

Improving Blockchain's Throughput via Small but Highly Fault-Tolerant Shards

Tirathraj Ramburn

A Thesis

in

The Department of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

February 2023

© Tirathraj Ramburn, 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Tirathraj Ramburn

Entitled: Improving Blockchain's Throughput via Small but Highly Fault-Tolerant Shards
and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (M. Comp. Sc.)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Chair
Dr. Sudhir Mudur

Examiner
Dr. Sudhir Mudur

Examiner
Dr. Hovhannes Harutyunyan

Supervisor
Dr. Dhrubajyoti Goswami

Approved by _____
Dr. Leila Kosseim, Graduate Program Director

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Improving Blockchain's Throughput via Small but Highly Fault-Tolerant Shards

Tirathraj Ramburn

Sharding increases concurrency in a blockchain system by splitting validators into groups (shards), thus allowing the system to increase throughput in proportion to the number of shards. Contemporary sharded systems assume that shards are ‘perfect’. Therefore, shards need to be formed in such a way that they have very low (negligible) probability of failure to be considered ‘perfect’. However, such an assumption has several limitations: 1) contemporary systems tend to use conservative parameters to satisfy the ‘perfect’ shard constraint. These include the use of large shard sizes or synchronous networks, which negatively affect throughput and performance; 2) in reality, shards can be faulty, and an invalid block validated by a faulty shard cannot be detected until after the block has been appended to the blockchain since there is no fault detection mechanism during transaction processing. We first present FlexiShard: a sharding protocol that uses a hybrid fault model (i.e., Byzantine and alive-but-corrupt faults) and Flexible Byzantine Fault Tolerance as its consensus algorithm, to allow for more practical parameters and create smaller shards that are responsive, more fault tolerant and more performant than contemporary bigger shards. Next, we present SecurShard: a hierarchical fault detection model that proposes mechanisms of combining shards into groups, one-to-all mapping of a transaction block to all shards in a group, and 100% consensus requirement to validate a block with finality; all these to ensure that a potentially invalid block is detected with high probability during transaction processing. This leads to the relaxation of the ‘perfect’ shard constraint and the possible usage of smaller shards, while still maintaining the collective fault tolerance of a system. Theoretical analyzes are presented for both models which demonstrate their advantages in terms of throughput, fault tolerance and performance over contemporary sharding systems.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Dhrubajyoti Goswami. This thesis would have been impossible without his guidance, care and encouragement. I will always be grateful to him for giving me this opportunity.

I am thankful to my family and friends for their love, support and encouragement over the years. Special mention to my parents who mean everything to me. This thesis is a culmination of their hard work and vision as well.

To my parents

Table of Contents

List of Figures.....	ix
List of Tables.....	x
Chapter 1. Introduction	1
1.1. Low throughput and scalability issue of blockchain.....	1
1.2. Sharding to improve throughput.....	2
1.2.1. Direction of our work & Chapter outline	3
1.3. Node and Shard behavior in blockchains	4
1.3.1. Honest node & Faulty node.....	4
1.3.2. Honest shard & Faulty shard	4
1.4. Impact of shard size on throughput and fault tolerance	4
1.5. Limitations of contemporary shard formation protocols	6
1.5.1. The ‘perfect’ shard assumption.....	6
1.5.2. Impact of conservative parameters on performance and fault tolerance.....	6
1.5.3. Impact of rigid parameters on fault tolerance	7
1.5.4. The ‘perfect’ shard assumption & the lack of faulty shard detection during transaction- processing.....	8
1.6. Contributions.....	9
1.6.1. FlexiShard	9
1.6.2. SecurShard.....	10
1.7. Thesis Outline	10
Chapter 2. Background & Related Works.....	11
2.1. Blockchain as an immutable ledger.....	11
2.2. Consensus in blockchains	11
2.2.1. Proof of Work (PoW)	12
2.3. Techniques to Scale Blockchain.....	13
2.3.1. Increasing Block Size.....	14
2.3.2. Increasing Rate of Block Creation.....	15
2.4. Decoupling Byzantine Fault Tolerance (BFT).....	15
2.4.1. Safety & Liveness.....	16
2.4.2. Network Model	16
2.4.3. Quorum sizes & Fault thresholds	16

2.4.4. Practicality of partial synchrony due to responsiveness	18
2.5. Node Selection & Randomness generation.....	18
2.6. Shard formation using Random Sampling (Node Assignment phase)	19
2.7. Contemporary sharded systems using BFT and Random Sampling	21
2.7.1. Elastico	21
2.7.2. OmniLedger	21
2.7.3. RapidChain.....	22
2.8. Fault detection in contemporary systems during transaction processing	23
2.8.1. Chainspace	23
2.8.2. OptiShard.....	23
Chapter 3. FlexiShard: a Flexible Sharding Scheme for Blockchain based on a Hybrid Fault Model	24
3.1. Calculation of shard failure probability in contemporary systems	24
3.2. Flexible BFT.....	26
3.2.1. Hybrid fault model of Byzantine and Alive-but-corrupt (abc) faults.....	26
3.2.2. Flexible quorum sizes and increased fault tolerance in partial synchrony	27
3.2.3. Impact of flexible quorum sizes qr and qc on fault thresholds	28
3.3. Calculation of shard failure probability in FlexiShard	29
3.4. FlexiShard's use cases & Theoretical analysis	31
3.4.1. Introduction of 3 new parameters: qr , qc & <i>byz:abc split</i>	31
3.4.2. <i>Use Case 1</i> : Determining good quorum sizes, and an optimal shard size, for a fixed number of faults (F), using different fault proportions (byz:abc split).	34
3.4.3. <i>Use Case 2</i> : Determining range of acceptable fault proportions for predefined quorum sizes, total faults F and shard failure threshold $P_{Threshold}$	38
3.5. Summary	40
Chapter 4. SecurShard: A Model for Hierarchical Fault Detection in Blockchain Sharding	42
4.1. Motivation	42
4.2. Terminologies and assumptions.....	43
4.2.1. Assumptions	43
4.2.2. Public Key Infrastructure (PKI).....	43
4.2.3. Transaction Processing and Intra-shard Consensus in SecurShard.....	43
4.3. SecurShard and its protocols.....	48
4.3.1. Formation of Ref shard and Txp shards	51
4.3.2. Randomized assignment of transaction blocks to Txp shards	52
4.3.3. Invalid transaction block and potentially faulty group detection	52

4.3.4. Faulty shard detection.....	53
4.4. Theoretical Analysis of SecurShard	55
4.4.1. Increased fault tolerance due to hierarchy and one-to-all mapping	55
4.4.2. Use case analysis	57
4.4.3. Performance Analysis of SecurShard	64
4.4.4. Choosing a good mapping factor k	66
4.5. Summary	68
Chapter 5. Conclusion & Future Works	70
5.1. Related Publications	71
References	72

List of Figures

Figure 1.1 Illustration of throughput in Bitcoin versus throughput in a sharded system.	2
Figure 1.2 Illustration of the Scalability Trilemma in blockchains (CAP Theorem).	3
Figure 1.3 Components of a sharded system from [6].	5
Figure 1.4 Trade offs between a small and a large shard.	5
Figure 2.1 Data structure of a block and a blockchain.	11
Figure 2.2 Forks in a blockchain.	12
Figure 2.3 Fork resolved in Bitcoin by selecting the longest chain.	13
Figure 2.4 The correlation between block size and its propagation time to 25% (red), 50% (green) and 75% (blue) of monitored nodes respectively in the Bitcoin network [25].	14
Figure 2.5 Increasing block size and/or rate of block creation increases throughput but decreases security in the Bitcoin system [25].	15
Figure 2.6 Quorum intersection in a partially synchronous BFT protocol.	17
Figure 2.7 The process of node selection and node assignment in sharded systems.	20
Figure 3.1 Classification of faults.	26
Figure 3.2 Valid values of qr and qc in Flexible BFT illustrated by the green region.	28
Figure 3.3 Effect of increasing quorum size(s) on Byzantine and/or abc faults in Flexible BFT.	29
Figure 3.4 Effect of increasing shard size M on failure probability: FlexiShard in partial synchrony obtains comparable shard sizes to RapidChain, but smaller shards than OmniLedger.	32
Figure 3.5 Effect of qc on shard failure probability at $F = N/3$ and $qr = 3/4$ for different <i>byz:abc splits</i>	35
Figure 3.6 Effect of qc on shard failure probability at $F = 0.45N$ and $qr = 4/5$ for different <i>byz:abc splits</i>	36
Figure 3.7 First intersection of $P_{Threshold}$ with a graph gives the corresponding optimal shard size.	37
Figure 3.8 Effect of <i>byz:abc split</i> on shard failure probability for different shard sizes. For this instance, $qr=qc=0.8$ and $F=N/3$	39
Figure 3.9 Impact of increasing quorum sizes on fault thresholds in FlexiShard.	40
Figure 4.1 Three shards S of size M split each into 4 smaller Txp shards of size m_{Tx} (1:4 mapping).	42
Figure 4.2 Data structure of an input, output and a transaction respectively. A transaction comprises of multiple inputs and outputs.	45
Figure 4.3 Transaction validation flowchart using the UTXO model.	46
Figure 4.4 Formation of the Ref shard.	49
Figure 4.5 Formation of $(T \times k)$ Txp shards by the Ref shard.	50

Figure 4.6 Communication diagram of an epoch processing in SecurShard.....	54
Figure 4.7 Proof a) of SecurShard's Lemma: All k shards being faulty implies that a big contemporary shard is faulty as well.	57
Figure 4.8 Proof b) of SecurShard's Lemma: A big contemporary shard being faulty does not imply that all shards resulting from its split are faulty as well.	57
Figure 4.9 Shard failure probability versus Ref shard size when $N=2500$, $F=625$ and $f < m_{Ref} / 3$	58
Figure 4.10 Failure probability when a contemporary shard of size $M=500$ is split into different number of fragments k	60
Figure 4.11 Failure probability when a contemporary shard of size $M=400$ is split into different number of fragments k	61
Figure 4.12 Failure probability when a contemporary shard of size $M=250$ is split into different number of fragments k	62
Figure 4.13 Probability of liveness attack for different throughputs and k values	63
Figure 4.14 Collective failure probability at different throughputs when $N_{Tx}=2000$ and $F_{Tx}=500$	64
Figure 4.15 Number of message exchanges per consensus versus mapping factor k when $M=500$ is split into k fragments.	65
Figure 4.16 Expected throughput for different mapping factors and different desired throughputs.....	66
Figure 4.17 Flowchart to choose a proper Ref shard size m_{Ref}	67
Figure 4.18 Flowchart to choose a proper mapping factor k and other parameters.	68
Figure 4.19 Illustration of the differences between contemporary sharded systems and SecurShard.	69

List of Tables

Table 2.1 Differences between Nakamoto Consensus (PoW) and traditional BFT protocols.....	18
Table 3.1 Differences in fault thresholds between contemporary systems and FlexiShard.....	31
Table 4.1 Failure probabilities for different mappings when $T=4$, $M=500$, $N_{Tx}=2000$ and $F_{Tx}=500$	60
Table 4.2 Failure probabilities for different mappings when $T=5$, $M=400$, $N_{Tx}=2000$ and $F_{Tx}=500$	61
Table 4.3 Failure probabilities for different mappings when $T=8$, $M=250$, $N_{Tx}=2000$ and $F_{Tx}=500$	62

Chapter 1. INTRODUCTION

1.1. Low throughput and scalability issue of blockchain

It has been twelve years since blockchain made its first appearance in the form of the Bitcoin cryptocurrency network [1]. Many new coins and cryptocurrencies have since been introduced with varying success. Their underlying core data structure, known as distributed ledgers, are now being used across various fields such as supply chains, voting systems, healthcare, copyright protection and so on.

However, contemporary blockchain systems such as Bitcoin and Ethereum [2] are known to have low transaction throughput and high transaction latency. Throughput and transaction latency are some of the key metrics used to evaluate the performance of a blockchain. Throughput is defined as the rate at which transactions are written (and therefore processed) to a blockchain, while transaction latency usually represents the time taken for a transaction to be confirmed or committed to the blockchain starting from the time it has been submitted to the blockchain. Blockchain transaction throughputs of around 7 transactions per second in Bitcoin and 20 transactions per second in Ethereum pale in comparison to highly-scalable centralized systems like VISA. VISA can process around 1700 transactions per second and claims to be capable of handling around 24000 transactions per second [3].

The issues of low throughput and high transaction latency in contemporary blockchains arise due to two main specifications: 1) a limit to the number of transactions that can be fitted in a block (block size), and 2) the rate of block creation which is set at around 10 minutes in Bitcoin. Reparameterization of these constraints to increase throughput (by increasing block size and/or rate of block creation) have their own limits (e.g., forking and reduced security) and are further discussed in Chapter 2.

Besides having low throughput, contemporary systems operating along the same principles as Bitcoin and Ethereum are considered to be unscalable because adding more resources (i.e., more nodes to process transactions) in such systems does not lead to a throughput increase. This is because every node in the blockchain network validates every transaction, and stores the whole blockchain (also known as blockchain state). While the high amount of validation work for every block makes contemporary blockchains highly secure, these blockchains can be deemed to be only as fast as their individual nodes instead of being as fast as the aggregated computation power of the whole network. Instead, addition of more resources in such systems can make these ‘already-inefficient-systems’ even more inefficient.

Scalability and improved throughput are critical moving forward as the world looks to adopt blockchain technology. Blockchain throughput clearly needs to be increased by a few orders of magnitude if we want to make blockchain from “possible” to “practical” and leverage its benefits in our day-to-day lives. Numerous scalability solutions already exist in the literature today – each with their own benefits as well as limitations. They include sharding, off-chain payment channels, sidechains, intelligent cryptographic techniques and so on. In this thesis, we focus on the approach of reengineering a blockchain’s architecture and consensus protocols via sharding to improve throughput, decrease transaction latency, and to make the blockchain more scalable. Sharding is discussed next.

1.2. Sharding to improve throughput

A common approach to scale blockchain systems is via sharding. The key idea of sharding is to split the nodes that participate in maintaining a blockchain system into different groups (called shards or committees), such that each group of nodes process a different batch of transactions. These participating nodes are also called validators. In essence, sharding exploits concurrency to increase the throughput of a system in proportion to the number of shards, because different shards work in parallel on different sets of transactions. In contrast, an unsharded network can be assumed to be one ‘big whole’ shard whereby every node processes the same set of transactions as shown in Figure 1.1. Sharding is therefore desirable because having T shards should theoretically lead to a throughput increase by a factor of T .

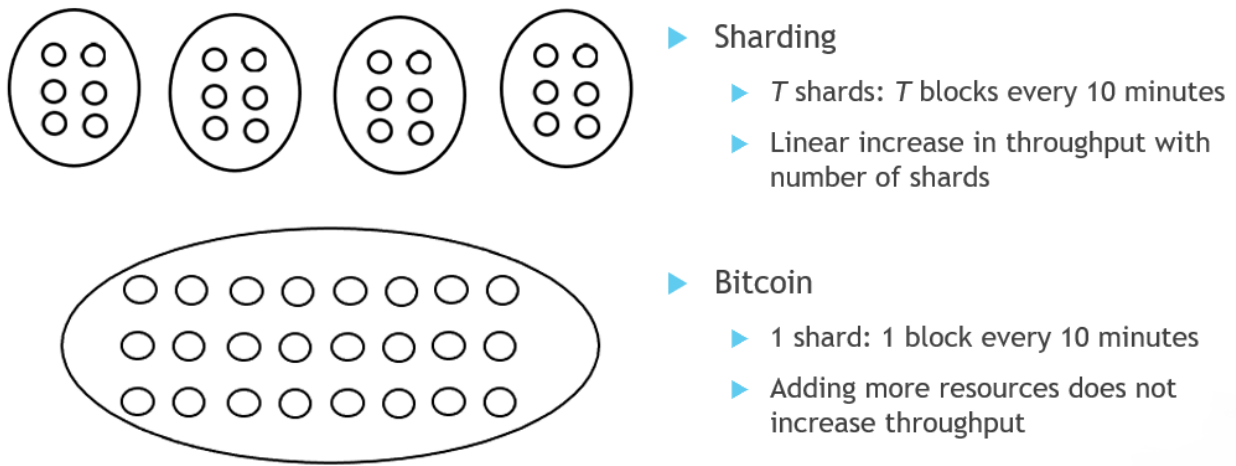


Figure 1.1 Illustration of throughput in Bitcoin versus throughput in a sharded system.

However, sharding lowers the collective fault-tolerance of a Blockchain system in general. So, there is a trade-off between performance and fault-tolerance in line with the CAP theorem (Consistency, Availability, Partition Tolerance) [4] illustrated in Figure 1.2. For example, for the Bitcoin system to fail, an adversary needs to have more than 50% ($\frac{N}{2}$) of the total mining power (or hash rate) of the system to forge blocks [5]. This failure threshold can get significantly lower in sharded blockchains: e.g., slightly more than $\frac{N}{3T}$ or $\frac{N}{2T}$ mining power, depending on the network model and the consensus algorithm, where N is the total number of nodes in the system and T is the number of shards. We assume that each node in the system has the same computing power here. In such failure scenario, the presence of even a single faulty shard can fail the entire sharded system.

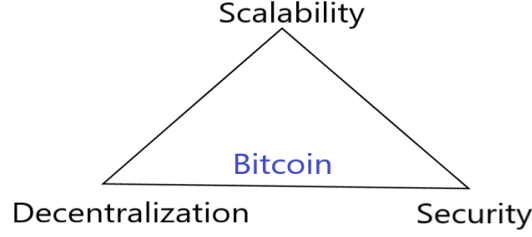


Figure 1.2 Illustration of the Scalability Trilemma in blockchains (CAP Theorem).

1.2.1. Direction of our work & Chapter outline

Liu et al. decompose blockchain systems into 7 functional components [6], as illustrated in Figure 1.3. They are:

1. Node selection.
2. Epoch Randomness.
3. Node assignment (i.e., shard formation).
4. Intra-shard consensus (reaching agreements inside shards).
5. Inter-shard consensus (also known as cross-shard transaction processing).
6. Shard reconfiguration.
7. Motivation mechanisms.

These components are not completely independent of each other, and decisions taken in a component may affect the behavior of other components. We observe that one way to improve blockchain throughput is by simply having more shards in a system (*number of shards = number of blocks per epoch = throughput*). For a fixed number of nodes N , this therefore means reducing the shard size M of every individual shard in a system since number of shards $T = \frac{N}{M}$. Reducing shard size however decreases the fault tolerance of a shard, and consequently reduces the fault tolerance of an entire sharded system (refer to sections 1.4 and 1.5). In this thesis, we therefore look at ways in which smaller shards can be formed, which however have good or comparable fault tolerance to larger shards. This is done by mainly taking design decisions in component 3 (Node assignment) and component 4 (Intra-shard consensus) of a sharded system.

Subsequently, in this Chapter, we first outline the differences between honest and malicious nodes and shards. This is followed by a discussion on how shard size M impacts fault tolerance and throughput T in a sharded system. We then discuss some limitations of the node assignment phase (component 3) of contemporary sharded blockchains - notably the use of large shard sizes and conservative parameters which may not always be necessary and practical. We conclude Chapter 1 by summarizing our contributions of the steps taken in Chapters 3 and 4 respectively to improve the process of shard creation in blockchains and obtain smaller shards than those used in contemporary systems, but with comparable or even better fault tolerance. The end goal is to increase throughput T by creating more shards, while still maintaining the collective security (fault tolerance) of a system.

1.3. Node and Shard behavior in blockchains

1.3.1. Honest node & Faulty node

A blockchain system can consist of two types of nodes: an honest (correct) node and a faulty (malicious) node. If a node is honest, it executes all the blockchain protocols honestly and always produces the correct results. In contrast, a faulty node englobes any other type of behavior that does not produce correct results. For example, if a node crashes and stops responding knowingly or unknowingly, it is considered faulty. If a node tries to append incorrect results (committing fraud) to the blockchain, it is also considered faulty.

In general, the most powerful type of fault that can exist in a blockchain is called a Byzantine fault [7]. Byzantine nodes exhibit arbitrary behavior – i.e., their behavior cannot be predicted. A Byzantine node can stop responding to other nodes and hence simulate a network failure (i.e., simulate crash faults); or it can try to collude with other malicious nodes to gain a majority vote and overrule the decision of honest nodes in the network. We use the term faulty, malicious and Byzantine interchangeably in this Chapter, unless specified otherwise. They all signify Byzantine behavior such that a Byzantine node can do anything which is computationally feasible to produce incorrect or inconsistent results and wrongly influence the blockchain.

1.3.2. Honest shard & Faulty shard

Similarly, a shard can be honest (correct) or faulty. If a shard is honest, it always produces the correct result. If a shard is faulty (malicious), it may produce the correct result if it wants to, or may not produce the correct result.

For a shard to be considered faulty in a contemporary sharded system, faulty nodes need to be present in that shard. If the number of faulty nodes in a shard is at most some fault threshold f (at most f), that shard is considered to be honest and will always produce correct results. The reverse is true as well, where the fault tolerance threshold f of a shard being exceeded implies that the shard is faulty. A faulty shard cannot be relied upon. Immediate actions need to be taken to disband faulty shards.

The fault threshold f of a shard in turn depends on the consensus algorithm (e.g., Nakamoto Consensus [1], or Byzantine Fault Tolerance (BFT)) (component 4) used inside that shard to agree on decisions [8] [9]. These fault parameters are important, and adjusting them can lead to interesting solutions to blockchain's low throughput, which is precisely what we do in Chapter 3.

1.4. Impact of shard size on throughput and fault tolerance

Contemporary systems usually divide their nodes into equal-sized groups or shards during the node assignment phase of sharding. Each shard is then assigned a different set of transactions to process. Shard size M plays an important role in determining the throughput T of a system. If smaller shards are used, more shards can be formed from a fixed number of nodes N . Latency in taking decisions is also lower for a small shard as there is a small number of message exchanges during consensus, thus making a smaller shard more performant. Small shards are therefore desirable in terms of performance and throughput.

However, they come at a cost of ‘sacrificing’ fault tolerance. Compared to a larger shard, a smaller number of nodes need to be corrupted to take-over or corrupt a small shard. In general, a smaller shard has a higher probability of failure since it is easier to corrupt a few nodes than a larger number of nodes computationally. Recall that a shard is considered faulty if the number of faulty nodes in that shard exceeds some threshold f . A good balance needs to be struck between shard size, throughput and fault tolerance. Many contemporary systems fail to do so due to their use of rather large shard sizes, which results in low throughput and negatively affects performance. We attempt to address this issue in Chapter 3. We next discuss the reasons which lead to contemporary systems prioritizing fault tolerance and adopting large shard sizes.

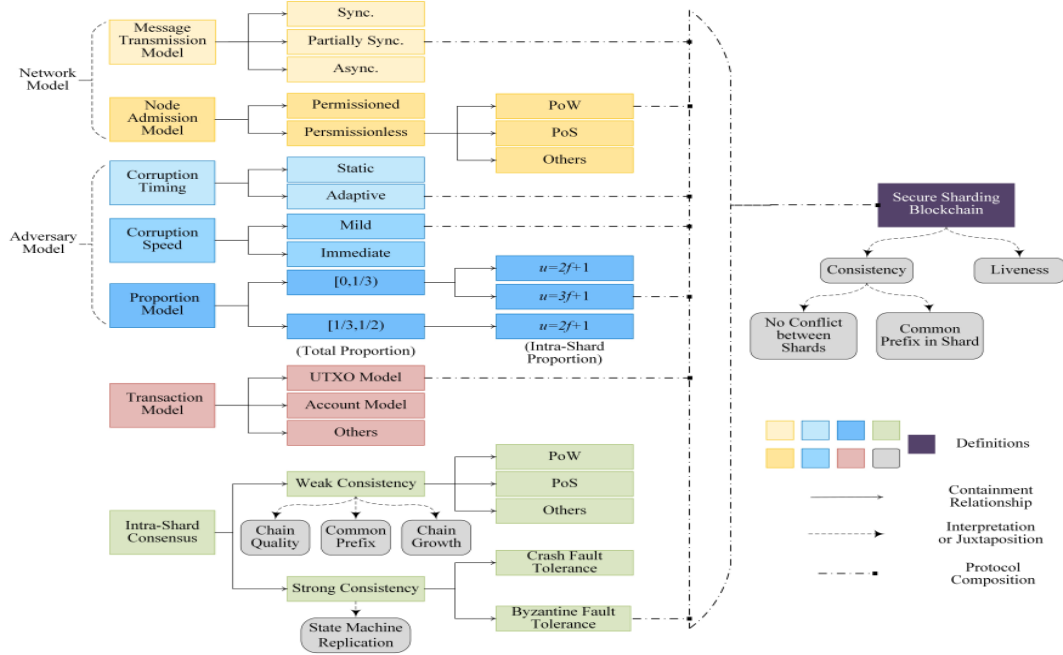


Figure 1.3 Components of a sharded system from [6].

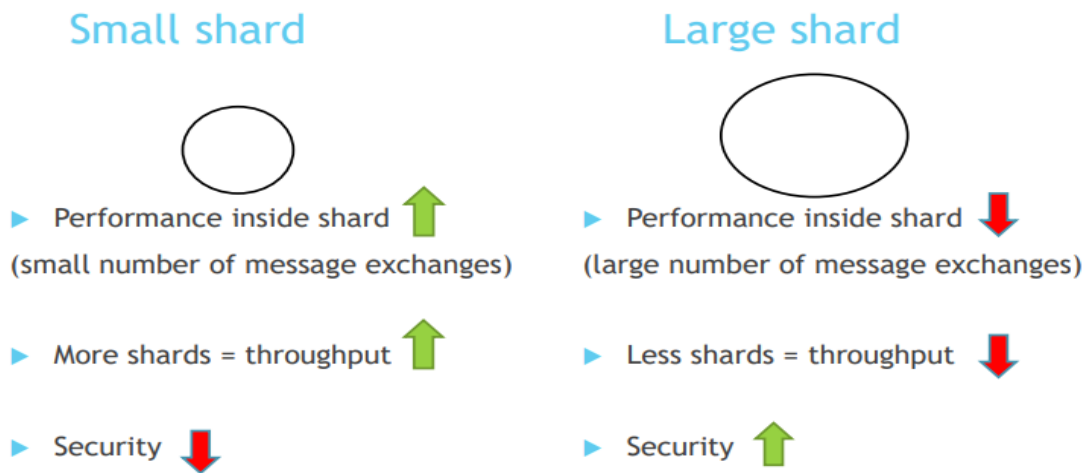


Figure 1.4 Trade offs between a small and a large shard.

1.5. Limitations of contemporary shard formation protocols

1.5.1. The ‘perfect’ shard assumption

There are usually no means to identify faulty nodes in a shard or a faulty shard itself, until after some incorrect/faulty transaction(s) has been appended to a blockchain. In other words, there are no fault detection mechanisms DURING transaction processing. Since blockchains are considered immutable, an incorrect transaction (e.g., a fraud) can cause irreversible damages. These include loss of trust in the system, an exodus of users and consequently the whole system being devalued. Mechanisms to reverse a set of incorrect transactions may be too costly as well. An incorrect transaction and/or block appended to a blockchain is therefore considered to be ‘catastrophic’ by the ecosystem.

Consequently, data that is appended to a blockchain must always be correct or ‘perfect’. Each shard processes a different batch of transaction in contemporary sharded systems, and the results of a shard are not verified by other shards before being committed. It is therefore required that each shard must always output correct results. In the presence of even a single faulty shard managing to append some invalid transaction(s) to the blockchain, the whole system is considered to have failed. In layman terms, every shard in a contemporary sharded system needs to be ‘perfect’ and can never fail.

1.5.2. Impact of conservative parameters on performance and fault tolerance

To adhere to this ‘perfect’ shard assumption, sharded systems attempt to form shards (during the node assignment phase (component 3)) in such a way that each shard has a very low or negligible failure probability (e.g., 1×10^{-10}). It can then be assumed with high confidence that these shards will always produce correct results and are ‘perfect’.

A common shard formation technique in contemporary systems is by using random sampling [10] [11]. When random sampling is used, shard failure probability is determined using the cumulative hypergeometric distribution. While we discuss this technique in more details in Chapters 2 and 3, it is worth noting that the hypergeometric distribution (and hence shard failure probability) is dictated by five parameters in a blockchain system:

1. The fault model (type of fault/adversary in the blockchain system).
2. The network model (synchrony, partial synchrony or asynchrony).
3. The consensus algorithm used for intra-shard decisions (component 4).
4. The total number of faulty nodes that can be allowed in the whole blockchain (also known as global fault threshold F).
5. The shard size.

Contemporary sharded systems tend to set conservative values for these parameters to obtain shards with low/negligible failure probability. For example, there exists one class of sharded systems [12] [13] that use the partially synchronous network model [14] [15] which is good and practical. However, they compromise on parameters 4 and 5 respectively, by setting a low global fault threshold (allowing low number of faults in the system) and using large shard sizes to obtain

negligible shard failure probabilities. A low fault threshold may not mimic the large number of attacks blockchain systems tend to suffer in reality, and hence may not be very practical. A large shard is not desirable as it tends to decrease throughput as well as increase latency (i.e., time taken to reach agreement on decisions).

To curb these performance limitations of large shard sizes, a second class of sharded systems tend to use the synchronous network model (parameter 1) [15]. This results in the formation of smaller shards which are as fault tolerant as larger shards. These small shards are more performant, and throughput increases as more shards can be created. Synchrony also allows the global fault tolerance threshold of a system (parameter 4) to be increased to more practical values as well such that a system can withstand a higher number of faults. However, synchrony as a network model is considered to be conservative and actually results in the loss of responsiveness [16] in a system. Although the reduction in shard size should lead to performance gains in such systems, the tradeoff of losing responsiveness can be detrimental to performance anyways [10]. We elaborate further on responsiveness and the tradeoff between synchrony and partial synchrony in Chapter 2.

a) Our aim

The key take-away from this subsection is that to adhere to the ‘perfect’ shard assumption, contemporary sharded systems generally adopt some conservative parameters which may not be practical in all scenarios, and which presumably limit systems from exploiting their full performance potential. In Chapter 3, we propose a new sharding protocol called FlexiShard. FlexiShard aims at striking a good balance between fault tolerance, throughput and shard size to benefit from the combined properties of contemporary synchronous and partially synchronous systems. FlexiShard attempts to use more practical parameters during shard formation to obtain shards that are small (hence performant), highly fault tolerant and responsive. The theoretical results obtained were promising. We summarize these contributions in section 1.6.

1.5.3. Impact of rigid parameters on fault tolerance

Besides conservatism, we observe that contemporary systems use rigid (fixed) values for shard size, fault thresholds and the type of faults that they can withstand. The use of conservative and rigid values in such systems is because these systems cater for worst case scenarios which may not happen very often. In practice, adversaries adapt and the number of corrupt nodes present in a system may change over time. The type of adversaries (i.e., fault model) blockchain systems deal with can also change.

Let F be the maximum number of faults a system can withstand (global fault threshold). If this assumption is violated when number of faults exceed F at some point in time, the whole system is considered to be unsafe. In general, if some system assumption is violated, a system may require a complete overhaul in terms of fault model, network model, consensus algorithm, fault thresholds and other components to conform to a different security environment. This is again a tedious and costly process.

Ideally, we want systems that can adapt to different security environments without making drastic changes to their components. For example, in the presence of a large number of faults, a system that could use a large shard size and other conservative parameters could be desirable. In the presence of a small number of faults though, if that system had the means to adapt and use a

smaller shard size for more throughput while still maintaining good fault tolerance, we could maximize the performance potential of that system. Doing so would involve the ability to use a range of values for fault threshold F and for other parameters (have a wider choice of values), instead of using fixed rigid values. In Chapter 3, we aim to achieve this property of flexibility as well. Note that some heuristic needs to be used to estimate the number of faults in a system, such as the rate of transaction failure. This is not the scope of this thesis.

1.5.4. The ‘perfect’ shard assumption & the lack of faulty shard detection during transaction-processing

We note that the ‘perfect’ shard assumption itself is conservative as it implies that the presence of a single faulty shard in a system signifies that the system has failed. Shards hence tend to be very large to obtain negligible failure probabilities and be ‘perfect’, which negatively impacts throughput and performance. The ‘perfect shard’ assumption also leads to another downside. Due to the assumption that shards will always produce correct results, contemporary systems do not have in place mechanisms for faulty shard/block detection during transaction processing. So, an invalid transaction block may get appended to the blockchain, sometimes with irreversible damage.

If sharded systems could detect potentially faulty blocks with high confidence DURING transaction processing instead of after, they would be able to allow the presence of some faulty shards in the system (until some threshold) as they could very likely detect and prevent any malicious behavior before irreversible damage is done. The ability to detect faulty shards and/or blocks during transaction processing could therefore relax the ‘perfect shard’ constraint, which is more practical. Smaller shards could also be used then which would increase throughput and decrease intra-shard communication cost (and therefore latency) during consensus.

We propose SecurShard in Chapter 4 to attempt to leverage these benefits of fault detection and help increase a blockchain system’s performance and throughput in the process. There is a disadvantage of smaller shard size though: reduced individual fault tolerance of a shard. In our work, this reduction is compensated by using collective fault tolerance of a group of shards, which is much higher, rather than the individual fault tolerance of a shard. This is elaborated in section 4.4.

1.6. Contributions

1.6.1. FlexiShard

In Chapter 3, we propose FlexiShard: a shard formation protocol that can be implemented by sharded systems to improve upon the issues of rigidity and conservative parameters that contemporary systems face today. FlexiShard aims to leverage the benefits of both synchrony and partial synchrony to obtain small, responsive and fault tolerant shards. The following are the contributions of FlexiShard:

- Changes component 4 to use Flexible BFT [17] as the consensus algorithm, in partial synchrony. Partial synchrony makes shards responsive. To the best of our knowledge, FlexiShard is the first application of Flexible BFT to blockchain and its sharding.
- Introduces and uses the hybrid fault model of Byzantine and alive-but-corrupt (abc) faults that was coined in the Flexible BFT paper [17]. Abc faults can be more practical and better suited to blockchains.
- Uses the notion of flexible quorums [17] in Flexible BFT to vary quorum sizes. In contrast, contemporary systems use fixed quorum sizes. Variable quorum sizes allow FlexiShard to: 1) increase both its global fault threshold F and intra-shard fault threshold f , and therefore tolerate a larger number of faults than contemporary systems; 2) be flexible and adapt to different security profiles. By tolerating a larger number of faults inside shards, smaller shards can be created (refer to section 3.1) without compromising fault tolerance. This is a major contribution.
- Introduces three new parameters for the calculation of shard failure probability due to the use of Flexible BFT and the hybrid fault model: qr , qc and $byz:abc\ split$. Consequently, we modify the traditional shard failure probability calculation to consider two types of faults. This is a major contribution (equations 3.10, 3.12).
- Flexibility of the new parameters provides different ways to shard a blockchain. In contrast, contemporary systems use only one approach for sharding. We describe two such ways (also called use cases or schemes) to shard a system and choose proper parameter values. By varying some parameters while keeping others constant, more such use cases can be devised. This is a major contribution as well.
- Theoretical analysis of FlexiShard demonstrates that we manage to get the combined benefits of both synchrony and partial synchrony. If proper parameter values are chosen, we can obtain small and responsive shards in partial synchrony that are comparable to synchronous shards in terms of size and fault tolerance.
- Theoretical analysis also shows that when conventional fault thresholds are exceeded, FlexiShard can obtain shards than can withstand more total faults than traditional BFT shards. FlexiShard can hence deliver similar/better performance and better fault tolerance than contemporary shards in extreme security environments.

1.6.2. SecurShard

In Chapter 4, we propose SecurShard: a hierarchical sharding protocol that can detect potential faulty blocks with high probability during transaction processing. SecurShard also attempts to relax the ‘perfect’ shard assumption and use smaller shards for decreased latency and therefore improved performance, while maintaining the collective fault tolerance of a system. SecurShard’s contributions are as follows:

- Splits an original shard S of a contemporary system into k distinct smaller shards and groups them to form T groups. T represents the throughput and is chosen by a user beforehand.
- $1:k$ mapping - Randomly assigns a transaction set (block) to k smaller shards in a group. In contrast, contemporary systems use a $1:1$ mapping of transaction block to shard. This is a major contribution.
- Introduces a Reference shard (Ref shard) that acts as a central authority to compare results from all the k shards in a group and consequently take preventive or corrective measures.
- Proposes protocols for the formation of Ref shard and transaction-processing (Txp) shards, and for the detection of potentially faulty blocks and/or shards.
- Detects potentially faulty blocks and/or shards with high probability, provided certain conditions are fulfilled. This is also a major contribution.
- Relaxes the ‘perfect’ shard assumption (but does not completely remove it): Ref shard must be perfect. If there are T transaction sets/blocks (hence T groups of k shards), only 1 shard from each group needs to be perfect to flag a potentially faulty group and preserve safety of the system (i.e., only T Txp shards out of $(T \times k)$ Txp shards need to be correct) This is a major contribution as well.
- Theoretical analysis shows that SecurShard’s $1:k$ mapping concept can provide better collective fault tolerance than contemporary systems if good shard splits (values of k) are found, while making a system more performant.
- Comes up with different fault tolerance and performance metrics to evaluate SecurShard. Results obtained support our goals of potentially faulty block detection and improved security when good shard splits and other parameter values are used.
- Provides guidelines to assist system administrators (users) in determining proper shard splits (mapping value k) and other parameter values.

1.7. Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 elaborates on some blockchain-related concepts, Byzantine Fault Tolerance, existing contemporary sharded systems and other background information. Chapter 3 discusses and theoretically analyzes FlexiShard: a flexible sharding scheme for blockchain based on a hybrid fault model and based on the application of Flexible BFT. Chapter 4 elaborates on SecurShard: a model for hierarchical fault detection for blockchain sharding. Finally, Chapter 5 concludes the thesis and discusses some future works.

Chapter 2. BACKGROUND & RELATED WORKS

2.1. Blockchain as an immutable ledger

Blockchain's underlying data structure is commonly known as a distributed ledger consisting of blocks. A block is a single unit of storage in blockchain. It contains data (such as transactions and other book-keeping related information), and a pointer (a reference) to a previous block. This pointer acts as a cryptographic link between two blocks such that multiple consecutive blocks collectively form a chain, hence the name Blockchain, as illustrated in Figure 2.1. The pointer or reference to a previous block is usually the hash value of that previous block.

A block's data and its pointer (the previous block's hash value) are combined and hashed together to produce a unique identifier (called "fingerprint") for that block. The hashing algorithm is selected such that any slight change to the block modifies the resultant hash value significantly (Avalanche effect). If a block B is tampered with, its hash value H changes. The pointer of the next block ($B+1$) that references the hash value H of the changed block no longer matches the modified hash value H' of the changed block B . The blockchain becomes invalid. To restore the blockchain to a valid state, all subsequent block pointers (and therefore blocks) need to be modified. This mechanism of hashing and chaining blocks detects potential tampering of data. Assuming that a chain is long enough, blockchain's proponents believe that modifying the hash reference of all subsequent blocks after tampering a given block is computationally unfeasible; hence the immutable property of blockchain. The cryptographic links also enable blocks (and hence transactions) to be timestamped and ordered.

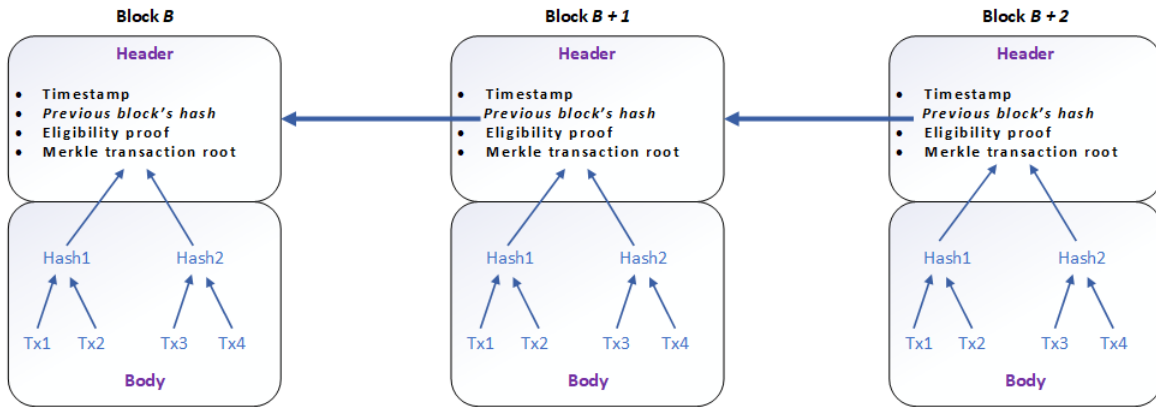


Figure 2.1 Data structure of a block and a blockchain.

2.2. Consensus in blockchains

Every node in a blockchain system maintains its own replica of the chain. Each replica needs to be up-to-date and consistent: i.e., all nodes in the system have to agree on a single and unique valid block of transactions at any given index of the chain. This common agreement has to be reached usually without the presence of a central authority to verify and validate transactions, especially in public blockchains. This is termed as the consensus problem.

Since blockchains generally use distributed peer-to-peer technology to propagate blocks to all nodes in the network, the time taken to propagate an approved valid block at some given index

is proportional to the diameter of the network. Before a block is fully propagated, nodes which are further in the network and which are unaware of the first approved block's existence in that epoch, can also attempt to publish their own blocks to the network provided certain conditions are met (e.g., solving the PoW puzzle discussed next). This can lead to forking, where at any given index, more than one distinct block is propagated in the network. Figure 2.2 illustrates some forks in a blockchain. When a fork happens, a chain diverges into at least two potential paths, which is unacceptable. A consensus protocol is therefore necessary to guarantee the total and final ordering of blocks so that there are no concurrently appended blocks. In layman terms, we want a single version of truth on all nodes in a blockchain system.

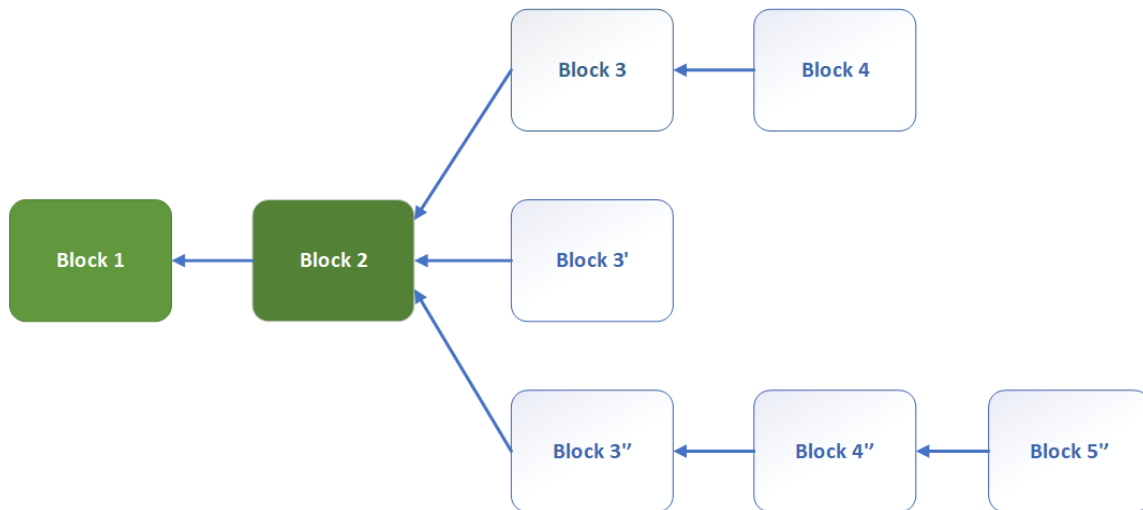


Figure 2.2 Forks in a blockchain.

2.2.1. Proof of Work (PoW)

Bitcoin solves the consensus problem using the Nakamoto Consensus, also known as Proof of Work (PoW) [1]. According to the Nakamoto Consensus, every validator node (also known as miners in Bitcoin) attempts to solve a cryptographic puzzle in an epoch (a round of processing). At the beginning of every epoch, a puzzle is given to all miners in the network. The difficulty of the puzzle is adjusted by the network such that every 10 minutes approximately, one miner in expectation can find the solution to the puzzle. That miner then proposes its block B at index Idx to the whole network. If all validation conditions are met (e.g., valid puzzle solution and valid block), every other node in the network accepts B at index Idx such that the single version of truth of the blockchain is preserved. 10 minutes was a design decision made to allow a miner sufficient time to solve the puzzle as well as to propagate its block in the network. In case of forks where multiple blocks exist at Idx (e.g., when multiple miners solve the PoW puzzle in the same epoch), Bitcoin miners wait for some time and select the longest chain as the valid chain. This fork resolution technique is illustrated in Figure 2.3. PoW's complexity makes a chain computationally secure as long as more than 50% of the computing power of its network is honest [5].

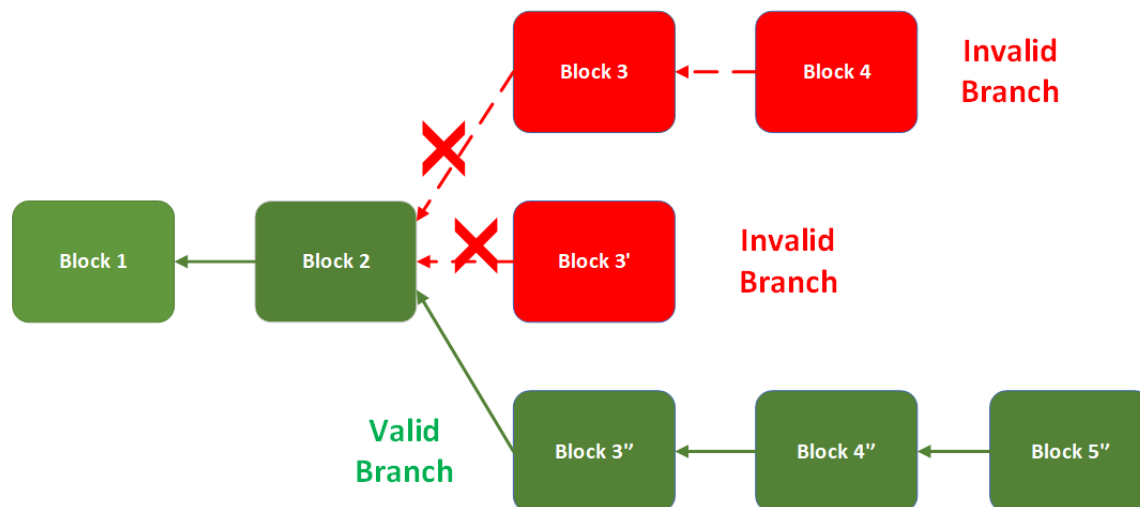


Figure 2.3 Fork resolved in Bitcoin by selecting the longest chain.

PoW is however very resource-intensive and inefficient in terms of energy consumption. Every epoch, all nodes compute the PoW puzzle but only one miner's solution is considered. The puzzle is renewed and the mining process starts again in the next epoch. This leads to wastage of resources of all other miners in an epoch. Powerful computing machines that consume a lot of energy are required as well for solving these puzzles. PoW also results in low throughput and high transaction latency. Addition of more nodes does not improve throughput or the system's performance. Instead, an 'already inefficient system' can become even more inefficient. PoW furthermore provides probabilistic block finality (or transaction commits) instead of instant finality (instant finality is desirable), such that users have to wait for multiple blocks to be appended to the blockchain to consider their transaction as committed with high probability. Other consensus alternatives to PoW exist such as Proof of Stake (PoS) [18], Proof of Burn (PoB) and Byzantine Fault Tolerance (BFT) [8] [9], which attempt to solve some of PoW's issues. In this thesis, we use and focus on BFT as the consensus algorithm. BFT is discussed in section 2.4.

2.3. Techniques to Scale Blockchain

Scalability solutions can be applied to the different hierarchical layers that make up a blockchain. There exist two main layers: Layer 1 and Layer 2. Layer 1 englobes the on-chain design of blockchain such as the consensus algorithm, block structure and the main chain itself. Layer 2 consists of off-chain approaches such as transaction-execution on separate channels, side chains, rollups [19] and off-chain computation with a view to alleviate the burden of the main chain [20]. Another special Layer called Layer 0 exists that deals with data propagation.

Ethereum [2] co-founder Vitalik Buterin outlines two prominent ways to scale a blockchain in Layer 1: firstly by 'simply increasing parameters', or secondly by trying to bring fundamental technical changes to the blockchain (changing the underlying architecture and protocols) [21]. The approach of 'increasing parameters' includes methods such as increasing the size of blocks in a blockchain to fit more transactions, or increasing the rate at which blocks are proposed in the blockchain. These techniques have physical limitations though: 1) 'normal' blockchain nodes such as personal computers cannot spend all their processing power on validating transactions. Resources must be allocated on such nodes to process transactions in their mempool, withstand

attacks and to run other standard applications simultaneously; 2) a linear increase in bandwidth and/or storage capabilities of such nodes also does not necessarily equate to linearly scaling the blockchain. The second approach of bringing fundamental improvements to the underlying architecture of a blockchain can work better and has more potential [20]. Techniques include: 1) sharding the network and/or blockchain state, 2) modifying the consensus algorithm and/or network model, 3) modifying the ledger data structure itself (e.g., using Directed acyclic graphs [22] [23] instead of linked lists) Some of these techniques are further discussed next.

2.3.1. Increasing Block Size

We discuss the impact of increasing block size in the context of Bitcoin as numerous studies have been conducted [24] [25] on information propagation in the Bitcoin network. These studies show that larger block sizes lead to higher probability of having forks in the network. Increasing block size implies increased time during the transaction-processing phase. Consequently, time to propagate a block to the whole network goes up and is amplified by the length of broadcasting hops/nodes. This is because at each hop, a whole block has to be verified before being relayed to its neighbours. During a long block propagation phase, other nodes may find the solution to the PoW puzzle and propose conflicting blocks – hence the direct correlation between increased block size and blockchain forks. Forks are bad as they lead to more wastage of mining efforts and reduce efficiency. Forks are therefore detrimental to throughput and scalability.

Increasing block size tends to make the process of block creation more centralized as well. Figure 2.4 is taken from a study done by Sompolinsky and Zohar [25]. Figure 2.4 shows that not only does block propagation time increase as a block gets larger, but that nodes closer to the origin of a proposed block can also get a head start to mine the next block as they receive the new block before other nodes in the network. Considerably, they have a higher chance of finding the PoW puzzle and proposing the next block themselves compared to nodes receiving the new block late. Hence, increasing block size has its limits and may not be the best way to increase a system's throughput on its own.

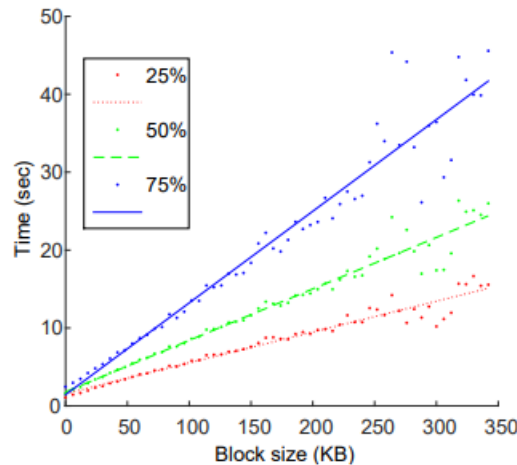


Figure 2.4 The correlation between block size and its propagation time to 25% (red), 50% (green) and 75% (blue) of monitored nodes respectively in the Bitcoin network [25].

2.3.2. Increasing Rate of Block Creation

We discuss the impact of increased rate of block creation in the context of Bitcoin. The difficulty of the PoW puzzle is adaptively adjusted by the Bitcoin protocol so that a block is mined every 10 minutes in expectation. To increase the rate of block creation, the difficulty of the PoW puzzle has to be reduced. In turn, more nodes can solve the puzzle in expectation every epoch and propose their own blocks, which can lead to forks. [25] shows that forking of a blockchain affects scalability as well as security of a system. Increasing rate of block proposal actually lowers the rate at which the main chain grows if there are multiple forks. In turn, it becomes easier for an attacker to replace a main chain if it creates blocks faster [25].

If both block size and rate of block creation is increased simultaneously, the problem of forking gets exacerbated and can make a system even more vulnerable. The outcomes of increasing block size and rate of block creation are summarised in Figure 2.5.

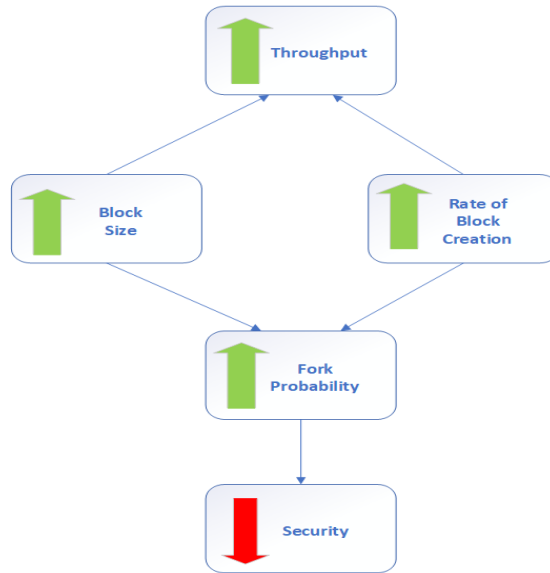


Figure 2.5 Increasing block size and/or rate of block creation increases throughput but decreases security in the Bitcoin system [25].

2.4. Decoupling Byzantine Fault Tolerance (BFT)

BFT protocols stem from a problem called the Byzantine General's Problem described in [7], whereby nodes have to reach a common agreement on some value without the presence of a central authority verifying and validating the nodes' decisions. The main problem in reaching agreement in [7] and in blockchains is the fact that parties may lie about their decisions in order to corrupt results, since participants in blockchains are very often unknown and unpredictable. Such behavior is termed as Byzantine, as described in section 1.3.1. BFT protocols guarantee correctness of decisions if the number of Byzantine faults in a system does not exceed some fault threshold. The fault threshold is influenced by the network model. Many contemporary sharded systems use BFT to reach agreements inside shards on various decisions such as transaction validation, order of blocks, certificate generation during cross-shard transactions, next block proposer and so on.

2.4.1. Safety & Liveness

Distributed systems and hence blockchains should guarantee 2 properties during execution:

1. Safety [26]: Nothing bad happens during execution.
2. Liveness: The system will eventually make progress and terminate (something good will eventually happen).

An example of safety could be that nodes in a shard will never accept an invalid block. BFT protocols are designed such that they are always safe if number of faulty nodes remain within some acceptable threshold f . Provided some conditions are met (e.g., a network is operating correctly), BFT protocols can provide liveness. These liveness conditions usually depend on the network model which is discussed next. The terms safety and liveness are used extensively in this thesis.

2.4.2. Network Model

The three prevalent network models in distributed systems are synchrony, asynchrony, and partial synchrony [15]. In a synchronous network, an adversary can only delay any message by a known finite time bound δ . Therefore, we are guaranteed that all messages will be delivered within δ time (i.e., liveness is guaranteed). Conversely, in asynchrony, this finite bound δ is unknown, but each message must eventually be delivered. Synchrony is not very practical in wide geographic networks such as the internet whereby nodes can suffer from unexpected interruptions and delays. The celebrated FLP 85 result [27] states that the consensus problem is impossible to solve in the presence of even a single faulty node in a purely asynchronous system.

Partial synchrony [14] aims at establishing a middle ground between synchrony and asynchrony. In partial synchrony, an adversary cannot withhold messages forever. Messages must be eventually delivered within some finite time upper bound δ (like in synchrony). This bound δ may not be known by honest nodes and can be chosen by the adversary (like in asynchrony). Formally, there exist some known finite time bound δ and a special event called Global Stabilization Time (GST) in partially synchronous networks. GST must be caused by an adversary to eventually happen after some unknown finite time γ such that after GST, any message must be delivered within the known time bound δ [15]. Partially synchronous protocols are hence asynchronous until GST and synchronous after GST. Protocols in partial synchrony are designed such that they are always safe but can provide liveness and termination guarantees after GST. Partial synchrony is deemed to be more practical and is used in major blockchain sharding systems [12] [13]. The practicality of partial synchrony over synchrony is further discussed in section 2.4.4. We use partial synchrony extensively in Chapters 3 and 4 of this thesis.

2.4.3. Quorum sizes & Fault thresholds

Traditional BFT protocols operate on the basis of quorum intersection [28] and generally involve two rounds of voting for reaching a consensus/agreement (e.g., deciding on a value v). The first round ensures safety within a view (an epoch can have multiple views) while the second round ensures safety across views. A quorum is a set of nodes in a voting round that is sufficient to decide on that value v . A quorum of a identical votes is required in each round respectively to decide on the value v . Any two quorums that vote in the first and second rounds need to have some number of nodes in common (i.e., quorum intersection) for correctness (safety) of v to be guaranteed.

In synchrony, if a shard needs to tolerate f faults, then there needs to be at least $2f + 1$ total nodes in the shard (i.e., shard size $M = 2f + 1$) and the quorum size a must be $f + 1$. In that case, the intersection of 2 quorums of size $f + 1$ will have at least 1 non-faulty node (note that number of nodes in quorum intersection $= 2a - M$). This non-faulty node will always vote for the correct value v in all voting rounds and is sufficient to inform all other nodes of the correct value in synchrony.

If partial synchrony is used instead, the shard size M increases to $3f + 1$ and quorum size must be set to $a = 2f + 1$. In that case, the intersection of any 2 quorums has at least $f + 1$ nodes. That is, any 2 sets of $2f + 1$ nodes (out of $M = 3f + 1$ total nodes) that vote in that view/epoch will have $f + 1$ common nodes. Since only f faulty nodes can be present in the shard, the intersection will always contain at least 1 nonfaulty node. That honest node will never deviate from the protocol. This guarantees safety in partial synchrony.

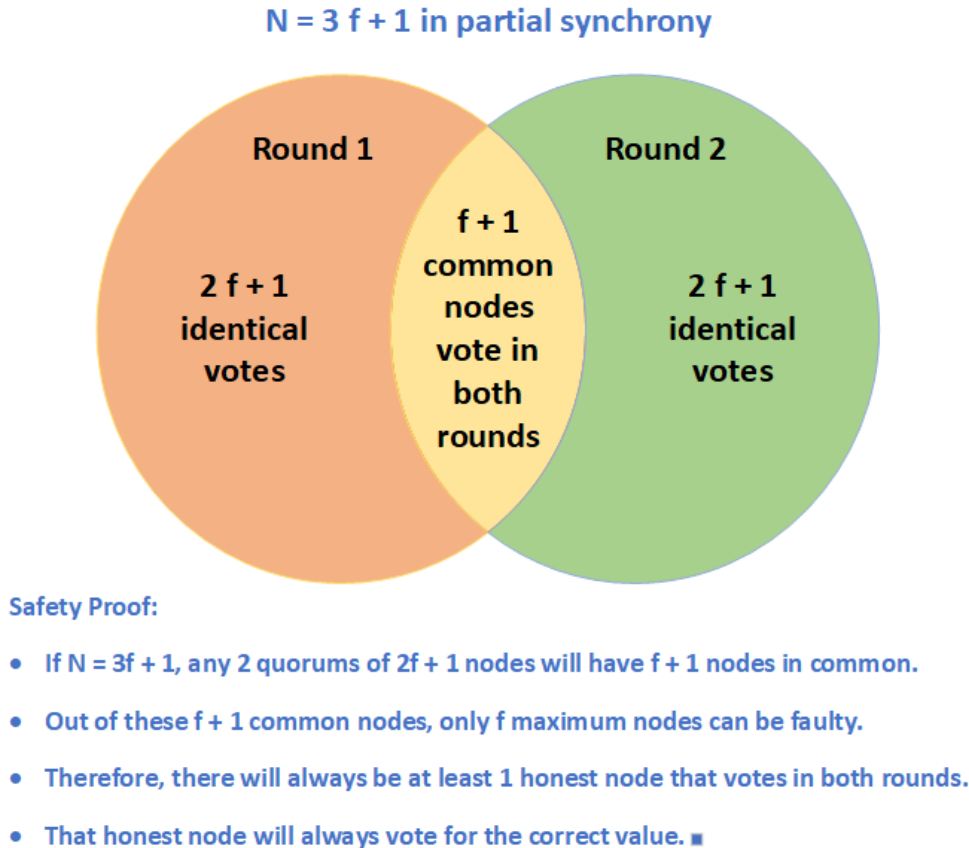


Figure 2.6 Quorum intersection in a partially synchronous BFT protocol.

Therefore, depending on the choice of network model, the fault threshold f of a shard varies. Synchronous models can tolerate less than $1/2$ Byzantine faults ($f < M/2$) in their shards while partially synchronous models can tolerate less than $1/3$ ($f < M/3$). While classic BFT protocols use a fixed quorum size a in each voting round, FlexiShard can use different quorum sizes in different rounds based on the Flexible BFT protocol [17] discussed in Chapter 3. The use of different quorum sizes in different rounds can lead to increased fault tolerance within a shard and other interesting results, which have been summarized in section 1.6.1.

2.4.4. Practicality of partial synchrony due to responsiveness

Synchronous BFT protocols require a time bound Δ to operate: meaning that they commit to messages at that fixed rate Δ . Therefore, synchronous BFT protocols are not responsive [16] [10]. A protocol is responsive if it operates at the actual network delay β instead of Δ . For example, suppose messages in practice take 1 to 2 seconds to be delivered over the internet. This represents actual network delay β . Δ is often a pessimistic value much greater than β ; e.g., 10 seconds. Hence, even if all messages are delivered within $\beta = 2$ seconds, synchronous protocols still wait for $\Delta = 10$ seconds before execution. Synchrony can therefore result in idling and inefficiency, can unnecessarily slow down a system and can hence affect performance. Partially synchronous BFT protocols which do not have a wait period in their view-change phase, such as PBFT [8] and Flexible BFT [17], operate at the actual network delay β ; therefore, they are considered to be responsive and are hence more practical to use in many situations. Table 2.1 outlines the main differences between Nakamoto consensus and classic BFT protocols.

	Nakamoto Consensus (PoW)	Synchronous BFT	Partially Synchronous BFT
Network model	Synchronous	Synchronous	Partially Synchronous
Fault tolerance (Fraction of faulty nodes)	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
Block commit time	Probabilistic finality over time	Instant finality	Instant finality
Scalability	Works well for large systems (scalable)	Works well for a few nodes	Works well for a few nodes
Latency	High	Low for few nodes	Low for few nodes
Responsive	No	No	Yes
Leader-based	Yes	Yes	Yes

Table 2.1 Differences between Nakamoto Consensus (PoW) and traditional BFT protocols.

2.5. Node Selection & Randomness generation

There is an infinite pool of nodes initially in public blockchains. A mechanism is needed to select a fixed number of nodes N to participate in an epoch. This process is known as establishing identities or node selection. Node selection is important to prevent malicious entities from spawning several other malicious nodes and taking over a network (Sybil attacks) [29]. Node selection is done by tying a potential validator financially to the network, or by limiting its computation power for example. In private blockchains, a trusted entity can establish identities using its own criteria. In public blockchains, identities can be established using PoW such that nodes wishing to participate in an epoch solve a given puzzle. This limits their computation power. Successful nodes are selected for participating in that epoch. The threshold on the number of participating nodes N can be set by adjusting the difficulty of the puzzle. An alternative way to

establish identities would be Proof of Stake (PoS) whereby nodes are selected based on the amount of assets they can commit in a network.

Blockchain systems also need some source of randomness for several important functionalities (e.g., for node assignment, PoW and block proposal). A trusted entity can introduce its own randomness in a private blockchain. Generating randomness is difficult in a public, trustless and distributed setting though and is a separate area of research on its own. While there are some proven techniques [30] [31], they are expensive in terms of resources. Randomness generation protocols for sharded blockchains need to scale as well. One way of achieving scalable randomness generation is by decomposing the initial problem into subproblems and generating randomness in smaller groups instead of a large group. However, we need randomness to securely sample these small groups. This essentially degenerates to a ‘chicken-and-egg problem’, whereby ‘sharding needs randomness needs sharding’. Blockchain systems use several techniques to generate randomness, such as [32] [33] and [34]. All exhibit different trade-offs. Although randomness generation is paramount in blockchains, the details of these different techniques are out of the scope of this thesis. We assume that some randomness generation protocol is already in place before FlexiShard or SecurShard is implemented by a system.

2.6. Shard formation using Random Sampling (Node Assignment phase)

Contemporary sharding protocols attempt to form ‘perfect’ shards by dividing validators into equally-sized shards using random sampling. Random sampling is used to spread faulty nodes as uniformly (evenly) as possible between shards. An even distribution of faulty nodes can ensure that there is a high probability of majority non-faulty nodes in any shard, i.e., the shard is non-faulty, provided some other requirements are met. When unbiased, complete random sampling works well because it is nondeterministic. Hence, an adversarial malicious node cannot predict in advance to which shard it is going to be assigned – making it harder to collude with other malicious nodes to take over and corrupt any particular shard.

The random sampling method dictates that if we sample from a population, the nature of the sampled group does not change significantly if the sampling is completely random (unbiased) [11]. For example, in a population that is 80% nonfaulty (hence 20% faulty), there is a small probability that a randomly sampled group will have more than 33% faulty nodes. In sharded systems, total number of nodes N represents the population while a shard M represents a sample.

The shard failure probability can be calculated using the cumulative hypergeometric distribution [10]. The calculation is described in Chapter 3. In a system with N total nodes, out of which F can be faulty maximally, let $prop_{Total} = F/N$ be the proportion of maximum allowable total faulty nodes in the entire system. If a shard size is M , total number of shards is T , and f is the maximum allowable number of faulty nodes in the shard, let $prop_{Shard} = \frac{f}{M} = \frac{fT}{N}$ be the proportion of maximum allowable faulty nodes inside a shard. Note that F (global fault threshold of the entire system) and f (shard fault threshold) are two different parameters.

If $prop_{Total} \approx prop_{Shard}$, then the cumulative shard failure probability based on the hypergeometric distribution is around 0.5, which is very high and undesirable. If $prop_{Total} < prop_{Shard}$, then the lower the value of $prop_{Total}$ is, the lower is the cumulative failure probability of a shard. That is, the lower the number of total faults F is in a system, or the higher the threshold of failure f is inside a shard, the lower is the failure probability of a shard [35]. The opposite is true

as well when $prop_{Total} > prop_{Shard}$. Thus, by varying fault proportions inside shards and/or in the entire system, a sharding protocol can be heavily influenced. Our goal in Chapter 3 is to maximize the value of $prop_{Shard}$ so that shard failure probability is minimized with an appropriate shard size. We do this by changing the intra-shard consensus algorithm in FlexiShard to a more fault tolerant one.

In partially synchronous contemporary systems using BFT, $prop_{Shard}$ is set to a fixed value of $\approx 1/3$ as shards can tolerate up to slightly less than $1/3$ of their nodes being faulty. Consequently, $prop_{Total}$ is set to a lower fixed conservative value of $1/4$ or $1/5$. Synchronous shards using BFT can withstand up to slightly less than $1/2$ faulty nodes; i.e., $prop_{Shard} \approx 1/2$. $prop_{Total}$ is then increased to around $1/3$ in such systems to tolerate more total faults while still maintaining low shard failure probability. This concept is extensively used in Chapter 3.

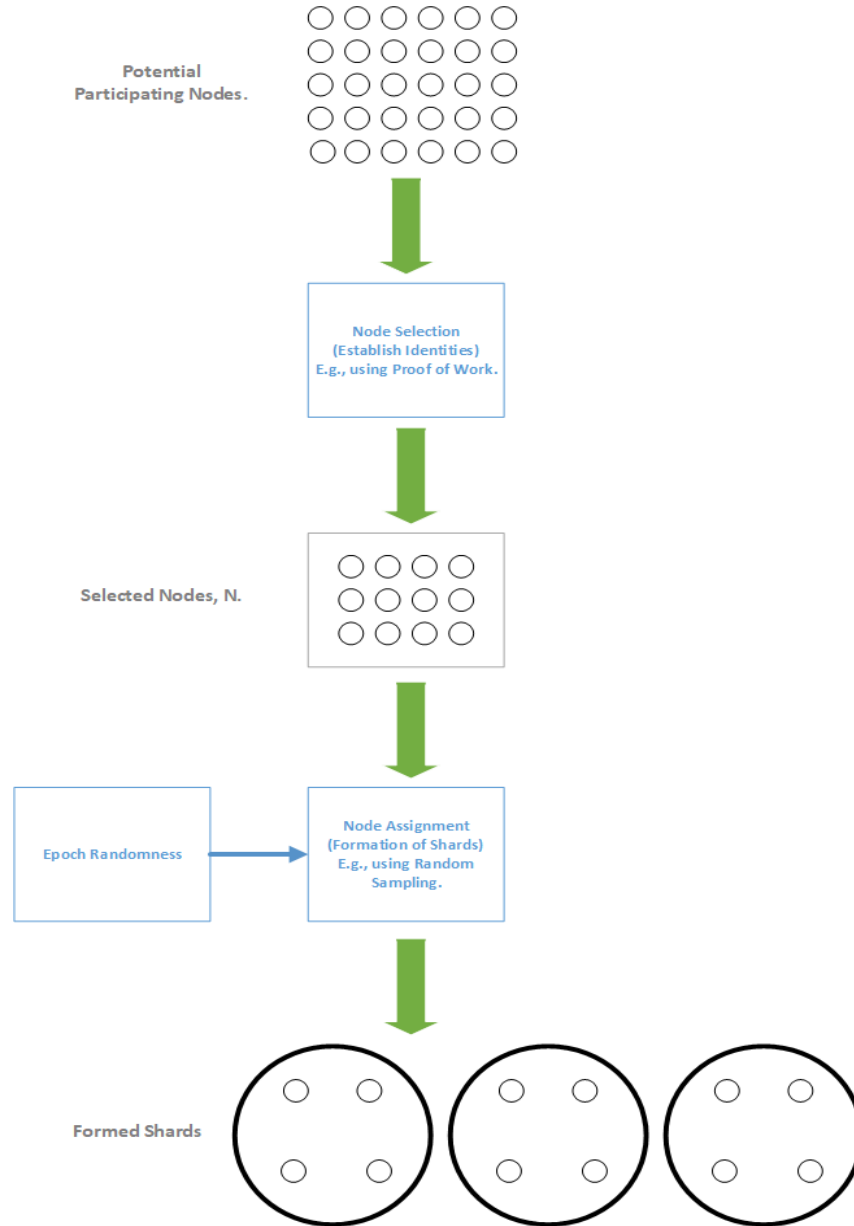


Figure 2.7 The process of node selection and node assignment in sharded systems.

2.7. Contemporary sharded systems using BFT and Random Sampling

2.7.1. Elastico

Elastico [13] is the first sharded system proposed for permissionless (public) blockchains in the literature. Identities are established by making potential participants solve a PoW puzzle. Nodes selected are then assigned to some shard, based on the least-significant bits (LSB) of the PoW's solution. Such node assignment technique is unfortunately biased, deterministic and therefore not secure. Adversaries in Elastico can keep solving and discarding PoW solutions until they find one that satisfies some desired LSB. They can then collude with other malicious parties so that all of them have the same LSB and get assigned to the same shard. Consequently, they can take over and corrupt that shard, thus failing the entire system.

Elastico uses PBFT as its consensus algorithm inside shards and operates in partial synchrony. $prop_{shard}$ is therefore set to $\approx 1/3$ while $prop_{Total}$ is set to $1/4$, which is relatively low and conservative. Since PBFT has quadratic communication cost, Elastico creates very small shards of around 100 nodes to minimize communication overhead. Shard failure probability is therefore very high. An analysis carried out by the authors of [12] shows that Elastico can fail with an overwhelming probability of 97% after only 6 epochs. Although Elastico manages to obtain gains in throughput by some orders of magnitude, the biased node-to-shard assignment process and high shard failure probability indicate fault tolerance issues.

2.7.2. OmniLedger

OmniLedger [12] attempts to fix some issues of Elastico by using random sampling to create shards so that the node-to-shard assignment process is nondeterministic and therefore unbiased. Randomness is generated using RandHound [34] which is a scalable distributed randomness generation protocol. RandHound is combined with a Verifiable-Random-Function-based (VRF) leader election algorithm [36] [37] to select an unpredictable and unbiased leader to execute the protocol.

OmniLedger uses a partially synchronous variant of BFT inside its shards called Byzcoin, which can withstand slightly less than $1/3$ faults (i.e., $prop_{shard} \approx 1/3$). Since $prop_{Total}$ should be less than $prop_{shard}$ to obtain low shard failure probability, OmniLedger sets total allowable faults in its system to be $1/4$ and obtains shard sizes of $M = 600 - 800$ for a realistic shard failure probability threshold of approximately 10^{-10} . A shard size of 600 - 800 is considered to be very large though and impacts performance negatively as Byzcoin's fallback mechanism has quadratic communication cost similar to PBFT's view-change [8]. The authors of RapidChain [10] claim that OmniLedger achieves low latency only when $prop_{Total} = 1/8$, which is a very low global fault threshold and which may not be very practical. Note that the fault thresholds of both Elastico and OmniLedger are similar: OmniLedger and Elastico both use conservative parameters as elaborated in section 1.5.

OmniLedger uses a proprietary protocol called Atomix to atomically process cross-shard transactions. Atomix is heavily dependent on users driving the protocol forward though, which may not be very practical, especially for light-weight users. Furthermore, the authors of OmniLedger claim that Atomix poses the risk of locking a user's funds if the latter crashes. This is

a potential flaw that Byzantine adversaries can exploit. Malicious nodes can thus carry out denial-of-service (DoS) attacks to lock arbitrary transactions [10].

An optimistic transaction validation protocol termed as ‘trust-but-verify’ is also implemented whereby trust (essentially security) is traded for performance. Users can choose to submit transactions to smaller shards with higher failure probability for quicker results. These transactions could then be verified later by bigger and more fault tolerant shards for finality. While this may be a good solution for micropayments (e.g., simple low-value transactions such as buying a coffee), this approach can be risky when it comes to dealing with high-value transactions. The authors of [10] claims that when $prop_{Total} = 1/4$, OmniLedger can only achieve high transaction throughputs of more than 500 transactions per second when this optimistic transaction validation protocol is used, which again comes at the expense of fault tolerance.

2.7.3. RapidChain

RapidChain [10] attempts to improve upon OmniLedger and Elastico in terms of fault tolerance by using a synchronous BFT protocol which is not responsive. The decision to switch to synchrony has a significant effect as shards can now tolerate up to slightly less than half of their nodes being faulty (i.e., $prop_{Shard} \approx 1/2$). Consequently, RapidChain can increase its total fault threshold $prop_{Total}$ to $1/3$. This threshold is significantly higher than the value of $1/4$ used in Elastico and OmniLedger, thus making RapidChain tolerant to more total faults, while obtaining reduced shard sizes. Shard sizes of approximately 300 can be obtained when failure threshold is set to 10^{-10} , compared to sizes of 700-800 in OmniLedger for the same failure threshold. RapidChain also implements some other techniques to optimise performance: the Information Dispersal Algorithm (IDA) introduced by Rabin [38] is adapted to gossip large blocks of transactions more efficiently using erasure codes [39]. RapidChain also improves upon its performance by pipelining its consensus phase.

Following the previous discussion on the non-responsiveness of synchronous BFT protocols in section 2.4.4, the authors of RapidChain argue that although their system is nonresponsive, it can obtain better performance in practice due to its smaller shard sizes. In comparison, OmniLedger uses much larger shard sizes which negatively impacts performance in terms of latency and throughput, hence hindering responsiveness anyway.

In FlexiShard, we observe that $prop_{Shard}$ and $prop_{Total}$ can be increased beyond the conventional bounds used in Elastico, OmniLedger and RapidChain to withstand even more total faults in the system and inside shards. By making shards more fault tolerant, FlexiShard can obtain relatively good (small) shard sizes and low shard failure probability. This is done by changing the fault model, the intra-shard consensus algorithm and by using variable quorum sizes during consensus. These techniques are elaborated in Chapter 3.

2.8. Fault detection in contemporary systems during transaction processing

2.8.1. Chainspace

Chainspace [40] has auditing mechanisms in place to identify potentially faulty shards. Honest nodes maintain enough information to be able to trace back inconsistencies to specific shards and therefore flag such malpractices. However, these faulty shards can only be detected after transaction processing, following which corrective measures can be undertaken against such shards. A full audit needs to be performed by an ‘auditor’ by potentially scanning the whole blockchain to detect such inconsistencies. Scanning the whole blockchain can obviously be very costly in terms of resources. In contrast, a system implementing SecurShard’s protocol could detect potentially faulty shards with high probability during transaction processing (informally in real-time) before irreversible damage is done. Fault detection in SecurShard can be relatively ‘less costly’ than Chainspace’s mechanism, as faulty shards do not need to be specifically detected to prevent a faulty block from being appended to the blockchain (refer to section 4.3). The amount of information and time that is required in *SecurShard* can therefore be significantly less (only final validation results from shards are required).

2.8.2. OptiShard

Optishard [41] is a hierarchical blockchain system that uses the ‘overlapping transactions’ method to detect potentially faulty shards during transaction processing. A central authority chooses a set of transactions (called overlapped set) and broadcasts it to all shards for validation. The overlapped set’s final result is then verified by the central authority. Shards that output incorrect results for the overlapped transaction set are flagged as faulty. However, if a malicious shard manages to process the overlapped transactions correctly but corrupts other transactions in a block, the latter will go undetected and the faulty block will still be appended to the blockchain. We also note that Optishard uses a semi-private blockchain model that shards its network deterministically (shards can be biased), and makes use of many trusted entities to increase resiliency. SecurShard’s protocol however focuses on fault detection with practicality as well. Therefore, we try to keep the number of trusted entities used to a minimum.

Chapter 3. FLEXISHARD: A FLEXIBLE SHARDING SCHEME FOR BLOCKCHAIN BASED ON A HYBRID FAULT MODEL

Following the discussions in sections 1.5, 2.4 and 2.6, we present FlexiShard: a new sharding scheme that aims to create small and responsive shards in partial synchrony, but with comparable fault tolerance to shards in synchronous systems. That is, FlexiShard aims to leverage the benefits of both synchrony and partial synchrony. Recall that the creation of smaller shards increases throughput in proportion to number of shards T , and decreases transaction latency inside shards. This is achieved by changing the consensus algorithm inside shards to a more fault tolerant one called Flexible BFT [17]. The main reason for using Flexible BFT is because the greater the number of faults a shard can tolerate, the smaller that shard can be to obtain a low failure probability. This is shown in section 3.1. Since Flexible BFT can be more fault tolerant than contemporary BFT algorithms, we presumed that its use in partial synchrony could theoretically result in smaller shards with low failure probability, in comparison to other partially synchronous systems such as OmniLedger [12]. However, application of Flexible BFT to blockchain sharding is not straightforward and adds complexity to the node assignment phase due to its unique properties (hybrid fault model and flexible quorums). We discuss and solve these challenges in sections 3.2 and 3.3. Theoretical analysis of FlexiShard confirms our presumptions. To the best of our knowledge, FlexiShard is the first application of Flexible BFT to blockchain and its sharding.

3.1. Calculation of shard failure probability in contemporary systems

We first discuss shard failure probability calculation in contemporary systems. This is done by using the cumulative hypergeometric distribution (sampling without replacement) and bounding the failure probability, as discussed in section 2.6. Failure probability can also be calculated using the binomial distribution (sampling with replacement). However, after the node selection process, the number of nodes N that can be sampled from is fixed and finite for an epoch(s). The hypergeometric distribution therefore gives a more realistic failure probability; although binomial distribution can still give a good approximation of failure.

Let X be the random variable denoting the number of faulty nodes in a sampled shard.

Let S be a shard of size $M=9$, and consider a partially synchronous system. Let number of faulty nodes in the shard be d , and let number of maximum faulty nodes in the whole system be F .

Failure probability of the shard is given by the hypergeometric distribution such that

$$P_{ShardFailure} = P(X = d) = \frac{\binom{F}{d} \binom{N-F}{M-d}}{\binom{N}{M}} \quad (3.1)$$

Contemporary systems bound the failure probability of a shard by summing up all the possibilities of failure. Although summing up all the failure possibilities is an overestimation of failure probability, we're then guaranteed that the real failure probability of a shard will never exceed that cumulative value. The actual shard failure probability can never be determined precisely as faulty and nonfaulty nodes in a shard cannot be differentiated. For S to be considered faulty, at least one third of its nodes need to be faulty since S is partially synchronous (refer to section 2.4.3); that is at least 3 nodes out of $M = 9$ need to be faulty. The cumulative failure probability for S is hence given by the cumulative hypergeometric distribution such that:

$$P(S \text{ is faulty}) = P(X = 3) + P(X = 4) + \dots P(X = 9)$$

$$P(S \text{ is faulty}) = P(X \geq \frac{M=9}{3}) = P(X \geq 3) = \sum_{d=3}^{M=9} \frac{\binom{F}{d} \binom{N-F}{M-d}}{\binom{N}{M}}$$

Formally, to determine the failure probability of a shard (cumulative), we should calculate the probability of obtaining more than f faulty nodes when randomly selecting a shard of size M without replacement from a population of N nodes that contains at most F faults. Recall that f is the maximum number of faulty nodes that a honest shard can withstand.

$$P_{Faulty} = P(X > f) = \sum_{x=f+1}^M \frac{\binom{F}{x} \binom{N-F}{M-x}}{\binom{N}{M}} \quad (3.2)$$

In partial synchrony, when $X \geq \frac{M}{3}$, a shard is considered to have failed. Failure probability is then given by

$$P_{FaultyPartialSync} = P\left(X \geq \left\lfloor \frac{M}{3} \right\rfloor\right) = \sum_{x=\left\lfloor \frac{M}{3} \right\rfloor}^M \frac{\binom{F}{x} \binom{N-F}{M-x}}{\binom{N}{M}} \quad (3.3)$$

In synchrony, a shard is considered faulty when $X \geq \frac{M}{2}$. Failure probability can then be obtained by

$$P_{FaultySync} = P\left(X \geq \left\lfloor \frac{M}{2} \right\rfloor\right) = \sum_{x=\left\lfloor \frac{M}{2} \right\rfloor}^M \frac{\binom{F}{x} \binom{N-F}{M-x}}{\binom{N}{M}} \quad (3.4)$$

From equations 3.3 and 3.4, it can be inferred that for some fixed value of N , F and shard size M , failure probability of a synchronous shard is lower than that of a partially synchronous shard. This translates to the fact that for the same shard failure probability, a smaller shard can be used in synchrony than in partial synchrony.

$$P_{FaultySync} \leq P_{FaultyPartialSync} \quad \text{for the same } N, F \text{ and } M.$$

$M_{Sync} \leq M_{PartialSync}$ for the same N , F and shard failure probability P_{Faulty} . M_{Sync} and $M_{PartialSync}$ represent the shard size of a synchronous and partially synchronous BFT shard respectively in contemporary systems.

In general, the more fault tolerant a shard is, the smaller that shard needs to be to have the same cumulative shard failure probability, in comparison to some less fault tolerant shard.

3.2. Flexible BFT

Flexible BFT [17] is a new state-of-the-art BFT algorithm that can withstand more total faults than traditional BFT algorithms. Recall that synchronous and partially synchronous BFT can withstand up to slightly less than half ($f < \frac{M}{2}$) and a third ($f < \frac{M}{3}$) of their nodes being faulty respectively. Flexible BFT, on the other hand, can support a range of fault thresholds and can withstand more than a third of its nodes being faulty in partial synchrony (up to $\frac{2}{3}$ total faulty nodes). Following previous discussions, Flexible BFT can therefore be used to obtain smaller shards theoretically than existing partially synchronous systems, for some fixed values of N , F and some fault threshold. This would consequently increase throughput of a system in partial synchrony. Flexible BFT achieves increased fault tolerance due to 2 main properties:

1. Use of a hybrid fault model comprising of Byzantine and alive-but-corrupt (abc) faults.
2. Flexible quorum sizes.

3.2.1. Hybrid fault model of Byzantine and Alive-but-corrupt (abc) faults

Contemporary BFT protocols consider Byzantine faults only. Byzantine faults attack both safety and liveness of a system. Flexible BFT introduces another type of fault called alive-but-corrupt (abc) fault which attacks safety but not liveness. Attacking liveness (i.e., preventing progress) would mean crashing off nodes (and eventually a system), cutting off communication channels, and devaluing a blockchain in the process for example, which does not benefit anyone. Instead, abc nodes can try to make gains by attacking safety (e.g., committing fraud and carrying out double-spending attacks [42]), but they also collaborate with others and help the system make progress if they cannot break safety. By collaborating with honest nodes in case of some failed attempt to attack a system's safety, abc nodes help maintain the system's value. These abc nodes can then attempt to break safety repeatedly during subsequent epochs until they are successful.

We believe that in a blockchain system, an abc-faulty node is a more rational adversary. Crashing and devaluing a blockchain system does not generally benefit an attacker in real life, as the latter usually invests considerable resources. Hence, if some system loses value, an attacker suffers from heavy monetary losses as well. Authors of Flexible BFT and we believe that a hybrid fault model of Byzantine and abc faults can be more practical and better suited to blockchain systems, especially when Byzantine nodes are not the predominant adversaries. FlexiShard is based on Flexible BFT and the associated hybrid fault model. Flexible BFT can operate in both synchrony and partial synchrony, but we focus on the latter as the network model in FlexiShard.

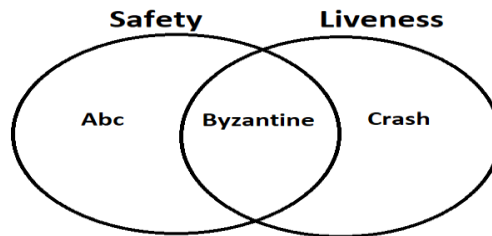


Figure 3.1 Classification of faults.

3.2.2. Flexible quorum sizes and increased fault tolerance in partial synchrony

The quorum size $a = 2f+1$ discussed in section 2.4.3 is optimal for partially synchronous BFT protocols which only consider Byzantine faults. When $N = 3f + 1$ in partial synchrony, increasing quorum size a can violate liveness (no quorum availability [43]) while decreasing a can result in having no honest nodes in the quorum intersection, which violates safety [28].

Let $q = a/M$ be the fraction of votes required to form a valid quorum. Then, in the partially synchronous BFT setting, $q = (2f+1)/(3f+1) \approx 2/3$ (we consider q to be $2/3$ for ease of expression).

$$q = \frac{a}{M} = \frac{2f+1}{3f+1} \approx \frac{2}{3} \quad \text{in partially synchronous BFT.}$$

Consequently, fraction of Byzantine faults that can be tolerated in a shard is less than

$$2q - 1 = 2\left(\frac{2}{3}\right) - 1 = \frac{1}{3}$$

In Flexible BFT, the hybrid fault model allows the second round's quorum size to be larger such that more abc faults can be tolerated at the expense of sacrificing some Byzantine faults. Let qr be the fraction of votes required to form a valid quorum in the first round of votes. Let qc be the fraction of votes required to form a valid quorum in the second voting round. Let $qc \geq qr$ [17].

Then the fraction of total allowable faults (i.e., Byzantine + abc faults) is less than $qr + qc - 1$, and the fraction of allowable Byzantine faults is at most $1 - qc$. The detailed reasons can be found in [17]. Consequently, the fraction of allowable abc faults becomes less than $qr + 2qc - 2$ (i.e., total faults – Byzantine faults). Increasing the values of qr and qc beyond the conventional value of $q = 2/3$ in Flexible BFT (using partial synchrony) allows shards to withstand more abc faults and less Byzantine faults, but more faults in total. This results in different fault-tolerance settings that can be used to obtain a range of shard sizes directly impacting performance and throughput. Therefore, instead of using a fixed quorum size of $q = \frac{2}{3}$ in both BFT voting rounds, use of Flexible BFT and varying quorum sizes to be more fault tolerant allows FlexiShard to obtain smaller shards which can perform better, but which are more resilient to existing contemporary shards in partial synchrony.

$$\text{fraction of total faults allowed} < qr + qc - 1 \quad (3.5)$$

$$\text{fraction of Byzantine faults allowed} \leq 1 - qc \quad (3.6)$$

$$\begin{aligned} & \text{fraction of abc faults allowed} \\ & < (\text{fraction of total faults} - \text{fraction of Byzantine faults}) \\ & < ((qr + qc - 1) - (1 - qc)) \\ & \text{fraction of abc faults allowed} < qr + 2qc - 2 \end{aligned} \quad (3.7)$$

The green region in Figure 3.2 represents valid quorum size values for qr and qc . The values of qr and the corresponding values of qc that can be chosen for a particular setting are bounded by equations 3.5, 3.6 and 3.7, and other additional constraints discussed in [17].

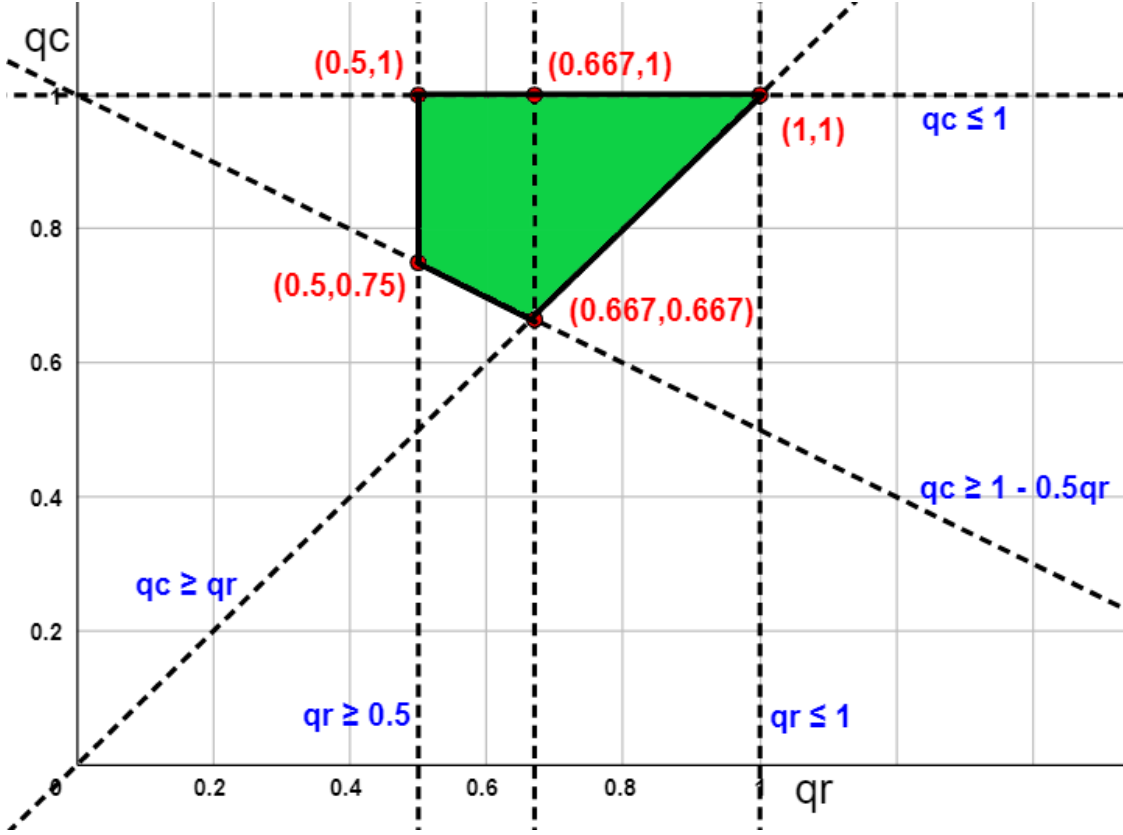


Figure 3.2 Valid values of qr and qc in Flexible BFT illustrated by the green region.

3.2.3. Impact of flexible quorum sizes qr and qc on fault thresholds

The quorum size q in classic BFT algorithms is fixed to $\frac{2}{3}$ in partial synchrony. If q is increased, there is a risk of losing liveness due to a property called quorum availability [43]. If q is decreased, safety can be compromised as there may not be enough honest nodes in the quorum intersection. In contrast, in Flexible BFT, if quorum size qc is increased, number of total faults that can be allowed in a shard or system in general increases according to equation 3.5. Equation 3.6 demonstrates that an increase in qc results in a decrease in the number of BFT faults that a shard can withstand. However, since the total number of faults allowed has increased, this implies that as qc increases, a shard can withstand more abc faults, albeit at the expense of sacrificing some Byzantine fault tolerance. This statement is confirmed by equation 3.7: logically, qc can be increased to tolerate more abc faults only, since they do not attack liveness. The property of quorum availability is therefore preserved when qc is increased, unlike in classic BFT setting.

Similarly, an increase in the value of qr implies an increase in the number of total faults and abc faults that a shard can withstand. Since $qc \geq qr$ [17], higher values of qr entails the use of higher values of qc . Therefore, increasing qr indirectly results in a decrease in Byzantine fault tolerance. By observing the behavior of a blockchain system, and by estimating the number of

faults and the type of faults using some heuristic, quorum sizes can therefore be changed accordingly to adapt to different security environments. In the presence of a large number of abc faults, where most nodes are observed to collaborate and make progress, larger qr and qc values could be used to tolerate a larger number of total faults inside a shard. If liveness is being attacked though such that many nodes appear to be crashing for example, quorum sizes could be decreased to accommodate more Byzantine faults. Flexible BFT therefore allows flexibility in fault thresholds, and also allows a shard to withstand more total faults than convention by increasing qr and/or qc beyond the traditional value of $q = \frac{2}{3}$. We leverage these properties in FlexiShard.

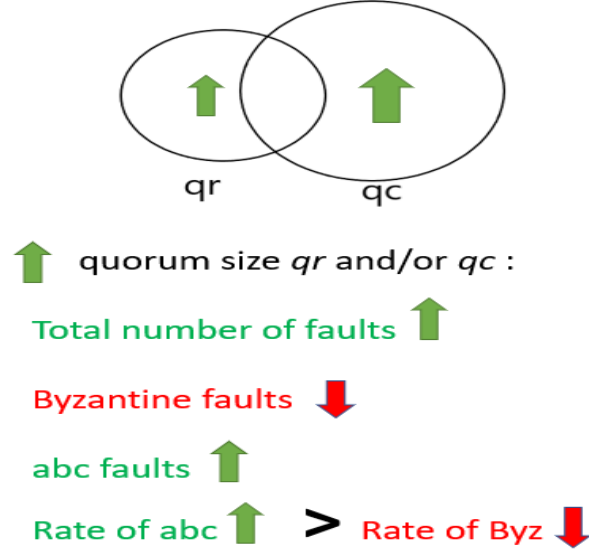


Figure 3.3 Effect of increasing quorum size(s) on Byzantine and/or abc faults in Flexible BFT.

3.3. Calculation of shard failure probability in FlexiShard

FlexiShard uses the cumulative hypergeometric distribution to calculate shard failure probability. However, the fault probability calculation is not as straightforward because there are two types of faults which need to be considered: a shard is faulty if either the number of Byzantine faults OR the number of abc faults have exceeded their respective allowable thresholds. The fault thresholds of Byzantine and abc faults in turn depend on the parameters qr and qc , and are calculated in terms of fractions using equations 3.6 and 3.7 respectively.

Let f be the maximum number of total faults that a shard in FlexiShard can withstand.

Let f_{byz} be the maximum number of Byzantine faults a shard in FlexiShard can withstand.

$$f_{byz} = \lfloor (1 - qc) \times M \rfloor \quad (3.8)$$

Let f_{abc} be the maximum number of abc faults a shard in FlexiShard can withstand. Then,

$$f_{abc} = f - f_{byz}$$

$$f_{abc} = \lfloor (qr + 2qc - 2) \times M \rfloor \quad (3.9)$$

Let Y and Z be the random variables denoting the number of abc and Byzantine faults respectively in a shard. The shard is non-faulty and will produce correct results if Y and Z are both within their allowable thresholds. As soon as one of them exceeds its threshold, Flexible BFT's safety requirement is violated and the shard is considered faulty. FlexiShard's shard failure probability can be calculated more conveniently by first calculating the cumulative probability that a shard is non-faulty, and then calculating the probability that the shard is faulty.

Let S' be a shard in FlexiShard that fails if either the number of abc faults or Byzantine faults exceed the value of 4 in that shard. We first find out all the possible possibilities that S' is safe and sum up their probabilities.

$$P(S' \text{ is faulty}) = P(f_{byz} > 4) \text{ OR } P(f_{abc} > 4).$$

$$P(S' \text{ is nonfaulty}) = P(f_{byz} \leq 4) \text{ AND } P(f_{abc} \leq 4).$$

$$P(S' \text{ is nonfaulty}) = P(f_{byz}=0 \text{ AND } f_{abc}=0) + P(f_{byz}=0 \text{ AND } f_{abc}=1) \dots + P(f_{byz}=0 \text{ AND } f_{abc}=4)$$

$$+ P(f_{byz}=1 \text{ AND } f_{abc}=0) + P(f_{byz}=1 \text{ AND } f_{abc}=1) \dots + P(f_{byz}=1 \text{ AND } f_{abc}=4)$$

•

•

•

$$+ P(f_{byz}=4 \text{ AND } f_{abc}=0) + P(f_{byz}=4 \text{ AND } f_{abc}=1) \dots + P(f_{byz}=4 \text{ AND } f_{abc}=4).$$

$$P(S' \text{ is faulty}) = 1 - P(S' \text{ is nonfaulty}).$$

Formally, let $P_{NonFaultyFlexi}$ represent the probability that a shard in FlexiShard is not faulty. Let $P_{FaultyFlexi}$ represent the shard failure probability of a shard in FlexiShard. Then,

$$\begin{aligned} P_{NonFaultyFlexi} &= P(Y \leq f_{abc} \text{ AND } Z \leq f_{byz}) \\ &= \frac{\sum_{f_{byz}=0}^{f_{byz}} \binom{F_B}{f_{byz}} \times \left(\sum_{f_{abc}=0}^{f_{abc}} \binom{F_A}{f_{abc}} \times \binom{N-F}{M-f_{byz}-f_{abc}} \right)}{\binom{N}{M}} \end{aligned} \quad (3.10)$$

$$P_{FaultyFlexi} = 1 - P_{NonFaultyFlexi} \quad (3.11)$$

In equation 3.10, F_A and F_B are the maximum allowable abc and Byzantine faults in the entire system respectively, and $F = F_A + F_B$ is the maximum allowable total faults in the entire system. When $qr = qc = 2/3$, Flexible BFT degenerates to classic BFT such that a shard can only support Byzantine faults. Similarly, when $qc = 1$, only abc faults are tolerated by a shard.

3.4. FlexiShard's use cases & Theoretical analysis

3.4.1. Introduction of 3 new parameters: qr , qc & $byz:abc$ split

Contemporary sharding systems are rigid (i.e., hard-coded) on the values of the fault parameters ($prop_{Total} = \frac{F}{N}$ and $prop_{Shard} = \frac{f}{M}$), shard size M , and shard failure probability threshold $P_{Threshold}$. The values of $prop_{Total}$ and $prop_{Shard}$ are determined by the network model and the intra-shard consensus protocol. When combined with the shard failure probability threshold $P_{Threshold}$, they determine shard size M . In synchronous systems, $prop_{Shard} \approx \frac{1}{2}$ while $prop_{Shard} \approx \frac{1}{3}$ in partial synchrony. To minimize shard failure, the value of $prop_{Total}$ needs to be much smaller than $prop_{Shard}$. Hence, systems tend to adapt conservative (low) values: $prop_{Total}$ is usually set to 1/3 in synchrony and 1/4 or 1/5 in partial synchrony. Recall that $prop_{Total}$ is the fraction of faulty nodes an entire system can withstand. This low number of faults may not be representative of real systems.

Since Flexible BFT is more fault tolerant than classic BFT protocols, the value of $prop_{Shard}$ can be much higher, such that shards can tolerate more total faults f . Consequently, a higher value can be used for $prop_{Total}$ as well. This implies that a system in general can withstand more total faults globally in FlexiShard. By varying the two new parameters qr and qc , FlexiShard can tolerate a variable number of faults both inside a shard ($f = f_{abc} + f_{byz}$) and in the whole system ($F = F_A + F_B$). Hence, the properties of flexibility, and increased fault tolerance resulting in throughput gains, can both be achieved in FlexiShard. Table 3.1 and Figure 3.9 demonstrates some possible values that can be used in FlexiShard.

	Synchronous BFT (e.g., RapidChain)	Partially Synchronous BFT (e.g., OmniLedger)	Partially Synchronous FlexiShard
Fault model	Byzantine	Byzantine	Byzantine + abc
Fault threshold inside shard, f	$f < \frac{M}{2}$	$f < \frac{M}{3}$	$\frac{M}{3} < f < \frac{2M}{3}$
Fault threshold in whole system, F	$F = \frac{N}{3}$	$(F = \frac{N}{5})$ OR $(F = \frac{N}{4})$	$\frac{N}{5} \leq F \leq \frac{N}{2}$
Shard size M for $P_{Threshold} = 10^{-10}$	≈ 280	≈ 700	≈ 350 for an instance where $qr = qc = 0.8$

Table 3.1 Differences in fault thresholds between contemporary systems and FlexiShard.

Figure 3.4 shows a semi-log plot for the failure probability of three different systems, each with total nodes $N=2000$. RapidChain obtains shard sizes of $M \approx 280$ when $P_{Threshold} = 10^{-10}$, while a particular instance of FlexiShard (with $qr = qc = 0.8$: $prop_{Shard} < 0.6$, $f_{byz}/M \leq 0.2$, $f_{abc}/M < 0.4$) yields shard size $M \approx 350$ with the same failure probability $P_{Threshold} = 10^{-10}$. When $F \approx N/3$, the probability of failure of the partially synchronous OmniLedger degenerates to around 0.5 (not shown in Figure. 3.4) which is considered very high. OmniLedger in its classic setting, which uses a conservative total fault threshold of $F \approx N/4$, still requires relatively large size of $M \approx 700$ to satisfy $P_{Threshold} = 10^{-10}$ in comparison to FlexiShard and RapidChain, although the latter two can tolerate more total faults in their entire system than OmniLedger. Recall that RapidChain uses the conservative model of synchrony to obtain small shard sizes while FlexiShard uses the more practical partial synchrony model to obtain comparable shard sizes to RapidChain. Depending on values of qr and qc used, some instances of FlexiShard obtain smaller shards than RapidChain's (e.g., when $qr = qc = 0.9$). FlexiShard hence demonstrates potential in leveraging the benefits of both synchrony and partial synchrony, and can be used in different settings to shard a blockchain.

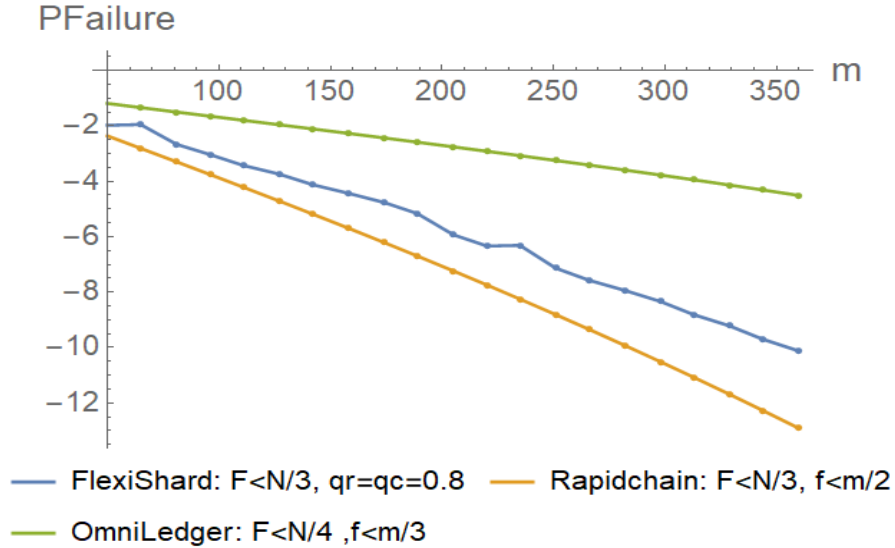


Figure 3.4 Effect of increasing shard size M on failure probability: FlexiShard in partial synchrony obtains comparable shard sizes to RapidChain, but smaller shards than OmniLedger.

It is important to note that the same choice of qr , qc and F can yield different instances (security profiles) in FlexiShard. Recall that $prop_{Total}$ should be small compared to $prop_{Shard}$. However, for the same value of $prop_{Total}$ and $prop_{Shard}$, there can be different proportions of Byzantine and abc faults in a system. The shard failure probability can be heavily influenced by the type of fault which is more dominant in the system. For example, suppose $F=500$ but the ratio of Byzantine to abc faults is 4:1. This means that there can be a maximum of 400 Byzantine faults and 100 abc faults present in the system. The values of qr and qc must then be picked such that the intra-shard consensus can withstand a higher number of Byzantine faults than abc faults. In this particular scenario, if very high values of qr and qc are used, the shard failure probability may be high because large values of qr and qc tend to sacrifice safety for liveness [17] and support more abc faults at the expense of Byzantine faults. In contrast, if the proportions were reversed (i.e., 400 abc faults and 100 Byzantine faults), high values of qr and qc would increase intra-shard fault tolerance to abc faults and hence presumably yield lower shard failure probability and shard sizes.

Therefore, alongside the parameters qr , qc , and F , the proportion of Byzantine to abc nodes is of paramount importance in a system using FlexiShard. We introduce a new parameter called *byz:abc split* to represent this fault proportion.

Recall that shard failure probability is heavily influenced by 5 parameters in contemporary systems. They are: 1) the fault model. 2) the network model. 3) the intra-shard fault threshold, f . 4) the global fault threshold, F . 5) the shard size, M . By applying Flexible BFT to sharding, and by modifying the fault model and the shard failure calculation (equations 3.10 and 3.11), three new parameters start heavily influencing the shard failure probability as well. They are:

1. qr : quorum size of the first round of voting in Flexible BFT.
2. qc : quorum size of the second round of voting in Flexible BFT.
3. *byz:abc split*: the proportion (or ratio) of Byzantine to abc faults present in the entire system.

Equation 3.12 is derived from equation 3.10 by substituting parameters f_{byz} , f_{abc} , F_A and F_B by these new parameters. In equation 3.12, parameter *byz:abc split* is represented by $byzProp = F_B / F = F_B / (F_A + F_B)$.

$$P_{NonFaultyFlexi} = \frac{\sum_{byz=0}^{\lfloor ((1-qc) \times M) \rfloor} \binom{\lfloor byzProp * F \rfloor}{byz}}{\binom{N}{M}} \times \left(\sum_{abc=0}^{\lfloor ((qr+2qc-2) \times M) \rfloor} \binom{F - \lfloor byzProp * F \rfloor}{abc} \times \binom{N-F}{M-byz-abc} \right) \quad (3.12)$$

Instead of being fixed though, the three new parameters qr , qc and *byz:abc split* (or $byzProp$) can be changed to obtain different security settings. Therefore, by keeping some parameters constant while varying others, it can be deduced that shard failure probability can greatly be influenced. FlexiShard can hence be used in many different ways to shard a system. That is, there are many more different combinations of parameters to shard a system. This is not the case in contemporary systems whereby quorum size is fixed and optimal, thus limiting the number of ways in which a blockchain can be sharded. Two such use cases (ways of sharding) are discussed next, including how to choose good parameter values to obtain low shard failure probabilities.

3.4.2. Use Case 1: Determining good quorum sizes, and an optimal shard size, for a fixed number of faults (F), using different fault proportions (byz:abc split).

a) Choosing good values of qr and qc .

Consider a common scenario where a system administrator has a good idea on the maximum number of faults the latter wants to allow in the entire system, i.e., F is known and is a fixed value. Assume also that the administrator has a good idea on the proportion or ratio of Byzantine to abc faults that he/she wants to allow in the system. Hence *byz:abc split* is also known.

Aim: The goal of the system administrator then becomes to determine potential quorum sizes that can be used in Flexible BFT inside shards, to obtain low failure probability. Let $P_{Threshold}$ be the maximum shard failure threshold that is allowed. $P_{Threshold}$ is chosen beforehand as well.

We propose scheme 3.1 (formally described next) to obtain acceptable quorum values to satisfy this goal. In scheme 3.1, the system administrator chooses the maximum shard size M_{max} he/she desires to satisfy $P_{Threshold}$. For fixed values of M , F , *byz:abc split*, and qr , the administrator determines the shard failure probability P_{Faulty} for different values of qc . Any pair of quorum sizes qr and qc resulting in $P_{Faulty} \leq P_{Threshold}$ is acceptable. If $P_{Faulty} > P_{Threshold}$, this signifies that the desired probability threshold cannot be met for any shard size $M \leq M_{max}$, when the chosen values of qr and qc are used. Then, the system administrator can either increase the value of M_{max} , or increase the values of qr and/or qc to obtain the desired shard failure threshold $P_{Threshold}$.

Scheme 1: Determine valid qr and qc for fixed F and *byz:abc split*

procedure QUORUMSIZE:

1. Choose F and *byz:abc split*.
 2. Choose $P_{Threshold}$. //maximum shard failure probability threshold
 3. Choose maximum shard size desired: M_{max} .
 4. Choose qr .
 5. Using equations (3.12 & 3.11):
 6. Plot graph of $F(qc)$. //shard failure probability vs qc .
 7. Range of qc : qr to 1. // $qr \leq qc \leq 1$.
 8. Check if $F(qc)$ intersects $P_{Threshold}$:
 9. if False:
 10. if $qr = 1$: stop. //shards with probability $\leq P_{Threshold}$ cannot be formed for parameters chosen in steps 1 - 4.
 11. else: go to 4 and increase qr . //when $qr < 1$
 12. else://true
 13. Valid qc values found when $F(qc) \leq P_{Threshold}$.
 14. Choose qc from valid region.
 15. return qr, qc .
-

Figures 3.5 and 3.6 show the effect of varying qc on shard failure for a fixed value of qr and $M_{max} = 300$. The different graph colors represent shard failure probability at different fault proportion of total faults (i.e., *byz:abc split*).

$prop_{Total} = F/N$ is set to $1/3$ (i.e., $F = N/3$) and 0.45 (i.e., $F = 9N/20$) respectively, and qr is set to $3/4$ and $4/5$ respectively in Figures 3.5 and 3.6. Let $P_{Threshold} = 10^{-5}$. N is set to 2000 nodes.

It can be deduced from Figure 3.5 that the region $0.79 \leq qc \leq 0.825$ yields shards with failure probability $\leq P_{Threshold}$ when *byz:abc split* is 3:7 (i.e., when the fault proportion is 30% Byzantine faults and 70% abc faults). Similarly, Figure 3.6 shows that desirable shards satisfying $P_{Threshold}$ can only be obtained when *byz:abc split* is 1:9 and $0.825 \leq qc \leq 0.93$. Figure 3.6 also shows that FlexiShard works well even when a higher number of total faults are present. $prop_{Total} = 0.45$ demonstrates that some FlexiShard instances represented in Figure 3.6 can tolerate close to half of their total nodes being faulty – largely exceeding conventional assumptions seen in OmniLedger ($prop_{Total} = 0.25$) or RapidChain ($prop_{Total} = 0.333$). $prop_{Total}$ (and therefore F) can be further increased in FlexiShard for some choices of qr and *byz:abc split*. This however decreases the range of possible values of qc that can satisfy $P_{Threshold}$.

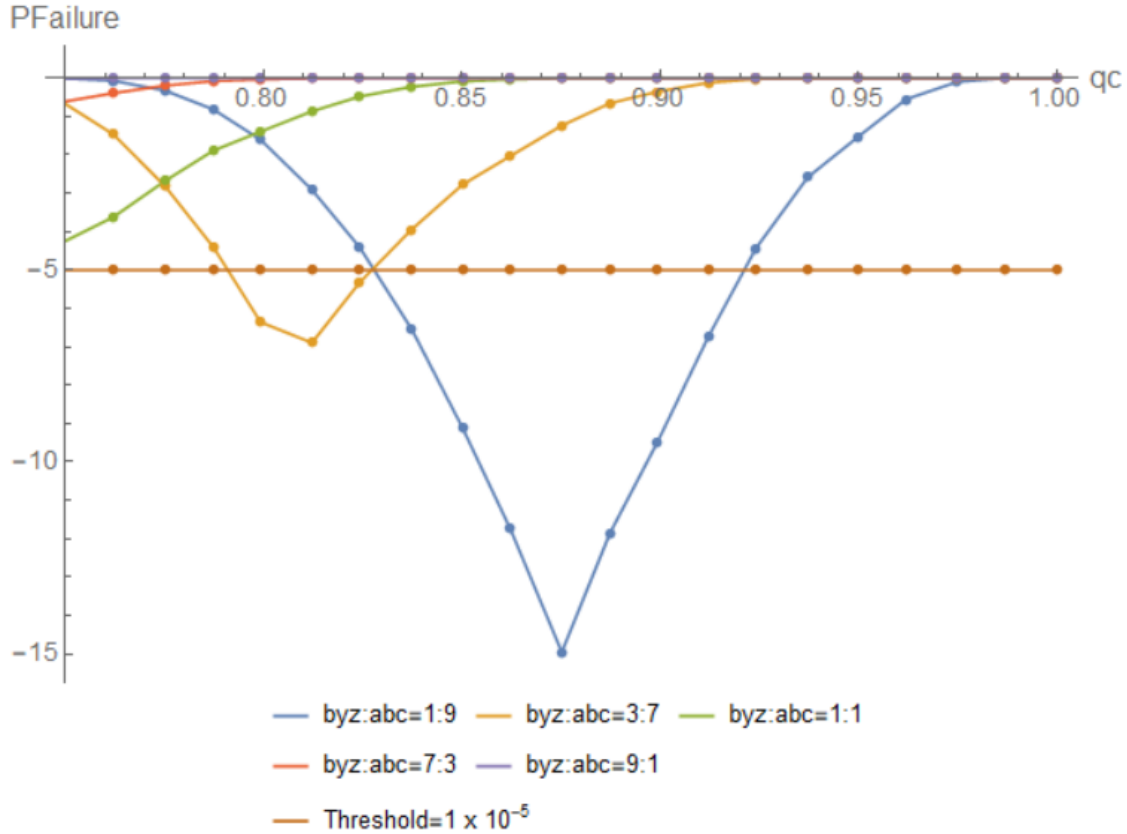


Figure 3.5 Effect of qc on shard failure probability at $F = N/3$ and $qr = 3/4$ for different *byz:abc splits*.

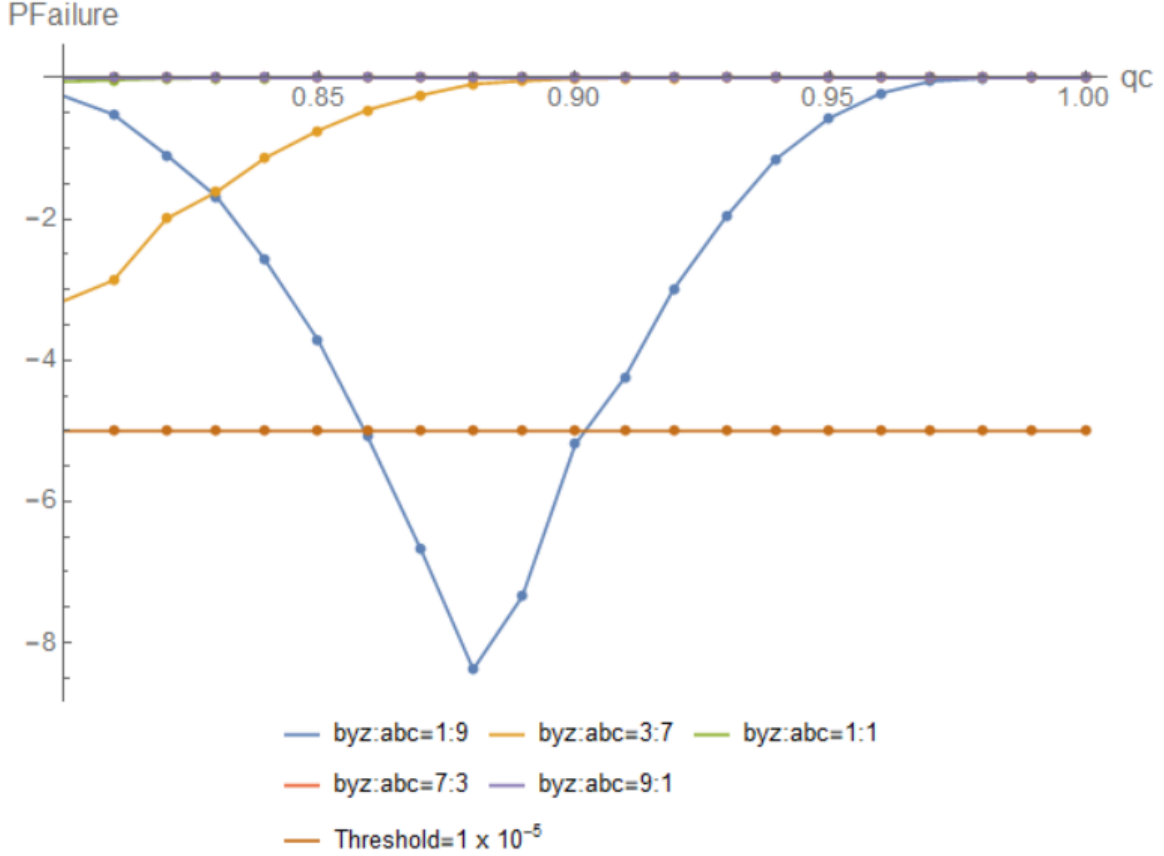


Figure 3.6 Effect of qc on shard failure probability at $F = 0.45N$ and $qr = 4/5$ for different $byz:abc$ splits.

The results obtained in Figures. 3.5 and 3.6 adhere to the discussions on random sampling and Flexible BFT. When qc increases, Byzantine fault tolerance in shards decreases while abc fault tolerance increases. Therefore, when the ratio of abc faults to Byzantine faults (i.e., $byz:abc$ split) is increased globally in the system, shards with increasingly lower failure probabilities are obtained. For example, when $qc = 0.85$ in Figure 3.5, shard failure probability is close to 1 when the proportion of Byzantine and abc faults are equal (1:1) (green graph). When $byz:abc$ split increases to 3:7 in favour of abc faults, shard failure probability goes down to around 10^{-3} (yellow graph) ; and decreases even further to around 10^{-9} when abc faults largely outnumber Byzantine faults in the system by a factor of 9 ($byz:abc = 1:9$) (blue graph).

b) Determining optimal shard size after choice of qr and qc .

In the previous, quorum sizes qr and qc have been determined using a fixed shard size, $M = 300$. This shard size may however not be optimal, i.e., there may exist an even smaller shard size that yields fault probability $P_{Threshold}$. Therefore, a system administrator can set all the parameters determined previously (qr , qc , F and $byz:abc$ split), and consequently deduce the effect of shard size M on shard failure probability by using equations 3.12 and 3.11, as shown in Figure 3.7. The aim is to determine the smallest shard size $M_{Optimal}$ that satisfies $P_{Threshold}$.

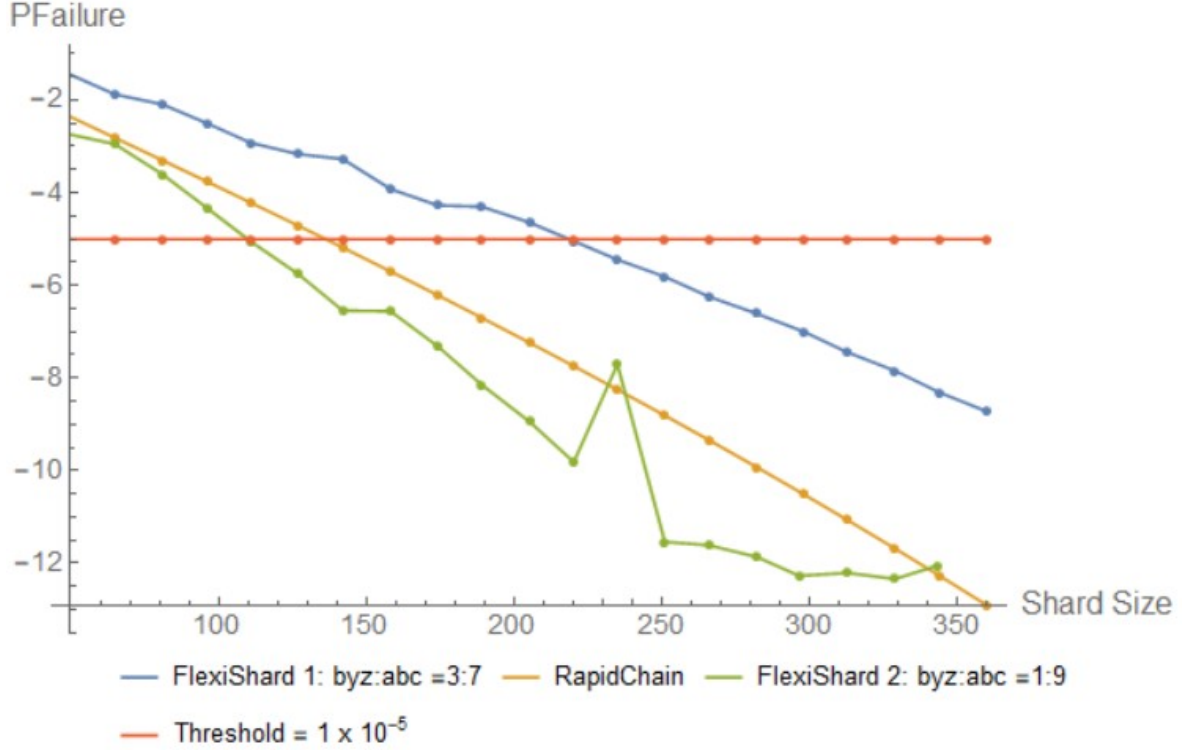


Figure 3.7 First intersection of $P_{Threshold}$ with a graph gives the corresponding optimal shard size.

Figure 3.7 shows the effect of increasing shard sizes on failure probability for RapidChain and two instances of FlexiShard. All three instances have $prop_{Total}$ set to $1/3$. FlexiShard 1 sets $qr = 3/4$ and $qc = 0.81$; FlexiShard 2 sets $qr = 3/4$ and $qc = 0.88$. The qr and qc values of both FlexiShard instances have been derived from scheme 1 in Figure 3.5, where $M = 300$ was used for calculations. However, Figure 3.7 clearly shows that $M = 300$ is not the optimal value for both instances. The optimal (smallest) shard size for failure probability threshold $P_{Threshold}$ is obtained at the first point of intersection between each of the 3 instances and $P_{Threshold}$. It can be deduced from Figure 3.7 that when $P_{Threshold} = 10^{-5}$, FlexiShard 1 has an optimal shard size of ≈ 210 , while FlexiShard 2 has an optimal value of ≈ 115 . For RapidChain, the optimal shard size is ≈ 140 . These results further demonstrate that FlexiShard can not only obtain shard sizes comparable to or even better than RapidChain's in certain cases, but can perform better in environments where fault thresholds start exceeding conventional bounds of $F=N/3$ or $F=N/4$. Recall that RapidChain uses the synchronous network model which is nonresponsive, while FlexiShard uses the more practical partial synchrony model.

Based on the previous discussions, the scheme for determining optimal shard size is formally described next.

Scheme 2: Determine optimal shard size

procedure OPTIMALSHARD ($F, P_{Threshold}, \text{byz:abc split}, qr, qc, M_{max}$):

1. Choose **minimum shard size** desired: M_{min} .
 2. Using **equations (3.12 & 3.11)**:
 3. **Plot** graph of $F(M)$. // shard failure probability vs shard size M .
 4. **Range of M :** M_{min} to M_{max} .
 5. $M_{optimal}$ = M-axis value at **first intersection** of $F(M)$ and $P_{Threshold}$
(i.e., when $F(M) = P_{Threshold}$).
 6. return $M_{optimal}$
-

3.4.3. Use Case 2: Determining range of acceptable fault proportions for predefined quorum sizes, total faults F and shard failure threshold $P_{Threshold}$.

In the previous use case, the *byz:abc split* (fault proportion of the system) was pre-determined and hardcoded by a system administrator to determine quorum sizes. However, there can be situations where the opposite is required. The system administrator may want to shard a system using fixed quorum sizes qr and qc . The reasons being that while higher quorum sizes do tend to result in more fault-tolerance and smaller shard sizes, more messages are required in a voting round when a quorum is large.

Additionally, different fault proportions may be required based on the failure characteristics of the system which can change dynamically at the start of every epoch. For example, an abc node might decide to attack liveness as well in some epoch(s) and thus transform into a Byzantine node, or vice-versa. In such scenarios, F remains constant but the proportion of Byzantine to abc nodes changes in the system. If abc faults disproportionately outnumber Byzantine faults such that *byz:abc split* goes outside some acceptable range and the value of f_{abc} in equation 3.9 is exceeded, but quorum sizes are too small, a shard can fail since small quorums tend to favor Byzantine faults more than abc faults. The reverse can happen as well: large quorum sizes can fail a shard if abc nodes suddenly transform into Byzantine nodes such that Byzantine nodes exceed the value of f_{byz} in equation 3.8. Therefore, FlexiShard can also be used to determine the range of *byz:abc splits* that can be supported by a system for some fixed quorum sizes qr and qc , and a fixed number of total faults F .

The results in Figure 3.8 represent instances of FlexiShard where $qr = qc = 0.8$, $F = N/3$ and $P_{Threshold} = 10^{-5}$ for shard sizes M set to 100, 200, and 300. It can be deduced from Figure 3.8 that it is not possible to form shards of size 100 when $P_{Threshold} \leq 10^{-5}$. Shards of size 200 satisfying $P_{Threshold}$ can be formed when the proportion of Byzantine faulty nodes in the whole system is between 0.2 and approximately 0.3 (hence abc faults proportion is between approximately 0.7 and 0.8). Shards of size 300 satisfying failure probability $\leq P_{Threshold}$ can be formed when the proportion of Byzantine faulty nodes is between 0.1 and 0.35. The minimum of each graph gives the optimal *byz:abc split* for that particular shard size. This analysis of obtaining a range of splits can be useful to a system administrator if it is estimated that the proportion of Byzantine faulty nodes (or abc faulty nodes) in the system never exceeds some value. Scheme 3 is formally described next.

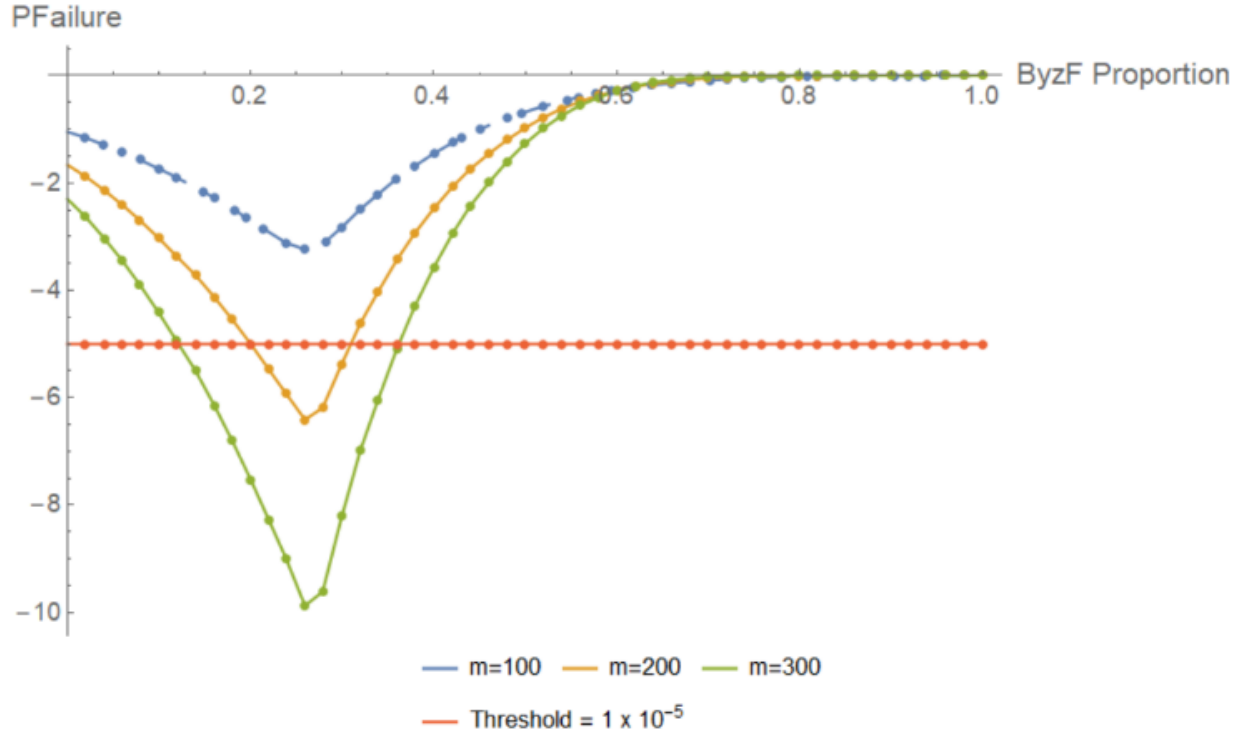


Figure 3.8 Effect of *byz:abc split* on shard failure probability for different shard sizes. For this instance, $qr=qc=0.8$ and $F=N/3$.

Scheme 3: Determine range of *byz:abc split* for fixed F , qr and qc

procedure SPLIT:

1. Choose F .
 2. Choose qr and qc .
 3. Choose maximum shard size desired: M_{max} .
 4. Using equations (3.12 & 3.11):
 5. Plot graph of $F(\text{byzProportion})$. // shard failure probability vs proportion of Byzantine nodes.
 6. Range of *byzProportion*: 0 to 1.
 7. Check if $F(\text{byzProportion})$ intersects $P_{Threshold}$:
 8. if False:
 9. stop. //shards with probability $\leq P_{Threshold}$ cannot be formed for chosen parameters in steps 1 - 3.
 10. else: //true
 11. // region below threshold $P_{Threshold}$ gives valid split values
 12. Valid *byz:abc split* proportions found when $F(\text{byzProportion}) \leq P_{Threshold}$.
 13. Choose *byzProportion* from valid region.
 14. $abcProportion = 1 - \text{byzProportion}$.
 15. return *byzProportion*, *abcProportion*.
-

Recall that the maximum desired shard size M_{max} is chosen for calculations in scheme 3. As in use case 1, this shard size may not be the optimal (smallest) size which yields probability $P_{Threshold}$. F , qr and qc are already fixed. Therefore, after choosing a valid *byz:abc split* from scheme 3, scheme 2 can be used to determine an optimal shard size for the chosen parameters. It can be observed from Figure 3.8 that when the *byz:abc split* = 0.3 : 0.7 (x-axis), $M = 100$ is too small for satisfying $P_{Threshold} = 10^{-5}$. There is no intersection between the graph of $M = 100$ and $P_{Threshold} = 10^{-5}$. Similarly, $M = 200$ or $M = 300$ are too large. The optimal shard size for $P_{Threshold} = 10^{-5}$ lies between 100 and 200 ($200 \leq M_{Optimal} \leq 300$) and can be determined using scheme 2.

Other use cases can be obtained by keeping some parameters constant while varying others in equations 3.11 and 3.12. It can be observed that each use case results in a different FlexiShard scheme to determine the sharding parameters. FlexiShard offers many more ways (schemes) to shard a blockchain compared to contemporary systems, alongside having the potential to increase throughput and reduce intra-shard transaction latency, subject to choice of proper parameters.

Shard size $M = 300$ in partial synchrony

► Contemporary	► FlexiShard 1	► FlexiShard 2	► FlexiShard 3
► Byzantine only	► Byz+ abc	► Byz+ abc	► Byz+ abc
► $qr = 2/3$	► $qr = 2/3$	► $qr = 2/3$	► $qr = 2/3$
► $qc = 2/3$	► $qc = 0.75$	► $qc = 0.9$	► $qc = 1$
			
► Total F ≤ 99 (33%)	► Total F ≤ 125 (41%)	► Total F ≤ 170 (56%)	► Total F ≤ 199 (66%)
► ByzF ≤ 99 (33%)	► ByzF ≤ 75 (25%)	► ByzF ≤ 30 (10%)	► ByzF = 0 (0%)
► abcF = 0 (0%)	► abcF = 50 (16%)	► abcF = 140 (46%)	► abcF = 199 (66%)
► Practical	► Practical	► Practical	► Very Optimistic

Figure 3.9 Impact of increasing quorum sizes on fault thresholds in FlexiShard.

3.5. Summary

In this Chapter, we first discussed an important result in section 3.1: the more fault tolerant a shard is, the smaller that shard needs to be to obtain a good (low) shard failure probability without compromising fault tolerance. Our aim was to increase the throughput and improve the performance of a blockchain system. Both can be achieved by simply creating smaller shards which should however be resilient.

Following the result from section 3.1, we therefore applied a new state-of-the-art and more fault tolerant BFT algorithm called Flexible BFT to blockchain and its sharding. By doing so, the

intra-shard fault threshold ($prop_{Shard}$) and global fault threshold ($prop_{Total}$) could both be increased to withstand more total faults both inside shards and in an entire system. This however required the slight relaxation of the fault model to consider abc faults (which are still powerful though) as well and sacrifice some Byzantine fault tolerance. By allowing a system to withstand more total faults, FlexiShard made it possible to use more practical values (i.e., higher fault thresholds and smaller shard sizes) instead of conservative contemporary values, and can hence be better suited to real-life blockchain systems. The property of flexible quorums in Flexible BFT also made it possible to use a range of fault thresholds, which gave FlexiShard the flexibility to adapt to constantly changing security environments. This again is a more practical feature of FlexiShard compared to contemporary systems which use fixed and rigid fault parameters.

The hybrid fault model meant that the traditional method of shard failure calculation had to be altered to consider abc faults as well. This new way of shard failure calculation (equations 3.10 and 3.12) led to the introduction of 3 new parameters (i.e., qr , qc , and $byz:abc\ split$) which could take on a wider range of values due to the property of flexible quorums. Hence, new combinations of different sharding-related parameters could be introduced by keeping some parameters constant while varying others. These combinations lead to different use cases (i.e., different ways of sharding a blockchain). We discussed two such main use cases and described formal protocols for these use cases. There are still many more sub-schemes that are possible, with the ultimate goal being to find optimal shard sizes and/or quorum sizes to obtain low shard failure probabilities for different scenarios. This concludes the Chapter.

Chapter 4. SECURSHARD: A MODEL FOR HIERARCHICAL FAULT DETECTION IN BLOCKCHAIN SHARDING

4.1. Motivation

In Chapter 3, a paramount assumption made both in systems implementing FlexiShard, and other contemporary sharded systems, is that shards are ‘perfect’ and always produce the correct results. Violation of this ‘perfect shard’ constraint due to a faulty shard appending a faulty (invalid) block to the blockchain usually leads to the entire sharded system being considered to have failed. This is because victims involved in such incorrect transactions (e.g., a double-spend attack) tend to incur huge losses and/or irreversible damages. In turn, the whole ecosystem can suffer from asset devaluation and loss of trust in the community. In this Chapter, we propose *SecurShard*, a hierarchical blockchain model to detect potentially faulty blocks (and faulty shards) with high probability during transaction processing instead of after transaction processing, thus preventing such faulty block(s) to be appended to the blockchain. The idea is as follows:

Let S represent a large shard of size M that would have originally been formed in a contemporary system. In *SecurShard*, S is split into k smaller shards which together form a group. A set of transactions (block) is then assigned to all of these k smaller shards in a group. In other words, a one-to- k mapping of transaction block to shards is applied, instead of a one-to-one mapping of a block to a shard in a contemporary system. 100% consensus is required among all the k shards in a group to validate a transaction. What is achieved in return are: 1) relaxation of the ‘perfect’ shard constraint (which is more practical and achievable), 2) smaller shard sizes giving better performance, 3) collective fault probability of k shards in a group can be made much smaller and hence more fault tolerant, with the proper selection of various parameters, as compared to individual fault probability of a shard in contemporary systems, and 4) capability of faulty shard/block detection before a block is appended to blockchain. A theoretical analysis of *SecurShard* is presented in section 4.4 in terms of its fault tolerance and performance, which shows the clear benefits of *SecurShard* as compared to contemporary systems, subject to the proper selection of various parameters.

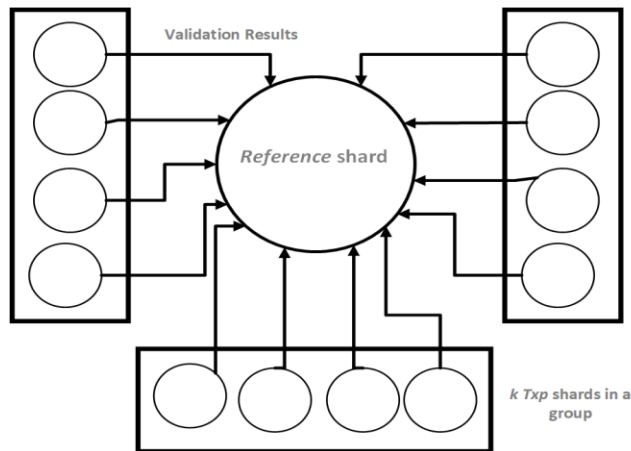


Figure 4.1 Three shards S of size M split each into 4 smaller Txp shards of size m_{Tx} (1:4 mapping).

4.2. Terminologies and assumptions

4.2.1. Assumptions

We assume the partially synchronous network model and further assume that communication channels between nodes with a public key infrastructure are established before sharding. An adversary is considered to be Byzantine only. Hence, shards described in this Chapter can tolerate less than $1/3$ faulty nodes (i.e., $f < M/3$). The maximum number of Byzantine faults F that can be present in the system is set to $N/4$. This threshold of $1/4$ is used to adhere to the parameters that other partially synchronous sharded blockchains use [12] [13]. We also adopt the UTXO model [1] in this Chapter to be consistent with other contemporary sharded blockchains.

These parameters and the fault model can be changed to obtain different levels of fault-tolerance and performance. For example, another system implementing SecurShard can adopt a hybrid fault model of Byzantine and alive-but-corrupt (abc) faults [35] [17] or Byzantine and crash faults [44]. Others could use a synchronous BFT protocol instead such that shards can tolerate less than $1/2$ nodes being Byzantine-faulty. The core protocol of SecurShard remains unchanged though as long as a variant of BFT is used for consensus.

In contemporary systems, some randomness is first generated to establish node identities, shard the network, propose new blocks and for other functionalities (refer to section 2.5). In RapidChain [10] (one of the prominent contemporary sharded systems), a randomly-elected small *initial committee* subsequently uses this randomness to shard the network into transaction-processing shards (i.e., *Txp* shards). Note that all shards in contemporary systems usually have the same role (i.e., being *Txp* shards only) and are considered ‘equal’ in hierarchy. Many protocols to generate randomness and to elect the initial committee already exist and are beyond the scope of this thesis. We assume that some protocols already exist to generate randomness, establish identities, and to elect a small initial committee for shard formation (e.g., [10]).

4.2.2. Public Key Infrastructure (PKI)

Assuming that nodes participating in an epoch of SecurShard have already established identities (using PoW for example) and a Public Key Infrastructure (PKI) exists, each node is assigned a unique identity and a unique cryptographic pair of keys (public key P_k , secret key S_k) related to that identity during the node assignment phase. A node signs the hash of a consensus message using its secret key S_k . A dictionary of node identity to key pair mapping is maintained and readily available in the system such that any node can query the public key P_k of any other node using the latter’s unique identity information.

4.2.3. Transaction Processing and Intra-shard Consensus in SecurShard

In contemporary systems, shards formed are usually equal in hierarchy and have the same role: to process transactions only. In SecurShard, we classify shards into two roles: reference shard (Ref shard) (acts as a central authority) and transaction-processing shards (*Txp* shards). *Txp* shards validate transactions inside a block while a reference shard usually validates the results of the *Txp* shards. The role of the Ref shard is discussed in section 4.3. We elaborate on the process of validating blocks, transactions and transaction inputs by the *Txp* shards next.

a) Txp Shard Leader and block proposal

Since some variant of BFT is used in SecurShard, a node is elected as the shard leader every epoch in accordance to the BFT protocol. The shard leader can be elected using various leader-election algorithms such as ‘round-robin’ among others [45]. This is a design choice that is left to a system administrator implementing SecurShard. Every epoch, a shard leader proposes a block. More formally, a leader chooses a set of transactions from its mempool (memory pool) and adds it to a new block along with the required metadata. The mempool is a data structure maintained by a node that contains a list of pending valid transactions submitted directly by a user or relayed by other peer nodes. The leader then broadcasts this block to every node in its shard. Note that all messages are signed by a sending node using its secret key S_k . For efficiency, instead of signing a whole message, each sender signs the hash of a message which is much smaller than the actual message. Let this be denoted by $H(<Message>)_{S_k}$. Signing the hash also guarantees that potential tampering of a message can be detected.

b) Block and transaction verification by Txp shard nodes

Upon receiving a block from the shard leader, every other node (also known as a validator) in the shard verifies the authenticity of the block. That is, the block received must have the current ‘epochId’ and must have been signed by the current shard leader (easily verified using shard leader’s public key P_k). The hash of the message received should also match the hash included in the message’s metadata, $H(<Message>)$ to ensure the message has not been tampered with.

Shard nodes then verify each transaction included in the block. Since the UTXO model [1] is used in this Chapter, transactions consist of inputs and outputs. More precisely, one or more outputs of previous transactions will be specified as the input(s) of an included block transaction as demonstrated in Figure 4.2. For a transaction tx handled by some shard S , if all its referenced outputs are also maintained in S , shard members can verify that transaction by themselves. tx is classified as an intra-shard transaction and consensus can be reached among shard members without communicating with other shards. However, most transactions in a sharded blockchain system are usually cross-sharded (cross-shard transactions). Multiple cross-shard transaction verification mechanisms already exist in the literature and is not the scope of this thesis. OmniLedger uses a user-based ‘proof-of-acceptance’ mechanism while RapidChain uses its proprietary ‘bridge transaction mechanism’ for example. These are design choices left to an administrator.

Shard nodes verify that each transaction input of a transaction has not been spent previously to prevent double-spends [42]. The traditional way of checking if a transaction input has been spent is to look at every single transaction since the creation of that input. In the worst case, the whole blockchain needs to be scanned. Scanning the whole blockchain for every transaction input is however very resource-intensive computationally and is not scalable. Nodes therefore scan the blockchain once and maintain a data structure called the UTXO Pool which contains all unspent transaction outputs since the beginning of the blockchain. If a blockchain is valid, every node should have the same UTXO pool. To check if a transaction input is spent, a node simply looks up for the input in its UTXO pool as detailed in Figure 4.3. If a transaction input is present in the UTXO pool, it is not spent and is therefore a valid input.

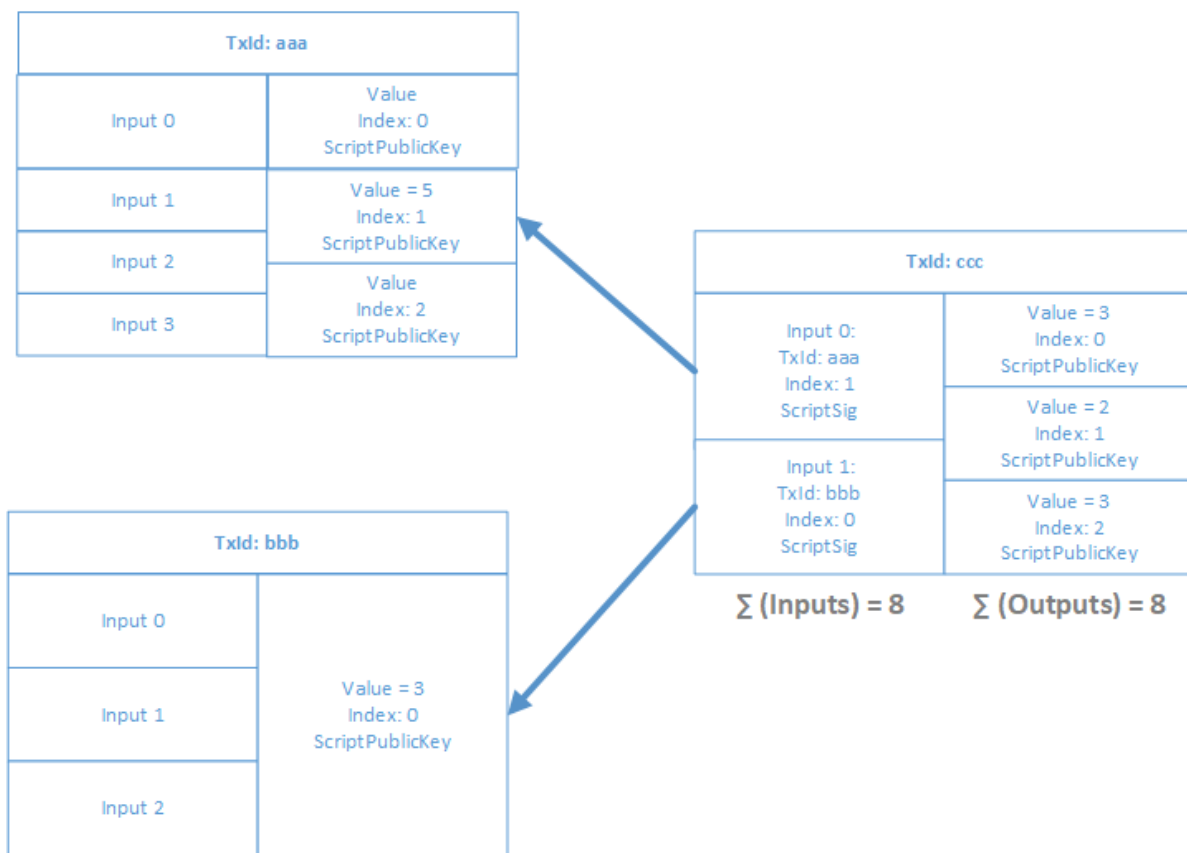
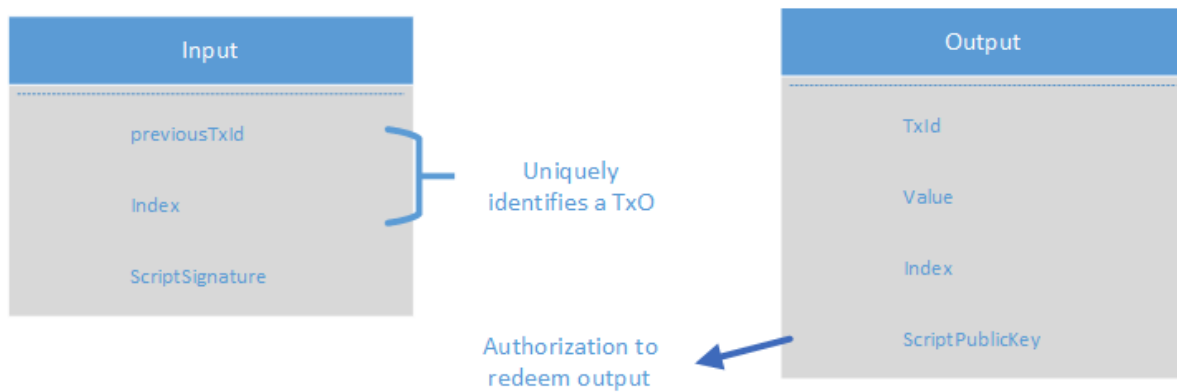


Figure 4.2 Data structure of an input, output and a transaction respectively. A transaction comprises of multiple inputs and outputs.

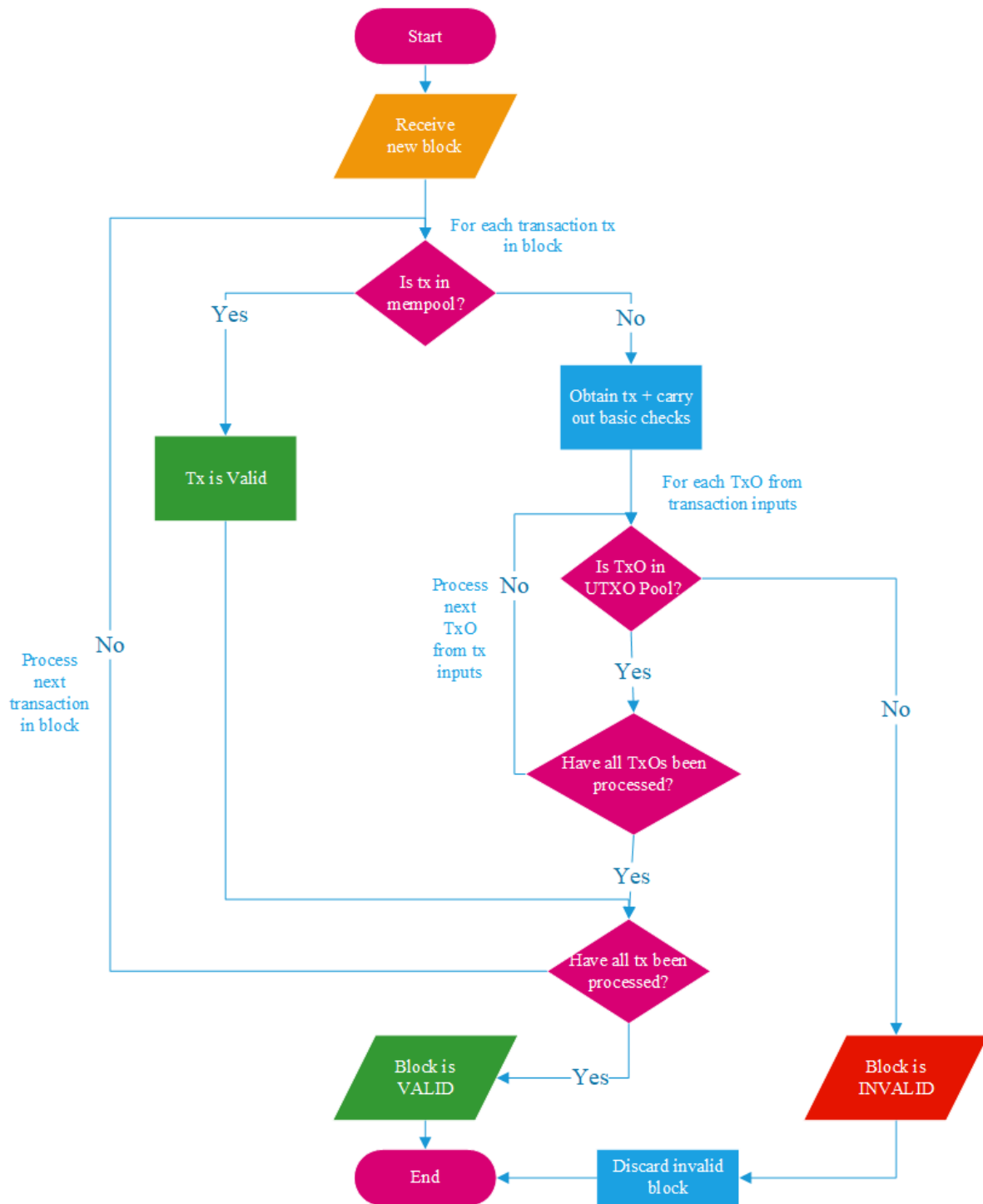


Figure 4.3 Transaction validation flowchart using the UTXO model.

Let U be the deterministic function that a node applies to a transaction input, represented by $input$. U returns 1 if that input is unspent (i.e., present in UTXO pool), otherwise U returns 0.

$$U(input) = \begin{cases} 1, & \text{input is unspent (i.e., valid)} \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

If all inputs to a transaction are unspent and the combined value of all the inputs equals or exceeds the combined value of the outputs, that transaction is deemed as valid as demonstrated by equation 4.2. All transactions in a block need to be valid for that block to be considered valid. The presence of even one invalid transaction should invalidate the whole block (equation 4.3).

Let Tx represent a transaction having a inputs and b outputs where $a \geq 1$ and $b \geq 1$.

Let $ValidateTx$ be the deterministic function that a node applies to a transaction Tx . $ValidateTx$ returns 1 if that transaction is valid, otherwise $Validate$ returns 0.

$$\begin{aligned} & ValidateTx(Tx) \\ &= \begin{cases} 1, & \left(\left(\sum_{i=1}^a input_i.value \geq \sum_{j=1}^b output_j.value \right) \wedge \sum_{i=1}^a U(input_i) = a \right) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \quad (4.2)$$

Let $ValidateBlock$ be the deterministic function applied by a node to validate a block B having d transactions. $ValidateBlock$ returns 1 if that block is valid (all transactions in the block need to be valid), otherwise $ValidateBlock$ returns 0.

$$ValidateBlock(B) = \begin{cases} 1, & \sum_{i=1}^d ValidateTx(Tx_i) = d \\ 0, & \text{otherwise} \end{cases} \quad (4.3)$$

If a block is valid following the previous discussion, a shard node signs only the Merkle root of the block with its secret key S_k instead of the whole block for efficiency. The node then sends the signed Merkle root to the shard leader. If PBFT is used (in partial synchrony), the leader waits for $2f + 1$ identical Merkle roots having distinct signatures, i.e., $2f + 1$ nodes inside a shard need to agree on the same Merkle root for consensus to have been reached and a block to be considered as valid by the shard. f is the maximum allowable faults inside a shard. After validation of a block by a shard, that block is not immediately appended to the blockchain by the shard leader unlike in contemporary systems. The consensus result is further validated using SecurShard's protocols to detect potentially faulty blocks and/or shards. We next elaborate on these protocols.

4.3. SecurShard and its protocols

Following from section 4.2.1, assuming that some protocols are already in place in a system to generate randomness, establish identities and elect a small initial committee for shard formation, an initial committee in a contemporary sharded system usually shards the whole network to form T transaction-processing shards (Txp shards) only. However, in SecurShard, an initial committee creates a single big shard called the reference committee or reference shard (Ref shard) of size m_{Ref} . The Ref shard acts as central authority and in turn proceeds to shard the remaining $N_{Tx} = (N - m_{Ref})$ nodes to create multiple distinct Txp shards of size m_{Tx} . The Ref shard then groups Txp shards together based on user (e.g., a system administrator) requirements to form T groups of k Txp shards each. T represents the throughput of the system and k represents the mapping factor of a group (number of smaller Txp shards in a group). These requirements and the choice of parameters (m_{Ref} , T and k) are decisions that a user must make beforehand and are discussed in the following sections.

The Ref shard subsequently assigns transaction sets (blocks) to each group. Since a group comprises of k Txp shards, the same block is assigned to k distinct shards in SecurShard – essentially creating a 1: all mapping of block to shards in a group. Each Txp shard individually processes the block (i.e., validates transactions) it was assigned and sends its result to the Ref shard. Ref shard finally compares the results of all the k Txp shards belonging to the same group and validates a block with finality (append to the blockchain) if and only if the result from all shards in that group match. Else, measures are taken such as ‘flagging’ a faulty group, re-sharding a group or re-sharding the entire network.

Recall that in contemporary sharded systems, each block is usually assigned to one shard only (1:1 mapping). Therefore, if a shard is faulty, an invalid block can be appended to the blockchain. There are no means to detect whether a shard (and therefore a block) could be faulty until after an invalid block has been appended and an audit has been performed, like in Chainspace [40] for example. Informally, an invalid block in a contemporary system can be detected only AFTER transaction processing.

In *SecurShard*, an ‘originally big’ Txp shard S of size M that would have been formed in a contemporary system is broken down into k equal-sized smaller Txp shards of size m_{Tx} each. We are going to demonstrate in section 4.4 that the introduction of the hierarchy in *SecurShard* and the one-to-all mapping of a block to k Txp shards can result in better fault tolerance without compromising performance. This approach also facilitates the detection of potentially faulty blocks (and therefore faulty Txp groups and/or shards) with high probability before a block is irreversibly validated and appended to the blockchain.

We also note that when quadratic BFT algorithms are used, the message–exchange complexity per consensus decision inside a Txp shard decreases by a factor of k^2 during intra-shard consensus as the mapping factor k increases. For linear BFT algorithms, message-complexity of intra-shard consensus inside a Txp shard decreases by a factor of k . The fault detection protocol also relaxes the requirement that all shards need to be ‘perfect’. Number of ‘perfect’ Txp shards required for the system to maintain safety [26] decreases by a factor of k . The detailed analysis is presented in section 4.4.

The following listed steps summarise the main protocols in SecurShard:

1. Formation of reference shard (Ref shard) and transaction-processing shards (Txp shards).
2. Randomized assignment of transaction blocks to Txp shards for validation.
3. Final validation of results by Ref shard.

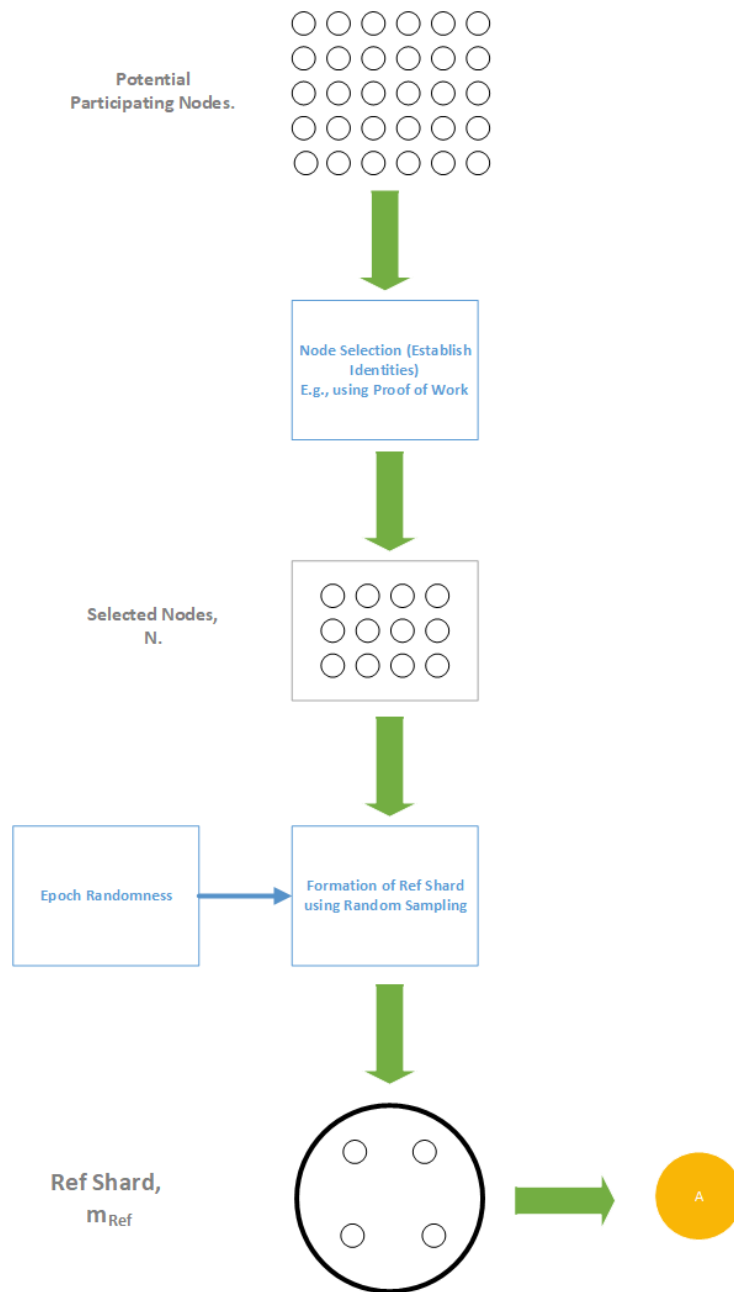


Figure 4.4 Formation of the Ref shard.

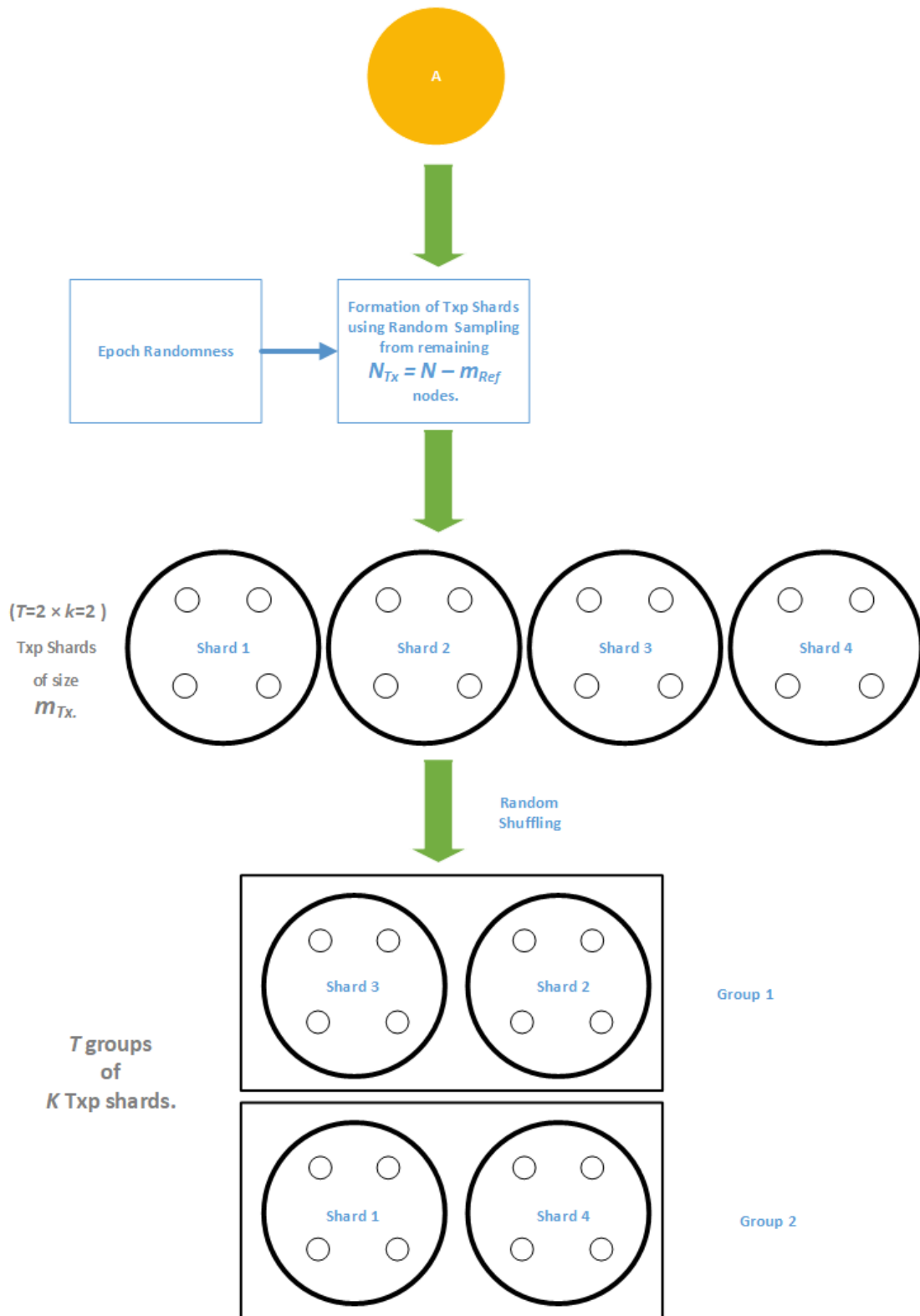


Figure 4.5 Formation of $(T \times k)$ Txp shards by the Ref shard.

4.3.1. Formation of Ref shard and Txp shards

The initial committee forms the Ref shard using the random sampling method. Control is then given to the Ref shard to drive the protocol forward. The Ref shard subsequently uses the random sampling method again to form multiple Txp shards from the remaining N_{Tx} nodes.

Let T be the throughput of a sharded system, i.e., the number of blocks that are validated per epoch (one round of processing). In contemporary systems, T equal-sized Txp shards are created such that $T=N/M$ where M is the shard size. Each shard ideally proposes and validates 1 distinct block every epoch (i.e., 1:1 mapping of block to Txp shard).

In contrast, in SecurShard, the Ref shard splits an original contemporary shard of size M into k smaller equal-sized Txp shards of size m_{Tx} each; here, $M = (k \times m_{Tx})$, $M \bmod k = 0$ (i.e., M must be a multiple of k), $1 < k < M$, and $m_{Tx} > 3$ for BFT to be applied. The split size k is called the *mapping factor*. The k smaller shards form a group that validates the same transaction block, i.e., a transaction block is mapped to all the k Txp shards in a group (one-to-all mapping). The benefits of k smaller shards as opposed to one bigger shard and the choice of a good mapping factor k are discussed later.

Failure probability of the Ref shard is obtained by substituting M to be shard size m_{Ref} in equation 4.4. Since the Ref shard is the bottleneck in SecurShard and needs to be fully trustworthy, a big shard size should be used to obtain a negligible failure probability as the Ref shard needs to be perfect in SecurShard. Multiple m_{Ref} values should therefore be trialed until the Ref shard failure probability desired by the user is attained.

$$P_{Faulty} = P(X > f) = P\left(X \geq \left\lceil \frac{M}{3} \right\rceil\right) = \sum_{x=\left\lceil \frac{M}{3} \right\rceil}^M \frac{\binom{F}{x} \binom{N-F}{M-x}}{\binom{N}{M}} \quad (4.4)$$

Following the formation of the Ref shard, total number of remaining nodes $N_{Tx} = N - m_{Ref}$. Assuming that the Ref shard contains the expected number of faulty nodes $E(F_{Ref}) = m_{Ref}/4$ when $F = N/4$, the number of faults remaining after formation of Ref shard, F_{Tx} therefore amounts to $F - E(F_{Ref})$. Consequently, failure probability of any Txp shard of size m_{Tx} is obtained by setting F to F_{Tx} , N to N_{Tx} and M to be m_{Tx} in equation 4.4.

Note that in general, if $F=N/w$ where $1 \leq w \leq N$ and $(N \bmod w) = 0$ (i.e., N is a multiple of w), then $E(F_{Ref}) = m_{Ref}/w$ if the hypergeometric distribution is used for the calculation of shard failure probability.

Unlike in contemporary systems where all shards (i.e., Txp shards) need to be nonfaulty, SecurShard requires that only 1 Txp shard in each group, and the Ref shard be nonfaulty to preserve safety (i.e., T ‘perfect’ Txp shards required instead of $(T \times k)$). Therefore, number of ‘perfect’ Txp shards required decreases by a factor of k . Section 4.4 presents a detailed analysis of *SecurShard* in terms of its enhanced fault tolerance, performance, expected throughput, and progress with respect to the different parameters (e.g., m_{Ref} , m_{Tx} , T , and k).

4.3.2. Randomized assignment of transaction blocks to Txp shards

In SecurShard, the Ref shard therefore creates $(T \times k)$ Txp shards. These Txp shards are then shuffled and randomly assigned to T distinct groups of size k each by the Ref shard.

During each epoch, the leader of every Txp shard proposes a block to the Ref shard. The block proposal mechanism has been discussed in depth in section 4.2. So $(T \times k)$ blocks get proposed. The Ref shard can subsequently do either of the following two options:

1. Randomly choose one of the k blocks proposed by each shard from a group and assign the chosen block to the same group that proposed the block.
2. Randomly choose T blocks from the $(T \times k)$ proposed blocks and randomly assign a block to each group.

If option 1 is opted for, the probability that a shard gets assigned the block it proposed is $(1/k)$. For option 2, that probability decreases to $(1/(T \times k))$. The randomization of block assignment adds another layer of fault tolerance to SecurShard. The choice of option is a design decision left to the system administrator.

Each of the k shards belonging to the same group individually validates the same block – essentially creating a 1:all mapping of block to Txp shards. During block validation, nodes inside each shard reach consensus using BFT. BFT leaders of each shard aggregate and send the results of transaction processing to the Ref shard. For example, the result can be a Merkle root with $(2f+1)$ signed signatures for partially synchronous BFT or $(f+1)$ signed signatures for synchronous BFT protocols where f is the maximum number of faulty nodes a shard can withstand. If the blockchain is sharded as well into T different chunks, shards belonging to the same group receive the same chunk/part of the blockchain. They can then take part in cross-shard transaction-verification by communicating with any of the shards in other groups having part of the blockchain and/or UTXO pool that is required. Numerous cross-sharding protocols already exist in the literature [10] [12] and are beyond the scope of this thesis. The process of transaction validation inside SecurShard's Txp shards has already been discussed in section 4.2.

4.3.3. Invalid transaction block and potentially faulty group detection

100% consensus is required among all the k shards in the same group for a transaction block assigned to the group to be validated. Results of shards in the same group are compared by the Ref shard and only if all results match that the block is validated with finality and appended to the blockchain. Therefore, if faulty shards are present in a group and they produce incorrect validation results for a transaction block, the group will be flagged as faulty as long as one Txp shard in the group and the Ref shard are nonfaulty. An invalid transaction block will consequently not be appended to the blockchain. For the purpose of efficiently comparing results, a common consensus decision inside a shard is to reach agreement on the Merkle root instead of signing and sending entire blocks during consensus. Systems implementing SecurShard can adapt to send different types of consensus results to the Ref shard.

4.3.4. Faulty shard detection

Detecting a potentially faulty group does not mean detecting a faulty shard though. Detecting a faulty group is sufficient to prevent an invalid block from being appended to the blockchain. However, more information is needed to specifically determine which shards are faulty within a group. If a group does not obtain 100% consensus on validation results, the Ref shard can then subsequently validate the proposed block itself (audit or replay transactions [40]) and compare its result with every shard's result in the group. Shards that do not have the same result are considered to be faulty. However, the approaches of replaying transactions or auditing are expensive and become a bottleneck as the Ref shard does duplication of work and essentially all the block validation – thus defeating the purpose of sharding and concurrency.

Since the main concern of SecurShard is the prevention of invalid blocks from being appended to the blockchain, there is essentially no need for specific faulty shard detection. When a group is flagged as faulty, there can be other measures as follows: a group can be flagged and given another chance instead of carrying out a full audit of all potentially faulty groups every epoch. Some systems use a scoring system to rank shards [46]. Ref shard can assign a confidence score to each group and re-shard a group if some score threshold is not met. A simple metric to rank shards and groups can be to compute the cosine similarity of the results from each shard. If multiple groups have low scores, the entire network could be re-sharded. Ref shard could then perform a full audit as well to pinpoint faulty shards, as re-sharding the network is costly anyway. The choice of detecting a faulty group or exact faulty shards is a design decision left to system administrators. Both have the same goal ultimately: to successfully detect an invalid block during transaction processing and prevent it from being appended to the blockchain.

A validated correct block can finally be disseminated to the entire network at the end of an epoch using some gossiping protocol. Reference shard issues a 'block append certificate' after appending a block with finality to the blockchain. A shard member belonging to a particular group can then gossip a confirmed block from its group and the 'block append certificate' to other groups. Preventive measures such as the cuckoo rule [47] [48] [10] can also be implemented, whereby a fixed number of nodes are constantly shuffled around between shards to keep a 'moving target' for an adversary. Additionally, the Ref shard could replay transactions every few epochs to identify shards that are most vulnerable. Subsequently it can apply the cuckoo rule or re-shard the vulnerable shards more often.

Figure 4.6 outlines the communication diagram of an epoch of processing in SecurShard. The contents of messages m1 – m9 are as follows:

- m1: Epoch Identity and Epoch Randomness (Signed by Ref shard members).
- m2: Epoch Identity, Epoch Randomness and Proposed block.
- m3: Signed Merkle Root of validated block.
- m4: Signed Merkle Root approved by Txp intra-shard consensus.
- m5: List of Merkle Roots approved by Ref intra-shard consensus for particular epoch.

- m6: Approved block to be appended to the blockchain.
- m7, m8: Block-Append-Certificate issued by Ref shard.
- m9: Gossiping of Block-Append-Certificates & approved blocks, to and from other groups.

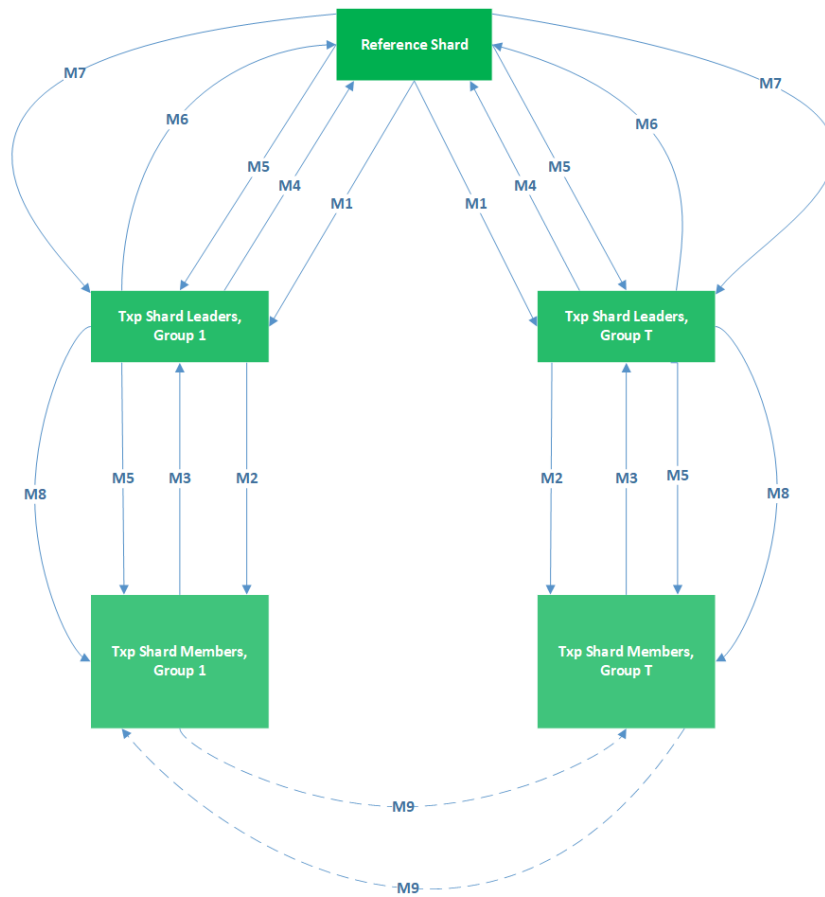


Figure 4.6 Communication diagram of an epoch processing in SecurShard.

4.4. Theoretical Analysis of SecurShard

4.4.1. Increased fault tolerance due to hierarchy and one-to-all mapping

Assume that S is a big Txp shard of size M that would be formed in a contemporary system. If S is faulty (malicious) and includes some invalid transaction(s) in a block, then that invalid block may be appended to the blockchain. There is no way of checking for invalid blocks before they are finalized.

In SecurShard, an invalid block can only be appended to the blockchain if and only if ALL k shards belonging to the group that is tasked to validate that block are faulty as well. However, if the faulty shard S is split into k smaller shards, we claim that there is no guarantee that all k smaller shards will always be faulty. With the introduction of a central authority requiring 100% consensus among shards, since 1 nonfaulty shard out of k is sufficient to detect a faulty group (and block), there are more chances that an invalid block gets detected if S is therefore split, than if S is not split. This is the argument that we use to say that our protocol can be more fault tolerant than contemporary systems if proper choices are made for parameters such as m_{Ref} , m_{Tx} , and k . We start with the following Lemma.

Lemma: A shard S of size M is split into k equal-sized smaller Txp shards of size m_{Tx} each, i.e. $k \times m_{Tx} = M$. If all k Txp shards are faulty, then it implies that the original big shard S is also faulty; however, the reverse is not true.

Proof a) *All k Txp shards being faulty implies that the original big shard S is faulty as well.*

The base case (or minimal case) is sufficient to prove this. In the base case, every small shard of size m_{Tx} is barely faulty. Assuming a partially synchronous BFT protocol is used inside shards, $1/3$ nodes being faulty is sufficient to corrupt a shard. Therefore, each Txp shard contains $(m_{Tx} / 3)$ faulty nodes.

If all the k barely-faulty shards are combined to form S , the number of faulty nodes in $S = (k \times m_{Tx}) / 3$. Since $M = (k \times m_{Tx})$, number of faulty nodes in $S = M / 3$, which signifies that $1/3$ of the total nodes in S are faulty. Therefore, S is faulty.

Proof b) *The big shard S being faulty does not necessarily imply that all k smaller Txp shards formed by splitting S will always be faulty as well for some k where $1 < k < M$ and $M \bmod k = 0$ (The reverse of proof a).*

We prove this by contradiction. Let the big shard S be barely faulty, and f_S be the number of faulty nodes in S . Then,

$$f_S = \frac{M}{3} = \frac{(k \times m_{Tx})}{3}$$

Now assume that all the k equal-size Txp shards of size m_{Tx} that are formed by splitting S are always faulty.

There can be a scenario where the split of S results in one of the shards of size m_{Tx} being completely faulty, i.e., all the m_{Tx} nodes of the shard are faulty. In such a case, let $f_{Remaining}$ be the number of faulty nodes remaining. Then,

$$f_{Remaining} = \frac{(k \times m_{Tx})}{3} - m_{Tx}$$

$$f_{Remaining} = \frac{(k - 3) \times m_{Tx}}{3}$$

In the worst case, each of the remaining shards contains the bare (minimum) number of faults required (i.e., $m_{Tx} / 3$), so that the shard is faulty, and the number of faulty shards can be maximized. As a result, the maximum number of faulty shards that can then be formed of the remaining faulty nodes is

$$\frac{f_{Remaining}}{m_{Tx}/3} = (k - 3)$$

Hence the total number of corrupt shards = $(k - 3) + 1 = (k - 2)$. Thus, there exist 2 shards out of the k shards that are not faulty, which is a contradiction. This concludes the proof. ■

The previous Lemma implies that the likelihood of all k shards being faulty in a group of SecurShard is lower than the likelihood that an original contemporary shard S is faulty if it is not split. The presence of one such nonfaulty smaller shard after splitting S is sufficient to prevent a potentially faulty block from being validated. If S was faulty but not split, a potentially faulty block would be validated with certainty, which is what we want to prevent. Hence, the one-to-all mapping scheme in SecurShard, coupled with the introduction of a Reference committee to compare results of all k shards in a group to reach 100% consensus, can lead to potential fault detection, better fault tolerance and collective resilience if proper values of the different parameters (e.g., m_{Ref} , m_{Tx} , T , and k) are chosen. This is discussed in the following.

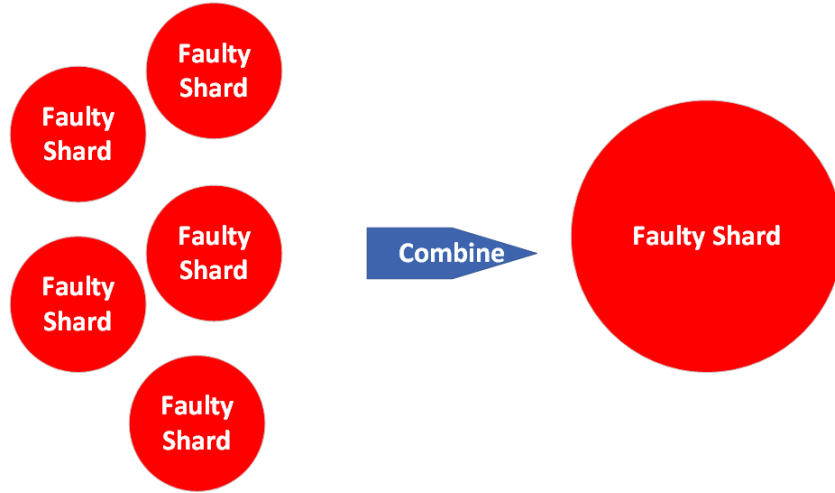


Figure 4.7 Proof a) of SecurShard's Lemma: All k shards being faulty implies that a big contemporary shard is faulty as well.

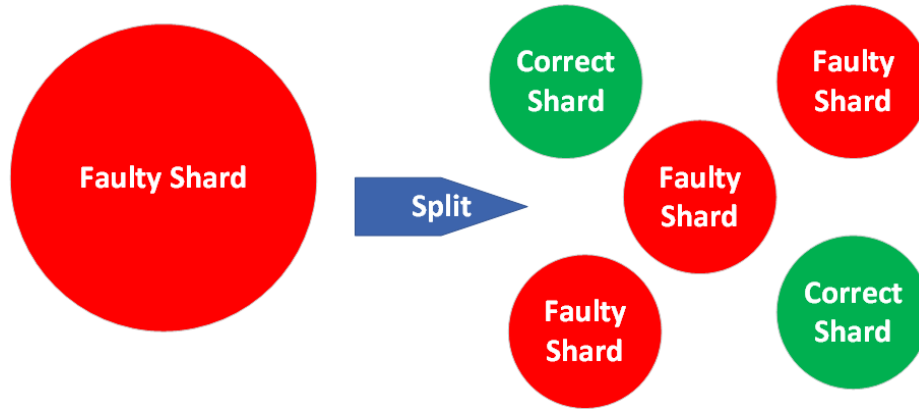


Figure 4.8 Proof b) of SecurShard's Lemma: A big contemporary shard being faulty does not imply that all shards resulting from its split are faulty as well.

4.4.2. Use case analysis

This section discusses a use case of SecurShard. Recall that the system's assumptions have been discussed in section 4.2. We analyze the following parameters:

a) Impact of shard size on shard failure probability

Shard failure probability is calculated using the cumulative hypergeometric distribution (equation 4.4). We can deduce from equation 4.4 that the failure probability of a shard decreases (i.e., shard becomes more fault tolerant) as the shard gets bigger. Since Ref shard needs to be perfect, a large value for m_{Ref} is desired. The right balance needs to be found between fault tolerance and performance since a big shard size negatively impacts performance (discussed later).

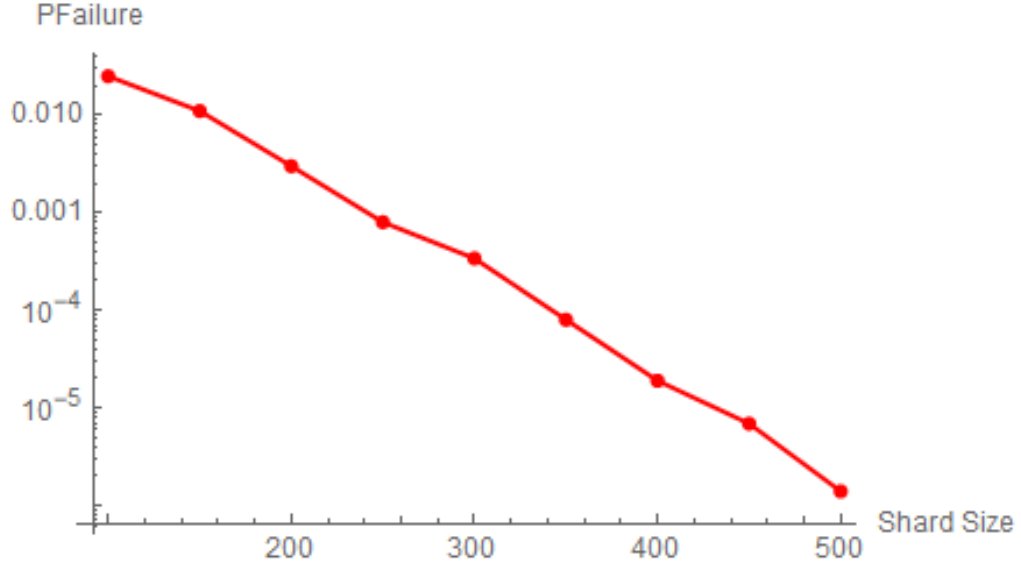


Figure 4.9 Shard failure probability versus Ref shard size when $N=2500$, $F=625$ and $f < m_{Ref}/3$.

b) Impact of mapping factor k on collective failure probability of a group of k Txp shards

Let $P_{TxpFailure}$ be the probability that an individual Txp shard of size m_{Tx} is faulty. $P_{TxpFailure}$ is obtained by substituting F by F_{Tx} , N by N_{Tx} , and M by m_{Tx} in equation 4.4. Recall from section 4.3 that $N_{Tx} = N - m_{Ref}$, $F_{Tx} = F - E(F_{Ref})$, and $m_{Tx} = \frac{N_{Tx}}{(T \times k)}$, where T is the desired throughput of the system. T needs to be chosen by a user beforehand.

From the previous paragraph, it can be seen that an increase in mapping factor k entails a decrease in shard size m_{Tx} . A decrease in shard size results in an increase in the probability $P_{TxpFailure}$ that an individual shard fails. Hence increasing mapping factor k increases the likelihood $P_{TxpFailure}$ than an individual Txp shard is faulty.

However, due to the Ref shard validation step where 100 % consensus is required, all k shards need to fail collectively in a group for an invalid block to be appended to the blockchain. Let $P_{CollectiveFailure}$ be the probability that all k Txp shards in a group are faulty, which is also the probability that an invalid block gets appended to the blockchain. It is given by:

$$P_{CollectiveFailure} = (P_{TxpFailure})^k \quad (4.5)$$

Let P_{Valid} be the probability that a transaction block gets correctly validated or a potentially invalid transaction block gets detected. It is given by:

$$P_{Valid} = 1 - P_{CollectiveFailure} = 1 - (P_{TxpFailure})^k \quad (4.6)$$

From equation 4.5, it can be inferred that the probability $P_{CollectiveFailure}$ that a group fails collectively decreases as the mapping factor k increases. Therefore, it can be inferred from equation 4.6 that the probability P_{Valid} that a block gets correctly validated increases with increasing k . In other words, the more shards an original contemporary shard S is split into, it is more likely that a transaction block mapped to that group is correctly validated, which is equivalent to saying that it is more likely that an invalid transaction block gets detected. When proper values of k are chosen, it can be observed that the collective failure probability of a group can be lower than the failure probability of an original big shard S . SecurShard can therefore not only be more resilient, but it can also enable detection of faulty transaction blocks during transaction-validation with high probability as compared to contemporary sharding systems where there is no such mechanism for faulty block detection.

Now let us consider a use case with total nodes $N=2500$ and total number of Byzantine faults present in the system, $F = N/4 = 625$. Let m_{Ref} chosen in this case be 500. The number of remaining nodes, $N_{Tx} = 2000$. The expected number of faulty nodes in the *Ref* shard $= E(F_{Ref}) = m_{Ref} / 4 = 125$. Hence the remaining number of faulty nodes in the system, $F_{Tx} = F - E(F_{Ref}) = 500$.

Let $T = 4$ be the desired throughput of the SecurShard system. In a contemporary sharding system, 4 Txp shards S , each of size $M=500$, would be formed. In contrary, in SecurShard each shard S is split into k smaller Txp shards. Tables 4.1, 4.2, 4.3, and Figures. 4.10, 4.11 and 4.12 illustrate the results from these possible splits for $k = 2, 4, 5$ and 10 at different shard sizes of S . It can be seen that as mapping factor k increases, individual Txp shard size gets smaller resulting in higher individual shard failure probability $P_{TxpFailure}$. However, the collective failure probability $P_{CollectiveFailure}$ of a group of k Txp shards decreases, therefore making the system more fault resilient. Note that when $k=1$, the results correspond to a contemporary sharding system with big shards S .

The randomization of block assignment to Txp shards (section 4.3.2) also introduces an additional layer of fault tolerance. The probability that a shard gets assigned the same block it proposed is inversely proportional to k . Hence, increasing mapping factor k reduces the chance that a shard validates the same set of transactions it proposes, furthermore demonstrating that a one-to- k mapping can result in better fault tolerance than contemporary systems.

Mapping factor, k	Shard size, m_{Tx}	Individual shard failure, $P_{TxpFailure}$	Collective group failure, $P_{CollectiveFailure}$	Liveness attack, $P_{LivenessAttack}$
1	500	6×10^{-7}	6×10^{-7}	N/A
2	250	7×10^{-4}	5×10^{-7}	1×10^{-3}
4	125	0.02	2×10^{-7}	8×10^{-2}
5	100	0.02	3×10^{-9}	9×10^{-2}
10	50	0.09	3×10^{-11}	6×10^{-1}

Table 4.1 Failure probabilities for different mappings when $T=4$, $M=500$, $N_{Tx}=2000$ and $F_{Tx}=500$.

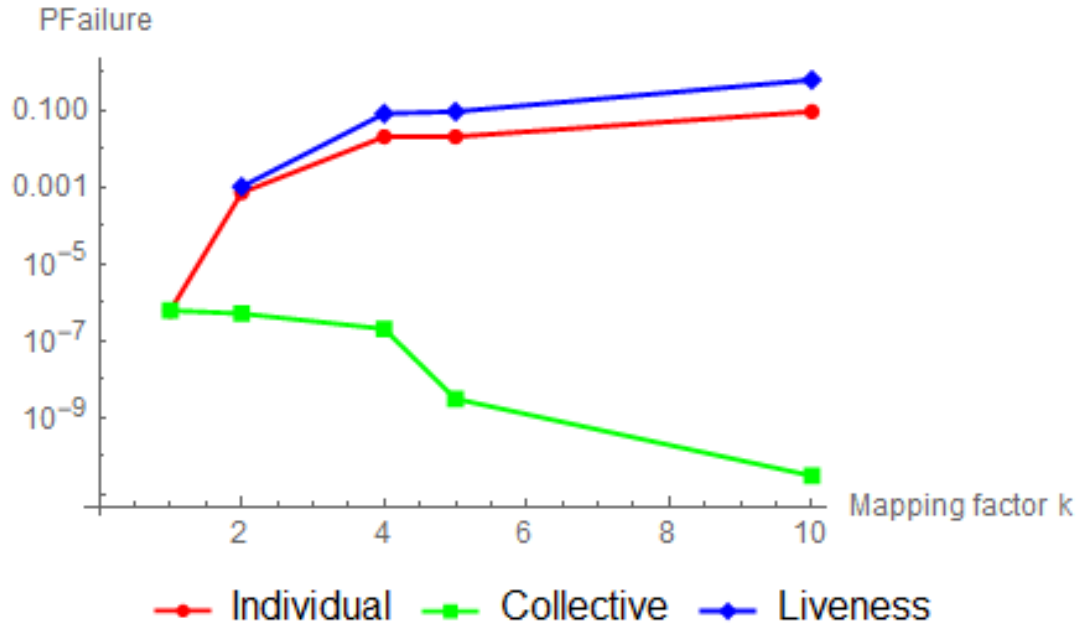


Figure 4.10 Failure probability when a contemporary shard of size $M=500$ is split into different number of fragments k .

Mapping factor, k	Shard size, m_{Tx}	Individual shard failure, $P_{TxpFailure}$	Collective group failure, $P_{CollectiveFailure}$	Liveness attack, $P_{LivenessAttack}$
1	400	1×10^{-5}	1×10^{-5}	N/A
2	200	3×10^{-3}	9×10^{-6}	6×10^{-3}
4	100	0.02	2×10^{-7}	8×10^{-2}
5	80	0.05	3×10^{-7}	2×10^{-1}
10	40	0.10	1×10^{-10}	7×10^{-1}

Table 4.2 Failure probabilities for different mappings when $T=5$, $M=400$, $N_{Tx}=2000$ and $F_{Tx}=500$.

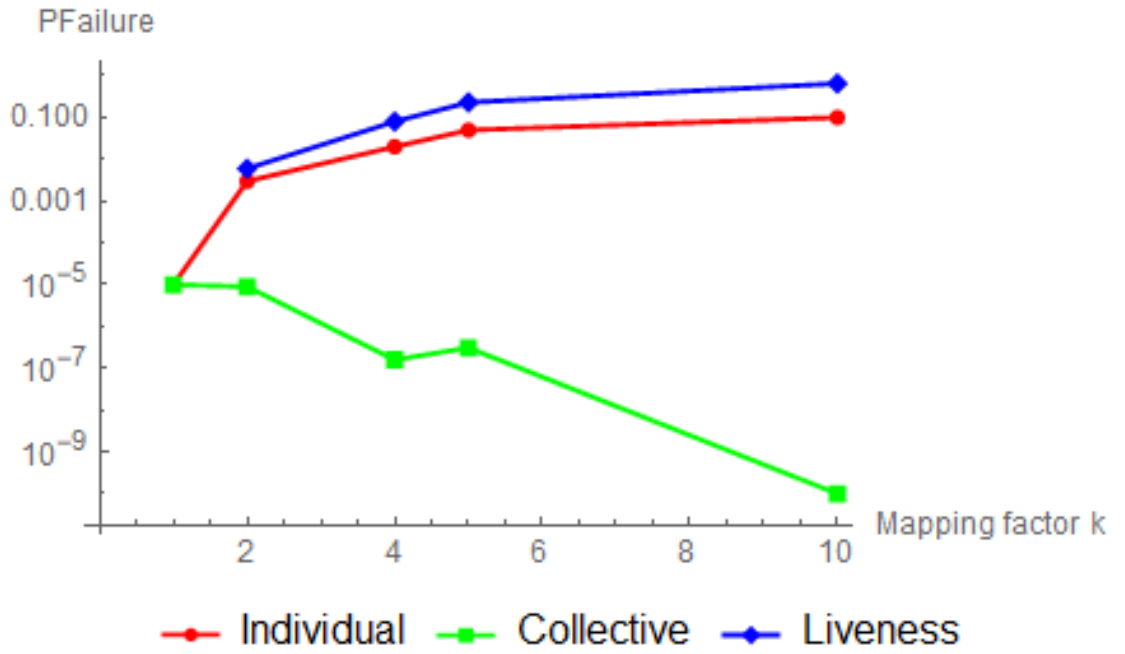


Figure 4.11 Failure probability when a contemporary shard of size $M=400$ is split into different number of fragments k .

Mapping factor, k	Shard size, m_{Tx}	Individual shard failure, $P_{TxpFailure}$	Collective group failure, $P_{CollectiveFailure}$	Liveness attack, $P_{LivenessAttack}$
1	250	7×10^{-4}	7×10^{-4}	N/A
2	125	0.02	4×10^{-4}	0.04
5	50	0.10	1×10^{-5}	0.40

Table 4.3 Failure probabilities for different mappings when $T=8$, $M=250$, $N_{Tx}=2000$ and $F_{Tx}=500$.

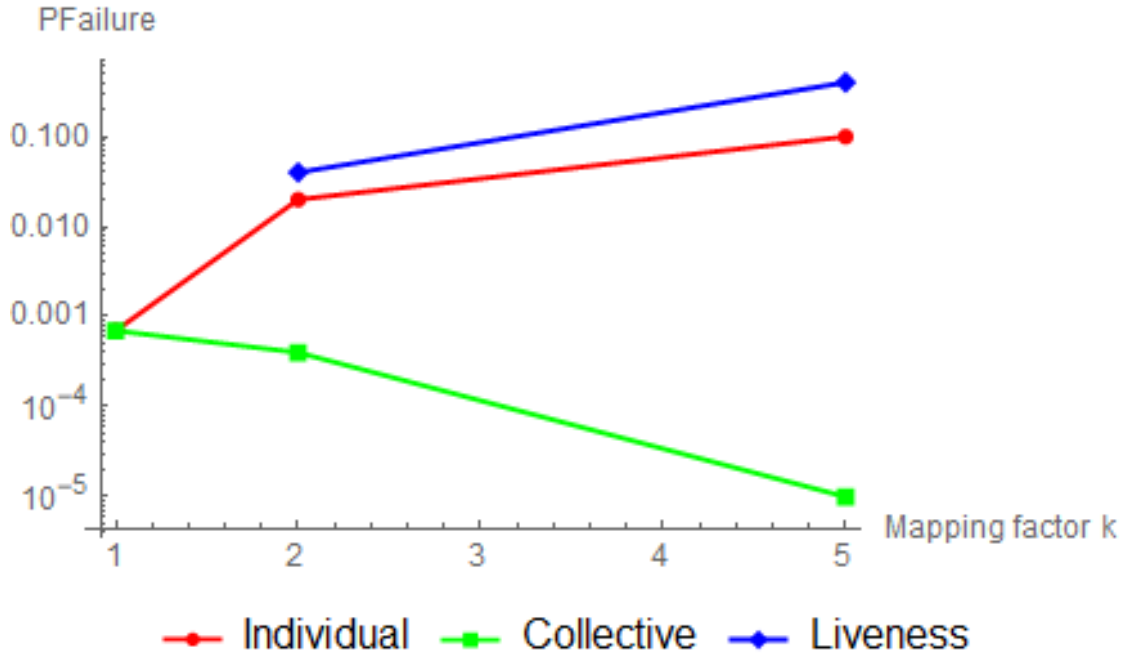


Figure 4.12 Failure probability when a contemporary shard of size $M=250$ is split into different number of fragments k .

c) Liveness Attack

Since a 100% consensus is needed among the k shards in a particular group to validate a block, the presence of a single faulty shard is sufficient to prevent the progress of that group. A single faulty shard can therefore always send invalid results in every epoch such that the group it belongs to never validates a block successfully and never makes progress. We term this lack of progress as a *liveness attack*.

Recall from the previous discussion that when mapping factor k increases, the probability $P_{TxpFailure}$ that an individual Txp shard is faulty increases, however the collective failure probability $P_{CollectiveFailure}$ decreases. Consequently, although the collective security of a system (safety property) improves with increasing values of k , the system becomes more prone to liveness attacks (prevents progress) as it becomes easier to corrupt a shard individually.

A group of k shards can suffer from liveness attack when at least 1 of the shards is faulty. Hence, the probability of liveness attack in the group is given by $P_{LivenessAttack}$. This can be formulated as follows:

$$P_{LivenessAttack} = 1 - P(\text{all shards in the group are nonfaulty})$$

$$P_{LivenessAttack} = 1 - (1 - P_{TxpFailure})^k \quad (4.7)$$

Figures 4.10 to 4.13 support our claim that a group is more prone to liveness attacks if it has more shards. Hence care should be taken in choosing a good mapping factor, because opting for the highest value of k does not necessarily lead to a better system.

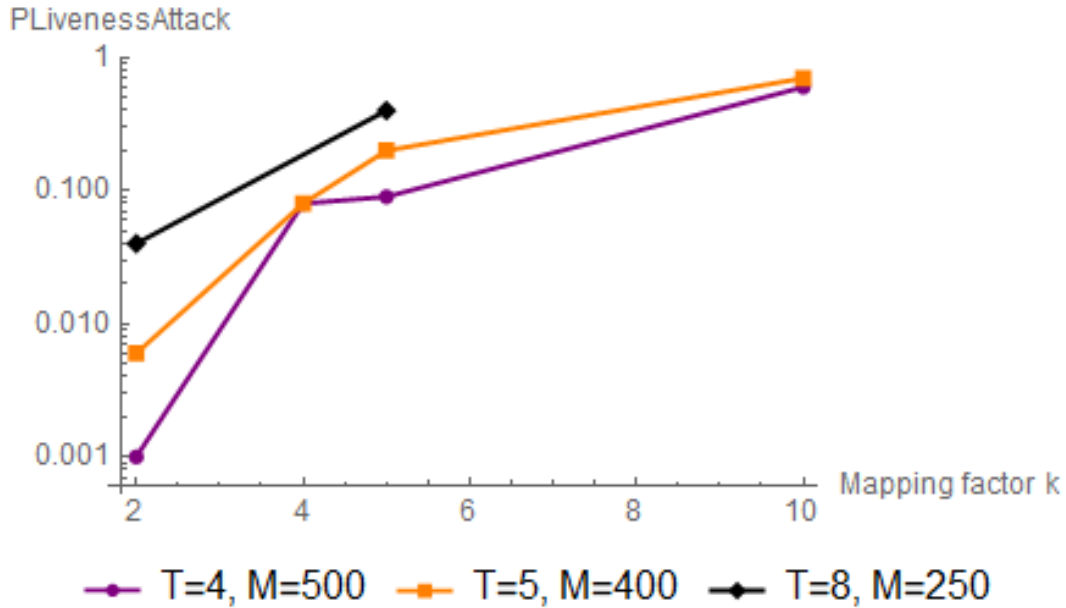


Figure 4.13 Probability of liveness attack for different throughputs and k values

d) Impact of throughput on collective fault tolerance of a group

Recall from section 4.3 that $m_{Tx} = \frac{N_{Tx}}{(T \times k)}$. Rearranging the previous equation, we get $T = \frac{N_{Tx}}{(m_{Tx} \times k)}$. The previous equation shows that throughput T is inversely proportional to both mapping factor k and shard size m_{Tx} . With increasing T , the ranges of possible values for both k and m_{Tx} decrease. Based on the previous discussion, an increase in T results in an increase in the collective failure probability $P_{CollectiveFailure}$ of a group of k shards and therefore a decrease in fault tolerance of the system. Decreasing T has the opposite effect. Figure 4.14 shows the collective failure probabilities for the current use case when desired throughputs are, $T = 2, 4$, and 8 respectively.

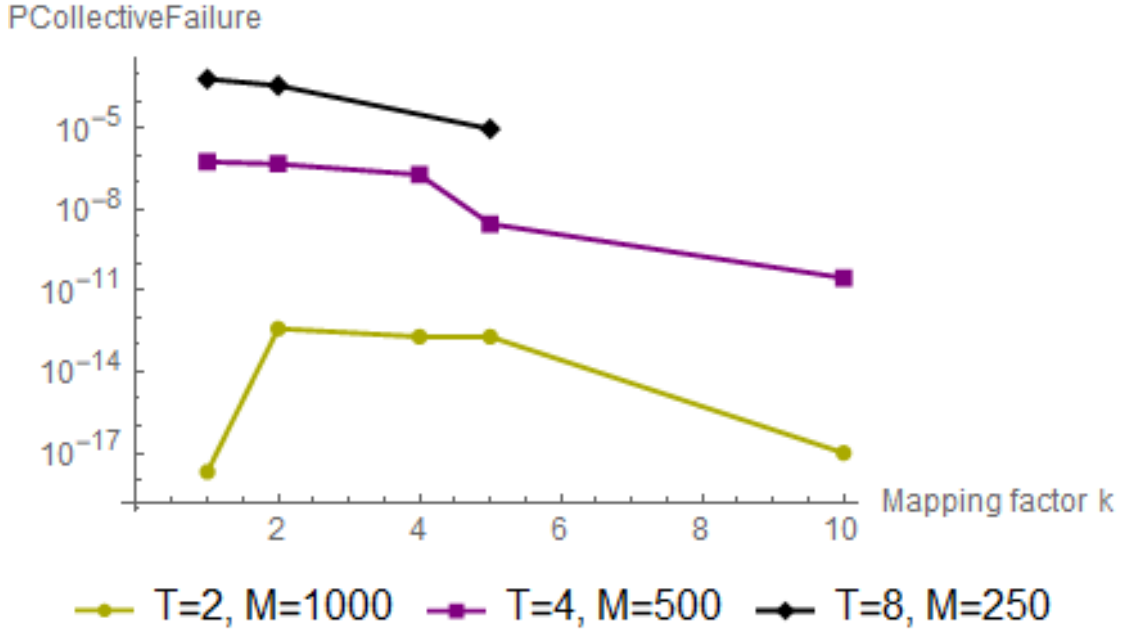


Figure 4.14 Collective failure probability at different throughputs when $N_{Tx} = 2000$ and $F_{Tx} = 500$.

4.4.3. Performance Analysis of SecurShard

a) Decrease in intra-shard message complexity for Txp shard

Classical BFT algorithms such as [8] [9] are quadratic in terms of message complexity. If M is the size of a contemporary shard S , intra-shard message complexity is $O(M^2)$ per consensus. In SecurShard, intra-shard message complexity per consensus becomes $O(m_{Ref}^2)$ for the Ref shard, and $O(\frac{M^2}{k^2})$ for the each of the Txp shards which work in parallel on the same transaction block. Hence, number of message exchanges decreases by a factor of k^2 inside Txp shards when a 1: all mapping is used with quadratic BFT algorithms. For large values of k , the reduction in number of message exchanges can considerably boost performance.

If a state-of-the-art linear BFT algorithm such as HotStuff [49] is used, message complexity per consensus of a Txp shard in SecurShard becomes $O(\frac{M}{k})$. This is a reduction by a factor of k which is still beneficial.

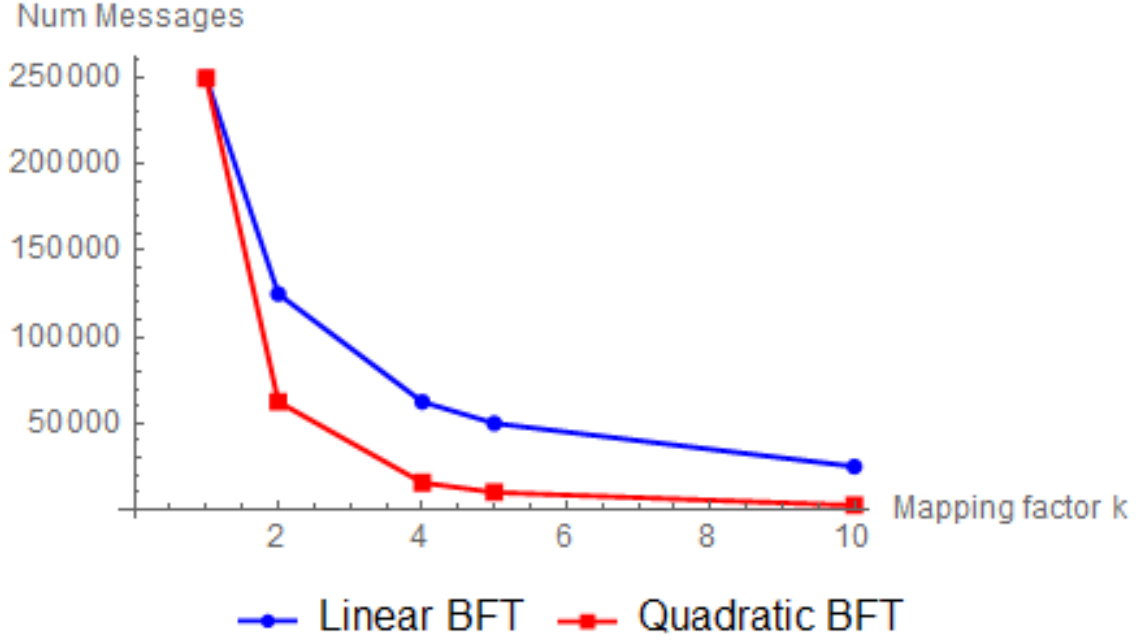


Figure 4.15 Number of message exchanges per consensus versus mapping factor k when $M=500$ is split into k fragments.

b) Expected Throughput $E(T)$ per epoch

Let $P_{Progress}$ be the Probability that a group of k shards makes progress during an epoch, i.e., the group does not suffer from a liveness attack and therefore correctly validates a transaction block during that epoch. As a follow up of equation 4.7, it can be formulated as:

$$P_{Progress} = 1 - P_{LivenessAttack} = (1 - P_{TxpFailure})^k \quad (4.8)$$

Let Y be the random variable denoting the number of groups out of all T groups that make progress in that epoch. Since each group has only 2 possible outcomes (progress or no progress) and their outcome are independent of other groups after being sharded, the binomial distribution can be used to calculate the probability that t groups out of T make progress every epoch where, $0 \leq t \leq T$. If ($t > 0$), we consider the system to be partially live. The probability that t out of T groups make progress in an epoch is given by:

$$P(Y = t) = \binom{T}{t} \times (P_{Progress})^t \times (P_{LivenessAttack})^{T-t} \quad (4.9)$$

Following from equation 4.9, the expected throughput $E(T)$ per epoch of a SecurShard system is given by:

$$E(T) = \sum_{t=0}^T t \times P(Y = t) \quad (4.10)$$

Equation 4.10 can be simplified to equation 4.11 below :

$$E(T) = T \times P_{progress}, \quad \text{where } 0 \leq E(T) \leq T \quad (4.11)$$

$E(T)$ represents the expected number of groups that will make progress during an epoch, which translates to the expected number of blocks that will be correctly validated in the system during that epoch. $E(T)$ is a useful metric that can be referred to when trying to find a good balance between fault tolerance (mapping factor k) and performance (liveness attack). Figure 4.16 illustrates the variation of expected throughput against mapping factors for 3 different configurations. We note that as mapping factor k increases, shards become more prone to individual failures. It becomes easier to corrupt at least 1 shard in a group. In turn, more groups become vulnerable to liveness attacks. As a consequence, expected throughput of a system decreases.

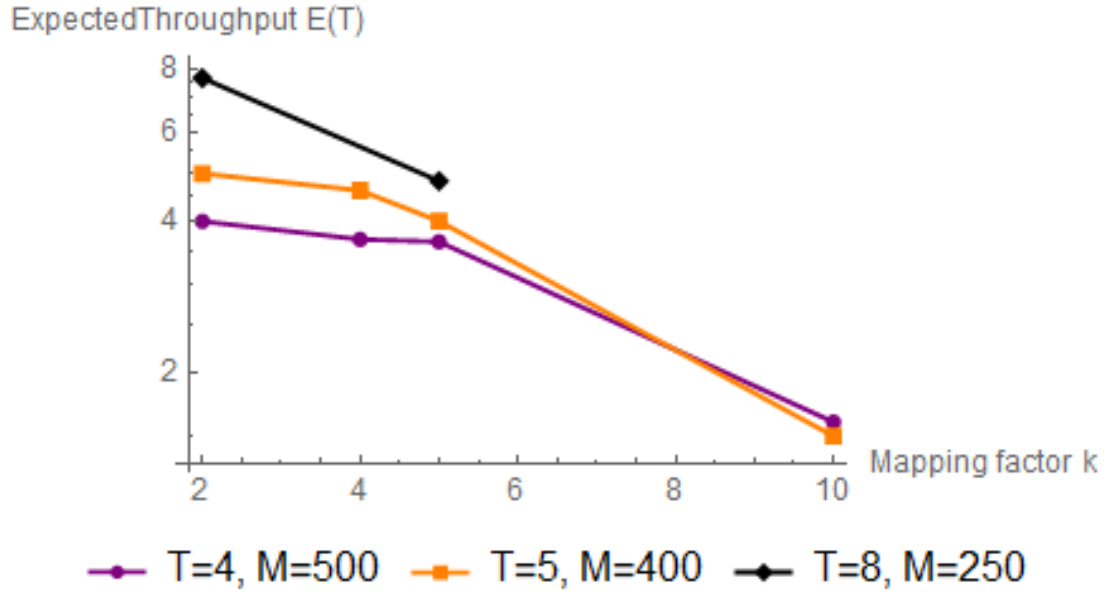


Figure 4.16 Expected throughput for different mapping factors and different desired throughputs.

4.4.4. Choosing a good mapping factor k

We analyze Figures 4.10 to 4.16, to come up with a good mapping factor k when $N=2500$, $F= N/4 = 625$, $m_{Ref}= 500$ and desired throughput $T= 4$.

From Figure 4.10, when $k = 2, 4$ or 5 , collective group failure probability seems to be comparable or even lower than if an original big shard S of size $M=500$ would have been used (i.e., when $k = 1$). However, analysis of liveness attack from Figure 4.13 shows that a group becomes vulnerable by 1-2 orders of magnitude when k increases from 2 to 4. Expected throughput from Figure 4.16 remains stable and close to the maximum desired throughput of 4 for all three k values.

Additionally, considering that number of intra-shard message exchanges decreases by a factor of k^2 when a quadratic BFT protocol is used, $k = 2$ or 4 seem to be reasonable values to choose in this use case. Figures 4.17 and 4.18 summarize the steps involved in choosing proper parameter values for SecurShard.

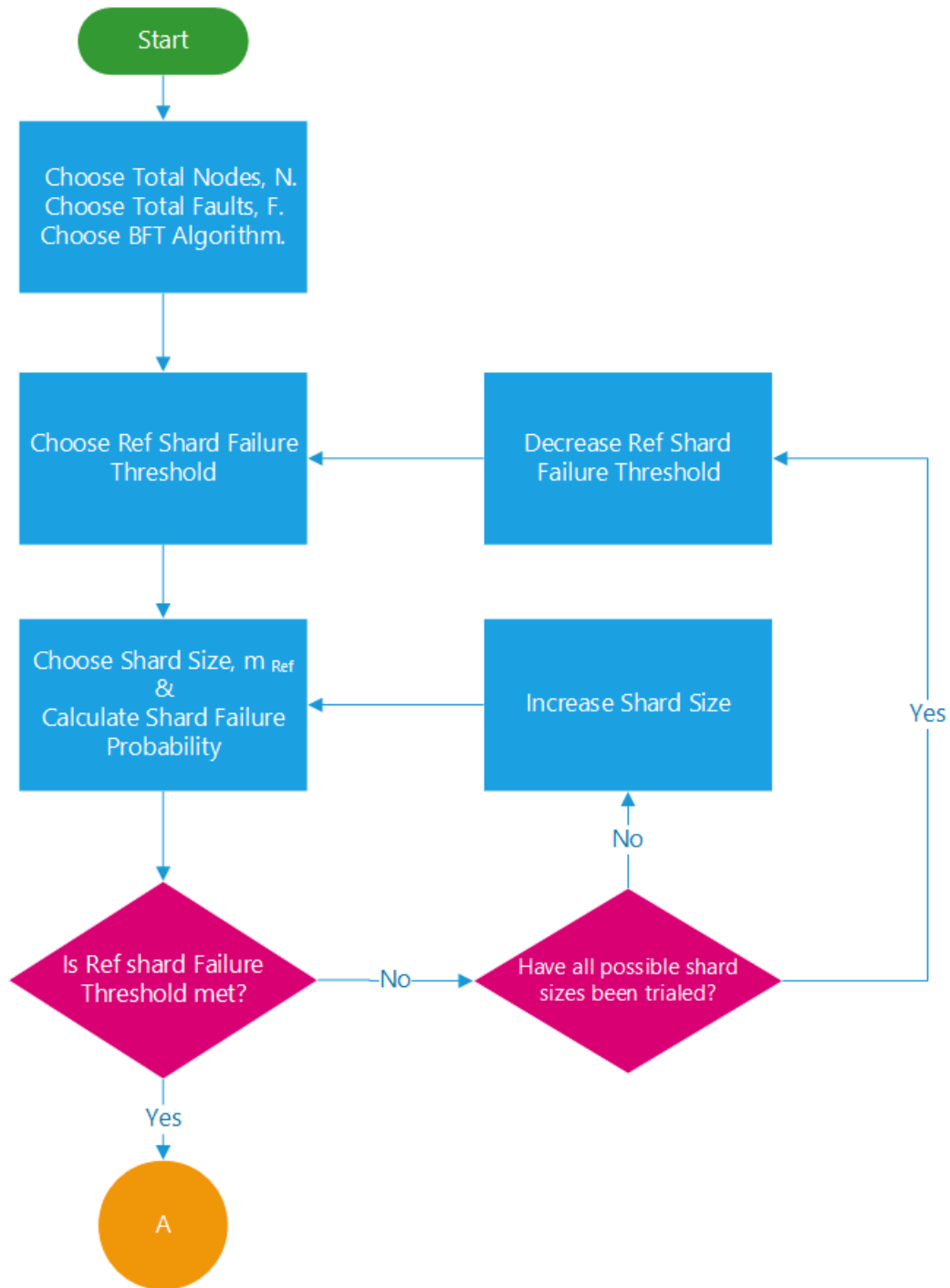


Figure 4.17 Flowchart to choose a proper Ref shard size m_{Ref} .

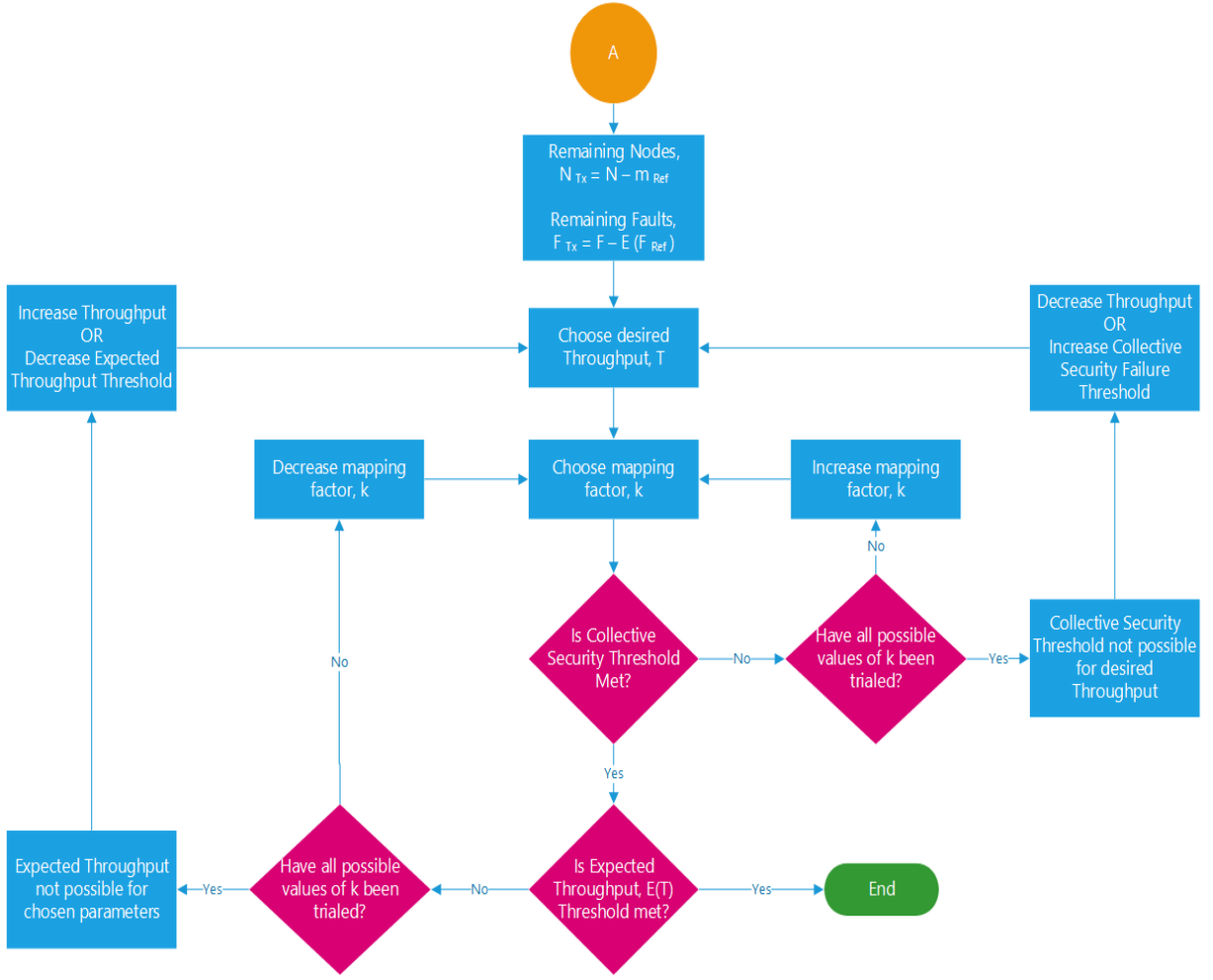


Figure 4.18 Flowchart to choose a proper mapping factor k and other parameters.

4.5. Summary

In this Chapter, we presented SecurShard: a model for hierarchical fault detection during transaction processing in blockchain sharding. The ability to detect potentially faulty blocks (and shards) can relax conservative assumptions like the ‘perfect’ shard constraint and large shard sizes. This can in turn allow the use of smaller shards which can lead to lower communication cost inside shards during intra-shard consensus (and therefore lower latency) and better performance.

SecurShard introduced the novel concept of dividing contemporary shards into groups of k shards each (k is called the mapping factor), 1 to k mapping of a block to all k shards in a group, and finally requirement to reach 100% consensus on the validation results of a block for it to be appended to the blockchain. Due to reduced shard size, there was a significant reduction in message complexity (by a factor of k^2) in Txp shards during intra-shard consensus when quadratic time BFT was used. There was a fair amount of reduction for linear time BFT as well (by a factor of k). The number of perfect shards required to maintain safety also decreased by a factor of k . Although there was an increase in failure probability of individual shard due to decreased shard size, the need

for reaching 100% consensus among all k shards in a group decreased the collective failure probability of the group. This collective failure probability of a whole group can potentially be much more fault tolerant than an original individual contemporary shard S . Theoretical and use case analyses of SecurShard were presented which showed that the SecurShard system can be made more fault resilient and performant in comparison to contemporary sharding systems by properly selecting the various parameters (especially the mapping factor k), while introducing the capability of potentially faulty block detection with high probability during transaction processing, and relaxing the conservative ‘perfect’ shard assumption. This concludes the Chapter.

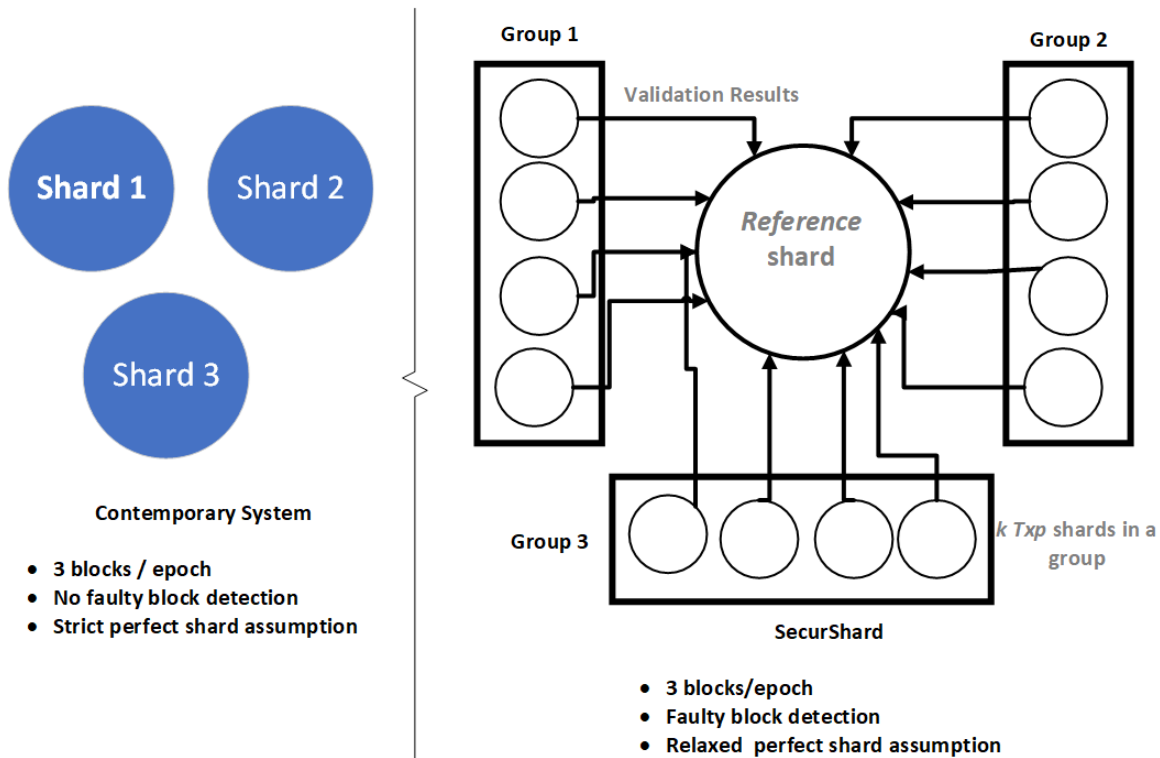


Figure 4.19 Illustration of the differences between contemporary sharded systems and SecurShard.

Chapter 5. CONCLUSION & FUTURE WORKS

In the first two Chapters, we discussed the use of partial synchrony by contemporary systems to obtain responsive shards which are however large in size. In synchrony, smaller fault tolerant shards can instead be created but these shards are not responsive (which affects performance anyway). The use of some form of conservative assumption or parameter in contemporary systems seems to stem from the fact that shards need to adhere to the ‘perfect’ shard assumption and can never fail. This in turn negatively affects throughput and/or performance.

In Chapter 3, by applying partially synchronous Flexible BFT to blockchain and its sharding, the benefits of both synchrony and partial synchrony are leveraged to obtain small, resilient and responsive shards which can help in increasing both throughput and performance while maintaining fault tolerance. Creation of smaller shards implies more shards in a system and hence more throughput in proportion to the number of shards. Use of smaller shards also decreases transaction latency (smaller number of message exchanges) which in turn improves performance.

Due to Flexible BFT’s hybrid fault model and flexible quorum sizes, three new parameters that can take on a range of values get introduced. Equations 3.10 and 3.12 are hence derived to consider these new parameters for calculation of shard failure probability. We observe that the range of values of these new parameters gives us new combinations that influence the shard failure probability. Hence, by varying some parameters while keeping others constant, we can come up with different ways to shard a blockchain. In the future, more different use cases (i.e., combination of parameters) need to be investigated, which can lead to other interesting results and sharding sub-schemes. FlexiShard needs to be implemented in practise as well to compare against the theoretical estimations. Good and suitable heuristics need to furthermore be developed to differentiate between abc and Byzantine faults, and to estimate fault proportions as accurately as possible in a system implementing FlexiShard.

In Chapter 4, we look at relaxing the conservative ‘perfect’ shard assumption itself. This is done by splitting an original contemporary shard into k smaller fragments, assigning a block to these multiple smaller shards for validation, and introducing a central authority that requires 100% consensus to append a block with finality to the blockchain. The benefits are increased collective fault resiliency in a system and the capability of detecting potentially faulty blocks with high probability in near real-time (i.e., during transaction processing). Faulty block detection and increased collective resiliency can allow the usage of smaller shards which improves performance. Furthermore, some faulty shards could be allowed in a system, which is more practical and better suited to real-life sharded systems that are subjected to a large number of adversaries, such as blockchains.

Possible future work for SecurShard includes improving the bottleneck due to centralization of the Ref shard (i.e., the Reference committee) by performing one of the following: (1) having a distributed Ref shard approach, or (2) removing the Ref shard altogether and making the leader of each shard in a group agree on the same result to reach consensus. For the first approach, synchronization and load-balancing protocols should be in place for the distributed Ref shards. In the second approach of removing the central authority altogether, the use of an all-to-all quadratic protocol would result in a message complexity of $O(k^2)$ which is much smaller than the message

complexity of $O(m_{Ref}^2)$ of the Ref shard, since the value of k is usually much smaller than m_{Ref} . However, the use of BFT would require the number of correct Txp shards in a group to increase from one to at least $\frac{2k}{3}$ in partial synchrony, and hence indirectly result in the use of bigger Txp shards to obtain low failure probabilities. Hence, trade-offs would need to be made anyway.

Ultimately, the concepts of SecurShard and FlexiShard could both be combined to obtain even smaller, responsive and more fault tolerant shards with fault detection capabilities during transaction processing. This is ongoing research and the results will be reported in our future works.

5.1. Related Publications

The following publications are a direct culmination of this thesis:

- T. Ramburn and D. Goswami, "FlexiShard: a Flexible Sharding Scheme for Blockchain based on a Hybrid Fault Model," in 2022 21st IEEE International Symposium on Parallel and Distributed Computing (ISPDC), Basel, Switzerland, 2022. pp 9-17.
- T. Ramburn and D. Goswami, "SecurShard: a Model for Hierarchical Fault Detection in Blockchain Sharding," in 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Dubai, United Arab Emirates, 2023.

REFERENCES

- [1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," May 2009.
- [2] V. Buterin, "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform," 2013.
- [3] "A Deep Dive Into Blockchain Scalability," 3 January 2020. [Online]. Available: <https://crypto.com/university/blockchain-scalability>.
- [4] E. A. Brewer, "Lessons from giant-scale services," *IEEE Internet Computing*, vol. 5, pp. 46-55, 2001.
- [5] N. Anita and M. Vijayalakshmi, "Blockchain security attack: A brief survey," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2019.
- [6] Y. Liu, J. Liu, M. A. V. Salles, Z. Zhang, T. Li, B. Hu, F. Henglein and R. Lu, "Building Blocks of Sharding Blockchain Systems: Concepts, Approaches, and Open Problems," *Comput. Sci. Rev.*, vol. 46, p. 100513, 2021.
- [7] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, p. 382–401, July 1982.
- [8] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, USA, 1999.
- [9] L. Ren, K. Nayak, I. Abraham and S. Devadas, "Practical Synchronous Byzantine Consensus," *CoRR*, vol. abs/1704.02397, 2017.
- [10] M. Zamani, M. Movahedi and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2018.
- [11] T. Hanke, M. Movahedi and D. Williams, "DFINITY Technology Overview Series, Consensus System," *CoRR*, vol. abs/1805.04548, 2018.
- [12] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.
- [13] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2016.
- [14] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, p. 288–323, April 1988.

- [15] I. Abraham, "Synchrony, Asynchrony and Partial synchrony," 1 June 2019. [Online]. Available: <https://decentralizedthoughts.github.io/2019-06-01-2019-5-31-models/>.
- [16] R. Pass and E. Shi, "Hybrid Consensus: Efficient Consensus in the Permissionless Model," in *International Symposium on Distributed Computing*, 2017.
- [17] D. Malkhi, K. Nayak and L. Ren, "Flexible Byzantine Fault Tolerance," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2019.
- [18] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.
- [19] V. Buterin, "An Incomplete Guide to Rollups," January 2021. [Online]. Available: <https://vitalik.ca/general/2021/01/05/rollup.html>.
- [20] V. Buterin, *Why Sharding is great: Demystifying the technical properties*.
- [21] V. Buterin, "The Limits to Blockchain Scalability," May 2021. [Online]. Available: <https://vitalik.ca/general/2021/05/23/scaling.html>.
- [22] "HEDERA: A PUBLIC HASHGRAPH NETWORK & GOVERNING COUNCIL," 2018.
- [23] S. Popov, "The tangle".
- [24] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," *IEEE P2P 2013 Proceedings*, pp. 1-10, 2013.
- [25] Y. Sompolinsky and A. Zohar, "Secure High-Rate Transaction Processing in Bitcoin," in *Financial Cryptography*, 2015.
- [26] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, Vols. SE-3, pp. 125-143, 1977.
- [27] M. J. Fischer, N. A. Lynch and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, p. 374–382, April 1985.
- [28] H. Howard, D. Malkhi and A. Spiegelman, "Flexible Paxos: Quorum Intersection Revisited," *ArXiv*, vol. abs/1608.06696, 2016.
- [29] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems: First International Workshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*, 2002.
- [30] C. Cachin, K. Kursawe and V. Shoup, "Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography (Extended Abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, USA, 2000.

- [31] A. Kate and I. Goldberg, "Distributed Key Generation for the Internet," in *2009 29th IEEE International Conference on Distributed Computing Systems*, 2009.
- [32] J. Bonneau, J. Clark and S. Goldfeder, "On Bitcoin as a public randomness source," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.
- [33] I. Bentov, A. Gabizon and D. Zuckerman, "Bitcoin beacon," *arXiv preprint arXiv:1605.04559*, 2016.
- [34] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer and B. Ford, "Scalable Bias-Resistant Distributed Randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.
- [35] T. Ramburn and D. Goswami, "FlexiShard: a Flexible Sharding Scheme for Blockchain based on a Hybrid Fault Model," in *2022 21st International Symposium on Parallel and Distributed Computing (ISPDC)*, 2022.
- [36] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989.
- [37] B. Schoenmakers, "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, Berlin, 1999.
- [38] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *J. ACM*, vol. 36, p. 335–348, April 1989.
- [39] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of The Society for Industrial and Applied Mathematics*, vol. 8, pp. 300-304, 1960.
- [40] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.
- [41] S. Kantesariya and D. Goswami, "Determining Optimal Shard Size in a Hierarchical Blockchain Architecture," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020.
- [42] G. Karame, E. Androulaki and S. Capkun, "Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin," *IACR Cryptol. ePrint Arch.*, p. 248, 2012.
- [43] D. Malkhi and M. Reiter, "Byzantine Quorum Systems," *Distrib. Comput.*, vol. 11, p. 203–213, October 1998.
- [44] H.-S. Siu, Y.-H. Chin and W.-P. Yang, "A note on consensus on dual failure modes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 225-230, 1996.
- [45] M. Christ, V. Nikolaenko and J. Bonneau, "Leader Election from Randomness Beacons and Other Strategies," November 2021. [Online]. Available: <https://a16zcrypto.com/leader-election-from-randomness-beacons-and-other-strategies/>.

- [46] M. Zhang, J. Li, Z. Chen, H. Chen and X. Deng, "An Efficient and Robust Committee Structure for Sharding Blockchain," *IEEE Transactions on Cloud Computing*, pp. 1-14, 2022.
- [47] S. Sen and M. J. Freedman, "Commensal Cuckoo: Secure Group Partitioning for Large-Scale Services," *SIGOPS Oper. Syst. Rev.*, vol. 46, p. 33–39, February 2012.
- [48] B. Awerbuch and C. Scheideler, "Towards a Scalable and Robust DHT," in *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, New York, NY, USA, 2006.
- [49] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, New York, NY, USA, 2019.