen Blockchains der fehlenden Skalierbarkeit gerecht werden.

0.2 Ziele

Das Hauptziel dieser Arbeit ist der Entwurf und die Implementierung einer skalierbaren Blockchain, welche sich die Technik des *Sharding* zu Nutze macht, um eintreffende Transaktionen parallel zu validieren und zu speichern. Im Spezifischen bedeutet dies, dass Transaktionen von einer distinkten Menge von Teilnehmern validiert und gespeichert werden. Im Gegensatz hierzu werden bei tranditionellen Blockchains die Daten von allen Teilnehmern verarbeitet mit dem Nachteil, dass sich mit der Zeit die Speicheranforderungen an die Knoten erhöhen. Dies kann unter Anderem dazu führen, dass gewisse Knoten vom Netzwerk ausgeschlossen werden. Die Blockchain in dieser Thesis mildert diesen Effekt dadurch, dass die Knoten nur einen Bruchteil aller Transaktionen verarbeiten.

0.3 Resultate

In Bezug auf die funktionalen Anforderungen kann festgehalten werden, dass die entwickelte Blockchain die Grundfunktionen des Sharding erfolgreich umgesetzt hat.
Um die nicht-funktionalen Eigenschaften respektive die Performance zu evaluieren, wurden drei Test Umgebungen aufgebaut. Besonderes Augenmerk wurde auf die Metrik 'Validierte Transactionen pro Sekunde' (TPS) gelegt.
Im ersten Setting wurde Bazo getestet mit einem Validator und einem Shard und je drei Testläufen. Das Ergebnis hat sich vielversprechend herausgestellt mit einer durchschnittlichen TPS von 52,6 Tx/sec, wobei das Maximum bei 55,2 Tx/sec lag. Die genauen Messungen können der Tabelle 5.1 im Kapitel Evaluation entnommen werden.
Der zweite Teil der Evaluation richtete sich gegen Bazo mit zwei Validatoren und zwei

Shards, wieder mit je drei Testläufen. Hierbei ging die Erwartung voraus, dass sich die TPS erhöhen sollte aufgrund von zwei parallel laufenden Shards. Erstaunlicherweise hat sich herausgestellt, dass sich die TPS etwas vermindert hat im Gegensatz zum Beispiel mit einem Shard. Hierbei lag die durchschnittliche TPS bei 44,8 Tx/sec mit einem Maxialwert von 48,8 Tx/sec. Eine genauere Analyse der Messungen von jedem der zwei Validatoren hat gezeigt, dass die Synchronisation der Shards auf jeder Blockhöhe eine nicht vernachlässigbare Verzögerung nach sich zieht. Während der Synchronisationsphase findet kein 'Mining' statt, sondern die Teilnehmer synchronisieren unter sich den globalen State der Blockchain. Bezogen auf jeden einzelnen Validator führte dies zu einer erheblich geringeren TPS als im Falle von einem Shard. Für die detaillierte Auswertung wird der Leser auf Tabelle 5.2 im Kapitel Evaluation verwiesen.
Ein letzter Testlauf wurde mit drei Validatoren und drei Shards durchgeführt. Hierbei hat sich gezeigt, dass die TPS Performance gestiegen ist mit einem Durchschnittswert von 58,3 Tx/sec und einer Maximalperformance von 59,5 Tx/sec. Somit konnte dargelegt werden, dass Sharding eine erhöhte Skalierbarkeit begünstigt.

0.4 Weitere Arbeiten

Bazo weist in seiner jetzigen Form einige Limitation auf, welche eine erfolgreiche Nutzung in Produktion erschweren. Zum einen funktioniert die Blockchain nur adäquat mit einem Validator pro Shard und zum Anderen besteht noch keine Massnahme, um Bazo von Ausfällen von Validatoren zu erholen. Des Weiteren gestaltet sich eine 51% Attacke durch das Sharding einfacher als in einer Blockchain ohne Sharding. Um diesen Schwächen gerecht zu werden, gibt es folgende Verbesserungspotentiale:

- Der Synchronisationsmechanismus vom State (Account Informationen) ist noch nicht in der Lage mit eine Fork (gleichzeitiges Generieren von mehreren Blöcken im gleichen Shard) umzugehen. Hier bedarf es eines Mechanismus, der im Falle eines Forks die Synchronisation von invaliden Blöcken zurückstellt.

- Ein weiteres Problem tritt auf, wenn ein Fork in einem Shard im letzten Block vor dem nächsten Epochenblock auftritt. In so einem Fall würde der nächste Epochenblock generiert werden, ohne dass die nötigen Blöcke zurückgerollt wurden. Dies führt zu einer Verfälschung im State. Hierbei müsste ein Fork ebenfalls richtig erkannt und die Generierung des nächsten Epochenblocks entsprechend aufgeschoben werden.

- In einer Blockchain mit Sharding gestaltet sich ein 51% Attackszenario einfacher als in einer Blockchain ohne Sharding, da hierbei nicht mehr 51% der Assets benötigt werden, sondern ein Bruchteil davon in einer bestimmten Epoche. Um dieser erleichterten Attacke vorzubeugen, könnte man die Zuweisung der Validatoren zu den einzelnen Shards verbessern, in dem man die Knoten auf jeder Blockhöhe einem anderen Shard zuweist.

- Des Weiteren könnte die Evaluation ausgeweitet werden durch Tests mit mehr als drei Shards, um herauszufinden, ob sich der Trend einer erhöhten TPS weiterentwickelt.

Abstract

This master's thesis documents the design, implementation and evaluation of the sharding enabled Bazo blockchain. The decision to leverage sharding as the layer 1 scalability technique was the result of reviewing recent literature with respect to mainstream approaches to scale blockchains.
The system supports sharded environments with one validator per shard. In case of multiple shards, the validators validate and store only a fraction of the whole incoming transaction load, and as such of the whole blockchain. Received transactions from the clients are assigned to shards based on their public address. The single shard chains synchronize the global state at every block height to make sure that the shards grow at the same pace. In order to do so, the new data type *state transition* was introduced which is emitted after every block creation from each shard. The state transition includes everything needed for the other shards to update their local state. Furthermore, the blockchain runs in fixed lengths of X blocks called epochs. After each epoch, the validators are randomly re-assigned to the single shards. The system was evaluated through unit tests sending up to 950 transactions per second to the blockchain. The set-up with one validator and one shard achieved a peak transaction validation rate of 55,2 Tx/sec.
In a similar set-up, Bazo was testet with two validators and two shards. Surprisingly, the overall TPS value decreased slightly with a peak of 48,8 Tx/sec. In this case, the main performance bottleneck turned out to lie in the synchronisation overhead of the two shards. Furthermore, the tests were extended to incorporate three valudators and three shards. Hereby, it was observed that the TPS capacity increased to a maximum of 59,5 Tx/sec, demonstrating that sharding has a positive influence on the scalability of Bazo.
Although the current system exhibits evident limitations and downsides, it nevertheless showed to be a first feasible step towards making Bazo more scalable while incorporate the promising technique of sharding.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Sina Rafati, for the continuous support of my master's thesis. His patience, motivation and profound knowledge in the field of blockchain technology is inspiring. Furthermore I would like to thank Dr. Thomas Bocek for his valuable advice and guidance in the weekly meetings.

Besides my supervisors I would also like to offer my sincere thanks to Prof. Dr. Burkhard Stiller, the head of the Communication Systems Research Group, for giving me the opportunity to work on such an interesting and currently relevant topic.

Contents

Chapter 1

Introduction

Two rapidly accelerating and emerging research areas in computer science with respect to secure and fast data storage are IoT (Internet of Things) and blockchains.
A blockchain is best described as a distributed ledger which is maintained concurrently by multiple peers and provides the possibility for users to insert data in it. Furthermore, sophisticated consensus mechanisms are employed among the peers to validate data input and store them on the blockchain. For this reason, such a system is especially appropriate for securely storing data and mitigating tampering approaches.
On the other hand, IoT denotes the primitive of several (usually many) devices communicating with each other while being connected to the Internet. This setting is particularly useful in application scenarios where automated data exchange is crucial.
Blockchains might be perceived as the missing link to settle scalability and redundant trust in the IoT industry. Leveraging blockchain techniques for IoT data offers a promising way to automatically store it in a tamper resistent and distributed system. This way, setting up a complex and centralized IT Infrastructure with the need of permanent maintenance can be omitted. This decentralized approach would eliminate any single point of failure. In addition, the cryptographic algorithms which are inherent to blockchains would make stored data more private.

1.1 Motivation

The Internet of Things (IoT) is growing at a fast pace. According to Gartner Research, the IoT will be including approximately 26 billion device units installed until 2020 [41]. Even though this growth is a potential sign for the unlimited use cases of IoT, a massive adoption of thereof still faces crucial challenges. One of the most important obstacles is that many IoT solutions are still expensive due to costs related to the deployment of centralized server farms.
Application areas in which blockchain technologies and IoT are involved are sensing, data storage, timestamping services, smart living applications, intelligent transportation systems, wearables, supply chain management, mobile crowd sensing, cyber law and security in mission-critical scenarios [20].

For instance, in terms of agriculture IoT, blockchains can be employed for tracability and monitoring purposes. In [46], an automated and decentralized pollution monitoring system is presented which basically saves IoT sensor data in the Ethereum blockchain.

When operating a blockchain in an IoT setting, especially for purposes of data storage, several requirements need to be addressed. The most important ones are known as high data sizes, great number of IoT nodes and data privacy.
Existing cryptocurrencies are known to have scalability issues, i.e., the number of transactions they are capable of processing per second is limited. As the number of applications leveraging public cryptocurrencies grow, the demand for processing high transaction rates with a comparatively low latency is increasing [63].
The Communications Systems Group (CSG) at the University of Zurich has recently been developing the Bazo blockchain from scratch [58, 4]. Although it already encompasses state-of-the-practice building blocks of a blockchain (e.g. transaction management, storage of data in blocks, consensus mechanisms), it is not designed for the purpose of IoT related use cases.
Especially the lack of Bazo's scalability makes it infeasable to use in an IoT environment. Imporving the scalability of Bazo will be the the main purpose of the thesis at hand.

1.2 Description of Work

This thesis covers the design and implementation of a scalable IoT enabled Blokchain for the purpose of storing IoT data while taking into account the aforementioned requirements.
Furthermore, the newly developed blockchain will be evaluated in order to assess the advantages compared to the current implementation of Bazo. During the evaluation phase, critical performance metrics will be examined. The most important metrics include block generation time, transaction validation rate, transaction issuance rate and average number of transactions in a block.
The technical goal of the thesis is to adapt the Bazo blockchain to incorporate the most applicable scalability methodology which was determined after thorough review of recent literature and assessment of its fit for Bazo.

1.3 Bazo Blockchain

The Bazo cryptocurrency [1] is a blockchain developed in 2017 from scratch at the University of Zurich mainly for research purposes. Unlike Bitcoin which employs an Unspent Transacion Output (UTXO) model, Bazo is a blockchain in which accounts are first-class citizens. This means that the union of all accounts make up the state of the network.
The initial development of Bazo employed Proof of Work (PoW) as the consensus mechanism [58]. Due to high energy consumption of the PoW consensus and the need of

[1]https://github.com/bazo-blockchain

increasing security of the blockchain, Bazo was adapted in 2018 to incorporate Proof of Stake (PoS) as the main consensus mechanism [4].
Based on the current implementation of Bazo, the main contribution of this thesis is an improvement of thereof with respect to its scalability.

1.4 Thesis Outline

The thesis at hand is structured into 7 main parts as follows: Section 2 familiarizes the reader with various approaches on how to scale blockchains. Chapter 3 and 4, as an elaboration of the practical parts, presents the proposal of the thesis including an outline of the chosen design and the implementation of the adjusted Bazo. An evaluation of the developed system is conducted in section 5. Further improvement potentials and future work are outlined in chapter 6. The paper concludes with section 7 which is devoted to a summary and conclusion part.

Chapter 2

Background and Related Work

The following chapter sheds light on different ways of improving the scalability of a given blockchain. The need for scalability becomes more important the more nodes are connected to the system and issue transactions at a high rate. The techniques which will be discussed in this chapter are depicted from recent literature. As such, they do not constitute the complete set of available scalability methods. Since blockchain scalability is a highly dynamic research area, new scalability proposals will be released on a regular time basis. For the sake of simplicity, only the methods mentioned in this chapter will be examined within the scope of this work.

2.1 Blocksize Increase

Probably the most straightforward and simplest approach of enhancing the scalability of a blockchain is to increase its blocksize. This way, more transaction fit into a single block in general, thus the achieved throughput of transaction per block interval would increases [37].
Despite its simplicity, this approach comes hand in hand with several drawbacks. A direct implication of the increase of the blocksize is a hardfork of the underlying blockchain. This would result in a split of the community as it happened with Bitcoin [56].
Due to the fact that traditional blockchains require every node to store the full history of the ledger in order to become part of the network, an increase of the blocksize would lead to an increased storage requirement of a full node. Depending on how much the storage requirement increases, nodes which are not capable of fulfilling these would be ruled out of the network in the long run.
This would mean that larger blocks would make full nodes more expensive to operate. This in turn would lead to some few centralized entities having enough power to be part of the mining/validator network. As a result of the reduced decentralization, users of the blockchain would need to put more trust into protocol[37].

The enourmous storage demand of Bitcoin when increasing the blocksize without any limitations can be made more clear when attempting to reach the throughput peak of VISA with roughly $47'000$ transactions per second during holidays in 2013. In case Bitcoin would use an average of 300 Bytes per transaction and assuming unlimited blocksize, an equivalent capacity to VISAs peak transaction volume would result in nearly 8 Gigabytes per Bitcoin block, every ten minutes on average. Extrapolated to one year, that would result in over 400 Terabytes of data [50].

2.2 IOTA

In the effort of developing a novel, scalable blockchain - especially for the application of IoT use cases - the IOTA Foundation [1] has released the eponymous crypotocurrency IOTA. In contrast to mainstream blockchains, IOTA employs a different data structure under the hood.
The distributed ledger technique of IOTA is based on a Directed Acyclic Graph (DAG) for storing transactions rather than that of cryptographically connected blocks as it is applied in mainstream cryptocurrencies. The DAG of IOTA is referred to as the *Tangle* [51]. An exemplary representation is shown in Figure 2.1. Whereas nodes in typical cryptocurrencies would represent a collection of transactions bundled as a block, the nodes in the Tangle represent single transactions. Any edge between two transaction denotes a validation path, e.g. $A \Longrightarrow B$ means that transaction A validates transaction B. An edge of the form $A \Longrightarrow ... \Longrightarrow ... \Longrightarrow ... \Longrightarrow F$ would mean that transaction A indirectly approves transaction F. Transactions without any confirmations yet are called *tips*. In Figure 2.1, they correspond to the grey nodes in the rightmost part of the Tangle.

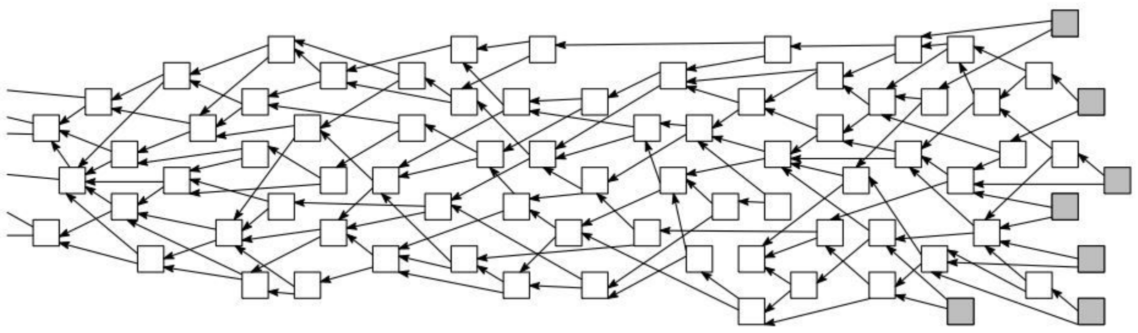

Figure 2.1: The Tangle - Grey Nodes: Tips - Time Progresses From Left To Right [51]

In order to issue a transaction to the Tangle, a user has to perform following tasks [51]:

- confirm two previous transactions
- perform a little PoW by solving a cryptographic puzzle similar to the handling in Bitcoin, but which is generally a lot easier to obtain

[1]https://www.iota.org/

All tokens of IOTA were generated when issuing the *genesis* transaction. No new tokens will be created in the future. Furthermore, IOTA does not employ a mining or validation mechanism as it is known from mainstream blockchains, e.g. Bitcoin or Ethereum. Therefore, there are no transaction fees in the traditional sense. The 'fee' to participate in the network is solely the accomplishment of the two aforementioned tasks. This characteristic enables to operate IOTA basically fee-free which is considered crucial in the context of micropayments. In establishing true micropayments, growing transaction fees are a major obstacle [32].
An additional entity which is part of IOTA is the *Coordinator* which is a special node contributing computing power and security measures to the network. He is responsible for periodically assigning a confidence level of 100% to transaction which have been residing in the Tangle long enough. Besides that, the Coordinator performs regular snapshots of the Tangle to keep the storage requirements minimized. In such an activity, he removes all events and accounts from the Tangle which have a non-positive balance. The notion of the centrally operated and closed source Coordinator might pose some threats to the decentralized nature of the cryptocurrency. IOTA intends to turn off the Coordinator once the network has grown strong enough to sustain itself [32].
In terms of scalability, IOTA does not exhibit an artificial thereshold or limit of transactions that can be verified within a certain time interval. In fact, every added transaction validtes two transaction, thus the scalability increases linearly with every new transaction in the Tangle. [51, 7, 32, 52].

2.3 Sharding

The idea of sharding has its origin from database management theory. Hereby, sharding denotes the horizontal partitioning of a database among multiple physical data stores holding its own distinct subset of the data. This split of a large collection across several servers enables the distributed management in terms of CRUD operations of a single table, thereby improving the scalability [64].
When distirbuting data across server clusters, a key has to be chosen for chunk ranges. This key is referred to as *shard key* and can consist of any combination of fields. Since the shard key determines how the data is distributed, choosing a proper key is crucial for the handling of the workload of the application [11].

Sharding in the context of blockchain research is perceived as one of the most effective scalability technique in recent literature. As a layer 1 scalability method (e.g. as opposed to blocksize increase), its application has a direct impact on the scalability of the underlying blockchain.
Hereby, sharding is referred to as the artificial division the workload of the blockchain, i.e. transaction processing, into single shards in such a way that the transactions can be validated and stored in parallel.
When applying sharding to a blockchain, there are several challenges one needs to be aware of. First, it needs to be ensured that the same transaction does not get assigned to several shard. This would lead to redundant work. To avoid this pitfall, an adequate shard key has to be determined. In most cases, the sender address of a transaction serves

as an appropriate shard key. Given that the number of shards remains the same, a transaction from a user will always be processed by one particular shard.
Another important point that needs to be taken care of when using sharding is to determine the right number of shards. With regards to this decision, it might be advisable to partition the network into shards based on the number of validators or miners, respectively.
Conceptually, it might be reasonable to have as many shards as possible to make the most out of sharding and parallel transaction processing. But having many shards also increases the communication and coordination effort among the shards. Apparently, there is a trade-off between the number of shards and the entailed synchronisation overhead.

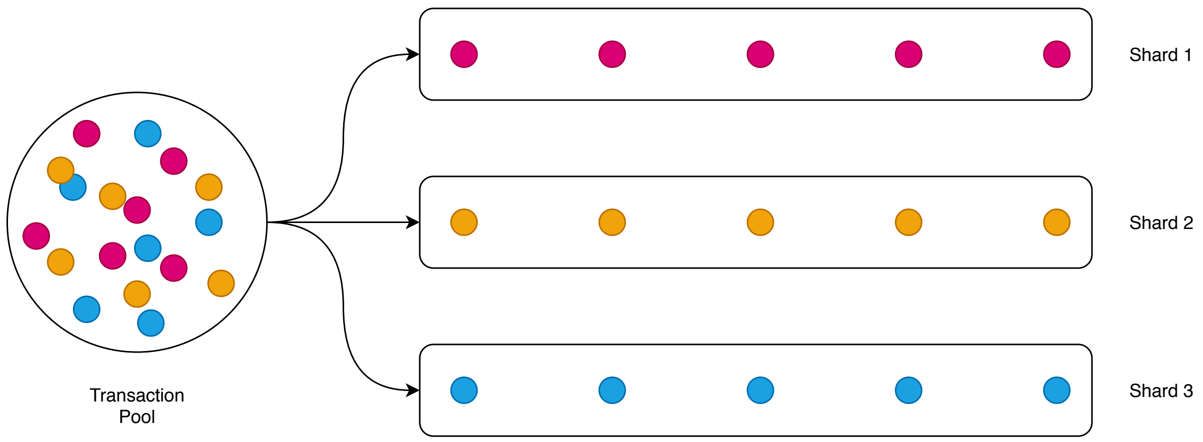

Figure 2.2: Exemplary Representation Of Sharding

Figure 2.2 outlines how transactions might be partitioned into distinct shards. Circles of a particular colour represent transactions from a specific sender. It can be seen that, for instance, transactions from the blue node are only processed in shard 3.
With regard to the scalability potential of sharding, it can be stated that this methodology is able enhance the scalability by its very design of enabling concurrent data validation and storage. However, due to the induced coordination work, the shards are not expected to grow at the same pace as in a setting without sharding.

2.4 Off-Chain state channels

This approach on improving the scalability of a blockchain builds on the assumption that, in case only two parties care about recurring transactions, the rest of the network actually does not need to be aware of those transaction having ever occurred. From a privacy perspective, it is preferable to have only the bare minimum of information stored on the blockchain.
Bidirectional state channels, often referred to as micropayment channels, create a relationship between two peers of the blockchain network to perpetually update balances. The main idea behind this application is that every party deposits some coins on the channel [65]. This initial balance is stored as a transaction on the main blockchain. Once the payment channel is initialized, any coin transfer between the two peers, i.e. any balance

adjustment, requires to be signed (2-of-2) by both parties. On terminating the payment channel, the final balance is also stored as a transaction on the blockchain. To put it differently, every intermediary coin exchange and balance adjustment can be discarded since it is not stored on the blockchain. This special characteristic of the payment channels allows a practically unlimited transaction rate per second. [50, 40].

A sample outline of such a two-way interaction channel is shown in Figure 2.3. As it can be seen, Alice and Bob have set up channel between them whereby Alice has initially deposited 5 coins. During the first balance distribution, Alice transferred 4 coins to Bob while in the second transaction Bob received another staked coin from Alice. From the three transactions of Figure 2.3, only the first and last one are actually stored on the blockchain.

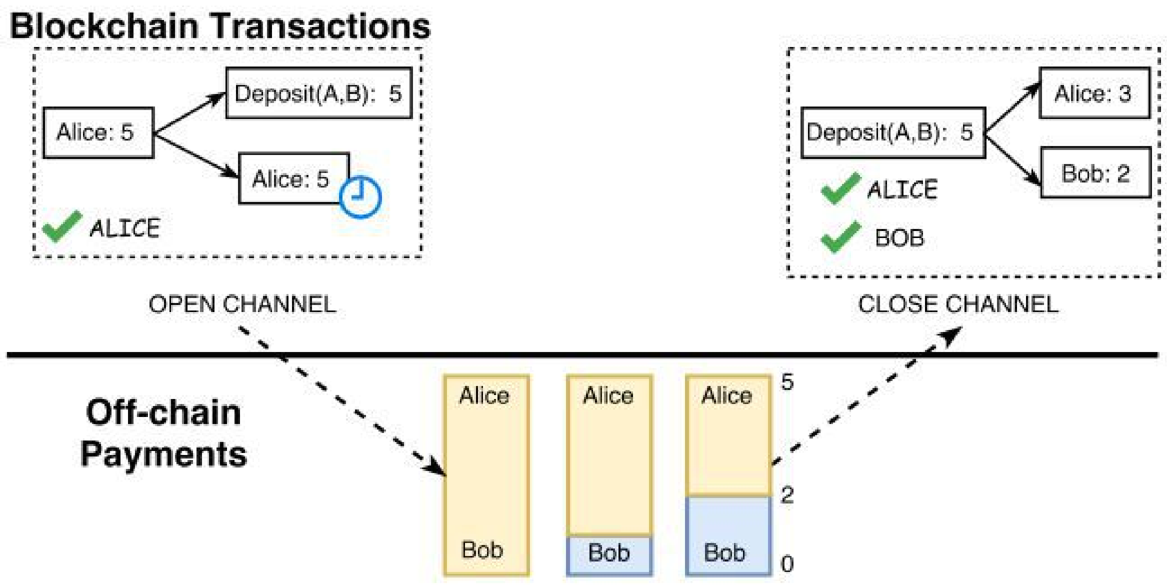

Figure 2.3: Paymentchannel Between Alice And Bob [40]

Even though this technique offers an apparent solution to scalability issues, i.e. reducing the load on the blockchain and improving transaction throughput, its employment conceals a couple of drawbacks.
First of all, due to the very design of payment channels, they are not applicable in situations where the corresponding group of users is *open*. This would esspecially cover environments where anyone can join at will without any permissions [13].
Another challenge of this approach are the locked up funds. Assets which are exchanged on a payment channel are locked up for the lifetime of the channel. Due to this property, payment channels might be perceived inappropriate in some use cases. To be able to spend the coins, the payment channel has to be terminated which results in issuing the final transaction on the blockchain [8]. The two on-chain transactions and security deposits of a state channel make them inefficient for one-time transactions [65]
Furthermore, micropayment channels may quickly lead to a complicated network infrastructure. The need to set up a paymnet channel with every peer of the network does not solve the scalability issue per se. In such cases, a network of payment channels needs to

be built over which assets can be transferred to the designated receiver. For instance, if party A does not have a direct communication with party D, but has a channel to C and at the same time C has one with D, then node A can take advantage of this situation and transfer assets to D via C [50].
Because coins can only be transferred over a channel to a maximum amount which is held by the security deposit of the payment channel, it has to be made sure that every single channel on such a network of channels, e.g. channels $A-C$ and $C-D$,is holding at least that amount of assets. In case any peer on this route would terminate its corresponding channel, a new route would need to be determined which leads to the peer of interest. This in turn requires sophisticated routing mechanisms.

The fact that intermediary transactions are not stored on the blockchain, but discarded, can be inconvenient in scenarios where the very intention is to store all transactions on the blockchain. For instance when using a blockchain as data storage, the user may be generally reluctant to the usage of payment channels because of this behaviour.

2.5 Scalability Methods Assessment

After taking into account the advantages and drawbacks of each scalability technique and assessing their applicability for improving Bazo, *sharding* has been chosen as the layer 1 scalability enhancement.
The approach of increasing the blocksize constitutes only a temporary solution with comparatively negative impacts on the long run as pointed out with the artificial example of Bitcoin.
Employing micropayment channels has several disadvantages in the setting of our IoT oriented blockchain. The fact that the funds are locked up in the channel makes this approach infeasible for our case since Bazo will be highly depending on permanent fund accessibility to issue transactions. Furthermore, the side effect of this approach that intermediary transactions get discarded is also against our very objective of storing lots of IoT data on the blockchain.
With respect to IOTA, we can argue that its architecture diverges too much from the design of Bazo which would require a complex architecture transition. In addition, the centralized nature of the Coordinator used in IOTA poses a great obstacle for our fully decentralized setting.

2.6 Related Work

This section is dedicated to frameworks released in recent literature which propose scalable cryptocurrencies employing the technique of sharding.

2.6.1 Elastico

Elastico is the first sharding-based crpytocurrency for permissionless, open blockchains which partitions the network into committees each of which is processing a disjoint set of transactions. The number of operated committees grows linearly in the total computational power of the network. Furthermore, the algorithm of Elastico proceeds in epochs. The evaluation of Elastico shows that it scales up the agreement throughput near linearly with the overall computational power and tolerates byzantine adversaries which control up to 1/4 computation capacity [39].

2.6.2 OmniLedger

The scale-out distributed ledger of OmniLedger uses a bias-resistent public-randomness protocol for choosing large, statistically representative shards that process transactions. Furthermore, it employs parallel intra-shard transaction processing and ledger pruning via collectively signed state blocks. State blocks are similar to stable checkpoints summarizing the entire state of a shard's ledger. At the beginning of each epoch, all validators get assigned to a shard based on the epoch randomness and bootstrap their states from the corresponding shard's most recent state block.
Its evaluation demonstrates that the throughput scales linearly in the number of active validators while keeping the confirmation time of transactions under two seconds [30].

2.6.3 RapidChain

The blockchain protocol proposed by [68] is resilient to Byzantine faults from up to a 1/3 fraction of its participants and achieves comparatively high throughputs via block pipelining. By using an efficienet cross-shard transaction verification technique, RapidChain avoids gossiping transactions to the entire network.
As with most of the sharded blockchains, the partitioning of RapidChain requires each node to store only $1/k$ of the entire blockchain where k denotes the number of committees. The empirical evaluations reveal a processing and confirmation capacity of more than $7'300$ tx/sec while ensuring a confirmation latency of approximately 8.7 seconds harnessing a network of $4'000$ nodes.

2.6.4 Zilliqa

Yet another blockchain leveraging the technique of sharding is Zilliqa, being under research and development for 2 years, with the aim to rival traditional centralized payment methods such as VISA or MasterCard. Similarly to Ethereum and Bazo, Zilliqa is an account-based system. Furthermore, the blockchain relies on creating and updating a directory service committee which is tasked to coordinate the sharding process and assign the nodes to the shards. Incoming transactions are sharded according to the sending accounts[62].
In contrast to other blockchains employing PoW consensus, Zilliqa only utilized PoW to establish identities and prevent sybil attacks. In terms of scalability, Zilliqa's design allows its transaction rates to roughly double with every few hundred added nodes. With respect to the fraction of malicious nodes, Zilliqa assumes this fraction to be less then 1/4 of the complete network. Furthermore, Zilliqa employs a special-purpose smart contract language leveraging the underlying architecture and following a dataflow programming style to provide a large scale and highly efficient computation platform [63].

2.6.5 QuarkChain

QuarkChain represents a two-layered blockchain, employing elastic sharding on the first layer and having a root chain as the second layer that confirms the blocks from the first layer without processing transactions. Interestingly, the root chain is guaranteed to hold 50 % of the total hashing power of the network. Thus, an advisory would need a minimum of $50\ \% \cdot 50\ \% = 25\ \%$ of the hashing power to initiate an attack. Besides that, the allocation of the hash power on the root chain is configurable, e.g. 25 % or 75 %.
The blockchain supports cross-shard transactions at any time with a confirmation latency in the order of minutes. With respect to the consensus mechanism, the root chain uses PoW, similar to Bitcoin and Ethereum. In case of forks, the shorter chain will be deemed invalid [22].

Table 2.1: Comparison Of Sharding Protocols

Protocol	**# Nodes**	**Resiliency**	**Throughput [Tx/sec]**	**Latency [sec]**	**Shard Size**
Elastico	1600	$t < n/4$	40	800	100
OmniLedger	1800	$t < n/4$	500	14	600
	1800	$t < n/4$	3500	63	600
RapidChain	1800	$t < n/3$	4220	8,5	200
	4000	$t < n/3$	7380	8,7	250
Zilliqa	3600	$t < n/4$	2828	-	600
Quarkchain	6450	-	10000	> 60	25

Table 2.1 summarizes the 5 Prototypes from this section with respect to their evaluations. In terms of the overall throughput, Quarkchain appears to have achieved the best performance with roughly 10'000 Tx/sec.
RapidChain has demonstrated resiliency against up to 1/3 of the nodes which is higher than all other prototypes. Also with respect to confirmation latency, RapidChain demonstrated the best figures.
The shard sizes of the test runs are in the order of hundreds except for QuarChain which has operated a comparatively small shard size.

Chapter 3

Design

In this chapter, a detailed description of the modified Bazo protocol and the underlying blockchain is undergone whereas each chapter depicts one specific design aspect.

3.1 Key Assumptions

To keep the complexity at a reasonable level, following assumptions were made:

One validator per shard Currently, the system does only support one validator per shard. The main reason is that multiple validators per shard can lead to forks of a shard chain. The state synchronisation among shards in Bazo operates on a block level with the assumption that per block height there should be only one block per shard. If forks occur within a shard, i.e. there is more than one block from a shard at some block height, the state synchronisation will not work appropriately. State and block height synchronisation is explained in detail in the subsequent chapters. This assumptions was not made initially at the beginning of the development of the thesis. The general approach was to support multiple validators per shard. This gets also evident in the following sections of the thesis where mechanisms are described as if multiple validators per shard are allowed. But due to technical complexities and time limitations, Bazo could only reach a state of working properly with one validator per shard.

Validators need to be running In order for a validator in some shard to continously create blocks, he needs to wait for validators of the other shards to create blocks as well in a synchronized manner. If a validator goes offline or crashes, then his shard will stop mining blocks. In such a case, the other validators will not be able to receive blocks from this particular shard, thus not be able to perform the synchronisation.

3.2 Entity Management

Bazo consists of three entities: *users*, *validators* and *leaders*. A user interacts with Bazo's infrastructure through issuing transactions in order to transfer funds or execute smart contracts. A validator is a node who takes part in the overall validation process of Bazo. While validators and users might issue transactions, users do not participate in the consensus protocol. For this reason, users only store the block headers whereas validators store the whole blocks of their assigned shard. The sphere of responsibility of a validator can be compared to that of a miner in the Bitcoin protocol.
Furthermore, a validator can become a leader who has the right to append the next block to the corresponding shardchain. The leader gets elected through fulfilling a Proof-of-Stake condition for a specific block height, i.e. leaders within a shard change block-wise whereby validators may only become leaders for their assigned shard. As pointed out in the aforementioned assumptions, Bazo can currently only deal with one validator per shard. This means, that in the current setting, every validator is also the leader of the corresponding shard. The concept of the leader becomes more important in case multiple validators per shard would be allowed.
In addition, each entity in Bazo possesses a private-public key-pair in order to digitally sign transactions. Bazo employs elliptic curve digital signature algorithm (ECDSA) as the primary signing algorithm. For the remainder of this paper, the ECDSA key-pair is referret to as the users *wallet keys* (pk_{wall}, sk_{wall}).

Figure 3.1 shows how a sharded split of the network might look like whereby it is partitioned into shards *1* to *N*. While users always remain in their shard based on their address pk_{wall}, validators regularly get assigned to a shard in a random fashion.
In case a user wants to join the set of thevalidators, it has to create an additional private-public key-pair using RSA and publish a *StakeTX* transaction and publish the public key. This key-pair does not correspond to the key-pair used to sign transactions. In the paper at hand, the key-pair generated using RSA is referred to as the users *commitment keys* (pk_{comm}, sk_{comm}).

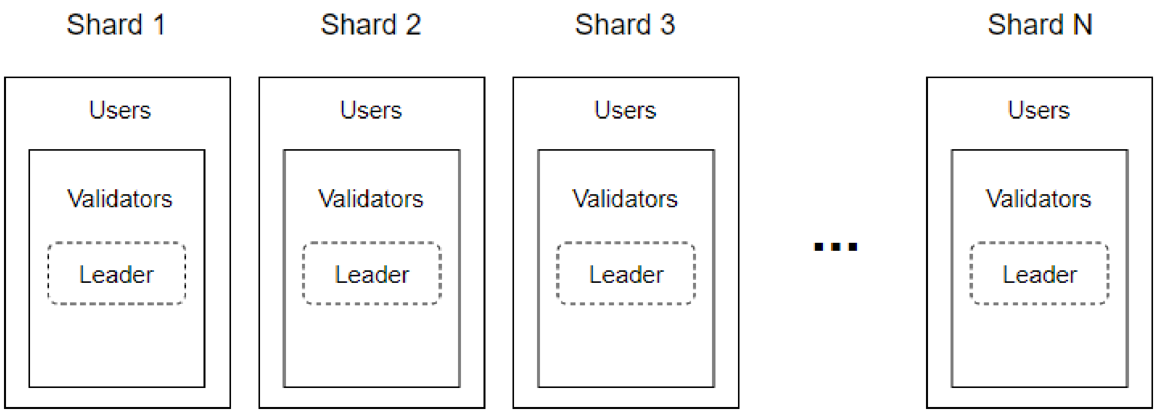

Figure 3.1: Overview Of The Entities In Bazo

3.3 Network Sharding

The Bazo network is divided into several subgroups called *shards*. Each shard has its own blockchain storage referred to as the *shard chain*. Note that the partitioning of the network is done on the basis of the validators, not the users. Dividing the network according to the number of users would not guarantee that every shard has a validator, leading to omission of transaction validation. For this reason, network partitioning and determining the required number of shards is based on the number of active validators in the network.

Transactions issued by users into Bazo are collected by all validators in a transaction pool, but validators only verify transactions based on the pk_{wall} of the transaction sender. For instance, validator V of shard S is responsible for user U and validates transactions where U is the sender of the transaction. Thus, user U is assigned to shard S.
The responsibility of a validator is restricted to the maintenance of his shardchain. Maintenance is defined as validating blocks and storing the full history of the shardchain. Unlike validators, users do not store the shar chain, but only their issued transactions.

3.4 Epochs

As in the case of most sharded blockchains, the protocol proceeds in some fixed length of blocks which are called *epochs*. Epochs have a predefined length, $length_{ep}$. For instance, assuming epoch e exhibits $length_{ep} = 100$ and the block height of the first epoch block is *0*, then this epoch would and at block height *101*, thus consisting of 100 intermediary blocks. This would result in the first shard block of epoch $e + 1$ to start at height 102, and so on.
At the end of each epoch, there is a *finalization* procedure during which the next epoch block is created and the number of new shards, *NofShards*, is determined based on the current number of validators in the network. Furthermore, the validators are randomly assigned to the single shards.

Figure 3.2 shows an exemplary excerpt of the Bazo network demonstrating how epochs work. The blockchain starts with eb^1, i.e. the epoch block at height *1* and then gets divided into three shards with shard blocks sb^{2-5}_{1-3} where the subscripts $1 - 3$ denote the shard ID and the superscripts $2 - 5$ represent the block height. The epoch in Figure 3.2 has a length $length_{ep}$ of 4 blocks, i.e. epoch block eb^1 is followed by 4 shard blocks, before the next epoch block eb^5 is created.

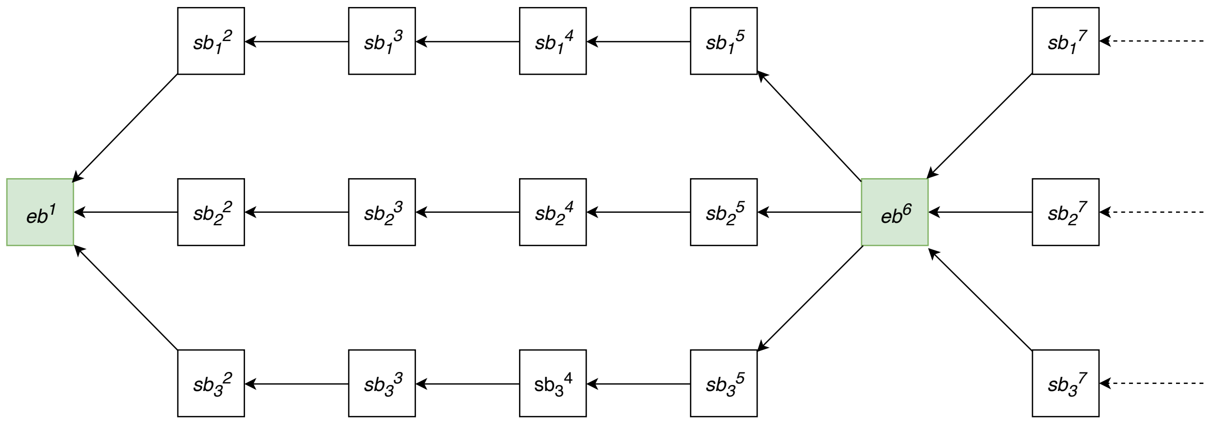

Figure 3.2: Exemplary Epoch Representation With Epoch Length $length_{ep} = 4$ And 3 Shards

3.5 Number Of Shards

The number of shards has an evident impact on the overall scalability of the network. As the validator network grows and the users issue more and more transactions, it is crucial to lighten the computational and storage requirements for each validator.
With respect to this decision, following cases can be distinguished:

- Employing only a single shard resembles the state of Bazo without any sharding
- Having a too small number of shards scales only sub-linearly as the number of validators increase, and as such leveraging the technique of sharding not to its full extent
- Too many shards increase the communication overhead between the shards to synchronize the block height and the global state.

Bazo divides the validator set into shards based on the system parameter *VALIDATORS_PER_SHARD* which is of Integer type. Due to the restriction of only supporting one valiadtor per shard, the default value of this parameter is set to 1. Once Bazo gets further improved to account for multiple validators per shard, this parameter can be set arbitrarily.
As such, *VALIDATORS_PER_SHARD* denotes the maximum number of validators per shard. Setting this factor to 3 would mean that at most 3 validators can share the same shard. Putting it differently, for every 3 validators, a new shard will be generated. For example, considering *VALIDATORS_PER_SHARD = 3* and the network consisting of exactly 3 validators, then there will be one shard. If another validator joins, the number of shards will increase to 2, until there are 7 valiadtors at which point another shard will be created. Therefore, the number of shards is evaluated according to:

$$NofShards = \frac{NofValidators}{VALIDATORS\_PER\_SHARD} \tag{3.1}$$

The dynamic adjustment of the number of shards is further illustrated in Figure 3.3. Starting off with 3 shards in the first epoch, the second epoch consists of 5 shards and the last one has only two.

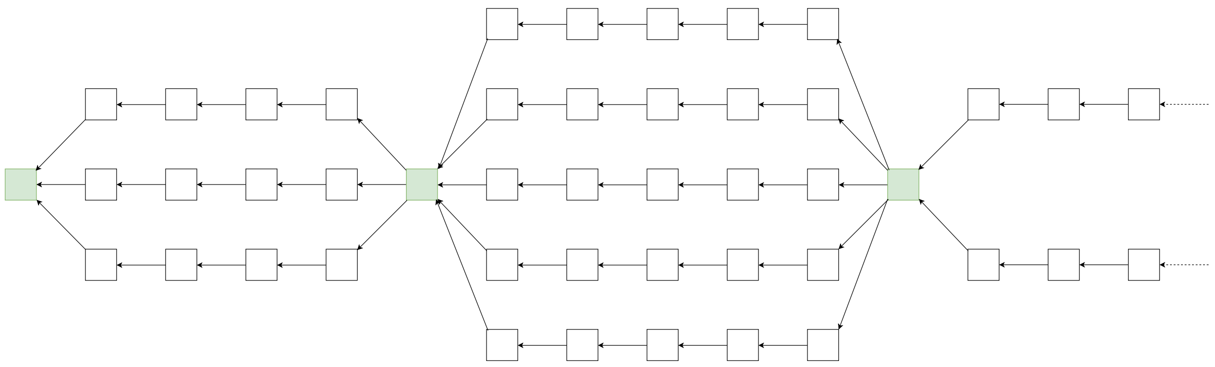

Figure 3.3: Dynamic Load-Balancing

3.6 Leader Formation

Users of Bazo intending to join the set of validators have to issue a *StakeTX* transaction containing the public commitment key pk_{comm}. As explained previously, in the current setting of the blockchain, every validator is also the leader of a certain shard. Nevertheless, Bazo includes a leader formation process for the situation of having multiple validators per shard which is explained in this section. Assuming to have several validators in each shard, solely taking part in the validator set does not guarantee a user the right to append a certain shard block. For appending the next shardblock, a validator has to become the *leader* of the shard. The leader election process is identical to fulfilling the PoS condition introduced in [4] which was later adjusted due to a security vulnerability in [6].

Every validator participates non-interactively in solving the PoS condition. As opposed to PoW, a validator is limited to exactly 1 H/s (hash per second) by including a *TimeIn-Seconds* factor in the PoS condition:

$$\frac{SHA3\text{-}512([Proof_{PrevBlocks}] \cdot Proof_{Local} \cdot BlockHeight \cdot TimeInSeconds)}{Coins} \leq Target \tag{3.2}$$

where

$$Proof_{Local} = RSA(sk_{comm}, SHA3\text{-}512(BlockHeight)). \tag{3.3}$$

The PoS condition consists of following parameters:

List of the Previous Proofs($Proof_{PrevBlocks}$) By including a list of the previous proofs of a particular shard, a stake grinding attack becomes infeasible.

Local Proof($Proof_{Local}$) All parameters are the same for each validator in the network except the local proof. The local proof individualizes the PoS condition to each validator and therefore, a validator with a low amount of coins also has the possibility to append a block to the blockchain.

Height of the Block($BlockHeight$) The height of a block is characterized by the number of previously added blocks in the blockchain plus one.

Amount of Coins($Coins$) By dividing up the number of coins that the leader possesses, the election process becomes proportional to the stake. Without this division every economically acting node would create new accounts that possess exactly the minimum staking in order to maximize its staking reward.

Difficulty of the PoS Condition($Target$) The difficulty in the PoS protocol can be adjusted with this global variable in order to determine the speed of the blockchain. As the number of validators in-/decreases the difficulty is adjusted accordingly. The $Target$ is recalculated and adjusted based on the measured average block interval [58].

If a validator falls below *Target*, he becomes the leader of his shard for a particular block height and thus, has the right to append the next block to the shardchain. The probability of a validator being elected to append the next block is in proportion to the stake he or she has. For example, a validator with 50 coins has a 5 times higher chance than a validator with 10 coins. A validator is only competing in fulfilling the PoS with the other validators assigned to the same shard.

3.7 Epoch Finality And Validator Assignment

En epoch ends after a predefined number of blocks, i.e. in case $length_{ep}$ blocks have been mined in every shard since the last epoch block. After $length_{ep}$ blocks in each shard, validators agree on an epoch block by taking part in the following PoS which is a slight adaption of the PoS condition from (3.2):

$$\frac{SHA3\text{-}512(Proof_{Local} \cdot BlockHeight \cdot TimeInSeconds)}{Coins} \leq Target \tag{3.4}$$

This PoS omits the factor $Proof_{PrevBlocks}$ due to it being different for all validators becasue they are mining in different shards. The definitions of the variables in (3.4) are exactly the same as in (3.2).

Note that the PoS condition in (3.4) is evaluated by all the validators accross the network in order to come up with the next epoch block. As opposed to mining an epoch block, mining a shard block through fulfilling (3.2) is only competed for with the validators in the same shard.
Furthermore, each epoch block contains the global state of the network which serves as a checkpoint for validators joining after a certain epoch. This way, a newly joining validator is not required to download the whole Bazo history starting from the *genesis* block.
The validator assignment in Bazo is carried out by the node who 'wins' the PoS in (3.4) and has the right to append the next epoch block. Assigning the validators to the shards works as follows:

- The single shards become an ID starting from *1* up to *NofShards*,e.g. if *NofShards = 4*, then we will have shards with the IDs *1, 2, 3* and *4*.
- The node in charge with the validator assignment loops through every shard index at a time and randomly selects a validator from the vaildator set and assigns this validator to the shard. Once the validator is assigned to a shard, he gets marked to have been assigned already by using a boolean flag. This prevents a re-assignment in subsequent iterations.
- In order to uniformly assign all validators to all shards, at each shard index, only one validator is assigned to that shard.
- The shard IDs are iterated as long as all validator have been assigned to a shard

This mechanics ensures following properties: (1) every shard has at least one validator, (2) validators are assigned in a truly random fashion without the chance of choosing in which shard they want to participate and (3) uniform distribution of validators among the shards to avoid having too much validators concentrating/pooling in a single shard.
Once the validator assignment has been completed, the node undergoing this task will internally create a mapping data structure **[64] byte - int** whereas **[64]byte** denotes the validator address and **int** represents the shard ID the validator belongs to. This mapping is included in the epoch block and then broadcasted to the whole network such that every validator is aware of which shard he belongs to.

Validators who have the intention to participate in Bazo, have to wait until the next epoch block is mined and the next epoch begins. This will ease the validator assignment to the shards at the end of each epoch.
In such a case, the *StakeTx* transaction of the user who wants to join the validator set is only evaluated in the last block before the next epoch block. This way, the user will be registered in the state as a validator right before the next epoch block.

3.8 Transaction Sharding and Processing

This section explains how transactions are assigned to the single shards and processed in Bazo. Every validator locally maintains a pool of transactions from the network in which every incoming transaction is stored. When a validator successfully competes in the local PoS of his shard and becomes a leader, he performs following steps:

1. The leader retrieves transactions from his local transaction pool with

   $$s = pk_{wall} \quad mod \quad NofShards, \tag{3.5}$$

   where

   s represents the assigned shard of the leader

   pk_{wall} is the wallet adderss of the transaction sender

   mod is the mathematical modulo operator

   NofShards refers to the total number of shards the network is partitioned into

2. For each transaction, the leader verifies the signature of the transaction using the public key pk_{wall} of the transaction sender. If the signature is valid, continue. Otherwise, the transaction contains an invalid signature and the algorithm stops.

3. Through retrieving the account information of the user from the global state, the validator checks whether the user has enough coins to accomplish the transactions. If this is the case, continue. In case the user has not enough coins, the transaction is deemed invalid and the algorithm stops.

4. Once the block is full, i.e. has reached the maximum amount of transactions that fit into a block, or, all transactions dedicated to the shard of the leader have been processed, the leader follows the standard block creation procedure which includes among others generating the Merkle root hash, creation of the Bloom filter and hashing the block

Once the procedure from above has been completed, the block is sent to the network for distribution. Upon receiving block b, validators of the shard s verify the block. If b is valid, the block is appended to the corresponding shardchain. In case the block is deemed invalid, the leader who proposed b gets slashed and the validators wait for another block of the same height.

3.9 Block Height and State Synchronisation

To make sure that all shard chains grow at the same pace, the blocks at some height h need to be synchronized among all shards. From a single shard's point of view, this implies to only continue creating the block for height $h + 1$ if the validators can be sure that the other shards have validated and stored the block at height h.

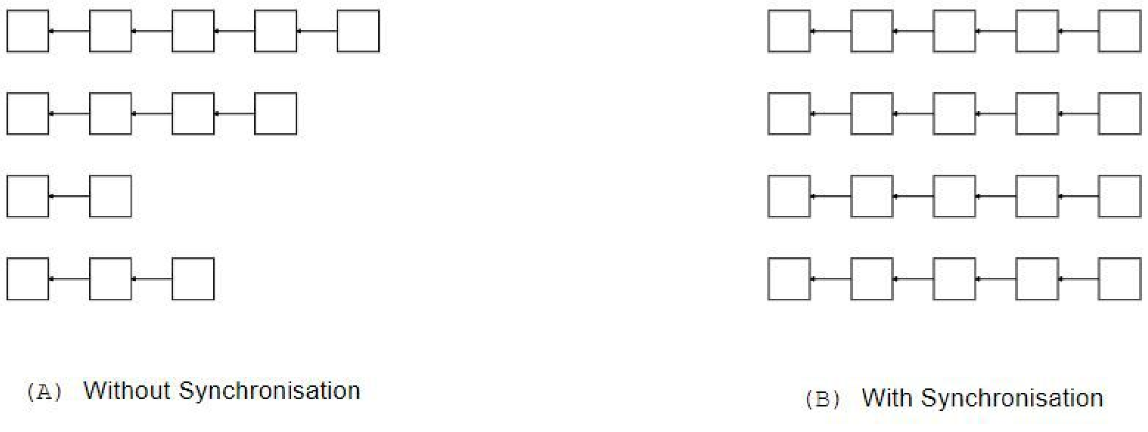

Figure 3.4: Due To Network Latency And The Speed Of The PoS, Shard Chains Might Diverge, Resulting In Different Block Heights

To operate appropriately, Bazo is synchronizing the shard chains at every height for two reasons:

- **State Synchronisation:** In order to keep the capability of issuing transactions to any user of the network, i.e. conducting cross-shard transactions, the validators at every block height need to be aware of the global state, i.e. the global account information of the nodes in the network.
- **Epoch Block Insertion:** Due to the reason that every epoch block is chained to the last blocks of each shard, every shard needs to be at the same height before the insertion of the next epoch block.

Within the scope of synchronizing the block heights among shards to determine if it is safe to start creating the next shard block, the validators also synchronize the global state. For this reason, a special data type, the *state transition*, has been introduced whose detailed structure is outlined in the implementation chapter. The state transition is created for every block mined and keeps track of how the account information changed based on the transactions in a block. In fact, every transaction basically represents a specific transition to the global state. The state transition for a block basically summarizes the cumulative changes in the accounts such as balance, transaction count and staking height.

The advantage of this approach is that every validator only has to apply *one* state transition per shard at a given block height. Figure 3.5 demonstrate how a sample state

transition might be generated. The state transition in the colored rectangle shows how the account characteristics changed during the creation of the block.
When validators receive state transitions from other shards, they process them one by one, thereby iterating through every account and applying the changes to their local account information, and as such synchronizing the global state.

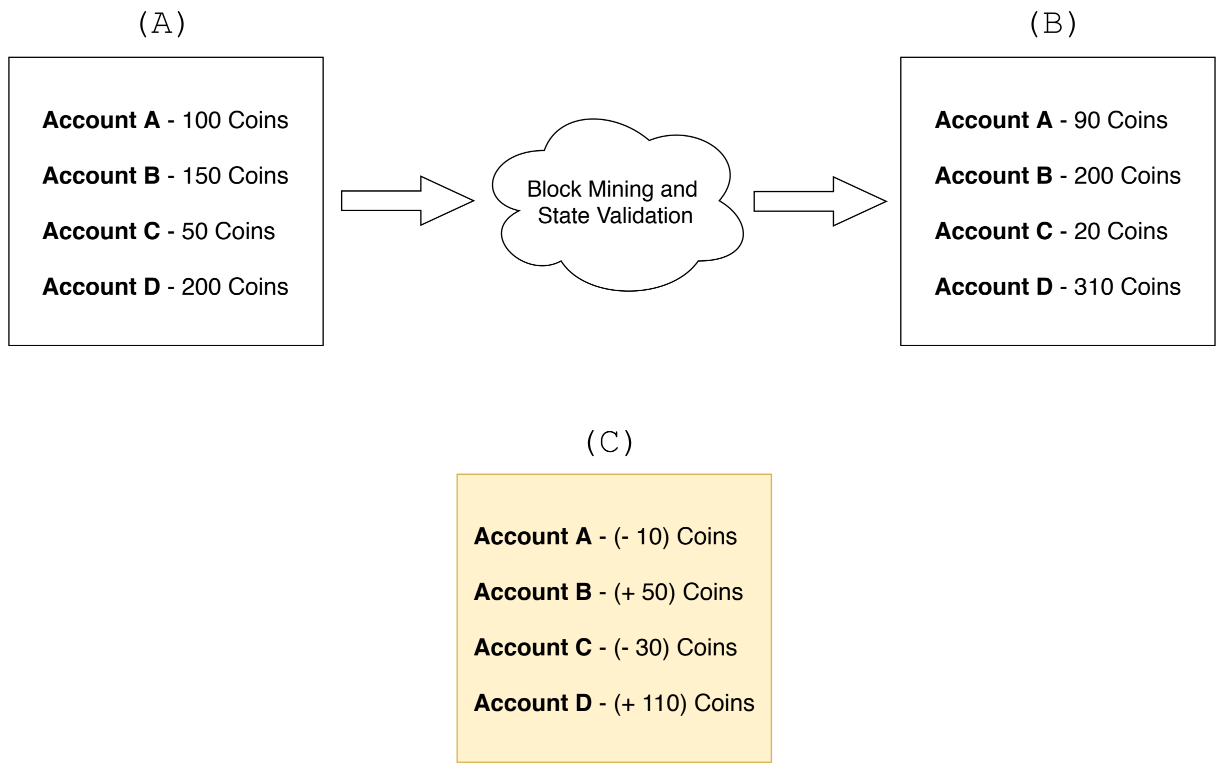

Figure 3.5: (A) - State Before Mining The Block; (B) - State After Validating Block And Applying The Transactions To The State; (C) - Generated State Transition To Be Distributed To The Network

When validators receive state transitions from the network, they have to perform at least one comparison operation to determine if it belongs to their shard or not. Furthermore, every state transition contains the block height for which it was created. Hence, validators can keep track of the number of state transitions received at block height h. If the number equals *NofShards*, they can be assured that the shard chains grow equally fast. Informally, the following algorithm is performed by every validator that receives state transitions from the network:

1. Compare the shard identifier of the state transition with my own shard. If it belongs to another shard, continue.
2. If $Height(state\ transition) > Height(last\ epoch\ block)$, where *Height* returns the height for the input, then continue. This means that the state transition is from the current epoch. If not, the state transition does not correspond to the current epoch, the algorithm stops.
3. With the hash of the state transition, check whether the received one has already been saved. This is done through comparing the state transition with the local

stash. The stash keeps track of the last 50 distinct state transitions fetched from the network. If the state transition has not been processed yet, store it in the stash. Otherwise the algorithm stops.

The aforementioned procedure is a continuously running routine in the background. Synchronisation of the state is *not* done yet at this step, i.e. whenever a state transition is received. The reason for this is that in order to properly process the state transitions, the validator needs to know exactly at which blockheight he is. That is why the synchronisation is done after the creation of a block and before the creation of the next block.

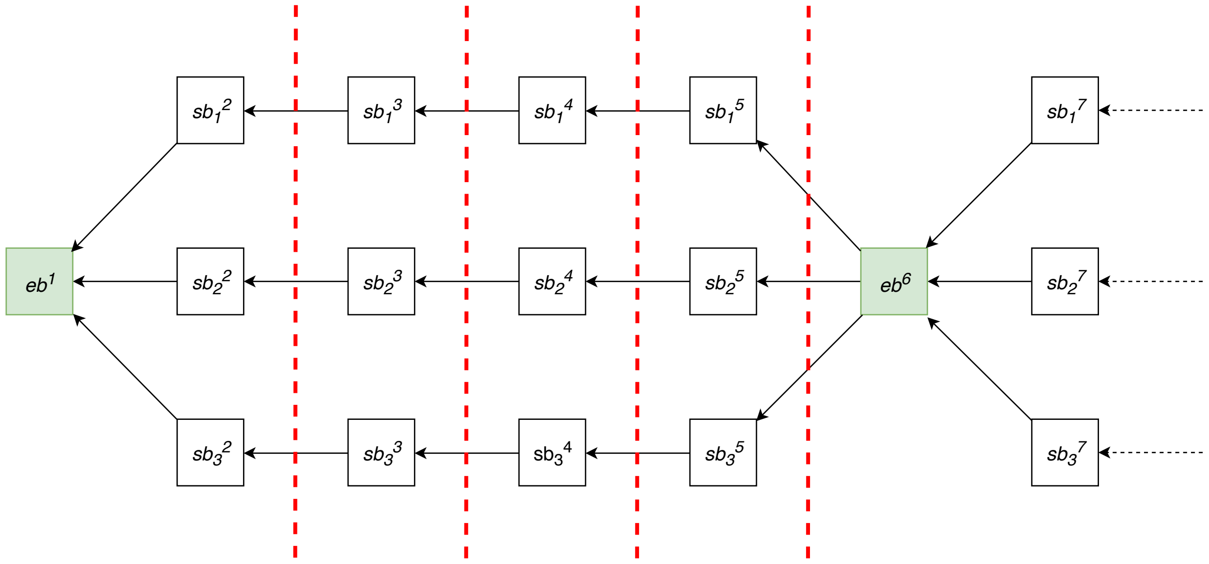

Figure 3.6: Point In Time When State Synchronisation Is Performed

The synchronisation points are highlighted in Figure 3.6. Whenever a validator reaches those stages, he performs following steps in order to synchronise his local state with the other shards:

1. Check the local stash of received state transitions and retrieve all of them which correspond to the height of my last block. If at this point, I already have received all state transitions from the other shards, then process each of them one by one. Processing a state transition consists of following sub-steps

   - Every state transition contains the relative change of the accounts, e.g. how much coins have been spent or received or by how much has the transaction counter been increased. As a validator, I iterate through all accounts and apply the adjustments to my local state. If an unknown account is encountered, then add it to my local state.
   - In addition, a state transition contains the hashes of the transactions which were validated during that block. To avoid starvation in my local transaction pool, I delete all transactions which were validated by the received state transition.

2. If state transitions from some shards for this block height have not been received yet, request them from the network. Wait for a maximum of 5 seconds to receive the response. If no response arrived for that particular shard, then continue to request the state transition from the next missing shard.
3. Repeat the last step until all state transitions from all shards have been received for the block height of my last block.

There are other ways to synchronize the global state at every block height among shards, such as processing - without validating and storing - the transactions from the other shards based on the transaction information contained in the received blocks. Recall that every validator stores incoming transactions in a transaction pool. It would be possible, for a given block, to retrieve the contained transaction hashes and pick them up from the mempool and apply them to the local state of a given validator.
This method has some evident drawbacks. Validators in a given shard would be required to process all transactions from other shards in order to be aware of the balance updates undergone in the other shards. This behaviour is against the very idea of sharding that the nodes should process a distinct set of transactions.
In addition, processing the other transactions from the mempool would not be enough to account for all account and balance updates of the other shards. The validators would also need to collect the transaction fees and block rewards from the other blocks, thus requiring not only to process their transactions, but also their blocks, e.g. to determine the benefiriary of all the fees.
With the technique of distirbuting state transitions, validators can account for any kind of balance updates, be it through receiving coins or through the collection of transaction fees.

Chapter 4

Implementation

This sections explains how the existing Bazo infrastructe is revised and extended on a technical level. Parts of the descriptions are taken from the previous publications [58, 4, 6] and updated where necessary

4.1 Transactions

4.1.1 Stake Transaction (StakeTx)

A node in Bazo with the intention to join the validator set can issue a stake transaction. Before issuing the StakeTx, the user has to generate a RSA key-pair, i.e. the commitment key-pair. Once this step is done, a stake transaction can be issued by including the commitment key pk_{comm} in the transaction. The StakeTx consists of the following parameters:

Fee A fee that has to be paid for the validator

IsStaking A boolean value that indicates whether the node wants to join or leave the set of validators.

Account The hash of the public key of the issuer

Commitment Key Included commitment key pk_{comm}. Note that this key belongs to a different key-pair than the one of the wallet of the user.

Signature The signature serves the purpose of authentication. The node digitally signs the transaction with its private key sk_{wall}

4.1.2 System Parameters (ConfigTx)

System parameters can be adjusted with this type of transaction without the need of a hard fork. The transaction must be signed by a root account in order to be accepted. Parameters to change are for example minimum staking amount, minimum waiting time, accepted time difference, slashing window size, validators per shard or the epoch length.

4.1.3 Funds Transactions (FundsTx)

Users can transfer funds from one account to another by issuing a FundsTx. In fact, transferring funds is the process of subtracting an amount of coins from the senders account and adding the same amount of coins to the receivers account. A FundsTX can be sent by validators and users.
A funds transaction has the following structure:

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

Fee A fee that has to be paid for the validator.

Transaction Counter The account nonce of the sender's account, prevents replay attacks.

From The public address pk_{wall} of the sender.

To The public address pk_{wall} of the receiver.

Signature The signature serves the purpose of authentication. The node digitally signs the transaction with its private key sk_{wall}

4.2 Blocks

Sharding introduces two new block types, namely epoch blocks and a slightly alternation of the existing block structure, i.e. shard blocks.

4.2.1 Shard Block

A shard block is almost identical to the existing block structure in Bazo. Formally, a shard block consists of the following properties:

ShardID The shard identifier of a block. This parameter is being set by the validator of a certain shard when creating the block

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 corresponding shardchain.

Number of Bloom filter Elements The number of elements that are in the bloom filter.

Bloomfilter The bloom filter can be queried with pk_{wall} whether the block contains a transaction of pk_{wall} or not. With a false-positive rate of about 10%, the size of the bloom filter is linearly increased or decreased to meet this target.

Time in Seconds (Nonce) The number of seconds a validator needed to fulfill the PoS condition.

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. Note that the transactions, i.e., the leaves of the Merkle tree, are ordered ascending before generating the Merkle root.

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

Commitment Proof This property stores a signed message of the Height that this block was created. In particular, $RSA(sk_{comm}, SHA3-512(BlockHeight))$ where sk_{comm} represents the private key that corresponds to the public commitment key pk_{comm} that was set in the initial StakeTx of the node. Other validators can use pk_{comm} to verify the proof.

Staking Proof The proof that the creator, i.e., the leader of this particular block is eligible to create it.

Height The height of a block refers to the number of previously appended blocks to the blockchain plus one.

Slashed Address A validator can submit a slashing proof when appending a block, i.e., it holds the address of the misbehaving node that must be punished.

Two Conflicting Block Hashes These two properties exhibit the block hashes where the same node has appended a block on two competing chains within the slashing window size.

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

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

4.2.2 Epoch Block

Epoch blocks are created once every shard has mined $length_{eb}$ intermediary shard blocks. Both block types are created interactively by fulfilling PoS conditions among the validators. While validators participate in the PoS against other shard members in creating the next shard block, the PoS in case of the epoch block is conducted by all validators of the network.
The epoch block consists of following fields:

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

Previous Shard Hashes This array is equal to the identifiers of the previous blocks in every shard.For instance, the epoch block of height *10* would include the hashes of all shard blocks at height *9*.

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

Height The height of a block refers to the number of previously appended blocks to the blockchain plus one.

Commitment Proof This property stores a signed message of the Height that this block was created. In particular, $RSA(sk_{comm}, SHA3-512(BlockHeight))$ where sk_{comm} represents the private key that corresponds to the public commitment key pk_{comm} that was set in the initial StakeTx of the node. Other validators can use pk_{comm} to verify the proof.

Global State Represents the global state of Bazo at the time when the epoch block was created. Technically speaking, it is a mapping of the form **[64]byte - Account** whereas **[64]byte** represents the public address of the account and **Account** stores the actual account information of a node. The benefit of including the global state in each epoch block is that new validators joining Bazo can refer to this state once they start validating in a perticular shard. Thus, new validators do not have to download the whole blockchain history and validate the transactions to create their version of the global state.

Validator-Shard Mapping Defines which validator is assigned to which shard. This property is being set by the node who has the right to append this epoch block. Once the epoch block is distributed to the network, each validator participating in the next epoch can refer to this property in order to figure out in which shard they will be validating transactions. This property is of type **[64]byte - int** whereby **[64]byte** represents the public address of the account and **int** defines the assigned shard.

NofShards Stands for the number of shards the network is partitioned into.

4.3 State Transition

State transitions keep track of the overall relative change of account information such as balance, transaction count or staking height. They are generated for every mined block and consumed by validators from the other shards in order to apply the state transition and update their local state.
A State Transition consist of following fields:

Relative State Change This parameter contains all accounts from the state and their relative changes in balance, transaction count and staking height. It is a map of the form **[64]byte - Account** where **[64]byte** is the public key of the account and **Account** represents the single adjustments of the account information of the address.

Height The height at which the state transition was created. This property is needed in the synchronisation procedure to filter the stash of state transitions and only retrieve those with the height of interest.

Shard ID The identifier of the shard in which the state transition was created. This field is utilized in the synchronisation step as well to check which shards have already been processed and which ones are still missing.

Hash Data ContractTx/FundsTxs/StakeTx/ConfigTxs The hashes of all transactions included in this state transition in sequential order. While processing a state transition, the validator can hereby delete already validated transactions form the local transaction pool to avoid unnecessary starvation of thereof.

Chapter 5

Evaluation

Testing Bazo by means of issuing transactions at a high rate turned out to be rather difficult due to a couple of reasons. First of all, it has to be made sure that transactions are sent to the network at a higher rate than the actual capacity of the network being able to handle and validate those transactions.
For instance, assuming Bazo - for a given block interval and block size - is capable of achieving a validation rate of 40 transactions per second (TPS), then, testing the blockchain through sending 10 transactions per second is technically not enough to leverage the 40 TPS of Bazo. This scenario would lead to partially filled blocks.
In order to properly stress the network, transactions need to be issued at a (much) higher rate than actually validated.

5.1 Set-Up of the Testcase

To overcome the aforementioned obstacle, sending transactions directly from the client application proved to be inefficient. The peak transaction sending rate with this approach was between 10 to 15. As a result, the tests exhibited TPS values of less than 10.
For this reason, a test environment has been set-up which does not issue transactions directly from the client application, but initiates them through unit tests in the miner application. The test scenarios can be found in the file **..\miner\sharding_test.go** of the miner application.
Using this approach, transactions can be sent to the network from artificial wallets and there is no need to open up a port for every node as it would be required when issuing transactions from the client application.
To run the unit tests, 20 nodes have been set up with corresponding wallet keys. A single test run consist of following two stages:

1. In the first step, the bootstrapping node (being a root account) transfers enough coins to the 20 wallets which were created initially. This makes sure that the wallets can continue sending valid transactions even under high transaction rates and for a long period of time.

2. During the second phase of the unit test, the 20 wallets - having enough assets - each transfer 1 coin per transaction to the bootstrapping node in a continuous fashion. This is achieved through spawning 20 concurrent go routines, each having the task of transferring funds from a dedicated wallet to the root account.

Furthermore, the configured block size is 20'000 Bytes and the epochs have a length of 9 blocks, i.e. every 10th block is an epoch block.
For every test run, the validators have to be up and running in the first place because the go routines in the unit test will be issuing transactions by writing to the respective ports of the validators.

Using this technique, it is possible to send transactions to the network at a remarkably higher rate than in the case of doing so through the client application. The subsequent section presents the performance metrics and measures for different test scenarios.

5.2 1 Validator - 1 Shard

While running the unit test for this scenario, all of the 20 go routines were continuously sending transactions to the root account. Every transaction is being written to the port of the validator, e.g. **127.0.0.1:8000**. The maximum achievable transaction sending rate with this technique is around 900 to 1000 transactions per second which is clearly better than the 10-15 transactions per second reached utilizing the client application.

Creating go routines running in parallel under the hood seem to be the key of issuing many transactions per second to the blockchain.
As a logical consequence, increasing the number of running go routines during the unit test should yield a higher transaction sending rate. But surprisingly, it does actually not matter if the unit test consists of much more than 20 go routines. A couple of test runs have shown, that there appears to be a network bottleneck in Windows which limits the maximum port write operations to around 1000 writes per second. Even when creating up to 500 concurrent go routines, the unit test struggles to send more than 950 to 1000 transactions per second to the network.

Because there does not seem to be an evident difference in the transaction rates, the unit tests only leverage 20 go routines to keep the complexity and CPU usage at a minimum.

Table 5.1 summarizes the test runs with one validator and one shard. The rightmost column presents the most important performance metric, namely the transaction validation rate per second.

Table 5.1: Testruns With One Validator And One Shard

# Testrun	Transactions sent per Second	Total Transactions Validated	Total Blocks Mined	Transaction Validation Duration [sec]	Sync Time [sec]	Avg Block Duration [sec]	Avg #Transactions in Block	TPS Overall [Tx/sec]
Run 1	803,3	33740	62	611	0	9,8	544,2	**55,2**
Run 2	941,2	73413	134	1435	0	10,7	547,8	**51,2**
Run 3	876,4	67483	113	1311	0	11,6	597,2	**51,4**

The columns have following definitions:

# Testrun The test identifier of each run

Transactions sent per Second The average rate at which transactions were sent to Bazo from the unit test.

Total Transactions Validated Total number of received and validated transactions during the test run

Total Blocks Mined Total number of created blocks during the time period between the first receipt of transactions and the mining of the last block which contains transactions

Transaction Validation Duration The time period between the first receipt of transactions until the creation of the last block which contains transactions.

Sync Time Total synchronisation time of the validator. The synchronisation duration is measured for each block separately and defined as the number of seconds needed to wait after having mined a block and being allowed to start mining the next block. Thus, during synchronisation, no mining is undergone. This metric sums up the synchronisation time of every block mined during the test run.

Avg Block Duration Average block creating interval in seconds. This measure also includes the synchronisation time and is defined as

$$\frac{Transaction\ Validation\ Duration}{Total\ Blocks\ Mined} \tag{5.1}$$

Avg # Transactions in Block Average number of transactions contained in one block, defined as

$$\frac{Total\ Transactions\ Validated}{Total\ Blocks\ Mined} \tag{5.2}$$

TPS Overall Number of validated transactions per second accross the test run, defined as

$$\frac{Total\ Transactions\ Validated}{Transaction\ Validation\ Duration} \tag{5.3}$$

Taking a closer look at Table 5.1 reveals that the TPS is actually quite high averaging 52,6 transactions per second with a peak TPS of 55,2 Tx/sec. In addition, the blocks are well-filled. Due to the block size restriction configured for the tests, a block can have a maximum of 607 transactions per block. During every test run of Table 5.1, all (intermediary) blocks were filled to the maximum except for the first and last block. During the creation of the first block, the transaction stream might not have reached 607 transactions yet and the last block usually only contains the remaining transactions.

In this set-up with one validator and one shard, the synchronisation time always equals 0 seconds because there is no need for it, since there is only one shard. The maximum rate at which the network was stressed is 941,2 transactions per second which is in accordance to the experienced maximum reachable port connections of 900 - 1000 per second.

5.3 2 Validators - 2 Shards

As explained in the design chapter, incoming transactions are sharded based on the public address of the sender. From the 20 nodes created for the unit test, transactions from 12 wallets were assigned to shard 1 and those of the remaining 8 wallets to shard 2.
Running tests with two validators and two shards has posed some non-trivial obstacles to the Bazo architecture and the way how transactions are received and propagated in the network.
In an initial run with the same set-up as in the case with one node and one shard, i.e. sending transactions only to the bootstrapping node at **127.0.0.1:8000**, the test runs were not really successful.
The reason for this is the way how incoming transactions are handled with multiple validators. When the bootstrapping node receives the transactions, he immediately broadcasts them to the other validators. Executing the same unit test with two validators revealed an uncommon behaviour of Bazo. Under the incoming transaction load at a high rate (approx. 900 - 1000 Tx/sec), the bootstrapping node was not always able to forward all transactions to the second validator. In other words, the second validator did not receive all the transactions which the first node received.
To make this problem more evident, recap that every transaction has a transaction counter which needs to be incremented for every transaction. Moreover, assume that the second validator (who might be missing some transactions) is responsible for incoming transactions of the address of **Wallet1**. In case the validator is missing the first transaction of **Wallet1**, i.e. the one with the transaction counter 0, all subsequent transactions from that address will be invalid for the validator, because the transaction with the counter 1 can only be validated if the one with the counter 0 has been approved. In turn, the transaction with the counter 2 will only be validated if the one with the counter 1 has been validated and so on.
In such a worst case scenario, the transactions from **Wallet1** can only be validated through the bootstrapping node who has received the initial transaction with the counter 0. For this to happen, the bootstrapping node needs to be assigned to the shard which is responsible for **Wallet1**. Under certain circumstances, that validator has to wait for a couple of epochs to be assigned to the shard of **Wallet1**.

To sum it up, transactions which do not reach the second validator can generally lead to a comparatively low TPS value due to the fact that the blocks are not well-filled and the network needs to evolve over a long period of time until all transactions are validated.
The first test run which exhibited this behaviour of lost transactions revealed a TPS of 8,8 Tx/sec. Obviously this value is far away form what Bazo is really able to achieve (see Table 5.1).
To overcome these issues, following workaround has been applied: During the execution of the unit tests, the wallets do not only send transactions to the bootstrapping node, but also explicitly to the port of the second validator, e.g. **127.0.0.1:8001**. This way, it can be assured that the second validator also receives every transaction.
The downside of this method is that the go routines are not anymore able to send transactions at the same high rate, but have to split their resources to send transactions to two ports instead of one.
For this reason, the test runs of this section exhibit roughly half the transaction sending rate of the scenario with one validator and one shard. Table 5.2 summarizes the performance measures with two validators and two shards.

Table 5.2: Testruns With Two Validators And Two Shards

# Testrun	Transactions sent per Second	Miner	Total Transactions Validated	Total Blocks Mined	Transaction Validation Duration [sec]	Sync Time [sec]	Avg Block Duration [sec]	Avg #Transactions in Block	TPS Overall [Tx/sec]
Run 4	416,7	Miner A	4000	8	157	60	19,6	500	25,4
		Miner B	6000	10	229	115	22,9	600	26,2
		Combined	**10000**	**17**	**229**	**60**	**13,4**	**588,2**	**43,7**
Run 5	588,2	Miner A	6000	11	205	170	18,63	545,5	29,3
		Miner B	4000	7	125	60	20,8	571,4	32
		Combined	**10000**	**18**	**205**	**60**	**11,4**	**555,6**	**48,8**
Run 6	503,4	Miner A	13990	26	552	351	21,2	538,1	25,3
		Miner B	17726	33	755	226	22,9	537,2	23,5
		Combined	**31716**	**59**	**755**	**226**	**12,8**	**537,6**	**42**

In Table 5.2, every compound of three rows represents one test run whereby the first two rows outline the measures for each validator and the third, bold formatted row summarizes the overall values for both validators for the period of the whole test.
During the tests with two validators and two shards, the obvious expectation was to experience an increase in the overall TPS metric. Unexpectedly, analyzing the TPS performance (last column in Table 5.2) revealed a slight decrease of thereof. The average TPS value is 44,8 Tx/sec with a peak at 48,8 Tx/sec.
Taking a detailed look at the measures of each miner during the test runs, it becomes evident that the synchronisation time, i.e. the time period during which no mining is performed, has a non negligible effect which leads to higher block intervals and lower TPS values for each validator. While having an average TPS of 52,6 Tx/sec in the setting of only one shard, the TPS decreases to an average of 26,8 Tx/sec with two shards per validator. With respect to the average number of transactions per block, it can be seen that also in this case the blocks are well-filled.

5.4 3 Validators - 3 Shards

Partitioning the 20 wallets into three shards resulted 7 wallets to be assigned to shard 1, 6 wallets to be handled in shard 2 and 7 wallets to be processed by shard 3.
Following test runs have been executed similarly to the case with two shards, i.e. all transactions were sent from the go routines to the three validators running at ports **127.0.0.1:8000**, **127.0.0.1:8001** and **127.0.0.1:8002**, respectively.
The test results are shown in Table 5.3. In this example, every compound of 4 rows represents one test. The bottom row of every test run summarizes the results for all validators.

Table 5.3: Testruns With Three Validators And Three Shards

# Testrun	Transactions sent per Second	Miner	Total Transactions Validated	Total Blocks Mined	Transaction Validation Duration [sec]	Sync Time [sec]	Avg Block Duration [sec]	Avg #Transactions in Block	TPS Overall [Tx/sec]
Run 7	421,4	Miner A	4856	9	244	65	27,1	539,5	19,9
		Miner B	4856	9	241	91	26,7	539,5	20,1
		Miner C	4197	9	215	95	23,8	466,3	19,5
		Combined	**13909**	**27**	**244**	**65**	**9**	**515,1**	**57**
Run 8	387,6	Miner A	4482	9	207	105	23	498	21,7
		Miner B	4251	9	211	70	23,4	427,3	20,1
		Miner C	4061	9	215	95	23,8	451,2	18,9
		Combined	**12794**	**27**	**215**	**70**	**7,9**	**473,8**	**59,5**

On close examination of the outcome, it becomes apparent that the TPS performance has recovered from the unexpected decrease in the setting with two shards. The average transaction validation rate amounts to 58,3 TPS with the maximum at 59,5 TPS.
Comparing the TPS performance per validator with the previous section reveals a slight decrease from approximately 25 Tx/sec with two shards to around 20 Tx/sec with three shards. It is noteworthy that the decrease in this case is not as severe as the decrease of the TPS from the test with one shard to the one with two shards, i.e. roughly from 51 Tx/sec to 25 Tx/sec.
The increase of the overall TPS can be explained through the fact, that the synchronisation time does not multiply with every joining validator. For instance, if a node needs to wait during synchronsation 10 seconds for the other node, assuming to have two shards, than the synchrnosation time does not necessarily double if there are three shards. In the best case, when waiting 10 seconds for some node, the validator might synchronize himself in the meantime with the third validator.
To put it differently, the synchronisation overhead does not necessarily double with every added shard. It can be observed that with three shards the TPS per node decreases slightly, but with the advantage of having another node with roughly the same TPS capacity, thus increasing the overall TPS throughput.
As observed with the examples of one and two shards, the blocks in this case are well-filled as well.

Chapter 6

Future Work

Bazo in its current implementation exhibits several limitations which impede a successful operation in production. On the one hand the blockchain only works appropriately with one validator per shard and on the other hand, there are no mechanisms to recover Bazo from failures of a validator. Furthermore, a 51% attack in a sharded environment - provided that there are multiple nodes per shard - is simpler to carry out. In order to do justice to these deficiencies, following improvements may be applied:

- To account for forks within a single shard, a sophisticated state synchronisation mechanism needs to be built which is capable of recognizing forks and rolling back the synchronisation for blocks which are not longer in the valid chain.

- Another issue with forks in a shard arises in case those forks happen at the end of a certain epoch and right before the creation of the next epoch block. In such cases, the blockchain should wait until the fork is resolved, such that there is only one validated block per shard and per block height.

- Sharding has the disadvantage that 51% attacks can be realized with actually less than 51% of the coins due to the staking power being fractioned among the shards. Currently, a validator is assigned to a shard for the lifetime of an epoch. In case of multiple validators per shard, a validator within a certain shard might rule the epoch with less than 51% of the coins and roll back blocks. To mitigate this attack scenario, the PoS condition might be revised to lighten the effect of possessing much more coins than the other validators of the same shard. Furthermore, a mechanisms might be considered to avoid a validator being assigned to a shard for a whole epoch. Making the assignment of validators per block height seems to be the most promising counter measure.

- To account for network failures or validators leaving the blockchain due to crashes, validators should be able to re-join Bazo and requesting the latest state from the network, such that blockchain can continue with the work and does not need to be launched from scratch.

- The evaluation has shown that the synchronisation overhead introduced a non negligible delay as soon as two shards are used. This in turn leads to a lower TPS value which makes the split into two shards unattractive.
  For this reason, the synchronisation mechanism could be revised to mitigate the communication overhead with two shards. It may be considered to use local states in each shard by taking advantage of self-contained proofs, thus omitting state synchronisation at every block height.

- However, testing with three shards has demonstrated that the TPS has increased again. To observe whether this trend remains with a higher number of shards, the blockchain needs to be tested with > 3 shards.

Chapter 7

Summary and Conclusions

The purpose of this thesis was in the first place to investigate different scalability techniques for blockchains through undergoing a thorough literature review. Based on the findings of this first phase, the goal was to design a scalability protocol and implement it for the Bazo blockchain.
As a result of the initial review of recent literature, the approach of *sharding* has been identified as being most appropriate for Bazo. In a next step, the required adjustments were designed and implemented to make Bazo incorporate sharding.
The main functional requirement of providing storage and validation sharding amongst the validators has been fulfilled. With respect to non-functional requirements, respectively, performance expectations, the sharded environment did not exhibit a contiunous increase in the performance. In fact, the validation capacity per second (TPS) decreased slightly when running on two shards due to the introduced communication overhead. This has led to higher block intervals and remarkably low TPS values from a single validator point of view.
Fortunately, extending the blockchain to three shards on the other hand has shown an increase of the overall TPS reaching a higher capacity than in the case with one shard, demonstrating that the technique of sharding in fact has a positive impact on the scalability of the blockchain.
In brief, it has been found that Bazo has made a successful first step towards leveraging the technique of sharding. Nonetheless, it is only fair to point out that there are still non-trivial limitations of the current system. From a functional perspective, there is the lack of support for multiple validators per shard and with respect to non-functional properties, improving the synchronisation mechanism might mitigate the communication overhead when operating with multiple shards.
In order to bring Bazo to a level of market readiness, there are still a couple of revisions required.

Bibliography

[1] R. Agrawal, P. Verma, R. Sonanis, U. Goel, A. De, S. A. Kondaveeti, and S. Shekhar. Continuous security in iot using blockchain. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6423–6427, April 2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8462513.

[2] LoRa Alliance. What is lorawan?(2018), 2015.

[3] Olivier Alphand, Michele Amoretti, Timothy Claeys, Simone Dall'Asta, Andrzej Duda, Gianluigi Ferrari, Franck Rousseau, Bernard Tourancheau, Luca Veltri, and Francesco Zanichelli. Iotchain: A blockchain security architecture for the internet of things. In *Wireless Communications and Networking Conference (WCNC), 2018 IEEE*, pages 1–6. IEEE, 2018. https://hal.archives-ouvertes.fr/hal-01705455/document.

[4] Simon Bachmann. Proof of stake for bazo, 2018.

[5] Sujit Biswas, Kashif Sharif, Fan Li, Boubakr Nour, and Yu Wang. A scalable blockchain framework for secure transactions in iot. *IEEE Internet of Things Journal*, 2018.

[6] Roman Blum. Cryptographic sortition for proof of stake in bazo, 2018.

[7] Quentin Bramas. The stability and the security of the tangle. 2018. https://hal.archives-ouvertes.fr/hal-01716111/document.

[8] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 361–377. Springer, 2017. https://www.tik.ee.ethz.ch/file/a20a865ce40d40c8f942cf206a7cba96/Scalable_Funding_Of_Blockchain_Micropayment_Networks.pdf.

[9] Daniel Burkhardt, Maximilian Werling, and Heiner Lasi. Distributed ledger. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–9. IEEE, 2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8436299.

[10] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016. https://pdfs.semanticscholar.org/f852/c5f3fe649f8a17ded391df0796677a59927f.pdf.

[11] Kristina Chodorow. *Scaling MongoDB: Sharding, Cluster Setup, and Administration.* " O'Reilly Media, Inc.", 2011.

[12] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Ieee Access*, 4:2292–2303, 2016. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7467408.

[13] Jeff Coleman. State channels wiki, 2016. https://github.com/ledgerlabs/state-channels/wiki.

[14] Marco Conoscenti, Antonio Vetro, and Juan Carlos De Martin. Peer to peer for privacy and decentralization in the internet of things. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 288–290. IEEE Press, 2017. https://iris.polito.it/retrieve/handle/11583/2665723/144242/peer-to-peer_for_privacy_and_decentralization_in_the_internet_of_things.pdf.

[15] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 173–178. ACM, 2017.

[16] Christopher Ehmke, Florian Wessling, and Christoph M Friedrich. Proof-of-property: a lightweight and scalable blockchain protocol. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 48–51. ACM, 2018.

[17] Joshua Ellul and Gordon J Pace. Alkylvm: A virtual machine for smart contract blockchain connected internet of things. In *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pages 1–4. IEEE, 2018. http://www.cs.um.edu.mt/gordon.pace/Research/Papers/bsc2018.pdf.

[18] Nicola Fabiano. The internet of things ecosystem: The blockchain and privacy issues. the challenge for a global privacy standard. In *Internet of Things for the Global Community (IoTGC), 2017 International Conference on*, pages 1–7. IEEE, 2017. https://fardapaper.ir/mohavaha/uploads/2017/11/The-Internet-of-Things-ecosystem-the-blockchain-and-privacy-issues.-The-challenge-for-a-global-privacy-standard.pdf.

[19] Kai Fan, Shangyang Wang, Yanhui Ren, Kan Yang, Zheng Yan, Hui Li, and Yintang Yang. Blockchain-based secure time protection scheme in iot. *IEEE Internet of Things Journal*, 2018.

[20] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. *IEEE Access*, 2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8370027.

[21] Bogdan Cristian Florea. Blockchain and internet of things data provider for smart applications. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4. IEEE, 2018. https://fardapaper.ir/mohavaha/uploads/2018/08/Fardapaper-Blockchain-and-Internet-of-Things-Data-Provider-for-Smart-Applications.pdf.

[22] QuarkChain Foundation. Quarkchain - a high-capacity peer-to-peer transactional system, 2018.

[23] Enrique Fynn and Fernando Pedone. Challenges and pitfalls of partitioning blockchains. *arXiv preprint arXiv:1804.07356*, 2018. `https://arxiv.org/pdf/1804.07356.pdf`.

[24] Pankaj Ganguly. Selecting the right iot cloud platform. In *Internet of Things and Applications (IOTA), International Conference on*, pages 316–320. IEEE, 2016. `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7562744`.

[25] Yuefei Gao and Hajime Nobuhara. A proof of stake sharding protocol for scalable blockchains. *Proceedings of the Asia-Pacific Advanced Network*, 44:13–16.

[26] Vincent Gramoli. The red belly blockchain. *invited talk, MIT*, 2017. `https://gramoli.redbellyblockchain.io/web/doc/talks/redbellyblockchain.pdf`.

[27] Concurrent Systems Research Group. The red belly blockchain experiments, 2017. `http://redbellyblockchain.io/papers/redbellyblockchain-experiments.pdf`.

[28] Aljosha Judmayer, Nicholas Stifter, Katharina Krombholz, and Edgar Weippl. Blocks and chains: Introduction to bitcoin, cryptocurrencies, and their consensus mechanisms. *Synthesis Lectures on Information Security, Privacy, & Trust*, 9(1):1–123, 2017.

[29] Kolbeinn Karlsson, Weitao Jiang, Stephen Wicker, Danny Adams, Edwin Ma, Robbert van Renesse, and Hakim Weatherspoon. Vegvisir: A partition-tolerant blockchain for the internet-of-things. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1150–1158. IEEE, 2018. `https://www.distributed-systems.net/my-data/var/icdcs2018/686.pdf`.

[30] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018. `https://eprint.iacr.org/2017/406.pdf`.

[31] Nir Kshetri. Can blockchain strengthen the internet of things? *IT Professional*, 19(4):68–72, 2017. `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8012302`.

[32] B Kusmierz. The first glance at the simulation of the tangle: discrete model, 2017. `https://assets.ctfassets.net/r1dr6vzfxhev/2ZO5XxwehymSMsgusUE6YG/f15f4571500a64b7741963df5312c7e7/The_First_Glance_of_the_Simulation_Tangle_-_Discrete_Model_v0.1.pdf`.

[33] Bartosz Kusmierz, Philip Staupe, and Alon Gal. Extracting tangle properties in continuous time via large-scale simulations. Technical report, working paper, 2018. `https://assets.ctfassets.net/r1dr6vzfxhev/64o6clgPcsUIIUGGYw8ksQ/5b8f1970bd06f0c29feb066a54fa6ee5/Extracting_Tangle_Properties_in_Continuous_Time_via_Large_Scale_Simulations_V2.pdf`.

[34] Alexandru Lavric and Valentin Popa. Internet of things and loraâ¢ low-power wide-area networks: a survey. In *Signals, Circuits and Systems (ISSCS), 2017 International Symposium on*, pages 1–5. IEEE, 2017.

[35] Alexandru Lavric and Valentin Popa. Loraâ¢ wide-area networks from an internet of things perspective. In *Electronics, Computers and Artificial Intelligence (ECAI), 2017 9th International Conference on*, pages 1–4. IEEE, 2017.

[36] Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using trusted execution environments. *arXiv preprint arXiv:1612.07766*, 2016. `https://arxiv.org/pdf/1612.07766.pdf`.

[37] Luke-jr. Block size limit controversy, 2015. `https://en.bitcoin.it/wiki/Block_size_limit_controversy`.

[38] Thomas Lundqvist, Andreas de Blanche, and H Robert H Andersson. Thing-to-thing electricity micro payments using blockchain technology. In *Global Internet of Things Summit (GIoTS), 2017*, pages 1–6. IEEE, 2017. `http://www.diva-portal.org/smash/get/diva2:1136256/FULLTEXT01.pdf`.

[39] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[40] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471. ACM, 2017. `https://publik.tuwien.ac.at/files/publik_268463.pdf`.

[41] Peter Middleton, Peter Kjeldsen, and Jim Tully. Forecast: The internet of things, worldwide, 2013. *Gartner Research*, 2013.

[42] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *CoRR abs/1702.05812*, 2017. `https://allquantor.at/blockchainbib/pdf/miller2017sprites.pdf`.

[43] Dennis Miller. Blockchain and the internet of things in the industrial sector. *IT Professional*, 20(3):15–18, 2018. `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8378971`.

[44] Bing Mo, Kuiren Su, Songjie Wei, Cai Liu, and Jianping Guo. A solution for internet of things based on blockchain technology. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 112–117. IEEE, 2018. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8476777`.

[45] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. .

[46] Sina Rafati Niya, Sanjiv S Jha, Thomas Bocek, and Burkhard Stiller. Design and implementation of an automated and decentralized pollution monitoring system with blockchains, smart contracts, and lorawan. In *NOMS 2018-2018 IEEE/IFIP Network*

Operations and Management Symposium, pages 1–4. IEEE, 2018. `https://files.ifi.uzh.ch/CSG/staff/Rafati/BPMS.pdf`.

[47] Asutosh Palai, Meet Vora, and Aashaka Shah. Empowering light nodes in blockchains with block summarization. In *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pages 1–5. IEEE, 2018.

[48] Jianli Pan, Jianyu Wang, Austin Hester, Ismail Alqerm, Yuanni Liu, and Ying Zhao. Edgechain: An edge-iot framework and prototype based on blockchain and smart contracts. *arXiv preprint arXiv:1806.06185*, 2018. `https://arxiv.org/pdf/1806.06185.pdf`.

[49] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017. `https://plasma.io/plasma.pdf`.

[50] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016. `https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf`.

[51] Serguei Popov. The tangle. *cit. on*, page 131, 2016. `http://www.descryptions.com/Iota.pdf`.

[52] Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the tangle. *arXiv preprint arXiv:1712.05385*, 2017. `https://arxiv.org/pdf/1712.05385.pdf`.

[53] Gowri Sankar Ramachandran and Bhaskar Krishnamachari. Blockchain for the iot: Opportunities and challenges. *arXiv preprint arXiv:1805.02818*, 2018. `https://arxiv.org/pdf/1805.02818.pdf`.

[54] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 2018.

[55] Mayra Samaniego and Ralph Deters. Blockchain as a service for iot. In *Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016 IEEE International Conference on*, pages 433–436. IEEE, 2016. `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7917130`.

[56] Mattias Scherer. Performance and scalability of blockchain networks and smart contracts, 2017. `https://umu.diva-portal.org/smash/get/diva2:1111497/FULLTEXT01.pdf`.

[57] AN Semtech. 120022. *LoRa Modulation Basics*, 2015.

[58] Livio Sgier. Bazo - a cryptocurrency from scratch, 2017.

[59] Nisarg Shah and S Sundar. Smart electric meter using lora protocols and lot applications. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1178–1180. IEEE, 2018.

[60] Pradip Kumar Sharma, Saurabh Singh, Young-Sik Jeong, and Jong Hyuk Park. Distblocknet: A distributed blockchains-based secure sdn architecture for iot networks. *IEEE Communications Magazine*, 55(9):78–85, 2017. `http://www.mihantarjomeh.com/wp-content/uploads/2018/07/4_5915577779562218259.pdf`.

[61] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: Serialization of proof-of-work events: confirming transactions via recursive elections, 2016.

[62] The Zilliqa Team. The zilliqa project - a secure, scalable blockchain platform. 2018. `https://docs.zilliqa.com/positionpaper.pdf`.

[63] ZILLIQA Team et al. The zilliqa technical whitepaper, 2017.

[64] Weinan Wang, Joseph E Magerramov, Maxym Kharchenko, Min Zhu, Aaron D Kujat, Alessandro Gherardi, and Jason C Jenks. Facilitating data redistribution in database sharding, April 23 2013. US Patent 8,429,162.

[65] Will Warren and Amir Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. *URl: https://github. com/0xProject/whitepaper*, 2017.

[66] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014. `http://gavwood.com/Paper.pdf`.

[67] Bin Yu, Jarod Wright, Surya Nepal, Liming Zhu, Joseph Liu, and Rajiv Ranjan. Iotchain: Establishing trust in the internet of things ecosystem using blockchain. *IEEE Cloud Computing*, 5(4):12–23, 2018.

[68] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018.

Abbreviations

PoW	Proof of Work
PoS	Proof of Stake
DAG	Directed Acyclic Graph
IOTA	Internet-of-Things Application
IOT	Internet-of-Things
ContractTx	Creating Account Transaction
FundsTx	Transferring Funds Transaction
ConfigTx	Changing Configuration/System Parameter Transaction
StakeTx	Staking Transaction

List of Figures

List of Tables

Appendix A

Installation Guidelines

Most of the installation guidelines are taken from the previous system installation instructions [58, 4].

A.1 Validator 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:

```
$ go get github.com/KuersatAydinli/bazo-miner
```

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

```
$ go build & bazo_miner start -d <node_name> -a <listening_port>
    -b <bootstrapping_port>
```

It is recommended to always execute **go build** as the first part of the command to make sure to use the most current compilation of the application.

The following command is a sample from the above generalized command, starting the validator **NodeA** at port **8000**:

```
$ go build & bazo_miner start -d NodeA -a 127.0.0.1:8000 -b
    127.0.0.1:8000
```

In order to start a second validator **NodeB** listening on port **8001** to connect to Bazo, execute following command:

```
$ go build & bazo_miner start -d NodeB -a 127.0.0.1:8001 -b
    127.0.0.1:8000
```

If the nodes are started for the first time, Bazo will automatically generate a separate directory for each node containing the wallet key, commitment key as well as a storage file for the blockchain. The next section further explains the structure of the key files.

A.1.1 Key Handling

All wallet 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 A.1 shows an example of a wallet key file:

Listing A.1: Bazo Wallet Key File

```
4dc418348a5c77263a70544b49ed07d78714e4df0efe277f4df20cc0a0583717
cec17514ae732cd02e0aff7f6d5125d202fc650b912c489da6ddea0fa153b904
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.

Besides wallets, Bazo also employs commitment keys which are needed in case a user intends to join the set of validators. Commitment keys are based on the RSA algorithm. Listing A.2 shows an exemplary excerpt of a commitment key file:

Listing A.2: Bazo Commitment Key File

```
y4++0vEIlHwPq2GVRQZLZpurhWkVLGM4gGgyqS77NaJSTFG7lCsLf94/oBwgu...
Sm1iXgoXEwNRT148j6I/i2K7kenb/gDBNeHjNAyjePbClHkF1qTubXA7XzUpm...
6jyMBvVhUNcGBO38LJULt/gDSH1TqabACnKvW2WQ/OrvYb3ORef+RtCDuYO6h...
3nmTTHR5ihaaed4c39P83/Zx3q/nlQSMRfS8COu/uBpuW1elA69v6ktLUo5Df...
```

A.2 Client Application

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

```
$ go get github.com/KuersatAydinli/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.

A.2.1 Transferring Funds Transaction

A transferring funds transaction is launched with the following command:

```
$ bazo_client funds --from <from_wallet> --to <to_wallet> --
    txcount <tx_count> --amount <amount> --fee <fee>
```

The command has following properties:

from The key file of the sender account (only public key needed).

to The key file of the receiver account (only public key needed).

txcount This integer parameter is linked to the sender account and needs to be increased with every newly created funds transfer transaction (starting at 0).

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.

For instance, to transfer 4000 coins from **NodeA** (WalletA.key) to **NodeB** (WalletB.key), execute following command:

```
$ bazo-client funds --from WalletA.key --to WalletB.key --
    txcount 0 --amount 4000 --fee 5
```

A.2.2 Stake Transaction

A transaction of this type can be initiated with the following command:

```
$ bazo-client staking enable --wallet <wallet_key_file> --
    commitment <commitment_key_file>
```

whereas

wallet Wallet key file of the sender account (to be found in the node directory).

commitment Commitment key file of the sender account (to be found in the node directory).

In order to let NodeB join the validator set, this command can be stated as follows:

```
$ bazo-client staking enable --wallet WalletB.key --commitment
    CommitmentB.key
```

Due to the fact that the key files are generated in the miner application, they have to be copied to the client application and renamed accordingly.

A.3 Complete Example with 3 Shards

The following sections guides the reader through the set up of running Bazo with three validators and three shards.

A.3.1 Generate Node Directories and Keys

Start the bootstrapping node with the following command executed from the miner application. In case the node is started for the first time, the node directory as well as the database and the wallet/commitment keys will be generated automatically.

```
$ go build & bazo_miner start -d NodeA -a 127.0.0.1:8000 -b
    127.0.0.1:8000
```

While the bootstrapping node is online, run following commands to launch the other two nodes:

```
$ go build & bazo_miner start -d NodeB -a 127.0.0.1:8001 -b
    127.0.0.1:8000
```

```
$ go build & bazo_miner start -d NodeC -a 127.0.0.1:8002 -b
    127.0.0.1:8000
```

As in the case of the bootstrapping node, the two additional miners will also get their corresponding keys and a local database to store the blockchain.

Note that it is required for the bootstrapping node to be running while starting up the other validators. The command line of the two validtors will indicate a succeeded connection to the bootstrapping node with the message **Adding a new miner: 127.0.0.1:8000**

Once the commands have been successful, the directory of the miner application should look as follows:

```
bazo-miner
 ├── cli
 ├── crypto
 ├── miner
 ├── NodeA
 │    ├── commitment.key
 │    ├── store.db
 │    ├── store.db.lock
 │    └── wallet.key
 ├── NodeB
 │    ├── commitment.key
 │    ├── store.db
 │    ├── store.db.lock
 │    └── wallet.key
 ├── NodeC
 │    ├── commitment.key
 │    ├── store.db
 │    ├── store.db.lock
 │    └── wallet.key
 ├── p2p
 ├── protocol
 ├── storage
 ├── vm
 └── ...
```

The epoch length is set to 9 blocks by default, i.e. every 10th block will be an epoch block. The epoch length can be customized in the configuration file **..\miner\configs.go**

When using Bazo with multiple validators, it is recommended to set the epoch length enough high to make sure the validtors are well connected when the next epoch starts. Generally, validators connect themselves first with the bootstrapping node and only after some time they discover the other validators through a neighbor request.

A.3.2 Transfer Funds and Enable Staking

Now that all three nodes have wallet and commitment keys, they are ready to join the network as validators and start mining.

To accomplish this, the bootstrapping node has to transfer enough coins to the two nodes and accept their staking transactions. For this reason, the **wallet.key** and **commitment.key** files of **NodeA, NodeB** and **NodeC** have to be copied to the client application and renamed appropriately.

Once done, restart **NodeA** with the above command in the miner application and apply following commands:

```
$ bazo-client funds --from WalletA.key --to WalletB.key --
   txcount 0 --amount 4000 --fee 5 & bazo-client staking enable
   --wallet WalletB.key --commitment CommitmentB.key
```

```
$ bazo-client funds --from WalletA.key --to WalletB.key --
   txcount 1 --amount 4000 --fee 5 & bazo-client staking enable
   --wallet WalletC.key --commitment CommitmentC.key
```

By doing so, **NodeA** transfers 4000 coins to **NodeB** and **NodeC**. At the same time, both validators issue a staking transaction. Note that the transaction counter in the second command needs to be incremented by one.

Issuing these commands in the first epoch will generate the required accounts for **NodeB** and **NodeC** and set them to staking.
At this point, **NodeB** and **NodeC** can be started in additional terminals through the previously given commands. They will connect themselves first to the bootstrapping node and after some seconds also recover each other.

As mentioned during the thesis, validators newly joining Bazo - as in this case with **NodeB** and **NodeC** - need to wait until the running epoch ends. Once both nodes receive the first epoch block from **NodA**, they will start mining blocks on their own while synchronizing the block heights among each other.
For every node, the miner application generates a two log files of the form **hash-prevhash-XXXX.txt** and **hlog-for-XXXX.txt** where **XXXX** is the port on which the nodes listen. The files can be found in the main directory.
The file **hlog-for-XXXX.txt** is basically a redirect of the print statements in the application to facilitate debugging whereas the file **hash-prevhash-XXXX.txt** records all blocks as a link from the previous hash to the actual block hash. The main purpose of this file is to give the user the ability of visualizing the blockchain. This can be done through copying the content of the file to the webpage **www.graphviz.it**.
Figure A.1 shows a sample run starting off with the bootstrapping node in the first epoch and two nodes joining in different shards in the second epoch.

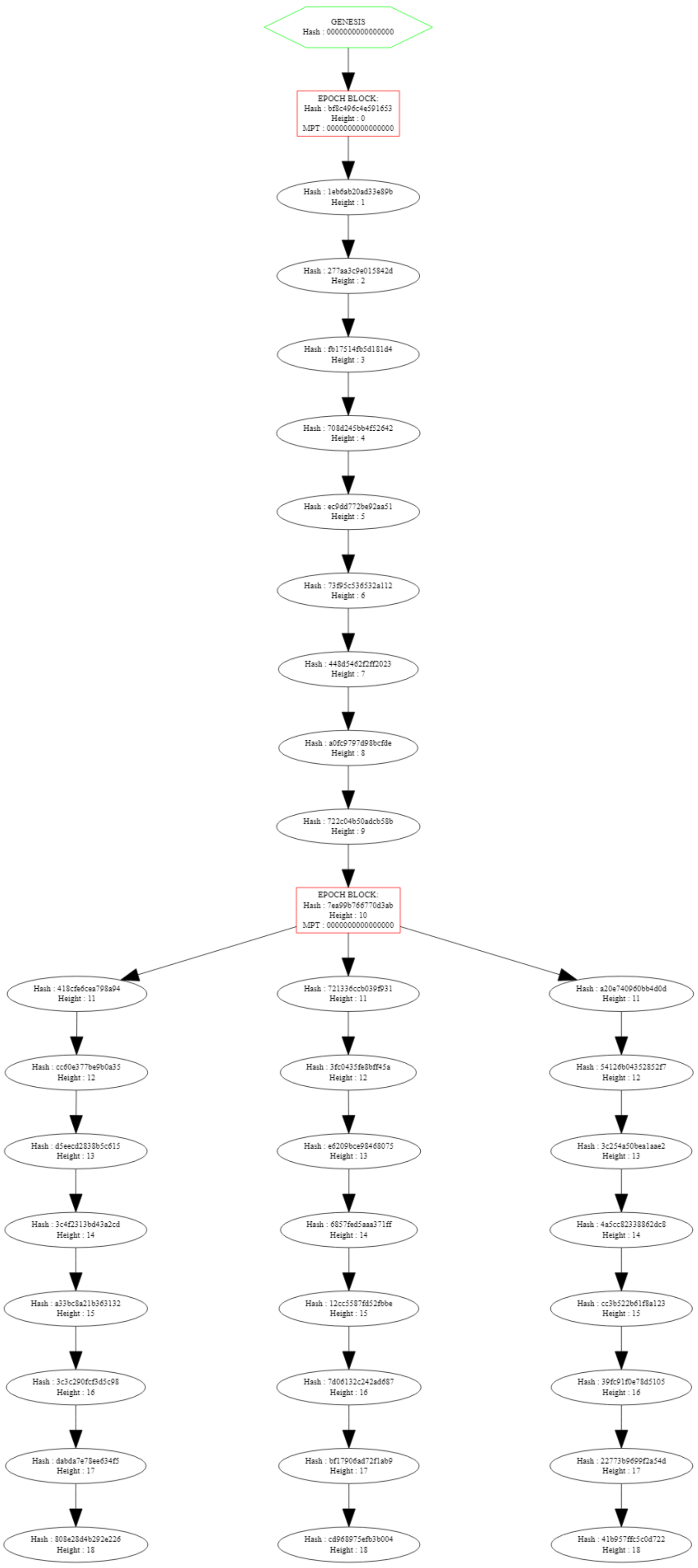

Figure A.1: Testrun With **NodeA** Starting As Bootstrapping Node And **NodeB** and **NodeC** Joining After The First Epoch

Appendix B

Contents of the CD

- Miner Application Source Code
- Client Application Source Code
- The LaTeX Source Code
- Final Thesis (.pdf)
- Final And Intermediate Presentation
- Related Work Papers