Distributed Fintech Infrastructure Testbed

Zhe Wu

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2020

Abstract

Distributed ledger technology is facing challenges from rapidly increasing transactions in mobile scenarios. Since the existing distributed ledgers are designed for servers, the mobile financial services require the new high-performance decentralized transaction platform to adapt to distributed ledger based on mobile devices, which will enhance security and reduce overhead for transactions. The existing consensus mechanism of the blockchain ledger cannot match the resource limitations of hardware devices, so it is necessary to adopt a new consensus algorithm.

This project proposes a distributed fintech infrastructure testbed to evaluate and compare the performance of the Byzantine fault tolerance (BFT) consensus mechanism in a mobile ledger system. An Android app of the testbed is developed to measure the performance metrics of the practical Byzantine fault tolerance (PBFT) and Zyzzyva protocols in mobile networks. The project presents a comparison between PBFT and Zyzzyva in normal case and failure case. The result shows that Zyzzyva has higher throughput and lower latency when all replicas are healthy, while PBFT performs better in the unexpected environment with failures. The proposed testbed can be applied to evaluations of more various consensus mechanism for the further optimization of the distributed ledger in mobile scenarios.

Acknowledgements

I would like to express my heartfelt gratitude to Dr Tiejun Ma and Dr Xiao Chen for their constant encouragement and guidance. They have walked me through all the stages of the writing of this dissertation. Without their consistent and illuminating instruction, this dissertation could not have reached its present form.

Besides, I would like to deeply indebted to everyone who help me during my study in Edinburgh. I appreciate them so much for spending the special and memorable year with me.

Table of Contents

Chapter 1

Introduction

1.1 Objectives

With the expansion of business transaction scenarios and the development of network communications, traditional financial services have taken advantages of mobile fintech infrastructures to achieve considerable growth [1]. The blockchain and distributed ledger are increasingly being applied in e-commerce, supply chain, and e-government transactions due to its traceable, unalterable, and decentralized characters [2][3]. However, the efficiency of the consensus mechanism and the limitation of equipment resources determine the system performance in transactions processing [4][5][6][7].

The project proposes a distributed fintech infrastructure testbed to test and evaluate the performance of mobile distributed networks with various consensus mechanisms. Two typical Byzantine fault tolerance BFT algorithms, PBFT and Zyzzyva, are run on the testbed to assess their robustness, throughput and delay. The testbed will associate fintech infrastructure such as distributed ledger with improving performance by switching appropriate consensus protocols.

1.2 Problem Statement

Mounting evidences suggested that the traditional centralized procedure has brought security risks, including data manipulation and node failures. Thus, the traditional financial transactions require the intermediary platform as the guarantee, which significantly increases the cost in transactions [8]. The latest transaction formats based on distributed ledger discarded the intermediary services, but they still face attacks from the network [9].

Besides, the rapid increase of transaction amount and popularity of mobile transaction require a mobile portable distributed fintech infrastructure, whose network support mobile nodes. The existing distributed ledger system cannot match the hardware condition of a mobile device, which only provides very limited computing resource and low power consumption [10][11][12]. The mobile transaction system requires to apply the most appropriate consensus mechanism to strike a balence between the hardware limitation and the high performance requirement. As a result, the testbed of mobile distributed fintech infrastructure is necessary to qualify and optimize the capacity of transaction systems for practical financial applications [13][14].

1.3 Our Contributions

The proposed distributed fintech infrastructure testbed (DFIT) in the project would support mobile nodes and clients to evaluate the performance of distributed fintech infrastructure and how their capabilities of risk elimination and processing efficiency.

The project extends the distributed ledger system to the mobile device, which is not only the client but the replica node to validate transactions and store data. It will broaden the distributed ledger to mobile scenarios for more complex application. The existing blockchains with the proof-of-X consensus algorithm are inefficient and lacks fairness and security for mobile scenarios, which is presented in section 2.3 in details. The proposed testbed focuses on the faster and reliable BFT algorithms, which are optimized further to adapt the mobile devices by trade-off between the limited resources and performance. The testbed constructs consortium blockchain network with PBFT and Zyzzyva protocols. It compares the two consensus algorithms in various mobile environments, which are simulated by changing the failed replicas, the request frequency, the string size of messages, and the frequency of failure. Our testbed measures the throughput and latency of transactions in normal case and failure case to evaluate the performance of the consensus algorithm in different mobile scenarios. The evaluation experiment shows the comparison result and the applicability of the mobile ledger system based on PBFT and Zyzzyva.

1.4 Organization

The rest of the dissertation is organized as follow. Chapter 2 introduces key concepts and challenges about the distributed ledger, followed by the model of distributed

fintech infrastructure testbed Chapter 3. Chapter 4 introduces the implementation details. Chapter 5 presents the performance evaluation experiment result and analysis, followed by conclusions in Chapter 6.

The Appendix A collates the used abbreviations in the dissertation.

Chapter 2

Related Work

The rapid growth of mobile financial service market is facing challenges from limited performance of fintech infrastructures. Meanwhile, the distributed ledger, blockchain, and consensus algorithm have been applied in the many financial platforms, such as digital currency, supply chain management tools and asset transaction settlements [15][16]. The chapter shows the necessary concepts and the status quo about the latest financial technologies application in mobile finance.

2.1 Distributed Ledger

The distributed ledger is a shared, replicated and synchronised database system among network nodes to record transactions between users [17][18]. It is a transparent, traceable, and unalterable database to record data in the decentralised network. It is applied in the trustless transactions without the participation of guarantee agency such as the bank [19]. A distributed ledger is constructed from data blocks, consensus mechanism, cryptographic algorithm, network routing, and running scripts.

2.1.1 Blockchain

The distributed ledger is based on blockchain, an event-driven stateful protocol for processing and custody of digital assets [20][21]. A block, the component of blockchain, consists of a header and a body shown in Figure 2.1 [22]. The block body stores transaction data, while the block header contains the following components: (1) the previous block hash, (2) the block height, (3) a nonce, (4) the Merkel root of block body, and (5) a timestamp [23]. The blockchain uses Merkel root, a hash value calculated

from block body, to verify the data correctness [24].

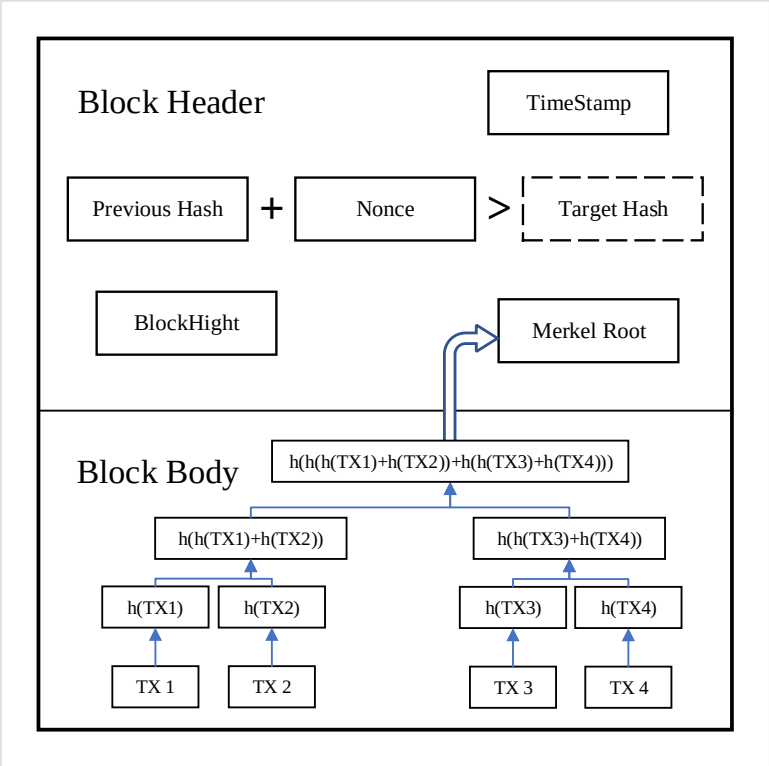


Figure 2.1: Structure of the data block

After being requested, a transaction would be packaged into blocks, and then multicast to all nodes for confirmation [25]. The data of valid transaction will be written into a new block to validate the transaction. Repeating procedure would generate a blockchain in Figure 2.2 to record all the transaction history.

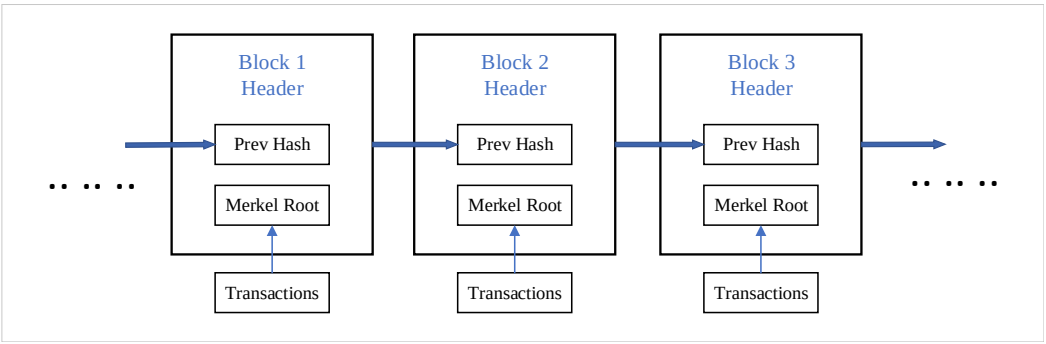


Figure 2.2: Example of blockchain structure

There are three blockchain systems summarised in Table 2.1. Because of their different properties, their applications are determined by the practical scenarios [26]. The public blockchain has transparent transactions records, whose replicas is unalterable. The public blockchain allows the external nodes joining freely, which leads the distributed system to spend more time to reach a consensus, so the system has a low throughput. The private blockchain is usually applied for in-company distributed system. It has strict access permission [27], but it allows administrators to customise a flexible consensus protocol to improve the data updating efficiency. Thus, it is supposed to be a more centralised system than the public blockchain, which has less security in case of transaction in multiple participants [28].

The distributed ledger applied in DFIT is the consortium blockchain, whose nodes are jointly managed by several independent participants [29]. It combines characteristics of public chain and private chain [30]. The consortium blockchain has the access permission, but its consensus mechanism is more flexible than public chains, whose efficiency is between the private blockchain and the public blockchain. It is suitable for the shared infrastructure for transactions in multiple participants.

Property	**Public blockchain**	**Consortium blockchain**	**Private blockchain**
Consensus determination	All miners	Nodes from multi-organizations	One organization
Permission	Public	Could be public or restricted	Restricted
Immutability	Hard to tamper	Possible to tampered	Possible to tampered
Efficiency	Low	High	High
Centralized	No	Partial	Yes

Table 2.1: Summary of different blockchain property comparison

2.1.2 Distributed Ledger Properties

The work of a distributed system requires cooperation between nodes, but the cooperation face challenges from uncertain factors of heterogeneous machines, crash nodes, malicious nodes, and unreliable networks [31]. An exemplary distributed network should have better properties as follow [32].

- *Scalability* - capacity to handle the growth as the number of users for the system increases.
- *Reliability* - ability to keep that nodes compute correctly, and the stored data will not be lost.

- *Availability* - the ratio of system uptime to total running time
- *Consistency* - agreement between multiple nodes in a distributed system to achieve a certain value
- *High performance* - ability to achieve high throughput and low latency.
- *Partition fault tolerance* - capacity to keep available services even if a part of the system has failed

All independent nodes should reach a consensus on a proposal within a limited time in the distributed ledger. However, any distributed system cannot simultaneously ensure strong CAP characteristics, consistency, availability, and partition fault tolerance [33][34]. Therefore, one of the CAP characteristics had to be weakened when designing the system.

To be a well-performing distributed system, DFIT should return consistent results if most of the machines work normally. Besides, the performance of the system can be scaled horizontally, and each node works asynchronously. It will maintain the consistency and availability with about 1/3 fault-tolerant.

2.2 Consensus Mechanism

2.2.1 Proof-of-X

2.2.1.1 Proof-of-work

The proof-of-work (PoW) is an economic countermeasure against malicious alter ledger, which requires users to perform a time-consuming computing task to prove their honesty [35][36][37]. The most commonly used algorithm for proof of work is the hash function because of its one-way process [38]. The hash function almost impossible to allow to derive a input value from its hash, so participants have to perform myriad exhaustive calculations to achieve proof of work [39]. The PoW blockchain usually assume that the longest chain as the legal chain to make nodes accept blocks from other nodes [20]. If a node refuses to accept the blocks that others legally add, it has to create more blocks to make its chain the longest, which is very expensive. The attacker must have more than half of the computing power of the entire network (“51% attack”), which makes the attack very expensive [40]. The PoW incentivize nodes adding blocks

by issuing tokens. The participants who has the large computing power tend to keep system health to prevent tokens devaluation [41].

However, PoW is not suitable as a shared financial infrastructure because it wastes a great amount of computing resources and energy on the proof. For instance, all Bitcoin miners spends power of 54TKWh on mining, the power consumption of entire New Zealand [42]. Besides, if the computing power is monopolised with appearance of large "mining pools", the PoW would lose its advantage of decentration.

2.2.1.2 Proof-of-stake

The proof-of-stake (PoS) is a method of pseudo-random election to select the node to confirms the transaction, which considers the time length of token existence, asset status, and randomness [43][44][45]. PoS requires participants to obtain and use the stake as a security deposit to stand on the side of the whole system interest. Participants will invest stake, which usually refer to the tokens or coin age, to the valid blocks, and the participants who choose the block with the most supporters will be rewarded. It is anti-malicious since any illegal behaviour leads to risks to lose their stake.

Although PoS is faster and economical than PoW due to its less consumption of the computing power, it is not supposed as an appropriate consensus mechanism in this project, because the rich have the dominant power to validate transactions, and the poor tend to break the system [46]. In PoS, the wealthy gain more probability to keep accounts, so the rich concerns about system health to protect their tokens, while the small users are less motivated to maintain the system. When the users with a small number of tokens are able to add the block, they might validate self-interest transactions ("tragedy of the commons"), such as allocating themselves new tokens. As a consequence, the blockchain bifurcation from the illegal transaction will reduce the value of tokens, hinder normal transactions, and damage interests of other users.

2.2.1.3 Delegated Proof-of-stake

The delegated proof-of-stake (DPoS) is a voting-based consensus algorithm from PoS, while participants of DPoS will select trustee nodes to validate transactions [47][48]. The trustee will confirm and package the transaction, and then broadcasts it to other replicas. The trustee should ensure the high performance and security of the system, and the incompetent trustee node would be voted out. Compared with PoS, DPoS improves security and DPoS does not require as much calculation as PoW. However, the

DPoS mechanism creates super nodes (trustee), which makes the system centralised.

2.2.2 Byzantine Fault Tolerant Algorithm

2.2.2.1 Byzantine Generals Problem

The concept of BFT is put forward to evaluate the system resistibility to nodes with unexpected behaviour. A challenge of distributed ledgers is the requirement of the consistency which is influenced by following factors, (1) node crash, (2) network communication failure or long delay, (3) difference in operating speed of nodes, or (4) malicious nodes destroy consensus [49].

In 1982, L. Lamport described the distributed consistency problem as the Byzantine generals problem [50]. The geographically isolated Byzantine generals (nodes) were making decisions on whether to attack their enemy, and they could communicate through messengers (communication channels). Since traitors in the generals (Byzantine nodes) would send random messages to interfere with decision-making, the generals (nodes) needed a mechanism to take the consistent action. Both crash nodes and Byzantine nodes should be tolerant in the model. It is verified that a distributed system should have more than 3f generals to tolerate f traitors [50][51]. Many BFT algorithms are designed to deal with the Byzantine generals problem, which is usually based on the value of the majority of the decisions a lieutenant observes [52], such as PBFT and Zyzzyva.

2.2.2.2 Practical Byzantine Fault Tolerance

The practical Byzantine fault tolerance (PBFT) algorithm is one of the best consensus algorithms for mobile consortium network. PBFT maintains liveness for less than $(N-1)/3$ Byzantine replicas with complexity less than origin BFT algorithm. [53][54][55]. When n nodes in the system, including up to f crash failure and f Byzantine nodes, the system will tolerate fault when there are more normal nodes than the Byzantine nodes since crash nodes do not respond as the equation 2.1 [50].

$$\begin{aligned} N - 2f &> f \\ f &\leq \frac{N-1}{3} \end{aligned} \tag{2.1}$$

PBFT provides better efficiency and fairness than Proof-of-X algorithm, because its security and stableness are based on decisions of replicas instead of expensive computing or stake certificate. Thus, PBFT dispenses with any incentive mechanism and

tokens. It has recently been applied in IBM Fabric, digital currency electronic payment of the People's Bank of China, and Tencent TrustSQL database.

This project would take PBFT as a base to implement a mobile fintech infrastructure system.

2.2.2.3 Zyzzyva

Zyzzyva is a scalable BFT algorithm with a simpler and faster consensus protocol [56]. Like the PBFT, Zyzzyva could tolerate up to f Byzantine nodes in a $3f+1$ nodes network. It allows to speculative operation by simplifying redundant communications and validations in no fault cases. Zyzzyva depends on the client to collect responses from replica nodes and detect the inconsistent result [57]. If the client got responses from all of replicas, it asserts the transaction has been executed; otherwise, it would activate a commit phase of protocol to associate replicas reach consensus. Zyzzyva protocol is based on a state machine replication protocol. The replicas would keep consistence by processing transactions in order and updating history. When a new or interrupted replica has detected the inconsistent history, it would fill the lost transactions with assistance from other replicas, thereby making whole system in the consistent state. It is supposed to be one of the best existing BFT for consortium blockchain.

2.3 Challenges of Mobile Fintech: A Chance for BFT

The fintech has provided economical efficient platform to improve user experience of financial service, but its security problem on transaction data is widely focused [58][59]. A existing empirical study presented that only 35% companies and costumers are confident about the technical security in transactions, which has led to the one of the most significant barriers to financial business growth [60]. Additionally, users in a transaction do not trust each other because of their different value orientations in most cases. Thus, the transactions must be endorsed by independent intermediary platform, such as PayPal. However, cost of transactions through the guarantee agency usually be expensive, and this cannot completely eliminate users' doubts about the guarantee company's reliability.

The distributed ledger with blockchain technology is supposed to be a feasible solution to security and trustlessness issues, which become popular recently [61]. The most existing distributed ledgers are based on server nodes, and only some of them

have mobile clients. The system cannot benefit mobile users because they have to collocate their transaction on server of trustee. Thus, the swift growth of mobile transactions requires distributed ledger system based on mobile nodes.

Nevertheless, the existing distributed ledger systems are inadequate for nodes on mobile devices [62]. Most of distributed ledger are running blockchain based on PoW [63]. A mobile node will spend a large amount of computing resources to choose the node to add new block. This procedure of proof reduces the performance of other app on mobile device and causes the faster resource collection cycle of the operating system. In worse cases, collection to active processes will degrade the user experience severely. Moreover, a large amount of power consumption reduces the limited battery life, and frequent charging influences the portability. The time cost for PoW reduces the efficiency of processing transactions.

Besides, the unbalanced computing power in the PoW system brings security risks as well. In a distributed network composed of servers and mobile devices, servers are considered to have more computing resources than mobile devices. Therefore, it is easier for server nodes to add block, which is unfair to mobile device users and has risk of 51% attack. Compared with proof-of-X mechanism, the BFT is more friendly to mobile users. It has a higher speed, lower resources consumption because it abandons the proof by computing tasks, which also extend the battery life. BFT is fairer because it free mobile devices from the competition with servers, which also improve the decentralized level [64]. Therefore, the proposed testbed takes the BFT algorithm to keep consistency.

Chapter 3

Architecture of Distributed Fintech Infrastructure

3.1 Distributed Fintech Infrastructure System

The proposed distributed fintech infrastructure testbed system is a peer-to-peer network composed of clients and nodes, which contains the mobile devices, personal computers and servers (Figure 3.1). The client in DFIT is a software to submit transaction requests and receive operation result. The node in DFIT is a database of the distributed system that executes requested operation and stores transaction data. According to practical scenarios, DFIT software can run both the client and node at the same time, or any one of them. All nodes in DFIT can serve as routers to forward messages, which support system fault tolerance because of no inexistence of irreplaceable super node.

As a shared fintech infrastructure, DFIT is designed to the consortium blockchain system to keep secure and decentralized system [65]. DFIT applies the BFT consensus mechanisms, PBFT and Zyzzyva to ensure the consistency of the ledger. Due to different device performance, the proposed system will be optimised to adapt to the resources and power consumption of mobile devices [66][67].

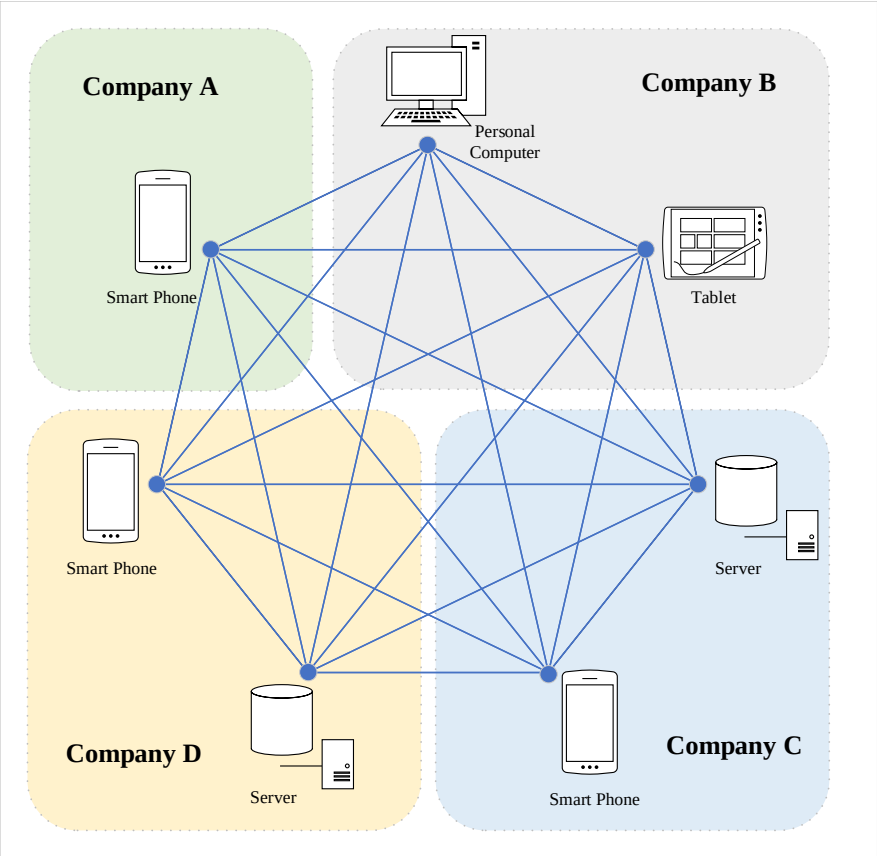


Figure 3.1: Distributed fintech infrastructure system

In DFIT, a transaction is defined as an asset transferring event between users. Imagine a practical mobile payment scenario. Alice, a cafe owner, wants to buy ten pounds of coffee beans from a coffee bean supplier Bob. They decide to use the digital wallet (client) connected to DFIT for transactions. The digital wallet obtains account balance information by reading the data in ledger. Figure 3.2 shows how the transaction executed in DFIT. To start a transaction, Alice will send a request to a specific primary node from her client. Then, the request will be multicast to other nodes. These nodes will add the valid transaction into their log and execute the operation in the request. By consensus mechanism, the operation result will be returned to the client. The procedure would not require any guarantee service.

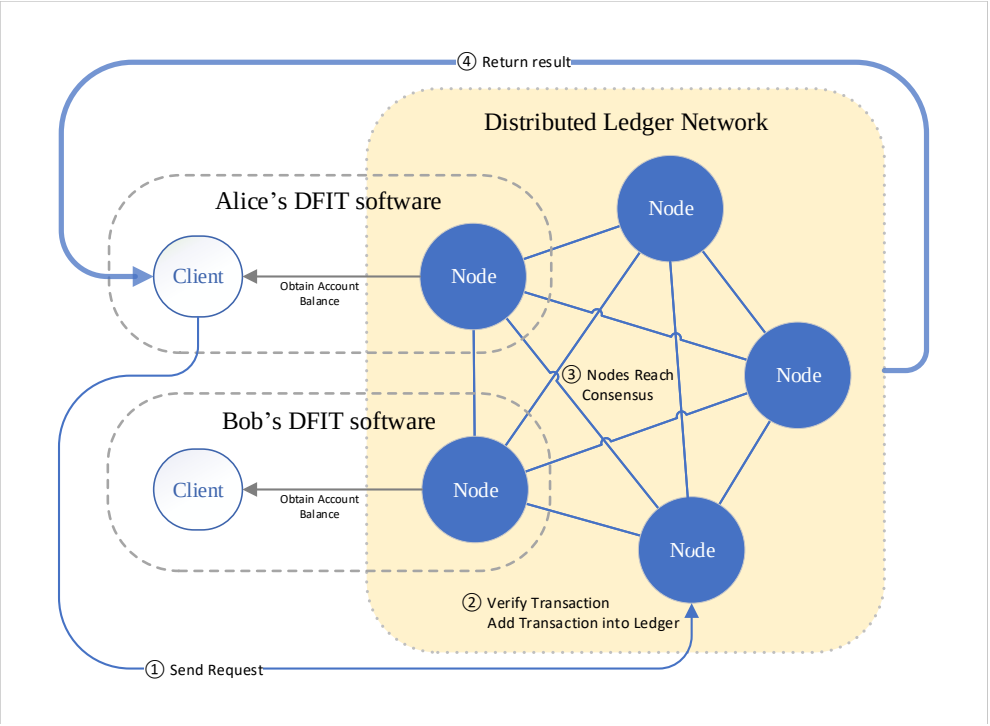


Figure 3.2: Transaction example in distributed fintech infrastructure

3.2 System Model

The testbed hypothesises that the distributed network consists of n replicas and a finite number of clients. No more than f replicas are Byzantine replicas, which exhibit arbitrary behaviour. The replicas transmit data by sending messages with ECC Schnorr signature. The hypothetical network attacker would control Byzantine replicas to crash, interrupt communication, messaging incorrect data, and make finite network delay. The system model assume that the attacker cannot break the system from cryptography, i.e. learning others' privacy key, forging signatures, and crack the collision-resistance hash.

The system maintains the liveness and security in case of $f \leq (n-1)/3$. When the client request an illegal transaction, the normal replicas should refuse the request consistently and return error code. To keep the liveness, a correct transaction is processed in finite time by correct replicas, and the system will give the correct result in case of finite delay in normal replicas communication.

The proposed testbed will test asynchronous ledger system for larger amount of

requests. Any request received by replicas is stored in buffer to join a task queue, and they will be processed in FCFS scheduling.

Chapter 4

System Implementation

Table 4.1 presents the technology stack for DFIT from top to bottom [68][69][70]. It consists of four layers. The data layer implements the data storage, message assembling and asymmetric encryption, which is followed by the network layer that presents the data transmission protocol for DFIT. The PBFT and Zyzzyva are implemented in the consensus layer to keep consistency and resist replica failures. Finally, the application layer is an interface for users to issue transaction requests and receive results.

Application layer	Android App			
Consensus Layer	PBFT	Zyzzyva		
Network Layer	P2P Networking	TCP/IP	Message Verification	
Data Layer	Data Storage	Message Assembling	Digest Technology	ECSchnorr Signature

Table 4.1: Technology slack for DFIT

4.1 Consensus Mechanism

The protocols of PBFT and Zyzzyva aim to process transactions consistently for the hypothesised model in Chapter 6. The distributed network contains N replicas, donated as $\{0,1,..,n\}$. The replica to multicast request is called the primary p which calculated from the system view v, $p = v \ mod \ N$. Other replicas are backups. View v can increment consecutively when view-change condition is activated.

The consensus mechanisms have been adjusted appropriately for mobile application. They have narrower watermarks and faster garbage collection than servers consensus. The message size in the mobile system is compressed for less occupancy of memory.

4.1.1 PBFT

The PBFT depends on the primary replica to forward the transaction to backup replicas [71]. It has a normal case 3-phase protocol, garbage collection mechanism, and view-change protocol. When the primary works in the normal case, the system would reach consensus by pre-prepare, prepare, and commit phases (Figure 4.1). If the primary cannot work to deal with the request, the backup replicas would start view-change to appoint a new primary in a round-robin manner.

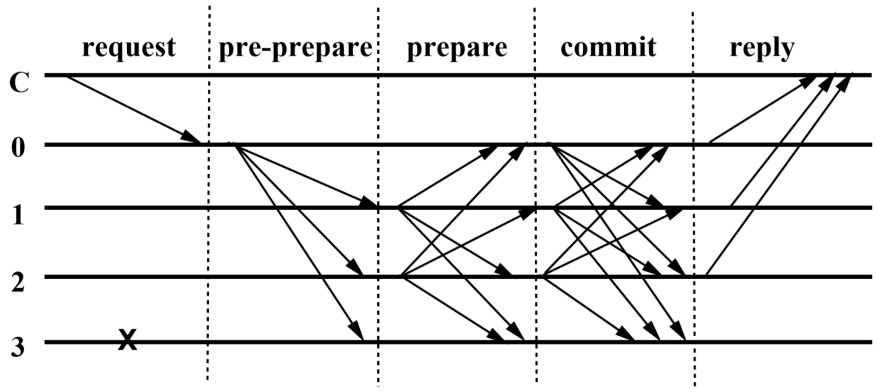


Figure 4.1: PBFT message flow in normal case [53]

Beginning

The client c sends a request message $< REQUEST, o, t, c >_{\sigma_c}$ of transaction to the primary p, where the o, t, c and σ_c are the operation, time seal, client and signature of client respectively. Meanwhile, it starts a timer to detect the disabled primary case.

Consensus Procedure

1) Normal Case: 3-phase Protocol

After receiving the request, The primary will assign a consecutive sequence number n to tab the valid request. There are pre-prepare, prepare and commit phase in normal case, which are shown in Table 4.2.

Phase	Node	Event	Behaviour	Message Assembling
Pre-prepare	**Primary**	Receive a request $<REQUEST, o, t, c>\sigma_c$ from the client	**Validate condition:** σ_c is correct. **if condition true then** Assign a consecutive sequence number n to tab the request; Multicast the pre-prepare message to backups and add it into local log; Enter prepare phase; **else** Discard the request; **end**	$<<PRE\text{-}PREPARE,v,n,d>\sigma_p, m>$ v : current view n : sequence number of the request d : digest of the request σ_p: signature of the primary p for pre-prepare message m : the request message
	Backup	Receive a pre-prepare message $<<PRE\text{-}PREPARE, v, n, d>\sigma_p, m>$ from the primary	**Validate condition:** (1) σ_p and d are correct, (2) v is the current view, (3) the request n has not been received in the current view, (4) n is between water marks, a reasonable range of n. **if conditions true then** Multicast the prepare message to other replicas; Add the received pre-prepare and the sent prepare message into local log; Enter prepare phase; **else** Discard the pre-prepare message **end**	$<PREPARE, v, n, d, i>\sigma_i$ v : current view n : sequence number of the request d : digest of the request i : local replica σ_i: signature of local replica i
Prepare	**Replica**	Receive a prepare message $<PREPARE, v, n, d, i>\sigma i$ from a replica	**Validate condition:** (1) σi is correct, (2) v is the current view, (3)n is between water marks. **if conditions true then** Store the prepare message into local log; **else** Discard the prepare message; **end**	
		Has received $2f$ valid prepare messages from different replicas	Predicade *prepared(m,v,n,i)* is true; Multicast commit message to other replicas; Enter commit phase;	$<COMMIT, v, n, D(m), i>sigi$ v : current view n : sequence number of the request $D(m)$: the digest of m in pre-prepare message of the request i : local replica σ_i: signature of local replica i
Commit	**Replica**	Receive a prepare message $<COMMIT, v, n, D(m), i>\sigma_i$ from a replica	**Validation Condition:** (1) σ_i and $D(m)$ are correct, (2) v is the current view, (3) n is between water marks. **if conditions ture then** Append the commit message into local log; **else:** Discard the commit message; **end**	
		Has received $2f$ valid prepare messages from other different replicas	Predicade *committed-local(m,v,n,i)* is true; Execute the requested operation; Return operation result to the client by reply message, and append the reply into local log;	$<REPLY, v, t, c, i, r>\sigma_i$ v : the current view t : time seal of the request c : client i : local replica i r : result of the operation σ_i: signature of local replica i

Table 4.2: PBFT normal case protocol

Finally, the client will cancel the timer and affirm the result of the request, when it

has collected $2f+1$ valid replies with the consistent view v and result r from various replicas.

2) Disabled Primary Case: View-change

If the client does not receive any reply from replicas before the timer expires, it multicasts the request message to all the replicas for replies. As Table 4.3 The backup received request will check the liveness of the primary, and start view-change when the primary is incompetent.

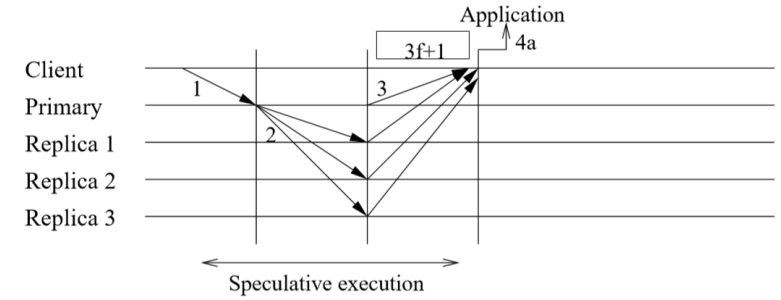
Phase	Node	Event	Behaviour	Message Assembling
View-Change Start	**Backup**	Receive a request *<REQUEST, o, t, c>*σ_c from the client	**if** σ_c **is correct then** **if request was executed then** resend the reply message from the local log; **else** forward the request to the primary; start a timer; **if a pre-prepare message is received before timer expiring then** stop the timer; start 3-phase protocol; **else** multicast view-change message to other replicas; enter view-change execution; **else** Discard the request;	*<VIEW-CHANGE,v+1,n,C,P,i>*σ_i $v+1$: the new view, n: sequence number of the request, C: checkpoint message set, P: the existing pre-prepare and prepare messages, i: local replica i, σ_i: signature of the local replica i
View-Change Execution	**New Primary**	Has received $2f$ valid view-change messages from different replicas	Multicast the new-view message; Change view to $v+1$;	*<New-View,v+1,V,O>*$\sigma_{p'}$ $v+1$: new view V: set of 2f+1 view-change messages, including 2f from others and one from the local, O: set of received pre-prepare and prepare messages in log, $\sigma_{p'}$: signature of the new primary,
	Backup	Receive the new-view message *<NEW-VIEW, v+1, V, O>*$\sigma_{p'}$ from the new primary	**Validate condition:** (1) $\sigma_{p'}$ is correct, (2) $v+1$ equals current view+1, (3) messages in O are consistent with the local log. **if conditions true then** Continue as 3-phase protocol according to the messages in O; Change view to $v+1$;	

Table 4.3: PBFT view-change protocol

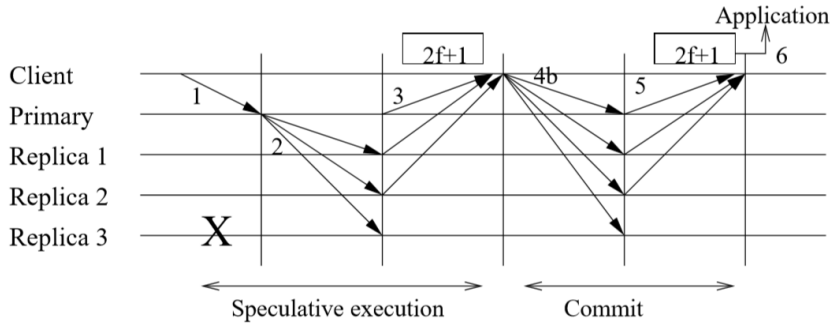
After view-change, the replicas will continue to process request in new view and reply request. If the new primary is not live as well, the view-change protocol will be run again until turning to a qualified primary.

4.1.2 Zyzzyva

Zyzzyva takes the speculative execution for the faster consensus in normal case [56]. The client checks if the speculative execution (Figure 4.2 (a)) covers all replicas; otherwise it will start to commit phase to ensure at least $2f+1$ normal replicas commit the transaction (Figure 4.2(b)). Replicas will chase up the application state by the fill-hole phase when they receive a message of unexpected state. If any conflict or no response happens in the system, the view-change will be started to generate the new primary [72].



(a) Fast execution



(b) Commit phase

Figure 4.2: Zyzzyva message flow [56]

Beginning

Zyzzyva protocol has the same beginning as PBFT, that the client sends a request $< REQUEST, o, t, c >_{\sigma_c}$ to the primary and waits for replies. It will start a timer to detect the liveness of the primary and backups for view-change phase and commit phase.

Consensus Procedure

1) Case 1: Normal Replicas

Replicas will execute operation for valid request speculatively Zyzzyva. As Table 4.4, the speculative execution phase makes request processed simpler than PBFT.

Phase	Node	Event	Behaviour	Message Assembling
Speculative Excution	**Primary**	Receive a request *<REQUEST,o, t, c>*σ_c from the client	**Validation condition:** σ_c is correct **if condition true then** Assign a consecutive sequence number *n* to tab the valid request, and update the maxn; Multicast the order-req message to backups; Execute the requested operation speculatively. **else** Discard the request	*<ORDER-REQ,v,n,*h_n*,d,ND>*σ_p*,m>* *v* :current view *n* :sequence number of the request hn:current history, hn = Digest(h_n*-1* , *m*) *d* : digest of the request *m* *ND* : the undtermined value for the operation σp : signature of the primary *m* : request message
	Backup	Receive a order-req message *<ORDER-REQ, v, n,* h_n*, d, ND>*σ_p*, m>* from the primary	**Validate condition:** (1)σ_c, h_n and *d* are correct (2)*v* is current view **if condition true then** **if *n* = max*n* +1 then** Execute requested operation speculatively; Update local history and max*n*; Send spec-response to the client, and add the order-req into the local log; **else if *n* > max*n* +1 then** Enter fill-hole phase **else:** Discard the order-req message **else:** Discard the order-req message	*<<SPEC-RESPONSE,v,n,*h_n*,H(r),c,t>*σ_i*,* *r,OR>* *v* :current view *n* :sequence number of the request, h_n :current history, h_n = Digest(h_n*-1* , *m*) *H(r)* : digest of the request *c* :client *t* : time seal of the request, σ_i : signature of the local replica *i* *r* : operation result *OR* : order-req message

Table 4.4: Zyzzyva speclative execution phase

If the client receives $3f+1$ valid replies with consistent result, it can affirm the operation result of the whole system. It will finish the transaction and cancel the timer.

While a replica receives a transaction with a sequence number higher than the expected next sequence number, $max_n + 1$, it will try to fill the lost transaction by fill-hole phase as Table 4.5 until it achieves the consistent state with other replicas.

Phase	Node	Event	Behaviour	Message Assembling
Fill-Hole	**Replica**	Receive a valid request with sequence number larger than the local max_n +1	Send the fill-hole message to the primary; Start a timer 1 to wait for the corresponding order-req from the primary; **If receiving filled oreder-req before timer 1 expiring then** Stop the timer; Start speculative execution; **else** Multicast the fill-hole message to other replicas; Start timer 2; **If receiving filled oreder-req before timer 2 expiring then** Stop timer 2; Start speculative execution; **else** Start view-change protocol;	$<FILL\text{-}HOLE,v,maxn+1,n,i>\sigma i$ v : current view max_n+1: sequence number of the request after the last received request n : sequence number of the lastest request i : local replica σ_i : signature of the local replica i
	Primary	Receive a fill-hole message $<FILL\text{-}HOLE, v, k,n,i>\sigma_i$ from a replica	**Validate condition:** (1)σ_i is correct, (2)v is current view. **if condition true then** Resend the order-req of requests from k to n ; **else** Discard the fill-hole message;	$<ORDER\text{-}REQ,v,n',h_{n'},d,ND>\sigma_p,m'>$ v : current view n' : sequence number of the filled request $h_{n'}$: filled history,hn = Digest(h_{n-1}, m) d : digest of the filled request m ND : undtermined value for the operation σ_p : signature of the primary m' : filled request message
	Replica	Receive a fill-hole message $<FILL\text{-}HOLE, v, k,n,i>\sigma_i$ from another replica	**Validate condition:** (1)σ_i is correct, (2)v is current view, (3)order-req of k exists in log. **if condition true then** Send the order-req of request from k to n **else** Discard the fill-hole message	$<ORDER\text{-}REQ,v,n',h_{n'},d,ND>\sigma_p,m'>$ v : current view n' : sequence number of the filled request $h_{n'}$: filled history, $h_{n'}$ = Digest(h_{n-1}, m) d : digest of the filled request m ND : undetermined value for the operation σ_p : signature of the primary m' : filled request message

Table 4.5: Zyzzyva fill-hole phase

2) Case 2: Disabled Backups

If the client receives between $2f$ and $3f+1$ consistent sepc-responses before timer expires, it will start the commit phase as Table 4.6.

Phase	Node	Event	Behaviour	Message Assembling
Commit	**Replica**	Receive a commit $<COMMIT, c, CC>\sigma_c$ from the client	**if σ_c is correct then** **if *CC* contains local history in current view then** Send local-commit to the client **if sequence number of request in *CC* is higher than local max_n then** Update maxn and add *CC* into local log **else** Enter fill-hole phase **else** Discard the commit message	$<LOCAL\text{-}COMMIT,v,d,h,i,c>\sigma_i$ v : current view d : digest of the request h : local history i : local replica c : client σ_i : signature of the local replica i

Table 4.6: Zyzzyva commit phase

When the client has collected $2f+1$ valid commit message for the same result, it can affirm the result for the transaction.

3) Case 3: The Disabled Primary

If the client receives less than $2f$ spec-response, it will multicast the request for view-change. The Zyzzyva view-change protocol is presented in Table 4.7.

Phase	Node	Event	Behaviour	Assembled Message
View-change Start	**Backup**	Timer for view-change expires, or receive a POM message	Multicast I-HATE-THE-PRIMARY message to other replicas and add one into the local log;	*<I-HATE-THE-PRIMARY,v>* σ_i v: current view σi: signature of local replica
	Backup	Receive *<I-HATE-THE-PRIMARY,v>* σ_i	**Validate condition:** (1)σi is correct, (2)v is current view. **if condition ture then** Add I-HATE-THE-PRIMARY into log; **if there are f+1 I-HATE-THE-PRIMARY messages in log then** Multicast view-change message; Start a timer to wait for new-view message; **if the timer expires before receiving new-view then** enter view-change excution phase; **else** start view-change protocol to v+2;	*<VIEW-CHANGE,v+1,CC,O,i>* σ_i $v+1$: next view CC: latest commit certificate O: order-req history i: local replica σ_i: signature of the local replica i
View-change Execution	**New Primary**	Has received 2f+1 valid view-change messages from different replicas, including one from the local	Multicast new-view message;	*<NEW-VIEW,v+1, P>*$\sigma_{p'}$ $v+1$: next view P: set of $2f+1$ view-change messages, including the one from the local $\sigma_{p'}$: signature of the new primary p'
	Replica	Receive new-view message *<NEW-VIEW,v+1, P>*$\sigma_{p'}$	**Validate condition:** (1)$\sigma_{p'}$ is correct, (2)v is current view. **if condition ture then** Stop timer for new-view; Multicast new-confirm message;	*<VIEW-CONFIRM,v+1,n,h,i>* σ_i $v+1$: new view n: being processed request before view-change h: the current history i: local replica σ_i: signature of the local replica i
	Replica	Receive view-confirm message *<VIEW-CONFIRM,v+1,n,h,i>*σ_i	**Validate condition:** (1)σ_i is correct, (2)$v+1$ is current view+1, (3)h is as same as local history. **if condition ture then** Save view-confirm in log; **if receiving *2f+1* view-confirm in log including one from the local then** Change view to $v+1$; **else** Discard the message;	

Table 4.7: Zyzzyva view-change protocol

4.2 Network Communication

The DFIT applies TCP/IP protocol for network communication, which can detect availability of communication channel by three-way handshaking before data transmission [73]. TCP/IP is independent of the hardware and operating system, which allows the DFIT network consisting of mobile phones, personal computers, and servers. Besides, the loss of data in package transmission should be avoided as far as possible because the reliability of financial data transmission is essential for users' transaction. Although the consensus mechanism can tolerate data error or loss partly, the retransmission and committing increase system overhead. TCP/IP protocol consider the liveness stipulates the correctness of data in transmission by checksum and acknowledgement [74].

The short connection mode is adopted in DFIT, which means the devices will disconnect connection and release resources immediately once they complete a message transmission. Many server networks apply the long connection which keeps live threads for each connection channel, but it is not practical for mobile devices. Maintaining the same number of live connections as devices in a network is a challenge to the limited hardware resources. The number of live threads is reduced in short connection mode to optimize the occupancy of resource, which reduces crash in DFIT.

4.3 Signature and Verification

Before devices sending messages, a signature generated by the asymmetric encryption mechanism is appended at the end of each message. The signature certificates authenticity of the message and its sender by private key encryption and corresponding public key verification [75]. DFIT supports to generate and verify the elliptic curve Schnorr (ECSchnorr) signature in messages, which is supposed to be an efficient ECC algorithm with high security [76].

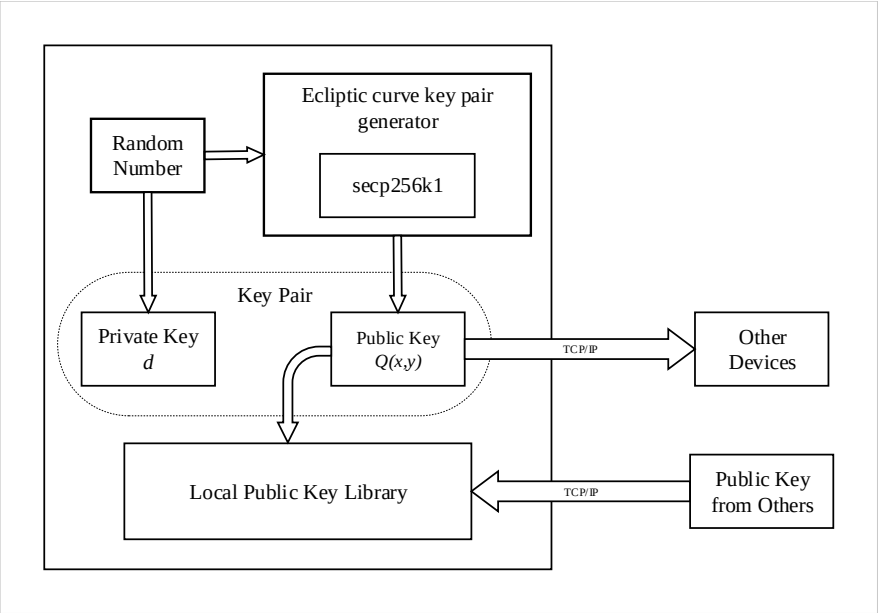


Figure 4.3: Key library and key pair generator

When the DFIT software was run on a device in network, it will start a key pair generator to create the public key and the private key as Figure 4.3. The public key will be transmitted to all devices in the network. The domain parameter of key pair generator is determined by the given elliptic curve, which is $secp256k1$ for DFIT [77].

Setup - The key pair generator set a random number d as the private key. $G(x,y)$ is a given point in domain parameter. The public key $Q(x,y)$ could be calculated as follow.

$$Q(x,y) = d \cdot G(x,y) \tag{4.1}$$

Sign - Generate a random number k. Calculate the commitment $R(x,y) = kG(x,y)$. If the signed message data is m, the Schnorr signature s will be generated by the following equation,

$$s = k + h(m,R,P) \cdot p \tag{4.2}$$

where h(m,R,P) is hash value for a combination string of m, R and P. The hash function in DFIT is SHA-256, which is also applied to digest generation in consensus. It is an anti-collision hash function to translate a string of any length to a 256 bits digest. It is used to verify the correctness of messages since any change will lead to an entirely different hash.

Verify - When a device receives a message, it will resolve the message into data

and signature. The message m will be verified as follow.

$$\begin{aligned} s \times G &= (k + h(m,R,P) \times p) \times G \\ &= kG + h(m,R,P) \times p \times G \\ &= R + h(m,R,P) \times P \end{aligned} \tag{4.3}$$

Thus, the message is verified if $s \times G = R + h(m,R,P) \times P$ is true.

ECSchnorr supports aggregate signatures that the signatures from different senders are merged into one signature, and verified by a merged public key. As a result, the verification of ECSchnorr is efficient and defends the reliability of messages from network attack, making DFIT stable and secure.

4.4 Mobile Fintech Infrastructure App

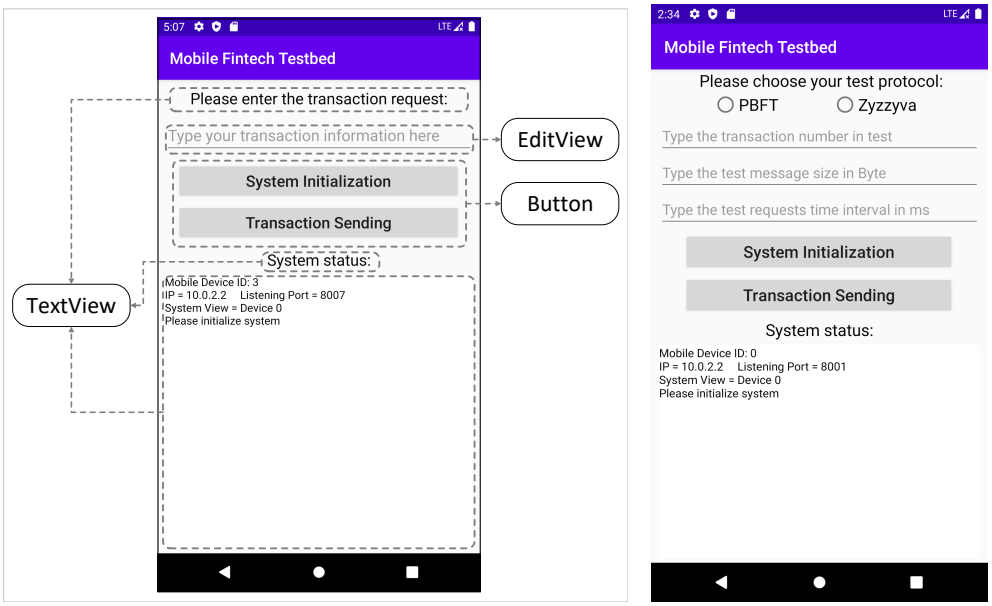
4.4.1 App Implementation in Android

The distributed ledger software implemented in this project is based on Android mobile devices, such as smartphones, tablets, PCs, smart televisions, and custom peripherals in the internet of things. The DFIT software is implemented on Android platform since it takes up the largest operating system market share (39%) [78].

Android apps are usually developed by Java programming and XML layout writing [79]. Since the human-computer interaction of mobile devices does not usually start from the same location, all Android apps would interact with the user by Activity which is the only visible component to users. Due to the limited resources and power consumption of mobile devices, the Android defines a life cycle for Activity to allocate and recycle limited resources appropriately. In Activity, the main thread is restricted to serve for human-computer interaction, which is only used to manipulate UI controls in the XML layout. Developers should use multiple threads to perform other tasks, such as listening ports, sending data and process messages.

4.4.2 User Interface

DFIT is carried on the Mobile Fintech Testbed app, whose UI is written in a top-down linear structure consist of Button, EditText and TextView controls (Figure 4.4(a)). [80][81]. Button is used to trigger the operation when it is pressed. EditView is an Android input control for scan the string that user typed. TextView is an output control to display the string of interactive information.



(a) Controls in UI (b) UI for large batch experiment

Figure 4.4: User interface of DFIT app

The app requires the user to first press ”System Initialization” to initialise the system. In this process, nodes in networks would exchange public keys for message verification later. After entering the transaction information string in EditView, the user needs to press the ”Transaction Sending” button to send the request. Then, a primary node, which is presented as ”view” in the screen, would execute the request, and start consensus phases. When the transaction written it into the ledgers, the client would receive replies and display the operation result in the TextView at the bottom. Another version of the app is implemented as the test script for the performance experiment in case of the large messages flow, which will be applied in Chapter 5.

Chapter 5

Experiment and Evaluation

5.1 Experiment Setup

This chapter presents a comparison of performance between PBFT and Zyzzyva on the DFIT prototype system to evaluate their applicability to different network environment. The experiment is run based on mobile device emulators on a 2.8 GHz Core-i7 personal computer. The emulators are integrated in the Android Studio IDE with 1536MB RAM and 6GB internal storage, which simulate the hardware performance of Google Pixel-2 smartphones. Each emulator is connected to a virtual router by port redirection to construct a virtual local area network. The experiment simulates a system consisting of four Android devices because of the limited PC hardware. The client and replica software are integrated together in the Mobile Fintech Testbed app. The experiment assumes that the received requests will not be lost so that all the requests in the task queue will be executed. The experiment set a network delay with random length from 50ms to 100ms to simulate the B4G/5G mobile network.

The experiment tests the system performance in case of a normal network with no failure at first. We take the time interval of requests sending and the message size as the two variables in the test, and then collect log for 1000 transactions. Namely, the client sends 1000 fixed size requests with given time intervals and records the operation result; then repeat the step when the client sends 1000 fixed interval requests with given message sizes. We can calculate the throughput and latency from the log in devices. After that, these steps in cases of f replicas failure and the primary sleeping are repeated for the performance observation of PBFT and Zyzzyva.

5.2 Result and Evaluation

5.2.1 Normal Case

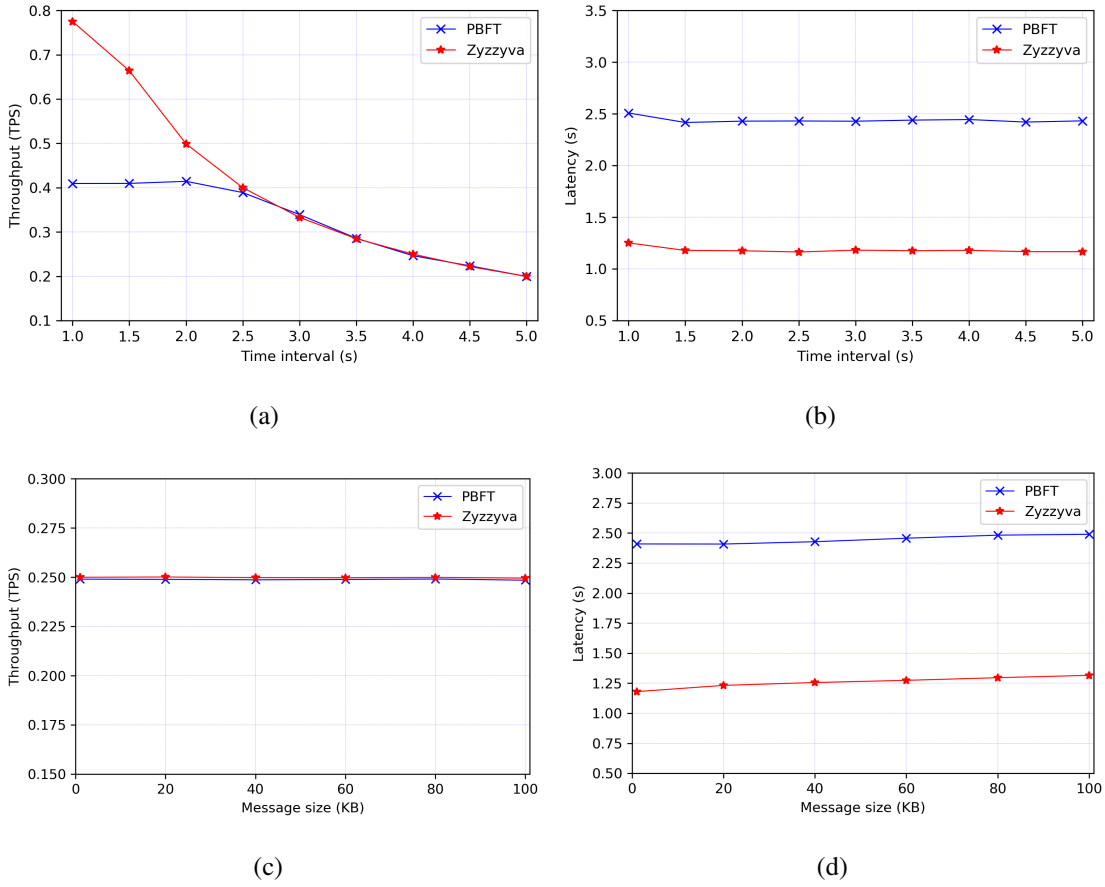


Figure 5.1: Performance of DFIT for 1000 transactions in normal case. (a) Throughput and (b) average latency when the request time interval varies from 1.5s to 5.0s. Message size = 1KB. (b) Throughput and (d) average latency for when the message size varies from 1KB to 100KB. Request interval = 4.0s.

Figure 5.1 (a) and (c) show the throughput of DFIT with different request time intervals and message sizes in normal case. With the increase of request time interval, the throughput of both mechanism declines in general, and approach the same value when the request interval is more than 2.5s. When requests are sent frequently, both the algorisms become saturation, whose speed of request is higher than processing speed. When the request interval is less than 2.5s, PBFT system has been saturated with a throughput of 0.41TPS; while the Zyzzyva's throughput is not saturated until the request interval is 1.0s to reach the maximum throughput of 0.78TPS, which is 90%

larger than PBFT. Regardless of the algorithm, the increase in message size within 100KB has almost no effect on the throughput rate of DFIT in message size change, since the time interval (4.0s) is greater than the execution time of a single request.

Figure 5.1 (b) and (d) show that the latency of PBFT is always higher than that of Zyzzyva for various request intervals and message sizes. Zyzzyva has a latency of 1.25s, which is 50% less than PBFT.

As a result, Zyzzyva has the higher throughput and lower latency in normal mobile system.

5.2.2 Failure Case: f Disabled Backups

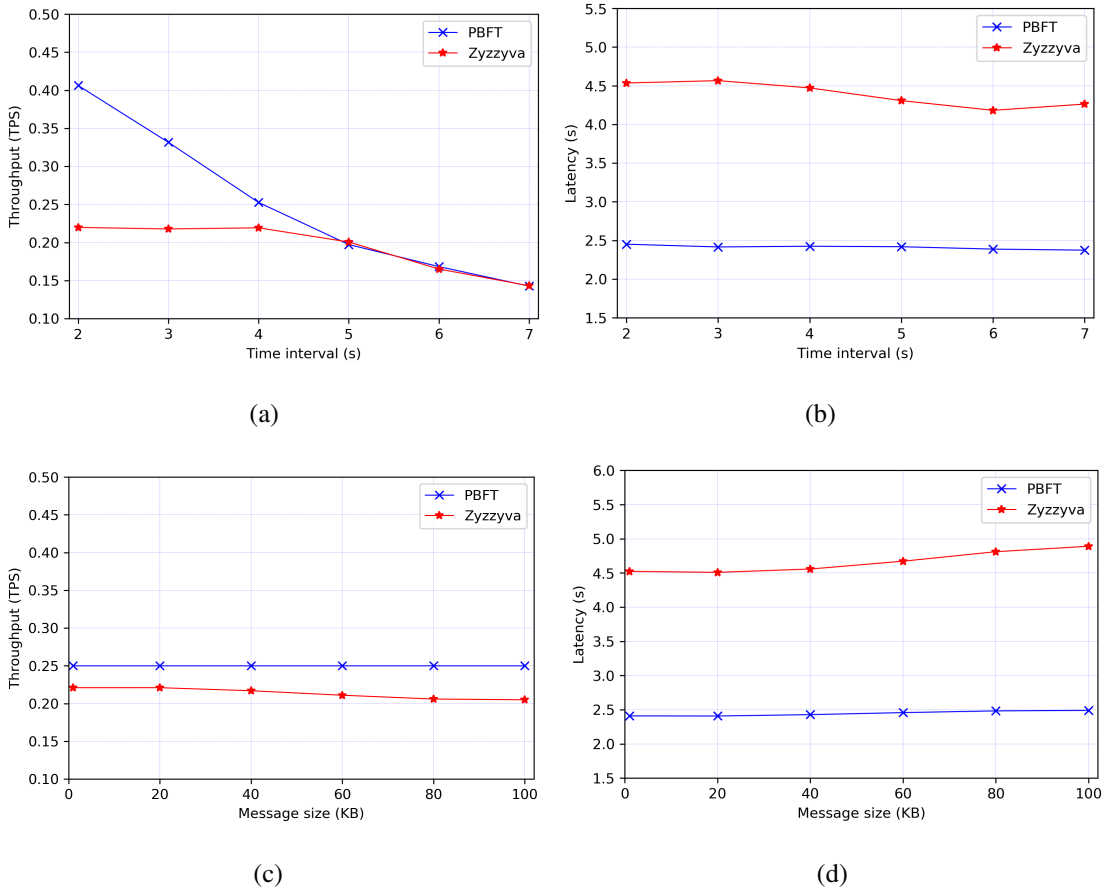


Figure 5.2: Performance of DFIT for 1000 transactions in case of f backup failure (f = 1). (a) Throughput and (b) average latency when the request time interval varies from 2.0s to 7.0s. Message size = 1KB. (c) Throughput and (d) average latency for when the message size varies from 1KB to 100KB. Request interval = 4.0s.

Figure 5.2 presents the case of f backups failure in mobile network. It can be

found that the system of PBFT has higher throughput and lower latency because PBFT is not sensitive to the backup failure case. It has the same throughput and latency as the normal case while Zyzzyva system performs a rapid performance reduction. The Zyzzyva throughput is only 52% of PBFT throughput in case of short request interval. Regardless of the request intervals and message sizes, Zyzzyva delay is always higher than PBFT. The change of the message size within 100KB have almost no impact on the system performance.

As a consequence, when there is f backups failure in system, PBFT has a higher throughput and lower latency.

5.2.3 Failure Case: The Disabled Primary

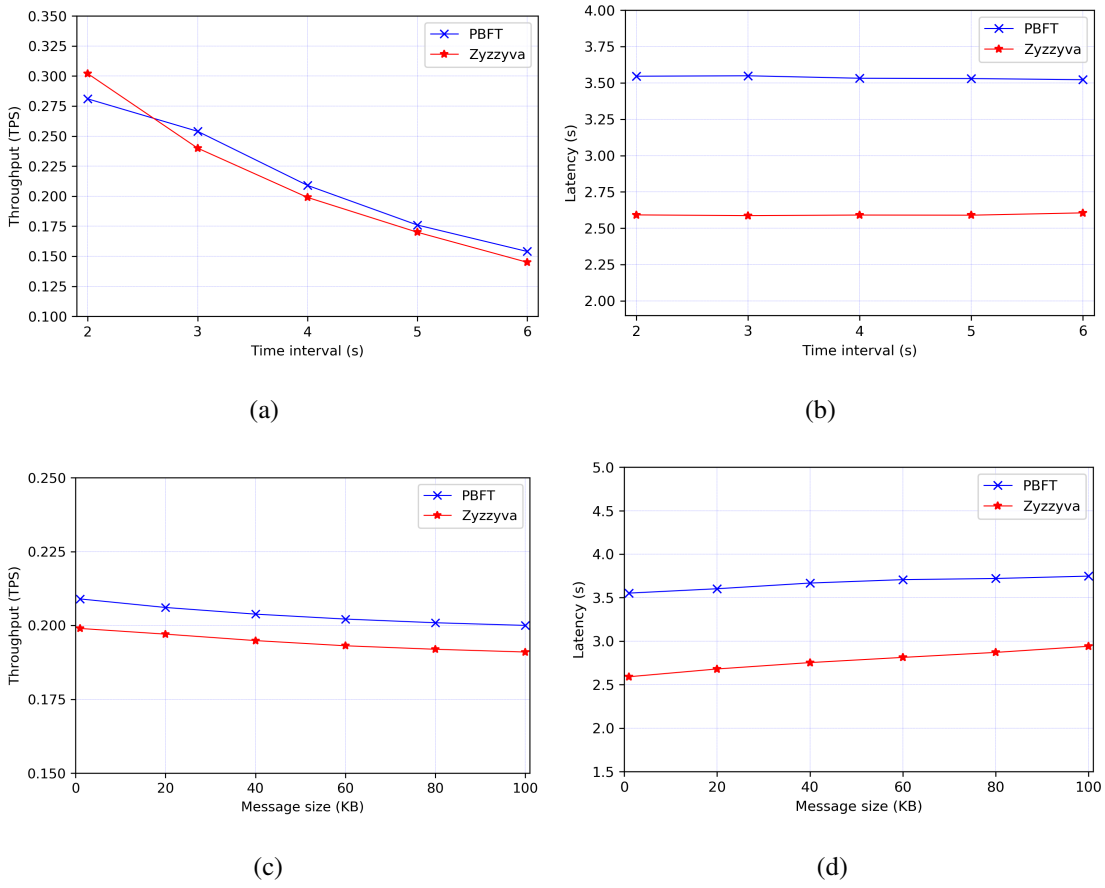
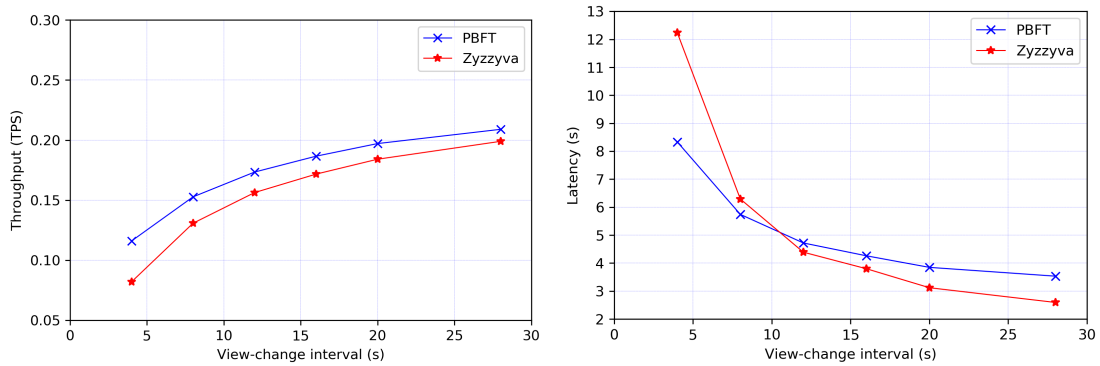


Figure 5.3: Performance of DFIT for 1000 transactions in case of the primary failure. Each current primary sleeps every 28 seconds to start a view-change, and revives until the new view advent. (a) Throughput and (b) average latency when the request time interval varies from 2.0s to 7.0s. Message size = 1KB. (c) Throughput and (d) average latency for when the message size varies from 1KB to 100KB. Request interval = 4.0s.

For the primary failure, Figure 5.3 shows an obvious decrease in throughput for the two BFT mechanisms. The throughput of PBFT is higher than Zyzzyva in despite of the message size changes, except to the case of the system saturation with busy requests.

The latency of view-change for the two algorithms is significantly higher than the normal case. For the period of 28s including normal work and view-change, PBFT has a higher latency than Zyzzyva, because the effect of long normal work has bigger impact than the view-change on performance. The higher throughput and latency imply that the DFIT has higher concurrency for PBFT than Zyzzyva.

Figure 5.4 shows the system performance with the various view-change frequencies in the network. With the longer time interval for the view-change, both the PBFT and



(a) (b)

Figure 5.4: Throughput and latency of DFIT for 1000 transactions in case of various view-change frequency. The time interval for view-change varies from 4s to 28s. Request interval = 4.0s. Message size = 1KB.

Zyzzyva have increasing throughput and decreasing latency. The throughput of PBFT is always higher than that of Zyzzyva. In case of the frequent view-changes in system, the PBFT has a lower latency than Zyzzyva due to its simpler view-change protocol.

5.3 Summary

Overall, Zyzzyva performs the higher throughput and lower latency in normal case, but it has the poorer performance than PBFT in case of replica failure. Zyzzyva has a simpler normal case protocol at the cost of more complex commit and view-change protocol, while PBFT has with better robustness to trades off the normal case and failure case. The growth of the message size within 100KB has not significant impact

on the performance for mobile systems. The experiment does not cover the situation of the excessively frequent requests because it will produce many finished threads but whose resources have not had time to be collected. The long queue of garbage collection will lead to a crash in Android.

Chapter 6

Conclusions

6.1 Project Summary

DFIT provides a practical evaluation platform for mobile distributed ledger system. The distributed ledger of mobile devices is very different from a system composed of servers, so it requires a unique method for optimization. The DFIT provides a reference for system optimization from the perspective of the consensus protocols in different system environment.

The DFIT experiment result shows that both PBFT and Zyzzyva are feasible for the mobile network. Their performance varies in different network environments, which can be used to optimize the network. Zyzzyva is appropriate for the stable network entirely consisting of health replicas, while PBFT fits for the network with unstable replicas and bad communication.

6.2 Future Work

In the project, the resource of PC limits number of available emulators, so there are more system features to be discovered, such as the influence of more replicas or more backup failures to the protocol performance.

Looking forward, this project offers a reference for the new protocol of distributed ledgers that can switch the consensus mechanism automatically according to the condition of the real network. In the future work, we will continue to focus on the optimization of mobile ledgers to take into account performance and security to energize more inclusive financial services.

Bibliography

[1] Philip Treleaven, Richard Gendal Brown, and Danny Yang. Blockchain technology in finance. *Computer*, 50(9):14–17, 2017.

[2] Alex Tapscott and Don Tapscott. How blockchain is changing finance. *Harvard Business Review*, 1(9):2–5, 2017.

[3] Erik Hofmann, Urs Magnus Strewe, and Nicola Bosia. *Supply chain finance and blockchain technology: the case of reverse securitisation*. Springer, 2017.

[4] Yonghee Kim, Jeongil Choi, Y-J Park, and Jiyoung Yeon. The adoption of mobile payment services for "fintech". *International Journal of Applied Engineering Research*, 11(2):1058–1061, 2016.

[5] In Lee and Yong Jae Shin. Fintech: Ecosystem, business models, investment decisions, and challenges. *Business Horizons*, 61(1):35–46, 2018.

[6] Sumit Agarwal, Wenlan Qian, Yuan Ren, Hsin-Tien Tsai, and Bernard Yin Yeung. The real impact of fintech: Evidence from mobile payment technology. *Available at SSRN 3556340*, 2020.

[7] Anna Omarini. Fintech and the future of the payment landscape: The mobile wallet ecosystem - a challenge for retail banks? *International Journal of Financial Research*, 9:97, 08 2018.

[8] Ashiq Anjum, Manu Sporny, and Alan Sill. Blockchain standards for compliance and trust. *IEEE Cloud Computing*, 4(4):84–90, 2017.

[9] Yan Chen and Cristiano Bellavitis. Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of Business Venturing Insights*, 13:e00151, 2020.

[10] Alia Asheralieva and Dusit Niyato. Learning-based mobile edge computing resource management to support public blockchain networks. *IEEE Transactions on Mobile Computing*, 2019.

[11] Xiaolong Xu, Xuyun Zhang, Honghao Gao, Yuan Xue, Lianyong Qi, and Wanchun Dou. Become: Blockchain-enabled computation offloading for iot in mobile edge computing. *IEEE Transactions on Industrial Informatics*, 16(6):4187–4195, 2019.

[12] Xiaolong Xu, Qingxiang Liu, Xuyun Zhang, Jie Zhang, Lianyong Qi, and Wanchun Dou. A blockchain-powered crowdsourcing method with privacy preservation in mobile environment. *IEEE Transactions on Computational Social Systems*, 6(6):1407–1419, 2019.

[13] Se Hun Lim, Dan J Kim, Yeon Hur, and Kunsu Park. An empirical study of the impacts of perceived security and knowledge on continuous intention to use mobile fintech payment services. *International Journal of Human–Computer Interaction*, 35(10):886–898, 2019.

[14] Fatou Ndiaye Mbodji, Gervais Mendy, Ahmath Bamba Mbacke, and Samuel Ouya. Proof of concept of blockchain integration in p2p lending for developing countries. In *International Conference on e-Infrastructure and e-Services for Developing Countries*, pages 59–70. Springer, 2019.

[15] Lee David Kuo Chuen and Low Linda. *Inclusive Fintech: Blockchain, Cryptocurrency and ICO*. World Scientific, 2018.

[16] Hazik Mohamed and Hassnian Ali. *Blockchain, Fintech, and Islamic finance: Building the future in the new Islamic digital economy*. Walter de Gruyter GmbH & Co KG, 2018.

[17] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. *International Journal of Research in Engineering and Technology*, 5(9):1–10, 2016.

[18] Svein Olnes, Jolien Ubacht, and Marijn Janssen. Blockchain in government: Benefits and implications of distributed ledger technology for information sharing. *Government Information Quarterly*, 34(3):355 – 364, 2017.

[19] Daniel Drescher. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Apress, USA, 1st edition, 2017.

[20] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[21] Sarah Underwood. Blockchain beyond bitcoin. *Commun. ACM*, 59(11):15–17, October 2016.

[22] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

[23] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[24] Yang Lu. Blockchain and the related issues: a review of current research topics. *Journal of Management Analytics*, 5(4):231–255, 2018.

[25] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.

[26] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*, pages 557–564. IEEE, 2017.

[27] Marc Pilkington. *Blockchain technology: principles and applications*. Edward Elgar Publishing, Cheltenham, UK, 2016.

[28] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 107:841–853, 2020.

[29] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

[30] Zhetao Li, Jiawen Kang, Rong Yu, Dongdong Ye, Qingyong Deng, and Yan Zhang. Consortium blockchain for secure energy trading in industrial internet of things. *IEEE transactions on industrial informatics*, 14(8):3690–3700, 2017.

[31] M. Tamer zsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

[32] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., USA, 2006.

[33] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, pages 343477–343502. Portland, OR, 2000.

[34] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[35] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[36] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure information networks*, pages 258–272. Springer, 1999.

[37] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.

[38] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

[39] Wei Ren, Jingjing Hu, Tianqing Zhu, Yi Ren, and Kim-Kwang Raymond Choo. A flexible method to defend against computationally resourceful miners in blockchain proof of work. *Information Sciences*, 507:161–171, 2020.

[40] Martijn Bastiaan. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In *Available at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf*, 2015.

[41] Giang-Truong Nguyen and Kyungbaek Kim. A survey about consensus algorithms used in blockchain. *Journal of Information processing systems*, 14(1), 2018.

[42] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 100:143 – 174, 2019.

[43] Pavel Vasin. Blackcoin's proof-of-stake protocol v2. *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, 71, 2014.

[44] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19:1, 2012.

[45] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[46] Andrew Poelstra et al. Distributed consensus from proof of stake is impossible. *Self-published Paper*, 2014.

[47] Daniel Larimer. Delegated proof-of-stake (dpos). *Bitshare whitepaper*, 2014.

[48] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, Cham, 2017. Springer International Publishing.

[49] ZiBin Zheng, Michael Rung Tsong Lyu, and HuaiMin Wang. Service fault tolerance for highly reliable service-oriented systems: an overview. *Science China Information Sciences*, 58(5):1–12, 2015.

[50] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982.

[51] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[52] Miguel Castro and Barbara Liskov. Byzantine fault tolerance can be fast. In *2001 International Conference on Dependable Systems and Networks*, pages 513–518. IEEE, 2001.

[53] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[54] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[55] Pedro Franco. *Understanding Bitcoin: Cryptography, engineering and economics*. John Wiley & Sons, 2014.

[56] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *SIGOPS Oper. Syst. Rev.*, 41(6):45–58, October 2007.

[57] L. J. Gunn, J. Liu, B. Vavala, and N. Asokan. Making speculative bft resilient with trusted monotonic counters. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 133–13309, 2019.

[58] Chris Skinner. *ValueWeb: How fintech firms are using bitcoin blockchain and mobile technologies to create the Internet of value*. Marshall Cavendish International Asia Pte Ltd, 2016.

[59] Timothy R Lyman, Mark Pickens, and David Porteous. Regulating transformational branchless banking: Mobile phones and other technology to increase access to finance. Technical report, The World Bank, 2008.

[60] Tom Kellermann. Mobile risk management: e-finance in the wireless environment. *Journals of Financial Sector*, pages 1–28, 2002.

[61] Djamel Djenouri, Lves Khelladi, and A Nadjib Badache. A survey of security issues in mobile ad hoc and sensor networks. *IEEE Communications Surveys and Tutorials*, 7(4), 2005.

[62] Kongrath Suankaewmanee, Dinh Thai Hoang, Dusit Niyato, Suttinee Sawadsitang, Ping Wang, and Zhu Han. Performance analysis and application of mobile blockchain. In *2018 international conference on computing, networking and communications (ICNC)*, pages 642–646. IEEE, 2018.

[63] Xu Wang, Guangsheng Yu, Xuan Zha, Wei Ni, Ren Ping Liu, Y. Jay Guo, Kangfeng Zheng, and Xinxin Niu. Capacity of blockchain based internet-of-things: Testbed and analysis. *Internet of Things*, 8:100109, 2019.

[64] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

[65] Angela Walch. The bitcoin blockchain as financial market infrastructure: A consideration of operational risk. *NYUJ Legis. & Pub. Pol'y*, 18:837, 2015.

[66] Yonggang Wen, Weiwen Zhang, and Haiyun Luo. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *2012 Proceedings Ieee Infocom*, pages 2716–2720. IEEE, 2012.

[67] Anthony I Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400, 2010.

[68] Yong Yuan and Fei-Yue Wang. Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1421–1428, 2018.

[69] Ittai Abraham, Dahlia Malkhi, et al. The blockchain consensus layer and bft. *Bulletin of EATCS*, 3(123), 2017.

[70] Alex Norta. Designing a smart-contract application layer for transacting decentralized autonomous organizations. In *International Conference on Advances in Computing and Data Sciences*, pages 595–604. Springer, 2016.

[71] Harish Sukhwani, José M Martínez, Xiaolin Chang, Kishor S Trivedi, and Andy Rindos. Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric). In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 253–255. IEEE, 2017.

[72] Roger Wattenhofer. *The Science of the Blockchain*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 1st edition, 2016.

[73] Behrouz A. Forouzan and Sophia Chung Fegan. *TCP/IP Protocol Suite*. McGraw-Hill Higher Education, 2nd edition, 2002.

[74] Steven M Bellovin. Security problems in the tcp/ip protocol suite. *ACM SIGCOMM Computer Communication Review*, 19(2):32–48, 1989.

[75] Behrouz A. Forouzan. *Cryptography amp; Network Security*. McGraw-Hill, Inc., USA, 1 edition, 2007.

[76] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[77] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[78] StatCounter. Operating system market share worldwide. [EB/OL]. https://gs.statcounter.com/os-market-share, Accessed August 14, 2020.

[79] Neil Smyth. *Android Studio 3.5 Development Essentials-Java Edition: Developing Android 10 (Q) Apps Using Android Studio 3.5, Java and Android Jetpack*. eBookFrenzy, 2019.

[80] Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike. *Android application development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.

[81] Jianye Liu and Jiankun Yu. Research on development of android applications. In *2011 4th International Conference on Intelligent Networks and Intelligent Systems*, pages 69–72. IEEE, 2011.

Appendix A

List of abbreviations

Abbreviation	Full form
B4G/5G	Beyond 4th or 5th Generation of broadband cellular network technology
BFT	Byzantine Fault Tolerance
CAP	Consistency, Availability, and Partition tolerance
DFIT	Distributed Fintech Infrastructure Testbed
DPoS	Delegated Proof-of-Stake
ECC	Elliptic Curve Cryptography
ECSchnorr	Elliptic Curve Schnorr signature
FCFS	First-Come-First-Serve
IDE	Integrated Development Environment
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PoW	Proof-of-Work
PoS	Proof-of-Stake

Table A.1: List of abbreviations