A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing

by

Saurabh Gupta

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2016 by the
Graduate Supervisory Committee:

Rida Bazzi, Chair
Gail-Joon Ahn
Maurice Herlihy

ARIZONA STATE UNIVERSITY

August 2016

ABSTRACT

The success of Bitcoin has generated significant interest in the financial community to understand whether the technological underpinnings of the cryptocurrency paradigm can be leveraged to improve the efficiency of financial processes in the existing infrastructure. Various alternative proposals, most notably, Ripple and Ethereum, aim to provide solutions to the financial community in different ways. These proposals derive their security guarantees from either the computational hardness of proof-of-work or voting based distributed consensus mechanism, both of which can be computationally expensive. Furthermore, the financial audit requirements for a participating financial institutions have not been suitably addressed.

This thesis presents a novel approach of constructing a non-consensus based decentralized financial transaction processing model with a built-in efficient audit structure. The problem of decentralized inter-bank payment processing is used for the model design. The two key insights used in this work are (1) to utilize a majority signature based replicated storage protocol for transaction authorization, and (2) to construct individual self-verifiable audit trails for each node as opposed to a common Blockchain. Theoretical analysis shows that the model provides cryptographic security for transaction processing and the presented audit structure facilitates financial auditing of individual nodes in time independent of the number of transactions.

To my parents.

ACKNOWLEDGMENTS

I would like to acknowledge that the central idea presented in this thesis, i.e. a transaction processing model without distributed consensus, was conceived by Rida Bazzi and Maurice Herlihy, and I express my utmost gratitude to them for giving me the opportunity to work on this interesting research problem. It would not have been possible without their ideas and insights that guided me throughout the duration of this work.

I am profoundly grateful and honored to have Rida Bazzi, Maurice Herlihy and Gail-Joon Ahn on my thesis committee, all of whom are meritorious scholars and have made exceptional contributions in this area of research. I am particularly thankful to Rida Bazzi for his exceptional guidance, constant support and patience as my adviser and mentor over the past year, which has been pivotal to this thesis. I am indebted to him for his in-depth reviews, vital inputs and critical insights that helped shape my work.

I thank Arizona State University for providing me the platform to pursue my interests. And finally, I would like to thank all the colleagues, friends and family who supported and motivated me during my time as a graduate student.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Bitcoin [21], as a cryptographic proof-of-work based decentralized value exchange mechanism, has demonstrated that it is practically possible for a distributed set of networked nodes, which don't trust each other, to arrive at a consensus about the validity of individual transactions and maintain the collective transaction history of the network, using a distributed ledger, without the need of centralized control. The success of Bitcoin and the Blockchain structure have generated significant interest in the financial community to understand whether the technological underpinnings of the cryptocurrency paradigm can be leveraged to improve the efficiency of financial processes in the existing infrastructure.

Decentralized inter-bank payment processing, smart contracts and distributed asset management have emerged as some interesting problems from this discussion. Various alternative proposals, most notably, Ripple [24] and Ethereum [28], aim to provide solutions to these problems in different ways. These proposals derive their security guarantees from either the computational hardness of proof-of-work or voting based distributed consensus mechanism, both of which can be computationally expensive. Furthermore, the financial audit requirements for a participating financial institution have not been suitably addressed. In this thesis, we present a novel approach of constructing a non-consensus based decentralized financial transaction processing model with a built-in efficient audit structure. We use the problem of decentralized inter-bank payment processing for the model design.

1.2 Inter-bank Payment Processing

Given banks B_1 and B_2 with respective customers c_1 and c_2, an inter-bank payment transaction T can be defined as an electronic transfer of amount x from customer c_1 at bank B_1 to customer c_2 at bank B_2 such that the completed atomic transaction consists of the following components.

1. Reduction of Liability for bank B_1 towards customer c_1 by amount x.
2. Increment of Liability for bank B_2 towards customer c_2 by amount x.
3. Electronic transfer of amount x from bank B_1 to bank B_2 as settlement towards the transfer of Liability.

Note that the changes in liability for a bank towards its customer are managed by the corresponding bank subject to its internal processes. Our discussion will focus on the third component of the transaction processing.

1.2.1 Centralized Payment Processing

FFIEC IT Booklets [13] describe various payment processing mechanisms used by the banking industry in US and mention that financial institutions are increasingly relying on third-party service providers to perform payment processing functions on their behalf. For instance, the most common methods for wholesale inter-bank payment processing are FedWire, operated by the Federal Reserve Banks, and CHIPS, operated by The Clearing House Payments Company, wherein participating banks rely on the corresponding service provider for payment processing. A simplified centralized payment processing model is depicted by the figure 1.1.

In the centralized payment processing model, *Bank-1* and *Bank-2* rely on the *Financial Institution*, at the center of the figure, to perform the payment processing

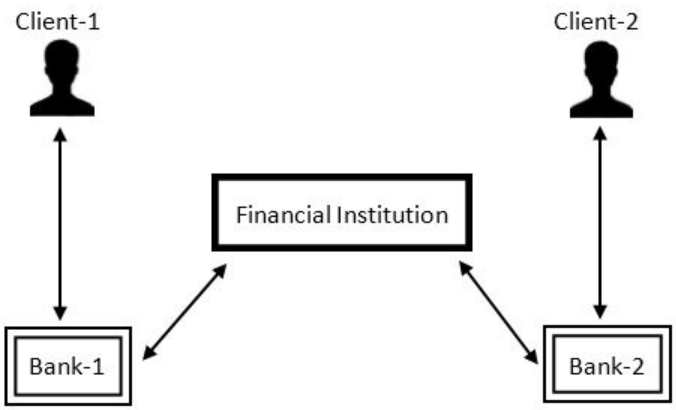

Figure 1.1: Centralized Payment Processing Model

on their behalf when, for instance, *Client-1* submits a transaction request at *Bank-1* for the transfer of amount X to *Client-2* at *Bank-2*.

1.2.2 Decentralized Payment Processing

Our model aims to replace the centralized control of the *Financial Institution* in figure 1.1 by a distributed network of banks, as depicted by figure 1.2, that perform the payment processing in a decentralized manner, while complying with regulatory requirements. The figure shows an example of the distributed network containing 4 banks, however, we generalize the problem to include n banks.

The key elements of the decentralized inter-bank payment processing problem are as follows. Given a distributed network consisting of process nodes $S = \{s_i : 1 \leq i \leq n\}$, where s_i is a server node federated by bank B_i, any inter-bank payment transaction T should

- be completed securely and atomically, to reflect the transfer of amount x from bank B_1 to bank B_2, in presence of Byzantine failures,
- have self-verifiable properties with non-repudiation, upon completion, for veri-

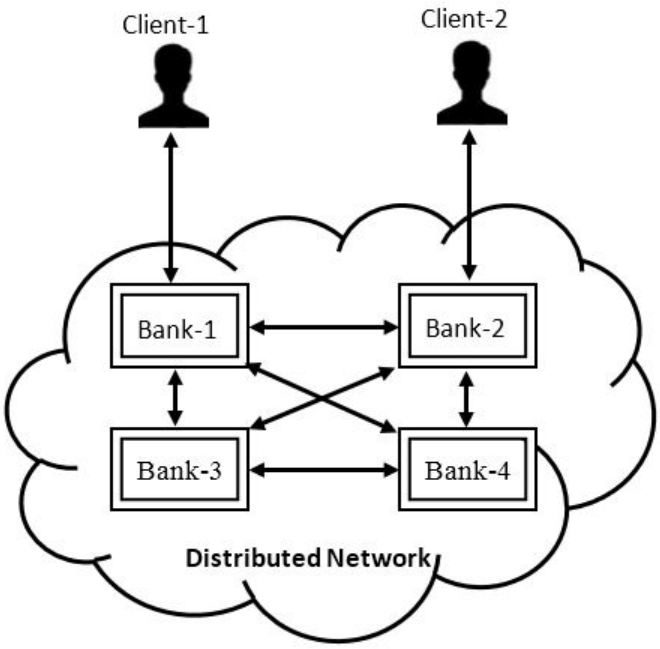

Figure 1.2: Decentralized Payment Processing Model

fication in case of future transaction processing and auditing, and

- be included in an immutable and self-verifiable audit structure for efficient financial auditing subject to regulatory requirements.

1.3 Financial Auditing

Financial Auditing is an important aspect of the financial infrastructure. As per US Government regulations [12, 25], public companies, including commercial banks, are required to publish periodic financial statements along with an auditor's attestation and the audit is required to be performed by an independent public accountant in case of large companies.

PCAOB Auditing Standards [22] define the objective of audit of financial statements as *"the expression of an opinion on the fairness with which they present, in all material respects, financial position, results of operations, and its cash flows in*

conformity with generally accepted accounting principles." Audit procedures include assessment of internal controls over reporting as well as substantive procedures for gathering evidence. Thus, it is important for a financial system to have efficient support for substantive procedures with strong internal controls in order to be efficiently audited.

We propose an audit structure that establishes periodic partial ordering of transactions into summarized checkpoints with respect to each participant and create individual audit trails for efficient auditing. Our results show that our audit structure facilitates financial auditing of individual nodes in time independent of the number of transactions.

1.4 Organization of the Thesis

The organization of the thesis is as follows:

- Chapter 2 discusses the related work in the field of cryptocurrency and distributed systems.
- Chapter 3 introduces the system model for the decentralized inter-bank payment processing problem.
- Chapter 4 presents the transaction model and authorization protocol, along with the detailed analysis of the protocol.
- Chapter 5 presents the audit structure used for efficient financial auditing, along with detailed theoretical analysis.
- Chapter 6 concludes the thesis and discusses future work on the problem.

Chapter 2

RELATED WORK

2.1 Distributed Consensus

Pease, Shostak and Lamport [23, 16] introduced the term *Byzantine Fault Tolerance* using the *Byzantine Generals Problem*, as a distributed consensus problem, where a set of f faulty process nodes may arbitrarily deviate from the defined process by tampering with or suppressing messages. Their results showed the solution requires at least $3f + 1$ process nodes.

Fischer et al. [14] showed that it is impossible to achieve distributed consensus in presence of even one faulty node using totally asynchronous model of computation. Practical solutions, like Bitcoin [21], adopt a model that allows partial synchrony in order to aim for probabilistic consensus over a period of time.

Our analysis shows that a non-consensus based transaction processing model can provide Byzantine fault tolerance in an asynchronous distributed network.

2.2 Cryptocurrency Systems

Back [6] first introduced, through Hashcash, the notion of cryptographic proof of computational work as a mechanism for secure value transfer over the Internet. This idea was later expanded and developed by Nakamoto [21] into Bitcoin as a cryptocurrency implementation wherein the network can securely achieve distributed consensus over the validity of financial transactions and collectively build a distributed immutable ledger, namely the Blockchain. Bitcoin has been extensively studied and analyzed most notably by Garay et al. [15], Clark et al. [11], Bonneau et al. [9],

Tschorsch and Scheuermann [27].

Factom, introduced by Snow et al. [26], aims at providing a mechanism for businesses to access the Bitcoin blockchain to record their transactions for generating an immutable audit trail. Wood [28] discusses Ethereum as a generalized platform with a Turing-complete scripting language upon which various decentralized transaction systems can be built.

As a computationally inexpensive alternative to proof-of-work mechanism, Bentov et al. [8] discuss pure proof-of-stake based protocols and their potential gains in terms of efficiency, while hypothesizing that the cryptographic security can possibly be worse relative to Bitcoin subject to short-term human behavior.

Alternatively, Ripple, introduced by Schwartz et al. [24], has shown that an efficient currency exchange and financial clearing system can be built upon a federated system using voting based-consensus mechanism. Mazieres [20] introduced Federated Byzantine Agreement, and Stellar Consensus Protocol as its construction, which uses two-phases voting mechanism to achieve consensus in a network that allows open membership and dynamic growth.

All the current proposals aim to solve the problem of achieving distributed consensus in order to provide cryptographic security for the transaction processing. Our work introduces the novel approach of using a non-consensus based transaction processing model.

2.3 Replicated Storage

Aiyer et al. [4] presented a bounded wait-free distributed register that does not require communication channels among processing nodes in order to improve message complexity of the protocol.

Aiyer et al. [5] introduced a MAC based signature scheme for distributed system

by constructing a matrix of MAC tags $\{h_{i,j} : 1 \leq i, j \leq n\}$, such that each tag $h_{i,j}$ is said to be signed by node i and can be verified by node j.

Our model derives ideas from both these papers. We use a model of communication where a sender node communicates with the processing nodes that do not interact with each other during the processing of the transaction. Also, our protocol constructs a set of digital signatures for a transaction similar to the matrix of MAC tags.

2.4 Distributed Checkpointing

For our audit structure design, we use distributed audit checkpoints. Distributed checkpointing was introduced by Chandy and Lamport [10] as a useful tool for computing the global state of a distributed computation. In their model, all the nodes coordinate in recording the local states of all nodes and channels into a distributed snapshot representing the global state.

However, we introduce a different approach wherein each node constructs its checkpoints comprising only of the transactions involving itself and proceeds to request authorization from the network to construct its own audit trail for financial auditing purposes.

2.5 Payment Systems

FFIEC IT Booklets [13] describe various payment processing mechanisms used by the banking industry in US as discussed in section 1.2.1. Commonly used electronic payment processing methods include The Automated Clearing House (ACH) [1] for

[1] ACH networks operate in accordance with guidelines by NACHA — The Electronic Payments Association. https://www.nacha.org/ach-network

retail payments, and FedWire [2] and CHIPS [3] for wholesale payments. These payment methods are centralized and rely on a service provider to process and settle payments on behalf of the participating commercial banks and financial institutions. Our model introduces a decentralized approach to payment processing and settlement where participating banks can process payments without the aid of any third party service provider.

[2] FedWire is operated by the Federal Reserve Banks. https://www.frbservices.org

[3] CHIPS is operated by The Clearing House (formerly known as the New York Clearing House Association). https://www.theclearinghouse.org

Chapter 3

SYSTEM MODEL

In this chapter, we firstly discuss the initial pre-conditions required to be satisfied in order for the proposed system model to be implemented. Then, we present the network model along with its assumptions. Finally, we discuss an overview of the customer transaction in view of the network model.

3.1 Pre-conditions

We assume that the bootstrapping of the proposed system will begin with an offline legal binding contract among a set of commercial banks $B = \{B_i : 1 \leq i \leq n\}$, such that each bank B_i will allocate a server node s_i and a minimum pre-funding amount x_i for the server node s_i in order to participate in the decentralized inter-bank payment processing, in presence of a set of legally authorized auditors $A = \{a_i : i \geq 1\}$ external to the banks in set B. Each bank B_i will be expected to maintain explicit private ledger entries specifying the allocation of amount x_i from its liquid accounts for the purpose of transparent reconciliation and auditing between its private ledger and the system's distributed ledger.

For the current state of this work, the execution of contracts is out of scope of the system and will be expected to be performed in traditional offline fashion. However, any changes proposed to the server node membership or pre-funding amount corresponding to a bank $B_i \in B$ will be required to be cryptographically signed by an auditor upon verification of B_i's private and distributed ledger histories for meeting contract compliance requirements.

3.2 Network Model and Assumptions

We define the network as composed of 3 categories of nodes:

- **Server Nodes** consisting of the set $\mathcal{S} = \{s_i : 1 \leq i \leq n\}$, where each server node s_i is federated by the corresponding bank B_i, while the set $\mathcal{S}$ is decentralized subject to offline contractual accountability requirements.

- **Interface Nodes**: We define an interface node e_i, federated by bank B_i, as an application server node acting as a transaction request interface for the bank's customers. The set of interface nodes is defined as $E = \{e_i : 1 \leq i \leq n\}$. The purpose of separation between interface nodes and server nodes is to provide a layer of abstraction between a bank's private ledger functionality and the distributed ledger functionality to be separately managed by interface nodes and server nodes respectively.

- **Auditor Nodes** comprising of logical or physical nodes corresponding to the set of auditors $\mathcal{A} = \{a_i : i \geq 1\}$ in accordance with the offline contract.

Figure 3.1 demonstrates an example of our network model consisting of four Banks and one Auditor Node. The solid boundary corresponds to the distributed network consisting of the server nodes, and the dashed boundaries correspond to individual banks, each containing the interface node and server node for corresponding bank. Note that an interface nodes interact with only the server node within its bank's boundary.

We assume that the auditor nodes and interface nodes are benign, while the up to f server nodes can be subject to Byzantine failures, such that $n = 3f + 1$.

We assume that each server node has a key pair $\langle sk,\ pk \rangle$, where sk is the secret signing key and pk is the public signature verification key, and the signature

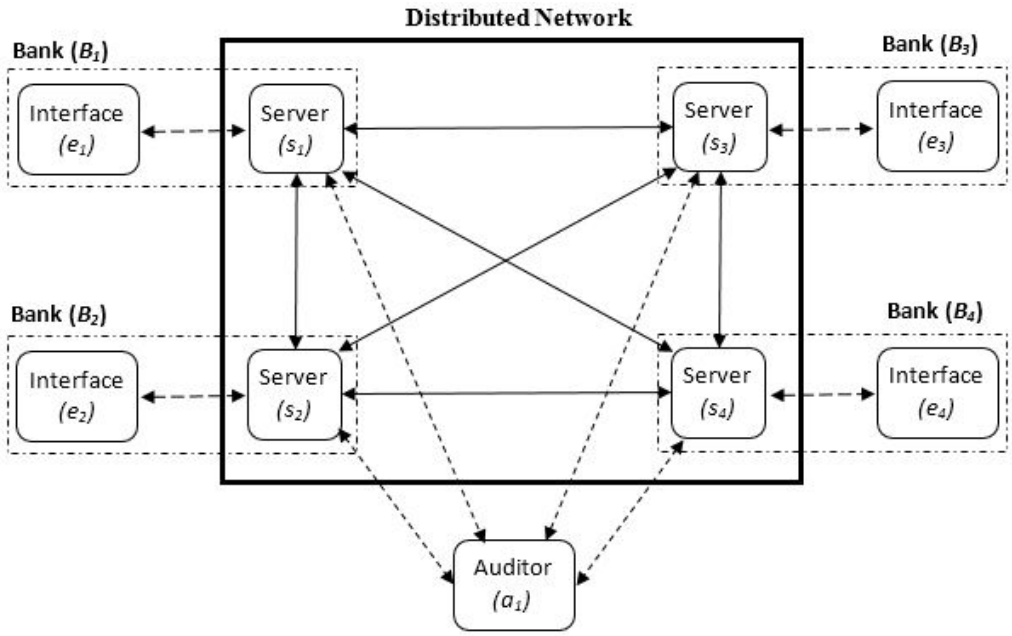

Figure 3.1: Network Model Example

verification key pk is known to every server and auditor node in the network. Additionally, we assume that there exists a Digital Signature scheme $\langle Sign,\ Verify \rangle$, where $Sign : d \leftarrow Sign_{sk}(m)$ is the function used to producing the digital signature d for the message m, and $Verify : b \leftarrow Verify_{pk}(m, d)$, such that $b \in \{true, false\}$, is the function that verifies the digital signature d for the message m and $Verify_{pk}(m, d) = true$ if and only if the signature d is valid for the message m. Additionally, we assume that signatures can not be forged.

We assume that interface nodes will validate and process customer transaction requests in accordance with the process specifications of the corresponding bank, and we do not wish to propose any changes to the internal transaction processing functions of any bank.

3.3 Transaction Overview

Let us consider a transaction submitted by customer c_1 using interface node e_1 at bank B_1 for payment of amount x to customer c_2 at bank B_2. The transaction processing will involve the following steps.

1. Interface node e_1 reduces the customer c_1's balance by amount x in bank B_1's private ledger

2. Bank B_1's server node s_1 performs a payment processing transaction to transfer amount x to bank B_2's server node s_2 using the distributed network of server nodes $\mathcal{S}$

3. Bank B_1 communicates the customer information for the transaction to Bank B_2 for reconciliation with the payment processing transaction in step 2 and transfer of amount x to customer c_2

4. Interface node e_2 increments customer c_2's balance by amount x in bank B_2's private ledger

Steps 1 and 4 involve changes in the private ledgers of the corresponding banks, and we assume that these changes will be managed by the corresponding banks subject to their internal processes.

Although steps 2 and 3 can intuitively be combined, we have mentioned them separately to make an interesting distinction. Note that the transaction in step 2 involves server s_1 and s_2, and can be agnostic of the customers involved in the overall transaction. Whereas, the communication in step 3 requires customer information to be transmitted. If the customer information is included in the payment processing transaction in step 2, it will be visible to all the server nodes in the network, which includes potentially Byzantine server nodes.

We do not wish to make the assumption whether this will be desirable to the banks involved. So, let us consider the various cases in terms of possible requirements according to the banks.

- **Case 1:** *Customer information can be transmitted as plain-text in the payment processing transaction according to the requirements.* In this case, steps 2-3 can be combined and customer information can be included as plain-text metadata to the payment processing transaction in step 2. The metatdata information can be used by bank B_2 for reconciliation and transfer of funds to the customer.

- **Case 2:** *Encrypted customer information can be transmitted in the payment processing transaction according to the requirements.* We assume that the encryption scheme will be decided as a part of the requirements and corresponding keys will be suitably shared among the banks. Now, steps 2-3 can be combined and the bank B_1 can provide its server node s_1 with the encrypted customer information, such that it can be decrypted by bank B_2 only, to be included as metadata to the payment processing transaction in step 2. The decrypted metatdata information can be used by bank B_2 for reconciliation and transfer of funds to the customer.

- **Case 3:** *Customer information can not be transmitted in the payment processing transaction according to the requirements.* In this case, the steps 2 and 3 will be required to be processed separately. After the payment processing transaction in step 2 is completed, bank B_1 can communicate the information to the bank B_2 using a separate private channel for reconciliation and step 4 processing.

Therefore, irrespective of the case adopted by the banks involved in the implementation of the system model, we assume that step 3 will be performed as a part of

the transaction processing. We will focus on the payment processing transaction in step 2 for our transaction model in chapter 4.

Chapter 4

TRANSACTION PROCESSING MODEL

In a traditional transaction model, each entity is associated with a balance value, which is a cumulative total of all the incoming and outgoing transactions for the entity, and the validity of a transaction is contingent upon availability of funds represented by the balance value.

In contrast, Bitcoin [21] represents a transaction as a mapping between input transactions and redeemable outputs. A transaction output is said to be an candidate for being used as an input for a new transaction if it has not been used as an input for any transaction, and such a transaction output is termed as an *unspent* transaction. As a result, the input-output mapping transaction model provides a proof of availability of funds that can be verified without the knowledge of the balance value associated with the transaction initiator if the distributed transaction processing model prevents a spent transaction output to be used as an input in a new transaction, referred to as a *double-spend* transaction. Bitcoin attempts to enforce this constraint by requiring eventual network consensus over complete transaction set using the Blockchain, where a block containing a *double-spend* transaction is rejected by all the non-faulty processing nodes and not included in the longest branch of the Blockchain.

We adopt a transaction model similar to the Bitcoin transaction model where a transaction is represented as an input-output mapping. However, our transaction processing model differs from Bitcoin such that each server node s_i is required to maintain only the set of transactions that are processed by the node s_i, instead of requiring it to maintain all the transactions. Also, each server node s_i validates a

transaction independent of the validations performed by other server nodes in the network and provides a signature for the transaction if s_i considers it valid according to the information available to it. A transaction is said to be authorized when it receives valid signatures from $2f + 1$ server nodes, and the resulting signature set is considered a proof of validity for the transaction. Each server node s_i, that signs a transaction, is required to maintain the transaction as a part of its transaction set, which is used by the node s_i to validate future transactions. Our results show that *double-spend* transaction can not be authorized in our model, without requiring network consensus over either the complete transaction set or the validity of individual transactions during processing.

Authorized transactions should become part of the permanent record of activity of servers involved in the transaction as a sender or receiver. In order to support auditing of transactions, each server should reconcile, in a permanent record and at regular intervals, the sequence of transactions it is involved in. Before being committed to this permanent record, a transaction is *new*, and after it is committed, the transaction changes status and is no longer *new*. The discussion of the details of this commitment process is given in Chapter 5.

The organization of this chapter is as follows.

- In section 4.1, we define the transaction data structures used in our model.
- Section 4.2 defines the validation structure to be used by a server node for validating transactions.
- Section 4.3 discusses the transaction processing protocol for a transaction sender node, receiver node and a processing node.
- In section 4.4, we analyze the transaction processing model.

- Section 4.5 discusses some efficiency improvement ideas pertaining to the transaction processing model.

4.1 Transaction Data Structures

The life-cycle of a transaction involves 2 stages:

1. **Proposed:** When a sender node initiates the transaction for authorization, the transaction is said to be a proposed transaction. We use T to denote a proposed transaction. The structural composition of a proposed transaction is described in definition 4.1.

2. **Authorized:** When a server node receives a proposed transaction T, it validates the transaction and responds with a signature if the proposed transaction is valid according to it. Once the sender node receives $2f + 1$ signatures from the network, the transaction is said to be authorized by the network. We use $\mathfrak{T}$ to denote an authorized transaction. The authorized transaction data structure is described in definitions 4.2.1 and 4.2.2.

Definition 4.1. *(Proposed Transaction) A proposed transaction T is defined as a 5-tuple $\langle s,\ id,\ I,\ O,\ d \rangle$, where*

- *s is the server node that initiates the transaction T.*
- ***id** is integer sequence number assigned to the transaction T.*
- *O is the set of redeemable outputs of the transaction T and is defined as $O = \{\langle j,\ s_j,\ x_j \rangle : s_j$ is the j^{th} recipient server node, and x_j is the amount to be received by it $\}$*

- *I is the set of input transactions for T and is defined as*

  $I = \{\langle \mathcal{T},\ j\rangle : \mathcal{T}$ is an authorized input transaction, and j is the output index to be used from the transaction $\mathcal{T}$ $\}$

- *d is the digital signature of the sender node s for the transaction $\langle s, id, I, O\rangle$.*

For simplifying the notations, we have included the authorized transactions $\mathcal{T}$ in the definition of input transactions set I. However, for implementation purposes, it can be replaced by a pair $\langle s, id\rangle$ where s is the sender node and id is the sequence number of the corresponding transaction $\mathcal{T}$.

Definition 4.2. *(Authorized Transaction) An authorized transaction $\mathcal{T}$, depicted by the figure 4.1, is defined as a pair $\langle T,\ D\rangle$, where $T = \langle s,\ id,\ I,\ O,\ d,\rangle$ is the underlying proposed transaction and D represents the set of $2f + 1$ server node signatures collected during the authorization of the transaction T.*

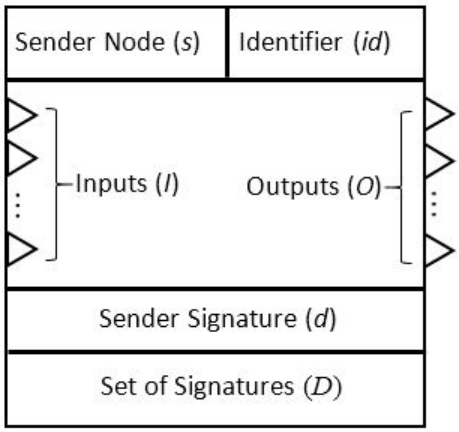

Figure 4.1: Transaction Model

Definition 4.2.1. *(Authorized Transaction) A transaction $\mathcal{T} : \langle T,\ D\rangle$ is said to be authorized if D contains $2f + 1$ valid signatures from distinct server nodes.*

$$authorized(\langle T, D\rangle) \triangleq (\forall (s_j, d_j) \in D,\ \ Verify_{pk_j}(T, d_j) = true)\ \ \wedge\ \ |D| = 2f + 1$$

Using the transaction model discussed above, each server s_i maintains the following data structures for the local image of the distributed transaction history.

1. **Authorized Transactions ($\mathcal{T}^A$)** is a set of transactions $\mathcal{T}$ authorized according to the server node s_i.

2. **New Transactions ($\mathcal{T}^N$)** is set of 3-tuples $\langle \mathcal{T},\ s,\ \theta \rangle$, where θ is the status of authorized transaction $\mathcal{T}$ with respect to the server node s involved in the transaction as a sender or receiver. When a transaction $\mathcal{T}$ is authorized, the status $\theta = new$ for the transaction $\mathcal{T}$ with respect to each server node s_j involved in it. When the transaction is committed to the permanent record by the corresponding server node, the status θ changes and is no longer *new*. The discussion of the details of this commitment process is given in Chapter 5.

Note that the each transaction $\mathcal{T}$ has one entry in the authorized transaction set $\mathcal{T}^A$, whereas it can have multiple independent entries in the new transaction set $\mathcal{T}^N$ because each transaction $\mathcal{T}$ has at least 2 server nodes involved, i.e. a sender and a receiver.

For simplifying the notations, we have included the authorized transactions $\mathcal{T}$ in the definition of new transaction set $\mathcal{T}^N$. However, for implementation purposes, it can be replaced by a pair $\langle s', id \rangle$ where s' is the sender node and id is the sequence number of the corresponding transaction $\mathcal{T}$. The $\langle s', id \rangle$ pair can be used to retrieve the corresponding authorized transaction from the authorized transaction set $\mathcal{T}^A$.

4.2 Transaction Validations

In order to ensure that a server's transactions are processed in order, we require that the server provide consecutive sequence numbers for its transaction. This requirement is enforced by having the server include the last authorized transaction

initiated by the server.

When a sender node s_i sends a proposed transaction $T : \langle s_i,\ id_c,\ I,\ O,\ d_i \rangle$ to the server nodes for authorization, it is accompanied by the last authorized transaction $\mathcal{T}_p$ for the sender node s_i. Note that the definition of the input transaction set I includes the authorized input transactions that can be used for validations by a processing server node even if the corresponding transactions are not present in its local authorized transaction set $\mathcal{T}^A$. For implementation purposes, the authorized input transactions can be required to accompany the transaction processing request, which serves the equivalent objective.

The transaction T is considered valid by a non-faulty server node if all the following conditions hold.

1. The transaction request parameters $\langle T, \mathcal{T}_p \rangle$ are structurally *well-formed*, i.e.

   (a) The transaction $\mathcal{T}_p$ is an authorized transaction with s_i as the sender node and sequence number used in $\mathcal{T}_p$ is one less than the sequence number id_c used in T

   (b) All the input transactions, in I, are authorized transactions received by the node s_i

   (c) The sum of amounts in the input set I is equal to the sum of amounts in the output set O

   (d) The signature d_i is the valid signature for the sender for s_i over the transaction $\langle s_i,\ id_c,\ I,\ O \rangle$

2. The transaction T is *functionally valid*, i.e.

   (a) All the input transactions, in I, are unspent transactions as per the authorized transaction set $\mathcal{T}^A$ of the processing node

(b) The sequence number id_c used in T is greater than all the sequence numbers used by the node s_i in previous transactions as per the authorized transaction set $\mathcal{T}^A$ of the processing node

Note that the structural validations depend upon the request parameters $\langle T, \mathcal{T}_p \rangle$ only, while the functional validations depend upon the authorized transaction set $\mathcal{T}^A$. Therefore, if a transaction is structurally valid according to one non-faulty node, it will be structurally valid according to all non-faulty nodes. However, the functional validity of a transaction may vary among non-faulty nodes depending upon the transactions in their authorized transaction set $\mathcal{T}^A$.

Also, the previous transaction $\mathcal{T}_p$ is unique with respect to each proposed transaction T because according to validation 1(a), it is required to be an authorized transaction with exactly the previous sequence number, and according to validation 2(b), the sequence number in T is required to be new, which will not be valid if $\mathcal{T}_p$ is used multiple times.

We now formally define the individual validations as building blocks for the structural and functional validations as mentioned above.

Definition 4.3. *(Amounts Validation) A transaction $T : \langle s,\ id,\ I,\ O,\ d \rangle$ is said to be valid in terms of the transaction amounts if the sum of input amounts and the sum of output amounts are equal.*

$$validAmounts(I, O) \triangleq \sum_{x_j \in O} x_j = \sum_{(T,j) \in I} T.O[j].x$$

Definition 4.4. *(Previous Transaction Validation) A transaction $\mathcal{T}_p : \langle T_p,\ D_p \rangle$ is said to be the valid previous transaction corresponding to the proposed transaction T if $\mathcal{T}_p$ is an authorized transaction and and $T.id = T_p.id + 1$ and both transactions were*

initiated by the same node.

$$validPrevTxn(T, \mathcal{T}_p) \triangleq authorized(\mathcal{T}_p) \ \wedge \ (T.s = \mathcal{T}_p.s) \ \wedge \ (T.id = \mathcal{T}_p.id + 1)$$

Definition 4.5. *(Own Inputs Validation) A node s is said to have included only its own authorized input transactions in the proposed transaction $T : \langle s,\ id,\ I,\ O,\ d \rangle$ if all the authorized transaction inputs in I correspond to the node s.*

$$ownInputs(s, I) \triangleq \forall (\mathcal{T}, j) \in I,\ authorized(\mathcal{T}) \ \wedge \ (\mathcal{T}.O[j].s_j = s)$$

Definition 4.6. *(Unspent Input Validation) A set of inputs I in a proposed transaction T_c, initiated by the node s, is said to be unspent if none of the input transactions has been used as an input for another transaction by the node s.*

$$unspent(s, I) \triangleq \forall (T, j) \in I,\ (T, j) \notin \{\mathcal{T}^A[s].I\}$$

Definition 4.7. *(Sequence Number Validation) A sequence number id_c in a proposed transaction T, initiated by the node s, is said to be valid if it is greater than the maximum sequence number used by the node s in previously authorized transactions.*

$$validId(s, id_c) \triangleq id_c > max\{id : id \in \mathcal{T}^A[s]\}$$

4.2.1 Structural Validations

When a server node s receives a transaction for authorization from the sender node s_j, the node s first validates the structural integrity of the transaction, i.e. ascertains whether it is well-formed. If the transaction is structurally invalid, the node s rejects the transaction. The corresponding validation is defined as follows.

Definition 4.8. *(Well-formed Transaction) A transaction T proposed by a node s_j, along with previous transaction evidence $\mathcal{T}_p$, is said to be well-formed if sender's*

signature is valid, all the inputs in T are s_j's own authorized transactions, and the transaction amounts and the previous transaction are valid.

$$wellFormed(T, \mathcal{T}_p) \triangleq (Verify_{pk_j}(T, T_c.d) = true) \ \wedge \ ownInputs(T.s, T.I) \\ \wedge \ validPrevTxn(T, \mathcal{T}_p) \ \wedge \ validAmounts(T.I, \ T.O)$$

4.2.2 Functional Validations

After validating the structural integrity of the transaction processing request using definition 4.8, the server node s performs functional validations to verify that the transaction is not a double-spend transaction using definition 4.6. In doing the validation, the server node s uses its local knowledge of the authorized transaction set $\mathcal{T}^A$. So it is possible that a non-faulty server node will consider a double-spend transaction valid because it may not be aware of the conflicting transaction. Nevertheless, if a transaction is a double-spend transaction, it will not be considered valid by enough servers required for authorizing it.

Similarly, the server node s also verifies, using its local knowledge, that the sequence number has not been used previously using definition 4.7.

Definition 4.9. *(Functionally Valid Transaction) A transaction T proposed by a node s is said to be functionally valid if all the inputs are unspent and the sequence number is valid.*

$$validTxn(T) \triangleq \ unspent(T.s, \ T.I) \ \wedge \ validId(T.s, \ T.id)$$

4.3 Transaction Protocol

We now present the transaction authorization protocol used by the server nodes to propose, authorize and accept transactions.

4.3.1 Sender Protocol

Without loss of generality, let us consider a transaction submitted by customer c_1 using interface node e_1 at bank B_1 for payment of amount x to customer c_2 at bank B_2. The interface node e_1 will send the transaction to the server node s_1 for processing. The server node s_1 will construct the transaction in the form of a pair (I, O) as the input and output transaction sets respectively, where one of the outputs in O will correspond to a server node s_2 for amount x, and initiate the protocol depicted in figure 4.2 for transaction authorization.

Algorithm 1 Sender Node s_i

```
1:  initialize:
2:      id_c ← 0    /* Current identifier value for node s_i */

3:  /* Process transaction specified as input set I and output set O*/
4:  function PROCESS-TXN(I, O)
5:      id_c ← id_c + 1
6:      T ← ⟨s_i, id_c, I, O, d_i⟩, where d_i = Sign_{sk_i}(⟨s_i, id_c, I, O⟩)

7:      /* Phase 1: Send txn authorization request along with last txn */
8:      send (PROCESS, T, T^A[s_i, id_c − 1]) to all server nodes
9:      initialize: D ← ∅
10:     repeat
11:         on receive (SIGNED, T, d_j) from server s_j do
12:             if Verify_{pk_j}(T, d_j) = true then
13:                 Add (s_j, d_j) to set D
14:             end if
15:         end
16:     until |D| = 2f + 1

17:     /* Phase 2: Send txn commit request with signature set D */
18:     send (COMMIT, T, D) to all server nodes
19:     receive (COMMITTED, T, D) from 2f + 1 servers
20:     COMMIT-TXN(⟨T, D⟩)
21:     return ⟨T, D⟩
22: end function
```

Figure 4.2: Transaction Sender Protocol

The sender node s_i maintains a current identifier counter value (id_c) to assign the id value to its transactions. The function PROCESS-TXN takes the transaction in the form of (I, O) as input and returns the authorized transaction (T, D) as output. It

begins by constructing the transaction (lines 5-6) and processes the transaction in 2 phases:

- **Phase 1 (Authorization):** Sender node s_i sends the authorization request to all servers (line 8). The transaction authorization request is accompanied by the previous authorized transaction (with identifier $id_c - 1$) as a proof that the node s_i is using sequential identifier values. Sender node waits for $2f + 1$ valid node signatures (lines 10-16). Once the transaction is authorized, s_i proceeds to phase 2.

- **Phase 2 (Commit):** Sender node s_i sends the transaction commit request along with the set D containing $2f + 1$ signatures to all server nodes (line 18) and waits for acknowledgements from $2f + 1$ server nodes (line 19). Once the acknowledgements are received, s_i commits the transaction (line 20) using the COMMIT-TXN procedure depicted in figure 4.5 and returns it (line 21) for internal processing, if any.

Note that the transaction is authorized at the end of phase 1 from the sender node's perspective, but only the sender node has the proof of authorization, i.e. the signature set D, at the end of phase 1. Phase 2 is used to communicate the signature set D to the network so that it can be read by the receiver node without depending on the sender node.

4.3.2 Receiver Protocol

In an ideal scenario, we would expect the sender node to communicate the committed transaction to the receiver node. However, since the sender node could be faulty, we do not assume that such a communication will take place in a timely manner. Therefore, we define a protocol, depicted in figure 4.3, for the receiver node, say

s_k, to read and process incoming transactions from the network.

Algorithm 2 Receiver Node s_k

```
1:  /* Get new transactions for the node s_k from the network */
2:  procedure GetNewTxns
3:      send (newTxns) to all server nodes
4:      initialize: T^R ← ∅, R ← ∅
5:      repeat
6:          on receive (txns, T*) from server s_j do
7:              T^R ← T^R ∪ T*
8:              Add s_j to set R
9:          end
10:     until |R| = 2f + 1

11:     for each T ∈ T^R s.t. T ∉ T^A ∧ authorized(T) do
12:         Commit-Txn(T)
13:     end for
14: end procedure
```

Figure 4.3: Transaction Receiver Protocol

The receiver node s_k sends the read request for new transactions to all server nodes (line 3) and processes responses from $2f + 1$ nodes (lines 5-10). Node s_k compiles a transaction set $\mathcal{T}^R$, which is the union of all the transaction sets $\mathcal{T}^*$ returned by the server nodes (line 6), and a set R to keep track of the server nodes that respond in order to satisfy the termination condition of $2f + 1$ responses (line 10). Once it receives $2f + 1$ responses, node s_k commits, using the COMMIT-TXN procedure, all authorized transactions that are not already committed (lines 11-13).

Firstly, we know that at most f nodes can be faulty, and as a result s_k is guaranteed to receive $2f + 1$ responses which guarantees termination. Now, let a set $\mathcal{T}^R$ denote the set of transaction authorized at the time when the node s_k initiates the protocol such that each transaction in $\mathcal{T}^R$ is initiated by a non-faulty node and has s_k as an output receiver. We know that each such transaction is written back to $2f + 1$ server nodes in the network by its sender node, i.e. at least $f + 1$ non-faulty nodes have the authorized transaction with the signature set. Therefore, s_k is guaranteed to receive each transaction from at least one server node during the processing of $2f + 1$

responses. So, s_k is guaranteed to have received all the transactions in $\mathcal{T}^R$ at the end of the receiver protocol.

4.3.3 Server Node Protocol

We now discuss the protocol used by server nodes, depicted in figure 4.4, to process transaction requests from sender and receiver nodes. Consider a server node s_j that receives the processing request.

Algorithm 3 Server Node s_j

```
 1: /* Process transaction authorization request from a sender node */
 2: on receive (PROCESS, T, T_p) from server node s_i do
 3:     if wellFormed(T, T_p) then
 4:         /* Commit missing transactions, if any */
 5:         COMMIT-TXN(T_p) if T_p ∉ T^A
 6:         for each T ∈ T.I s.t. T ∉ T^A do
 7:             COMMIT-TXN(T)
 8:         end for
 9:         /* Sign the transaction if it is well-formed and valid */
10:         if validTxn(T) then
11:             d_j ← Sign_{sk_j}(T)
12:             Add ⟨T, ∅⟩ to T^A
13:             send (SIGNED, T, d_j) to server node s_i
14:         end if
15:     end if
16: end

17: /* Process transaction commit request from a sender node */
18: on receive (COMMIT, T, D) from server node s_i do
19:     if authorized(⟨T, D⟩) then
20:         COMMIT-TXN(⟨T, D⟩)
21:         send (COMMITTED, T, D) to server node s_i
22:     end if
23: end

24: /* Process new transactions request from a receiver node */
25: on receive (NEWTXNS) from server node s_k do
26:     T^R ← {T : ⟨T, s_k, new⟩ ∈ T^N ∧ T.s^I ≠ s_k}
27:     send (TXNS, T^R) to node s_k
28: end
```

Figure 4.4: Server Node Protocol for Transaction Processing

The transaction authorization protocol is represented in lines 1-23.

- **Phase 1 (Authorization)**: When the node s_j receives the transaction autho-

rization request for transaction T with previous transaction $\mathcal{T}_p$ from the node s_i (line 2), s_j processes it only if the transaction request is well-formed as per the definition 4.8 (line 3). If either the previous transaction $\mathcal{T}_p$ or any of the input transactions is not a committed transaction, s_j commits those transactions for further processing (lines 5-8). Now, if the transaction is functionally valid (line 10), s_j saves the transaction (line 12) and sends the signed transaction to the sender node (line 13).

- **Phase 2 (Commit)**: When the node s_j receives the transaction commit request for transaction T with signature set D from the node s_i (line 18), s_j verifies whether the transaction is authorized (line 19). If the validation succeeds, s_j commits the transaction and responds to the sender node.

The new transaction request protocol for the server nodes is represented in lines 25-28. The node s_j responds to requests from server nodes s_k (line 25) by constructing the set $\mathcal{T}^R$ of new transactions that involve the node s_k as a receiver (line 26) and sending the set to the node s_k in response (line 27).

The transaction commit procedure COMMIT-TXN is depicted in figure 4.5. When a transaction is committed, it is stored in two transaction sets: authorized transactions (line 6) and new transactions (line 7). The new transaction set stores one copy of the transaction for each node involved in the transaction as input or output. The procedure also ensures that commit is not performed for already committed transactions.

Server nodes commit missing transactions (fig 4.4, lines 5-8) for two reasons. Firstly, it helps them improve their local knowledge of the authorized transactions in the network. Secondly, if a faulty sender node terminates sender protocol at the end of phase 1, the transaction will not be committed by any server node in the

Algorithm 4 Server Node s_i

```
1: initialize:
2:       T^A ← ∅, T^N ← ∅

3: /* Store the transaction in authorized and new transaction sets */
4: procedure COMMIT-TXN(⟨T, D⟩)
5:     if T ∉ T^A ∨ ⟨T, ∅⟩ ∈ T^A then
6:         Add ⟨T, D⟩ to T^A
7:         Add {⟨T, s, new⟩ : s ∈ T.O ∪ T.I} to T^N
8:     end if
9: end procedure
```

Figure 4.5: Transaction Commit Procedure

network. As a result, it will not be available as a new transaction for any server node and consequently, the receiver will not receive the transaction during the receiver protocol. However, we know that the sender node will be required to send the this authorized transaction while processing the next transaction, and each server node can commit the transaction at this time and guarantee progress.

4.4 Analysis

For our analysis of the transaction processing protocol, we consider the following properties:

- *Theorem 4.1:* If non-faulty sender node initiates a transaction, the sender protocol is guaranteed to terminate.
- *Theorem 4.2:* All transactions initiated by non-faulty sender nodes will be accepted as authorized transactions by their corresponding non-faulty receiver nodes.
- *Theorem 4.3:* A transaction is authorized if and only if it is structurally and functionally valid.

Theorem 4.1. *If non-faulty sender node initiates a transaction, the sender protocol*

is guaranteed to terminate.

Proof. If the sender node is non-faulty, then its transaction must be valid because a non-faulty sender node validates the transaction before sending it to the network and the sender node has complete knowledge of all the transactions sent by it in the past, which prevents it from accidentally constructing a double-spend transactions. Therefore, we can say that the transaction must be structurally and functionally valid as per definitions 4.8 and 4.9 respectively.

So, as per algorithm 3 (fig 4.4), each non-faulty server node will sign the transaction. We know that there are at most f faulty nodes, i.e. there are at least $2f + 1$ non-faulty nodes in the network, so the sender is guaranteed to receive at least $2f + 1$ signatures, which may or may not contain signatures from faulty nodes, in phase 1. Therefore, the sender will be able to complete phase 1 and proceed to phase 2.

We know that the sender node verifies each received signature during construction of the signature set D (fig 4.2, lines 12-13), so the signature set D must be valid and as a result, it will be accepted as an authorized transaction, as per definition 4.2.1, by each non-faulty server node during phase 2. So, the sender will receive at least $2f + 1$ acknowledgements during phase 2 and terminate the protocol. □

Theorem 4.2. *All transactions initiated by non-faulty sender nodes will be accepted as authorized transactions by their corresponding non-faulty receiver nodes.*

Proof. Using the argument from Theorem 4.1, we can say that if a non-faulty sender node initiates a transaction T, the sender is guaranteed to complete phase 2 and terminate. This means that the transaction T will be authorized at the end of phase 1, and the authorized transaction will be acknowledged by at least $2f + 1$ nodes during phase 2. As per algorithm 3 (fig 4.4), during phase 2, a non-faulty server node commits the transaction T before acknowledging it (lines 20-21). Since at most

f nodes can be faulty, we can conclude that the transaction will be committed by at least $f + 1$ non-faulty nodes at the end of phase 2. We consider the worst case that when the sender node received $2f + 1$ acknowledgements during phase 2, f of them were sent by faulty nodes which did not commit the transaction, and exactly $f + 1$ non-faulty nodes committed the transaction T.

As per algorithm 2 (fig 4.3), when a non-faulty receiver node requests new transactions from the network, it only processes the first arrived $2f + 1$ responses before termination. If a transaction T is committed by $f + 1$ non-faulty nodes and a non-faulty receiver, say s_r, of the transaction T executes the receiver protocol, the node s_r is guaranteed to receive a response from at least one of those $f + 1$ non-faulty nodes, say s_i, as a part of the first arrived $2f + 1$ responses. Since the s_i is non-faulty, it will communicate the transaction T in its response. As the transaction T is authorized, the receiver s_r will accept the transaction as authorized as per definition 4.2.1. □

For the proof of *Theorem 4.3*, we consider the following lemmas.

- *Lemma 4.1:* If a transaction is authorized, then it must be structurally valid.
- *Lemma 4.2:* A sender node s_i completes phase 1 for a transaction T, with sequence number id_c, if and only if each of the preceding transactions initiated by the node s_i, with sequence numbers $id < id_c$, are authorized and are committed by at least $f + 1$ non-faulty nodes.
- *Lemma 4.3:* If a transaction is authorized, then it must be functionally valid.

Lemma 4.1. *If a transaction $\mathcal{T}$ is authorized, then it must be structurally valid.*

Proof. We know that each non-faulty server node performs structural validations using the request parameters only, and as a result, if a transaction is structurally invalid according to definition 4.8, it will be invalid according to every non-faulty

server node (fig 4.4, line 3). Therefore, a structurally invalid transaction will be rejected by each non-faulty server node. Since there are at most f faulty nodes and at least $2f+1$ non-faulty nodes, the structurally invalid transaction can not be signed by more than f node (all faulty). Hence, a structurally invalid transaction can not be authorized as per definition 4.2.1. □

Lemma 4.2. *A sender node s_i completes phase 1 for a transaction T, with sequence number id_c, if and only if each of the preceding transactions initiated by the node s_i, with sequence numbers $id < id_c$, are authorized and are committed by at least $f + 1$ non-faulty nodes.*

Proof. We know that the sender node s_i is required to include the previous authorized transaction $\mathcal{T}_p$, with sequence number $id_c - 1$, as a transaction processing request parameter in phase 1 of the processing of the transaction T, and it is validated during the structural validations as per definitions 4.8 and 4.4. As per the hypothesis, the sender node s_i is able to complete phase 1 for a transaction T, which means that it has received $2f + 1$ signatures and is considered authorized as per definition 4.2.1. From lemma 4.1, we know that if the transaction T is authorized, it must be structurally valid. Therefore, the transaction $\mathcal{T}_p$ must be authorized and its sequence number must be $id_c - 1$.

If the transaction $\mathcal{T}_p$ is authorized, the sender node s_i must have completed phase 1 of the sender protocol for it. Let us consider the two cases for the processing of the transaction $\mathcal{T}_p$.

- *Case 1:* If the sender node s_i is non-faulty, we know from the arguments in theorems 4.1 and 4.2 that the sender node s_i must have completed phase 2 for the the transaction $\mathcal{T}_p$ and it must be committed by at least $f + 1$ non-faulty nodes.

- *Case 2:* If the sender node s_i is faulty, it may not have completed or even initiated phase 2 for the transaction $\mathcal{T}_p$. For the worst case consideration, we assume that the sender node s_i did not initiate phase 2 for the transaction $\mathcal{T}_p$. As a result, no non-faulty node can have committed the transaction $\mathcal{T}_p$ because transaction commit is a part of phase 2 processing for the server nodes.

  As discussed earlier, the sender node s_i sends the transaction $\mathcal{T}_p$ as a request parameter in the succeeding transaction T and the transaction T is authorized. This means that the transaction T has been signed by at least $f + 1$ non-faulty nodes. We know from algorithm 3 (fig 4.4), that a non-faulty server node commits the previous transaction $\mathcal{T}_p$ (line 5) before signing the transaction T. This means that each of the (at least) $f + 1$ non-faulty nodes would have committed the authorized transaction $\mathcal{T}_p$ before signing the transaction T, and as a result, the transaction $\mathcal{T}_p$ must be committed by at least $f + 1$ non-faulty nodes.

Therefore, we have established that if a transaction T, sent by server node s_i with sequence number id_c, is authorized, then the previous authorized transaction $\mathcal{T}_p$, sent by server node s_i with sequence number $id_c - 1$, must be committed by at least $f + 1$ non-faulty nodes.

We can apply the same argument to say that the transaction $\mathcal{T}'_p$, sent by server node s_i with sequence number $id_c - 2$, must be committed by at least $f + 1$ non-faulty nodes because its succeeding transaction $\mathcal{T}_p$ is authorized and the sender node s_i must have completed phase 1 for it. Therefore, using induction, we can conclude that each of the preceding transaction sent by sender node s_i, with sequence numbers $id < id_c$ must be committed by at least $f + 1$ non-faulty nodes. □

Lemma 4.3. *If a transaction $\mathcal{T}$ is authorized, then it must be functionally valid.*

Proof. As per functional validation definition 4.9, a transaction $\mathcal{T}$, initiated by the sender node s_i, can be functionally invalid in two cases.

- *Case 1:* The sequence number id_c, used in the transaction $\mathcal{T}$, is invalid.

  We know that the sender node s_i is required to include the previous authorized transaction $\mathcal{T}_p$, with sequence number $id_c - 1$, as a transaction processing request parameter in phase 1 of the processing of the transaction T, and it is validated during the structural validations as per definitions 4.8 and 4.4. As the transaction $\mathcal{T}$ is authorized, it must be structurally valid according to lemma 4.1, and as a result, sequence number id_c in the transaction $\mathcal{T}$ must be one greater than the sequence number in the transaction $\mathcal{T}_p$. So, the sequence number id_c can not be arbitrarily large. It can only be invalid if it is not greater than the largest sequence number used node s_i in previous transactions.

  As the transaction $\mathcal{T}$ is authorized, the sender must have completed phase 1 for it, and using lemma 4.2, we can say that each of the preceding transaction sent by sender node s_i, with sequence numbers $id < id_c$ must be committed by at least $f + 1$ non-faulty nodes. Therefore, if the sequence number id_c is not the largest sequence number used node s_i so far, there must be another authorized transaction with the same sequence number id_c, and it must be committed by at least $f + 1$ non-faulty nodes. So, at least $f + 1$ non-faulty nodes would have detected the transaction $\mathcal{T}$ as functionally invalid and rejected it. As a result, the transaction $\mathcal{T}$ can only be signed by at most $2f$ nodes, which is not sufficient for the transaction $\mathcal{T}$ to be authorized as per definition 4.2.1. Hence, if the transaction $\mathcal{T}$ is authorized, its sequence number id_c must be valid.

- *Case 2:* The transaction $\mathcal{T}$ is a *double-spend* transaction, i.e. at least of the inputs used in $\mathcal{T}$ has been used as an input by another transaction.

Let us consider that there exists a transaction $\mathcal{T}'$ initiated by the node s_i that used an input which has been re-used by the transaction $\mathcal{T}$. If the sequence number of $\mathcal{T}'$ is greater than or equal to the sequence number of $\mathcal{T}$, then the transaction $\mathcal{T}$ will not be authorized as per the argument in case 1. Alternatively, if the sequence number used in $\mathcal{T}'$ is less than the sequence number used in $\mathcal{T}$, then using lemma 4.2, we can say that the transaction $\mathcal{T}'$ is authorized and is committed by at least $2f+1$ server nodes prior to the functional validation of the transaction $\mathcal{T}$. This means that the transaction $\mathcal{T}$ will be detected as a double-spend transaction by at least $f+1$ non-faulty nodes, and can be signed by at most $2f$ nodes, which is not sufficient for the transaction $\mathcal{T}$ to be authorized as per definition 4.2.1. Hence, if a transaction $\mathcal{T}$ is authorized, it can not be a *double-spend* transaction.

Hence, a functionally invalid transaction can not be authorized as per definition 4.2.1. □

Theorem 4.3. *A transaction is authorized if and only if it is structurally and functionally valid.*

Proof. Combining lemmas 4.1 and 4.3, we can say that if a transaction is authorized, it must be structurally and functionally valid. Conversely, if a transaction is either structurally or functionally invalid, and the transaction is authorized, it would lead to a contradiction of the the hypothesis in lemmas 4.1 or 4.3 respectively. □

Corollary 4.3.1. *A double-spend transaction can not be authorized.*

Proof. We know that a double spend transaction is considered functionally invalid as per definitions 4.9 and 4.6. Therefore, as per theorem 4.3, it can not be authorized. □

4.5 Efficiency Improvement Ideas

As mentioned earlier, the authorized transactions $\mathcal{T}$ are included in the input transactions I of a proposed transaction and in the new transaction set $\mathcal{T}^N$. For efficient implementation, these sets can be defined with pair $\langle s, id \rangle$ where s is the sender node and id is the sequence number of the corresponding transaction $\mathcal{T}$. Corresponding validations can be suitably modified to retrieve the authorized transactions, before validation, from the authorized transaction set $\mathcal{T}^A$ using the $\langle s, id \rangle$ pair.

Note that the size of a pair $\langle s, id \rangle$ is constant, whereas the size of the corresponding transaction depends on the number of inputs, outputs and the server node signatures. As a result, the size of the pair $\langle s, id \rangle$ is much smaller as compared to the transaction data structure.

4.5.1 Receiver Protocol

We now discuss the efficiency concerns and potential improvements concerning the receiver protocol depicted in figure 4.3 and figure 4.4 (lines 25-28).

As discussed in subsection 4.3.2, using the protocol, a node is guaranteed to receive all the newly authorized transactions wherein it is the receiver. However, it can receive all the transactions again that received during the last execution of the protocol along with the transactions that were authorized after the last execution.

We can control this redundancy in successive responses to the receiver protocol by modifying the protocol as follows.

- The receiver node s_i constructs a known transactions set $\mathcal{T}^K$ of $\langle s, id \rangle$ pairs using its new transaction set $\mathcal{T}^N$ such that all the corresponding transactions involved s_i as a receiver.

- The receiver node s_i sends the set $\mathcal{T}^K$ as a request parameter in the receiver

protocol to all server nodes and collects $2f + 1$ responses before termination.

- Each server node processes the request by constructing the response transaction set $\mathcal{T}^R$ similar to the construction shown in figure 4.4 (line 26) with a modification that it excludes all the transactions that have been included in the request parameter $\mathcal{T}^K$, and sends the response set $\mathcal{T}^R$ to the node s_i.

- The receiver process the received transactions in the same way as shown in figure 4.3.

Now, observe that the receiver's known transactions set $\mathcal{T}^K$ may contain a lot of transactions, but they are specified in the minimal possible way using a $\langle s, id \rangle$ pair, and as a result, the size of the set $\mathcal{T}^K$ will be smaller than set of corresponding transactions. Using the set $\mathcal{T}^K$, each server node reduces the size of its response set and the number of transactions received and processed by the receiver node during the protocol will correspond to the newly authorized transactions only.

Future work can introduce new model parameters to further optimize the protocol.

Chapter 5

AUDIT STRUCTURE

PCAOB Auditing Standards [22] define the objective of audit of financial statements as *"the expression of an opinion on the fairness with which they present, in all material respects, financial position, results of operations, and its cash flows in conformity with generally accepted accounting principles."* Audit procedures include assessment of internal controls over reporting as well as substantive procedures for gathering evidence.

Before we discuss our audit structure, let us discuss an audit procedure in context of a financial transaction processing system. We assume that, given a server node s_i and set $\mathcal{T}_i^C$ of transaction involving s_i as input or output, there exists a deterministic summarizing function SUMMARIZE : $(s_i, \mathcal{T}_i^C) \rightarrow \sigma$ that produces a summary σ of the transaction set $\mathcal{T}_i^A$ with respect to the node s_i, and the function is known to all nodes in the network. By virtue of this assumption, we defer the definition of the summarizing function to the financial experts so that it conforms to the requirements of substantive procedures to be used in the financial audit.

Figure 5.1 illustrates some potential examples of the transaction set summary. Given the set of transactions shown in figure, where transactions 1-5 are sent by s_1 and transactions 6-10 are received by s_1, the possible summaries for the node s_1 can be as follows.

- $\sigma_1 = \mathit{Sum}(\mathit{Received\ by}\ s_1) - \mathit{Sum}(\mathit{Sent\ by}\ s_1)$
- $\sigma_2 = \langle\ \mathit{Sum}(\mathit{Sent\ by}\ s_1),\ \mathit{Sum}(\mathit{Received\ by}\ s_1)\ \rangle$
- $\sigma_3 = \{\langle s_i, \mathit{Total}\rangle : \mathit{Total} = \mathit{Sum}(\mathit{Received\ by}\ s_1\ \mathit{from}\ s_i) - \mathit{Sum}(\mathit{Sent\ by}\ s_1\ \mathit{to}\ s_i)\}$

#	Sender	Receiver	Amount
1	s_1	s_2	100
2	s_1	s_2	75
3	s_1	s_3	60
4	s_1	s_3	80
5	s_1	s_3	110
6	s_2	s_1	80
7	s_2	s_1	90
8	s_2	s_1	150
9	s_3	s_1	95
10	s_3	s_1	50

σ_1	Total	40

σ_2	Sent	425
	Received	465

σ_3	Node	Total
	s_2	145
	s_3	-105

σ_4	Node	Sent	Received
	s_2	175	320
	s_3	250	145

Figure 5.1: Transaction Set Summary Examples

- $\sigma_4 = \{ \langle s_i, \ Sent, \ Received \rangle : Received = Sum(Received \ by \ s_1 \ from \ s_i), Sent = Sum(Sent \ by \ s_1 \ to \ s_i)\}$

Let $\mathcal{T}^C$ be the complete set of authorized transactions for the entire network, i.e. a union of all the authorized transactions sets of all server nodes in the network, at the time of the audit. And let $\mathcal{T}_i^C$ be the set of authorized transactions that involve the node s_i as a sender or a receiver at the time of the audit. We define the audit procedure as follows.

Definition 5.1. *(Audit) Given a complete set of authorized transactions $\mathcal{T}^C$, a set of transactions $\mathcal{T}_i^C$ involving node s_i as a sender or receiver and a summary value σ at a particular time, an audit of the node s_i is defined as a procedure of verifying the following criteria.*

1. $\sigma = \text{SUMMARIZE}(s_i, \mathcal{T}_i^C)$, *i.e. the value σ corresponds to the set $\mathcal{T}_i^C$ for the node s_i as per the summarizing function* SUMMARIZE,

2. *the set $\mathcal{T}_i^C$ is* complete, *i.e. it contains all the transactions in $\mathcal{T}^A$ involving node s_i as a sender or receiver, and*

3. *the set $\mathcal{T}_i^C$ is* sound, *i.e. it does not contain any transaction not in $\mathcal{T}^C$*

Let us analyze this audit procedure in absence of any audit structure. We discuss naive approaches for the analysis, but we use the minimal possible lower bounds of processing time required in order to be representative of an optimal solution. We consider two scenarios as follows.

1. *Bitcoin Blockchain:* In Bitcoin, the network maintains a common Blockchain structure and the Blockchain is expected to contain all the completed transactions at any given time. So, we assume that downloading the Blockchain from the network will provide the auditor with the correct transactions set $\mathcal{T}^C$, which can be used as a reference for verifying the completeness and soundness of the set $\mathcal{T}_i^C$, provided by node s_i, and it would require at least $\Omega(|\mathcal{T}_i^C|)$ time. Now, the auditor can verify the summary σ, provided by node s_i, with respect to the set $\mathcal{T}_i^C$ and it would also require at least $\Omega(|\mathcal{T}_i^C|)$ time. Therefore, the audit procedure on the Bitcoin Blockchain can be expected to take at least $\Omega(|\mathcal{T}_i^C|)$ time. Additionally, an auditor will be required to query the network to download the Blockchain in order to validate the completeness and soundness of the set $\mathcal{T}_i^C$ provided by the node s_i.

2. *Our Transaction Model:* In our model, none of the server nodes are expected to maintain the complete set of authorized transactions. So, in order to compile the set $\mathcal{T}^C$, the auditor needs to request authorized transaction sets from all the nodes in the network and construct a union of all the responses. The auditor is expected to receive at least $2f + 1$ responses as f can be faulty, i.e. $O(f)$ responses each of size $O(|\mathcal{T}^C|) \approx O(|\mathcal{T}_i^C|)$. The union operation will take at least $\Omega(|\mathcal{T}_i^C|)$ time. Then, the auditor can perform the verification of the set $\mathcal{T}_i^C$ and the summary σ, provided by node s_i, which would again require at least $\Omega(|\mathcal{T}_i^C|)$ time. So, the audit procedure can be expected to take at least $\Omega(|\mathcal{T}_i^C|)$ time.

Again, the auditor will be required to query the network in order to compile the set $\mathcal{T}^C$ so as to validate the completeness and soundness of the set $\mathcal{T}_i^C$ provided by the node s_i.

Therefore, in either scenario, the audit procedure time complexity depends upon the number of transactions in the set $\mathcal{T}_i^C$, which can be significantly large for a bank and is very inefficient for an audit. Furthermore, the auditor will be required to query the network to construct the set $\mathcal{T}^C$ as an accurate representation of the set of transactions in the system that can be used as a point of reference in validating the completeness and soundness of the set $\mathcal{T}_i^C$ provided by the node s_i. We also note that validating the soundness of the set $\mathcal{T}_i^C$ can be done without constructing the set $\mathcal{T}^C$ in presence of verifiable evidence, e.g. set of server signatures in our model, however, the validation of the completeness of the set $\mathcal{T}_i^C$ requires set $\mathcal{T}^C$ to ensure that the node s_i has not withheld any transactions from the auditor.

We propose an audit trail model that (1) provides the auditor with the guarantee that the set of transactions provided by the corresponding node is sound and complete, without querying any other node in the network, and (2) allows the audit procedure to be completed in time independent of the number of transactions. This would significantly improve the efficiency of the audit procedure, thereby reducing the cost of an external audit.

5.1 Audit Trail Model

Financial audit is a periodic activity, where the periodicity can be monthly, quarterly, yearly or some other established time period. And therefore, the audit of a particular period needs to distinguish the transactions that occurred during the concerned period from the ones that occurred during other periods. We do not assume a specific periodicity or the start/end parameters of those periods in our model. We as-

sume that the participating entities will define time-periods as per their requirements and utilize our model accordingly.

Each node s_i constructs a linear *audit trail* consisting of only the transactions that involve node s_i as either the sender or a receiver, and this audit trail can be used as the node s_i's transaction set $\mathcal{T}_i^C$ during an audit. The audit trail is sub-divided into a connected sequence of *audit checkpoints*, each of which represents a consolidated record of financial activity, both the transactions and the summary, for the concerned node during a specified time-period.

To illustrate the idea, we use an example. Let us consider a scenario where the bank B is subject to annual auditing, i.e. at the end of each year, the bank B is required to publish an audited summary of its financial activity for the year. So, for the year-end audit, the bank B is required to provide the auditor with the set of transactions that occurred during the year involving it as either the sender or a beneficiary. Let us assume that the bank B decides to perform daily checkpointing, i.e. it constructs one audit checkpoint every day to represent its financial activity for the day. Consequently, at the year-end audit, the bank B can provide the auditor with the corresponding section of its audit trail consisting of 365 (or 366) checkpoints, such that the first checkpoint represents day 1 of the year and the last checkpoint represents the last day of the year. Figure 5.2 used to demonstrate an example audit trail for this scenario, where the time-period k represents day 1 of the year and time-period $k + 364$ represents last day of the year. The *(*)* symbol is used to denote a reference to the previous checkpoint (pointed to by the respective arrows). If the auditor can be provided with the guarantee that the audit trail construction ensures completeness and soundness of the underlying transaction sets, and that each summary can be guaranteed to be an accurate representation of the corresponding transaction set, then the auditor is only required to validate the consolidated value

of the 365 checkpoint summaries, which is independent of the number of transactions in the underlying transaction sets.

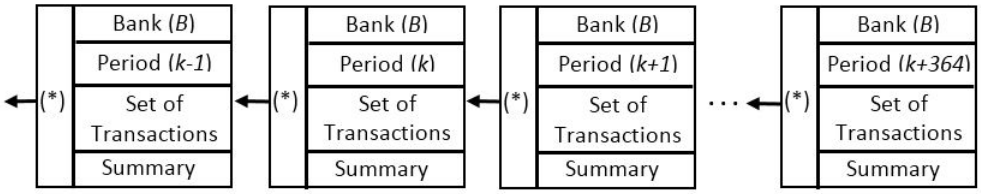

Figure 5.2: Audit Trail Example

In this way, our model allows each participating bank to define its own frequency of checkpointing and its correlation with the periodicity of its audit requirements.

It is important to note that, in our model, each node constructs its own *audit trail* independent of other nodes, unlike the Bitcoin [21] Blockchain where the entire network shares a common linear chain of transaction blocks. Furthermore, an audit trail in our model differs from the Bitcoin Blockchain in terms of its correlation to the processing of underlying transactions. In Bitcoin, a transaction is considered as completed when it has been included sufficiently deep in the Blockchain, which means that the frequency of block creation constrains the productivity of the Bitcoin network in terms of transaction processing. Whereas, in our model, transaction authorization is performed independently of the audit trail construction.

The key characteristic considerations of the model are as follows.

- When a node s_i constructs an *audit checkpoint*, it should include *all* the transactions that involve s_i as the sender or receiver of the transaction and were authorized during the time-period to which the audit checkpoint corresponds.

- For any node, the set of transactions in every audit checkpoint is required to be *mutually exclusive* from the transaction sets in all other audit checkpoints.

- Each audit checkpoint is required to include an accurate *summary* for its set of transaction.

- Each node is required to link its audit checkpoints together in a linear *audit trail*, such that each audit checkpoint contains a reference to the previous audit checkpoint.

We enforce the above mentioned constraints by requiring each audit checkpoint to be validated and signed by at least $2f+1$ processing server nodes in the network, and the resulting signature set is considered as a proof of validity for an audit checkpoint. An audit checkpoint validated and signed by $2f+1$ nodes is referred to as an *authorized* checkpoint. Similar to our transaction processing model, each processing server node validates a checkpoint using its local knowledge only and no server node is required to maintain all the checkpoint information for any node. Each server node s_i, that signs a checkpoint, is required to maintain the checkpoint as a part of its local checkpoint set and also keep track of the transactions that have been included in a checkpoint by a node. The node s_i uses this local knowledge to validate future checkpoints.

We would like to make an important distinction before discussing how a server node validates the transaction set in a checkpoint. For the purpose of abstraction, we refer to the constituents of an audit checkpoint as transactions. It should be construed that an audit checkpoint contains transactions only in a symbolic sense, i.e. it contains verifiable references to the corresponding authorized transactions.

As discussed in chapter 4, each authorized transaction is added to the new transactions set $\mathfrak{T}^N$ for each server node involved in it as a sender or receiver. Initially, the status of the transaction is set to be *new*. Each server node constructs its audit checkpoints by including all its *new* transactions from its local copy of the set $\mathfrak{T}^N$. Similarly, each server node validates a checkpoint using its local copy of the set $\mathfrak{T}^N$.

Once, the checkpoint is authorized, the corresponding entries are removed from the set $\mathcal{T}^N$ by the checkpoint initiator as well as all the server nodes that sign it, and those entries no longer considered *new*. If a server node detects that a transaction included in the audit checkpoint being processed is not a *new* transaction, it rejects the checkpoint. We show that our protocol guarantees that no authorized transaction can be included in more than one authorized audit checkpoint by any server node.

We also consider the possibility that, during the construction of an audit checkpoint, the server node may not be aware of a newly authorized transaction, or a transaction being processed at the same time, where it is the receiver. So we allow the server nodes to exclude such a transaction at most once from its checkpoint processing, i.e. it is required to include that transaction in the subsequent checkpoint. This constraint is enforced by a notification mechanism. If a server node detects that a *new* transaction is excluded from the audit checkpoint being processed, the node notifies the checkpoint initiator about the transaction, and changes the status of the transaction in the set $\mathcal{T}^N$ as *notified*. If a server node detects that a *notified* transaction is excluded from the audit checkpoint being processed, it rejects the checkpoint. We show that our protocol guarantees that an authorized transaction is required to be included in one of the subsequent two audit checkpoints by the receiver node of the transaction.

As a result, if a sender node s_i completes phase 2 of transaction authorization protocol for a transaction $\mathcal{T}$ at time t, then

- the node s_i must include the transaction $\mathcal{T}$ only in its first checkpoint constructed after time t, and
- each node s_j, involved in the transaction $\mathcal{T}$ as a receiver, must include the transaction $\mathcal{T}$ in exactly one of its first two checkpoints constructed after time

t.

At any given time *t*, the *audit trail* for any node must contain all its audit checkpoints authorized by the time *t*, such that each audit checkpoint contains a reference to the previous authorized checkpoint.

For audit checkpoints, we consider a stronger reference mechanism which ensures that an authorized checkpoint must be rendered immutable once it is referenced in another checkpoint as its previous authorized checkpoint. Similar to Bitcoin [21] Blockchain, we use collision-resistant hash values as references for the authorized checkpoints because it can be assumed, by virtue of the definition of collision-resistant hash functions (CRHF), that an efficient (i.e. probabilistic polynomial time) server node will not be able to modify the checkpoint without affecting a detectable change in the corresponding hash value. Therefore, when we symbolically include the authorized checkpoint in its succeeding checkpoint, it can be construed that a collision-resistant hash value of the said checkpoint is used as the reference mechanism.

Using this *audit trail* construction, we derive an interesting property that if a checkpoint in an audit trail is verified to be authorized, then each checkpoint, that can be traced from its sequence of preceding checkpoint references in the audit trail, must also be authorized. We use this property of our audit trail model to show that the audit procedure (definition 5.1) can be performed in time independent of the number of the transactions and the auditor is not required to query any other node in the network to validate the soundness or completeness of the underlying transaction set.

The further discussion of the audit trail model in this chapter is organized as follows.

- In section 5.2, we define the audit checkpoint and audit trail data structures

used in our model.

- Section 5.3 defines the validation structure to be used by a server node for validating audit checkpoints.
- Section 5.4 discusses the audit checkpoint processing protocol for a checkpoint initiator node and a processing node.
- In section 5.5, we analyze the audit trail model in detail.
- Section 5.6 discusses some efficiency improvement ideas pertaining to our audit trail model.

5.2 Audit Data Structures

In this section, we define and discuss the data structures used by our audit trail model.

5.2.1 Audit Checkpoints

The life-cycle of an audit checkpoint involves 2 stages:

1. **Proposed:** When a initiator node sends the checkpoint for authorization to the network, the checkpoint is said to be a proposed checkpoint. We use α to denote a proposed checkpoint. The structural composition of a proposed checkpoint is described in definition 5.2.
2. **Authorized:** When a server node receives a proposed checkpoint α, it validates the checkpoint using its local knowledge and responds with a signature if the proposed checkpoint is valid according to it. Once the initiator node receives $2f+1$ signatures from the network, the checkpoint is said to be authorized. We

use β to denote an authorized checkpoint. Authorized checkpoint data structure is formally defined in definitions 5.3.1 and 5.3.2.

Definition 5.2. *(Proposed Checkpoint) A proposed checkpoint α is defined as a 6-tuple $\langle s, \delta, \mathcal{T}^C, \sigma, \beta_p, d \rangle$, where*

- *s is the server node which initiates the checkpoint.*
- *δ is the integer time-period value for the checkpoint. It is used in a similar manner to the sequence number id used in the transaction model.*
- *$\mathcal{T}^C$ is the set of transactions that involve node s as input or output during the time-period $\delta - 1$ to δ. The transaction set is validated by server nodes using their local knowledge of the new transaction set $\mathcal{T}^N$.*
- *σ is the summary for the checkpoint calculated by the summarizing function* SUMMARIZE. *The summary is validated by the server nodes while authorizing the checkpoint.*
- *β_p is the reference to previous authorized checkpoint for the node s, such that if δ_p is the time-period for the checkpoint β_p, then $\delta - \delta_p = 1$.*
- *d is the digital signature of the sender node for the checkpoint $(s, \delta, \mathcal{T}^C, \sigma, \beta_p)$.*

The definition of the time-period δ has been left abstract in order to account for individual financial reporting and auditing requirements. An example of time-period can be a day, where s creates an audit checkpoint each day. Similarly, a node can define its time-period δ to be less than or more than the length of the day depending upon their requirements.

For simplicity of the discussion, we have defined the set $\mathcal{T}^C$ as a set of transactions. However, it can replaced by a set of $\langle s', id \rangle$ pairs, where s' is the sender node and id

is the sequence number of the concerned transactions. Also, as discussed in section 5.1, the reference β_p to the previous authorized checkpoint should be understood as a collision-resistant hash value of the checkpoint.

Definition 5.3. *(Authorized Checkpoint) An authorized checkpoint β, depicted by the figure 5.3, is defined as a pair $\langle \alpha,\ D \rangle$, where $\alpha = \langle s, \delta, \mathcal{T}^C, \sigma, \beta_p, d \rangle$ is the underlying proposed checkpoint and D represents the set of $2f+1$ server node signatures collected during the authorization of the checkpoint α.*

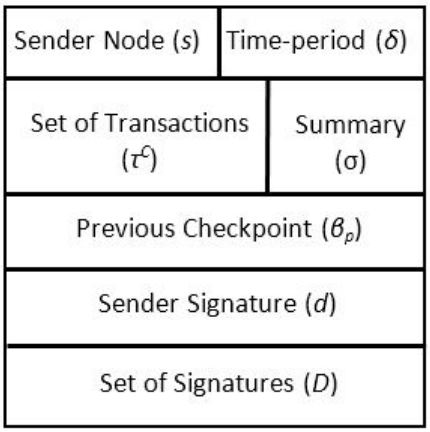

Figure 5.3: Audit Checkpoint Model

Definition 5.3.1. *(Authorized Checkpoint) A checkpoint $\beta : \langle \alpha,\ D \rangle$ is said to be authorized if D contains $2f+1$ valid signatures from distinct server nodes.*

$$authorized(\langle \alpha, D \rangle) \triangleq (\forall (s_j, d_j) \in D,\ \ Verify_{pk_j}(\alpha, d_j) = true) \ \wedge \ |D| = 2f+1$$

5.2.2 Audit Trail

For the discussion of an audit trail, we denote an authorized checkpoint as a composite notation $\beta[s_i, \delta]$, where s_i is the initiator node and δ is the time-period value used in the checkpoint.

Using the definitions for authorized audit checkpoints, we define an audit trail in terms of the structural requirements as follows.

Definition 5.4. *(Audit Trail) Given an audit period length l defined in terms of number of audit checkpoint time-periods and a starting time-period δ, an audit trail for the node s_i is defined as an audit checkpoint sequence $\beta^T_{s_i,\ \delta,\ l}$ composed of the checkpoints $\langle\ \beta[s_i, \delta],\ \beta[s_i,\ \delta + 1],\ \ldots\ \beta[s_i, \delta + l - 1]\ \rangle$ where*

- $\forall \beta[s_i, k] \in \beta^T_{s_i,\ \delta,\ l},\ \beta[s_i, k]$ *contains a reference β_p to the previous authorized checkpoint $\beta[s_i, k-1]$.*

We define an authorized audit trail as composed of a sequence of authorized audit checkpoints as follows.

Definition 5.5. *(Authorized Audit Trail) An audit trail $\beta^T_{s_i,\ \delta,\ l} = \langle\ \beta[s_i, \delta],\ \beta[s_i,\ \delta + 1],\ \ldots\ \beta[s_i, \delta + l - 1]\ \rangle$ for the node s_i, where $\forall \beta[s_i, k] \in \beta^T_{s_i,\ \delta,\ l},\ \beta[s_i, k]$ contains a reference β_p to the previous authorized checkpoint $\beta[s_i, k-1]$, is said to be authorized if*

- $\forall \beta \in \beta^T_{s_i,\ \delta,\ l},$ *authorized(β), i.e. all the audit checkpoints in the audit trail are authorized as per definition 5.3.2.*

5.2.3 Authorized Checkpoints Set

Using the audit data structures discussed above, each server s_i maintains the following data structure for maintaining the local image of the distributed checkpoint history.

- **Authorized Checkpoints (β^A)** is the set of checkpoints β authorized by the server node s_i.

5.3 Audit Structure Validations

In order to ensure that a server's checkpoints are processed in order, we require that the server provide consecutive time-period values for its checkpoints. This requirement is enforced by having the server include the last authorized checkpoint initiated by the server in the request parameters.

When a sender node s_i sends a proposed checkpoint $\alpha : \langle s, \delta, \mathcal{T}^C, \sigma, \beta_p, d \rangle$ to the server nodes for authorization, the last authorized checkpoint for the sender node s_i, for which β_p is the reference, is accompanied by the checkpoint α as an evidence. We slightly abuse the notation for simplicity and use β_p to denote the previous authorized checkpoint during the discussion of the validations, however, it should be understood that β_p included in the data structure α is only a reference.

To ensure that the completeness and soundness of the transaction set included in checkpoints, we make use of the new transaction set $\mathcal{T}^N$ constructed during the transaction commit procedure. As noted in chapter 4, initially, each transaction is committed with a status *new* for each node involved in the transaction as the sender or a receiver. When a node constructs its checkpoints, it includes all its *new* transactions using its local copy of the new transaction set $\mathcal{T}^N$, and each server nodes validates this criteria using its local copy of the new transaction set $\mathcal{T}^N$. We note that during checkpoint construction, a server node is expected to be aware of all the transactions where it was the sender, however, it can be unaware of some recently authorized transactions, or the ones being processed, wherein it is a receiver. Therefore, if a server node s_i detects that *new* transaction, sent by the checkpoint initiator, has been excluded from the checkpoint being processed, the node s_i rejects the checkpoint as invalid; whereas, if the node s_i detects that a *new* transaction, expected to be received by the checkpoint initiator, has been excluded from the checkpoint being

processed, the node s_i notifies the checkpoint initiator about the transaction, changes the status of the transaction as *notified* and signs the checkpoint under a conditional expectation that if the checkpoint initiator excludes this *notified* transaction from the next checkpoint, the node node s_i will reject it as invalid.

The audit checkpoint α is considered valid by a non-faulty server node s_i if all the following conditions hold.

1. The checkpoint α is structurally *well-formed*, i.e.
   (a) The summary value σ is valid for the transaction set $\mathcal{T}^C$ in the checkpoint with respect to the initiator node s
   (b) All the transactions, in set $\mathcal{T}^C$, are authorized transactions where node s is either the sender or receiver
   (c) The checkpoint β_p is an authorized checkpoint initiated by the node s with the time-period value one less than the time-period value δ in checkpoint α
   (d) The signature d is the valid signature for the sender node s over the checkpoint $\langle s, \delta, \mathcal{T}^C, \sigma, \beta_p \rangle$
2. The checkpoint α is *functionally valid*, i.e.
   (a) All the transactions in set $\mathcal{T}^C$ are *new* or *notified* transactions for node s as per the new transaction set $\mathcal{T}^N$ of the processing node s_i
   (b) All the *notified* transactions, as per the new transaction set $\mathcal{T}^N$ of the processing node s_i, for the node s are included in the set $\mathcal{T}^C$
   (c) All the *new* transactions, where the node s is the transaction sender node, as per the new transaction set $\mathcal{T}^N$ of the processing node s_i, are included in the set $\mathcal{T}^C$

(d) The time-period value δ_c used in α is greater than all the time-period values used by the node s in previous checkpoints as per the authorized checkpoint set β^A of the processing node s_i

Similar to the transaction model validations, the structural validations depend upon the proposed checkpoint α and the accompanying previous authorized checkpoint β_p, while the functional validations depend upon the new transactions set $\mathfrak{T}^N$ and the authorized checkpoints set β^A. Therefore, if a checkpoint is structurally valid according to one non-faulty node, it will be structurally valid according to all non-faulty nodes. However, the functional validity of a checkpoint may vary among non-faulty nodes depending upon the transactions in their new transaction set $\mathfrak{T}^N$ and the checkpoints in their authorized checkpoints set β^A.

Also, the previous checkpoint β_p is unique with respect to each proposed checkpoint α because according to validation 1(c), it is required to be an authorized checkpoint with exactly the previous time-period value, and according to validation 2(d), the time-period value in α is required to be new, which will not be valid if β_p is used multiple times.

We now formally define the individual validations as building blocks for the structural and functional validations as mentioned above.

Definition 5.6. *(Summary Validation) A proposed checkpoint* $\alpha : \langle s, \delta, \mathfrak{T}^C, \sigma, \beta_p, d \rangle$ *is said to have a valid summary* σ *if the* SUMMARIZE *function produces* σ *for the comprising set of transactions* $\mathfrak{T}^C$ *corresponding to the node* s.

$$validSummary(s, \mathfrak{T}^C, \sigma) \triangleq \ (\sigma = \text{SUMMARIZE}(s, \mathfrak{T}^C))$$

Definition 5.7. *(Previous Checkpoint Validation) A checkpoint* β_p*, included in the proposed checkpoint* α*, is said to be the valid previous checkpoint for* α *if* β_p *is an*

authorized checkpoint initiated by the same node and the time-period value in α is one more than the time-period value in β_p.

$$validPrevCP(\alpha, \beta_p) \triangleq authorized(\beta_p) \ \wedge \ (\alpha.s = \beta_p.s) \ \wedge \ (\alpha.\delta = \beta_p.\delta + 1)$$

Definition 5.8. *(Authorized Transactions Validation) A proposed checkpoint α, initiated by node s is structurally valid in terms of the included transactions $\mathcal{T}^C$ if all the transactions are authorized transactions and involve s as input or output.*

$$authorizedTxns(s, \mathcal{T}^C) \triangleq \forall \mathcal{T} \in \mathcal{T}^C, \ authorized(\mathcal{T}) \ \wedge \ (s \in \mathcal{T}.O \ \cup \ \mathcal{T}.I)$$

Definition 5.9. *(Transaction Set Validation) The transaction set $\mathcal{T}^C$ included in a checkpoint α proposed by node s_i is valid if*

- *all the transactions in $\mathcal{T}^C$ are* new *or* notified *transactions for node s_i,*
- *all the* notified *transactions for node s_i are included in $\mathcal{T}^C$ and*
- *all the* new *transactions initiated by node s_i are included in $\mathcal{T}^C$*

$$\begin{aligned} validTxnSet(s_i, \mathcal{T}^C) \triangleq \ & (\ \forall \mathcal{T} \in \mathcal{T}^C, \ \langle \mathcal{T}, s_i \rangle \in \mathcal{T}^N \ ) \\ & \wedge \ (\ \forall \langle \mathcal{T}, s_i, notified \rangle \in \mathcal{T}^N, \ \mathcal{T} \in \mathcal{T}^C \ ) \\ & \wedge \ (\ \forall \langle \mathcal{T}, s_i \rangle \in \mathcal{T}^N, \ \mathcal{T}.s = s_i \implies \mathcal{T} \in \mathcal{T}^C \ ) \end{aligned}$$

Definition 5.10. *(Time-period Validation) A time-period δ_c in a proposed checkpoint α, initiated by the node s, is said to be valid if it is greater than the maximum time-period used by the node s in previously authorized checkpoints.*

$$validPeriod(s, \delta_c) \triangleq \delta_c > max\{\delta : \delta \in \beta^A[s]\}$$

5.3.1 Structural Validations

When a server node s_i receives a checkpoint for authorization from the sender node s, the node s_i first validates the structural integrity of the checkpoint, i.e. ascertains whether it is well-formed. If the checkpoint is structurally invalid, the node s_i rejects the checkpoint. The corresponding validation is defined as follows.

Definition 5.11. *(Well-formed Checkpoint) A checkpoint α proposed by a node s is said to be well-formed if all the transactions in the transaction set $\mathfrak{T}^C$ are authorized, and the summary, the previous checkpoint and the sender node signature are valid.*

$$\begin{aligned} \mathit{wellFormed}(\alpha) \triangleq\ & \mathit{validPrevCP}(\alpha, \alpha.\beta_p)\ \wedge\ \mathit{authorizedTxns}(\alpha.s,\ \alpha.\mathfrak{T}^C) \\ & \wedge\ \mathit{validSummary}(\alpha.s, \alpha.\mathfrak{T}^C, \alpha.\sigma)\ \wedge\ (\mathit{Verify}_{pk_j}(\alpha, \alpha.d) = \mathit{true}) \end{aligned}$$

5.3.2 Functional Validations

After validating the structural integrity of the checkpoint using definition 5.11, the server node s_i performs functional validations to verify that the transaction set $\mathfrak{T}^C$ is valid using definition 5.9. In doing the validation, the server node s_i uses its local knowledge of the new transaction set $\mathfrak{T}^N$. So it is possible that a non-faulty server node will consider an invalid transaction set as valid because it may not be aware of a *notified* transaction or an excluded transaction that was initiated by node checkpoint sender node. Nevertheless, if a checkpoint is invalid, it will not be considered valid by enough servers required for authorizing it.

Similarly, the server node s_i also verifies, using its local knowledge, that the time-period value has not been used previously using definition 5.10.

Definition 5.12. *(Functionally Valid Checkpoint) A checkpoint α proposed by a node*

s is said to be functionally valid if the transaction set and the time-period are valid.

$$validCP(\alpha) \triangleq validTxnSet(\alpha.s,\ \alpha.\mathcal{T}^C)\ \wedge\ validPeriod(\alpha.s,\ \alpha.\delta)$$

5.4 Audit Checkpoint Protocol

As mentioned earlier, the set $\mathcal{T}^C$ in a proposed checkpoint α can be considered as a set of transaction identifying pairs $\langle s', id \rangle$ for implementation purposes, where s' is the sender node and id is the sequence number of the concerned transactions. However, in context of the protocol discussed in this section, we assume that the corresponding set of authorized transactions is sent by the initiator node to all server nodes for validation, irrespective of the representation of the set $\mathcal{T}^C$, of the proposed checkpoint α. We revisit this point in our efficiency improvement section 5.6 and discuss ideas to improve the message complexity of the protocol.

We now present the audit checkpoint authorization protocol used by the server nodes to propose and authorize checkpoints.

5.4.1 Sender Protocol

Each server node s_i periodically constructs an audit checkpoint and processes its authorization using the PROCESS-CHECKPOINT procedure depicted in figure 5.4. Unlike the transaction authorization protocol, the audit checkpoint authorization protocol only uses one phase of communication with the network.

The sender node s_i maintains a current time-period counter value δ_c to assign the δ value to its checkpoints. It begins by constructing the checkpoint which involves the following steps.

1. The time-period δ_c is incremented for the current checkpoint (line 6)

2. The transaction set $\mathcal{T}^C$ is constructed using the new transactions for the node

Algorithm 5 Sender Node s_i

```
1:  initialize:
2:      δ_c ← 0    /* Current time-period value for node s_i */

3:  /* Process checkpoint authorization */
4:  procedure PROCESS-CHECKPOINT
5:      /* Construct the checkpoint to be authorized */
6:      δ_c ← δ_c + 1
7:      T^c ← {T : ⟨T, s_i⟩ ∈ T^N}
8:      σ   ← SUMMARIZE(s_i, T^c)
9:      β_p ← β^A[s_i, δ_c − 1]
10:     d_i ← Sign_{sk_i}(⟨s_i, δ_c, T^c, σ, β_p⟩)
11:     α   ← ⟨s_i, δ_c, T^c, σ, β_p, d_i⟩

12:     /* Send request for checkpoint authorization */
13:     send (CHECKPOINT, α) to all server nodes
14:     initialize: D ← ∅, T^M ← ∅
15:     repeat
16:         on receive (SIGNED-CP, α, d_j, T^E) from server s_j do
17:             if Verify_{pk_j}(α, d_j) = true then
18:                 Add (s_j, d_j) to set D
19:                 T^M ← T^M ∪ T^E
20:             end if
21:         end
22:     until |D| = 2f + 1

23:     /* Commit checkpoint and the notified new transactions */
24:     COMMIT-CP(⟨α, D⟩, ∅)
25:     for each T ∈ T^M s.t. T ∉ T^A ∧ authorized(T) do
26:         COMMIT-TXN(T)
27:     end for
28: end procedure
```

Figure 5.4: Audit Checkpoint Sender Protocol

s_i as per the new transactions set $\mathcal{T}^N$ (line 7)

3. The summary value σ is calculated using the SUMMARIZE function for the transaction set $\mathcal{T}^C$ (line 8)

4. The previous authorized checkpoint β_p is retrieved from the authorized checkpoints set β^A using the time-period $\delta_c - 1$ (line 9)

5. The checkpoint α is constructed as a 6-tuple (line 11) using the digital signature d_i (line 10).

Once the checkpoint is constructed, s_i sends the checkpoint authorization request

to all server nodes (line 13) and waits for $2f+1$ valid signatures (lines 14-22). When a server node s_j responds (line 16), the signature d_j is accompanied by a set of transactions $\mathcal{T}^E$, which consists of transactions that are new transactions involving s_i and are excluded from the checkpoint transaction set $\mathcal{T}^C$. A non-faulty server node s_j expects the transactions in $\mathcal{T}^E$ to be included by s_i in its next checkpoint. So, the node s_i collects the transactions notified by server nodes in the set $\mathcal{T}^M$ (line 19) and commits the transactions in $\mathcal{T}^M$ if they have not already been committed and are authorized transactions (lines 25-27). When $2f+1$ signatures are received, s_i commits the checkpoint (line 24) using the COMMIT-CP procedure depicted in figure 5.6.

5.4.2 Server Node Protocol

The corresponding protocol used by server nodes is depicted in figure 5.5.

Algorithm 6 Server Node s_j

```
1:  /* Process checkpoint authorization request from a sender node */
2:  on receive (CHECKPOINT, α) from server node s_i do
3:      if wellFormed(α) then
4:          /* Commit missing transactions, if any */
5:          for each T ∈ α.T^C s.t. T ∉ T^A do
6:              COMMIT-TXN(T)
7:          end for
8:          /* Sign the checkpoint if it is well-formed and valid */
9:          if validCP(α) then
10:             d_j ← Sign_{sk_j}(α)
11:             T^E ← {T : ⟨T, s_i⟩ ∈ T^N ∧ T ∉ α.T^C}
12:             COMMIT-CP(⟨α, d_j⟩, T^E)
13:             send (SIGNED-CP, α, d_j, T^E) to server node s_i
14:         end if
15:     end if
16: end
```

Figure 5.5: Server Node Protocol for Audit Checkpoint Processing

When the node s_j receives the checkpoint authorization request for audit checkpoint α from the node s_i (line 2), s_j processes it only if the checkpoint is well-formed as per the definition 5.11 (line 3). If any of the transactions in the checkpoint are not

committed, s_j commits those transactions for further processing (lines 5-7). Now, if the checkpoint is functionally valid (line 9), s_j digitally signs the checkpoint (line 10) and constructs the set of excluded transaction $\mathcal{T}^E$ consisting of new transactions involving s_i that have not been included in the checkpoint (line 11). Node s_j commits the checkpoint along with the excluded transactions (line 12) and sends the signed checkpoint to the sender (line 13).

The audit checkpoint commit procedure COMMIT-CP is depicted in figure 5.6. A checkpoint commit involves three tasks. Firstly, the checkpoint $\langle \alpha, D \rangle$ is stored as an authorized checkpoint β^A (line 5). Secondly, the new transactions included in the checkpoint $\alpha.\mathcal{T}^C$ are removed from the new transactions set $\mathcal{T}^N$ (lines 7-9) because they are not considered new transactions after the checkpoint commit. And finally, the excluded set of transactions $\mathcal{T}^E$ are updated as *notified* in the new transactions set $\mathcal{T}^N$ (lines 11-13) to make sure that they are not excluded by the sender node again in the next checkpoint.

Algorithm 7 Server Node s_i

```
1:  initialize:
2:      β^A ← ∅

3:  procedure COMMIT-CP(⟨α, D⟩, T^E)
4:      if α ∉ β^A then
5:          Add ⟨α, D⟩ to β^A
6:          /* Remove checkpointed txns from the new txn set */
7:          for each T ∈ α.T^C do
8:              Remove ⟨T, α.s⟩ from T^N
9:          end for
10:         /* Update status for notified txns */
11:         for each T ∈ T^E do
12:             T^N[T, α.s].θ = notified
13:         end for
14:     end if
15: end procedure
```

Figure 5.6: Audit Checkpoint Commit Procedure

Server nodes commit missing transactions (fig 5.5, lines 5-7) for two reasons. Firstly, it helps them improve their local knowledge of the authorized transactions

in the network. Secondly, if a faulty sender node terminates sender protocol while processing a transaction at the end of phase 1, the transaction will not be committed by any server node in the network. As a result, it will not be available as a new transaction for any server node and consequently, the receiver will not receive the transaction during the receiver protocol. However, we know that the sender node will be required to send this authorized transaction while processing the checkpoint, and each server node can commit the transaction at this time and guarantee progress for the transaction receiver node.

5.5 Analysis

During our analysis of the audit structure, we denote an authorized checkpoint as $\beta[s_i, \delta]$, where s_i is the initiator node and δ is the time-period value used in the checkpoint. This provides a composite notation to refer to an authorized checkpoint with its identifying characteristics.

5.5.1 Checkpoint Processing Protocol

For our analysis of the audit checkpoint processing protocol, we consider the following properties:

- *Theorem 5.1:* If non-faulty node initiates a checkpoint, the sender protocol is guaranteed to terminate.

- *Theorem 5.2:* All checkpoints initiated by non-faulty sender nodes will be authorized.

- *Theorem 5.3:* If a transaction $\mathcal{T}$ is authorized after the checkpoint $\beta[s_i, \delta]$ is authorized and before the checkpoint $\beta[s_i, \delta + 1]$ is proposed, and the node s_i is either the sender or a receiver of the transaction, then s_i must include $\mathcal{T}$ in

exactly one of the committed checkpoints $\beta[s_i, \delta + 1]$ or $\beta[s_i, \delta + 2]$.

- *Theorem 5.4:* If a transaction $\mathcal{T}$ is not authorized, then it can not be included in any authorized checkpoint.

Theorem 5.1. *If non-faulty node initiates a checkpoint, the sender protocol is guaranteed to terminate.*

Proof. We observe that the sender node has complete knowledge of all the transactions initiated by it in the past, which prevents it from accidentally excluding an authorized transaction where it was the sender node. It also has the knowledge of the transactions that it has included in the past checkpoints, which prevents it from accidentally including a transaction more than once. Also, if any transactions, where it was the receiver node, were excluded in the previous checkpoint due to lack of local knowledge, it would have received the notification from the server nodes and it would have committed the *notified* transactions according to each of the server nodes (fig 5.4, lines 19, 25-27). This ensures that those transactions are not excluded during the current checkpoint construction, so a non-faulty sender not can exclude *notified* transactions. It keeps track of its time-period values, which can not be invalid for any checkpoint. It also has the previous authorized checkpoint which is required to be included in the current checkpoint. And finally, the summary value is calculated as per the summarizing function SUMMARIZE and it will be valid.

So, if the sender node is non-faulty, then its checkpoint must be valid because a non-faulty sender node constructs the checkpoint as per these specifications (fig 5.4, lines 6-11). Therefore, we can say that the checkpoint must be structurally and functionally valid as per definitions 5.11 and 5.12 respectively. So, as per algorithm 6 (fig 5.5), each non-faulty server node will sign the checkpoint. We know that there are at most f faulty nodes, i.e. there are at least $2f + 1$ non-faulty nodes in the

network, so the sender is guaranteed to receive at least $2f + 1$ signatures, which may or may not contain signatures from faulty nodes. Therefore, the sender will be able to complete the protocol and terminate. □

Theorem 5.2. *All checkpoints initiated by non-faulty sender nodes will be authorized.*

Proof. Using the argument from Theorem 5.1, we can say that if a non-faulty sender node initiates a checkpoint, the sender is guaranteed to complete the protocol and terminate. This follows that the checkpoint will be will be signed by at least $2f + 1$ nodes and consequently, will be authorized as per definition 5.3.1. □

For the proof of *Theorem 5.3*, we consider the following lemmas:

- *Lemma 5.1:* If a transaction $\mathcal{T}$ is authorized and the node s_i is either the sender or a receiver of the transaction, then s_i can include $\mathcal{T}$ in at most one authorized checkpoint.

- *Lemma 5.2:* If a transaction $\mathcal{T}$ is authorized after the checkpoint $\beta[s_i, \delta]$ is authorized and before the checkpoint $\beta[s_i, \delta + 1]$ is proposed, and the node s_i is either the sender or a receiver of the transaction, then s_i must include $\mathcal{T}$ in at least one of the authorized checkpoints $\beta[s_i, \delta + 1]$ or $\beta[s_i, \delta + 2]$.

Lemma 5.1. *If a transaction $\mathcal{T}$ is authorized and the node s_i is either the sender or a receiver of the transaction, then s_i can include $\mathcal{T}$ in at most one authorized checkpoint.*

Proof. As per COMMIT-CP procedure defined in algorithm 7 (fig 5.6), once the checkpoint β is committed by a node s_j, all the transactions in the transaction set $\beta.\mathcal{T}^C$ are removed from the new transaction set $\mathcal{T}^N$ for the node s_i. If a checkpoint β is authorized, it is signed and committed by at least $2f + 1$ nodes (fig 5.5, lines 10-12), out of which at least $f + 1$ nodes must be non-faulty. So, if s_i includes a transaction

$\mathcal{T} \in \beta.\mathcal{T}^C$ in another checkpoint β', at least $f+1$ nodes will be able to detect it as invalid as per definition 5.12 and will not sign the checkpoint β', and as a result, the checkpoint β' will not be authorized. □

Lemma 5.2. *If a transaction $\mathcal{T}$ is authorized after the checkpoint $\beta[s_i, \delta]$ is authorized and before the checkpoint $\beta[s_i, \delta+1]$ is proposed, and the node s_i is either the sender or a receiver of the transaction, then s_i must include $\mathcal{T}$ in at least one of the authorized checkpoints $\beta[s_i, \delta+1]$ or $\beta[s_i, \delta+2]$.*

Proof. As per algorithm 7 (fig 5.5), if the transaction $\mathcal{T}$ is a new transaction according to node s_j and the new checkpoint $\beta[s_i, \delta+1]$ does not include $\mathcal{T}$, the node s_j notifies the node s_i and changes the status of the transaction $\mathcal{T}$ for the node s_i as *notified*. Because the transaction $\mathcal{T}$ is authorized after the checkpoint $\beta[s_i, \delta]$ is authorized and before the checkpoint $\beta[s_i, \delta+1]$ is proposed, the transaction $\mathcal{T}$ is a new transaction for the checkpoint $\beta[s_i, \delta+1]$ according to at least $f+1$ non-faulty nodes, and they will all change its status to *notified* upon its exclusion from $\beta[s_i, \delta+1]$. If the next checkpoint $\beta[s_i, \delta+2]$ does not include $\mathcal{T}$, then at least $f+1$ nodes will not sign the checkpoint, and consequently, it will not be authorized. Therefore, the node s_i must include the transaction $\mathcal{T}$ in at least one of its next 2 checkpoints $\beta[s_i, \delta+1]$ or $\beta[s_i, \delta+2]$. □

Theorem 5.3. *If a transaction $\mathcal{T}$ is authorized after the checkpoint $\beta[s_i, \delta]$ is authorized and before the checkpoint $\beta[s_i, \delta+1]$ is proposed, and the node s_i is either the sender or a receiver of the transaction, then s_i must include $\mathcal{T}$ in exactly one of the committed checkpoints $\beta[s_i, \delta+1]$ or $\beta[s_i, \delta+2]$.*

Proof. The result follows from combining Lemma 5.1 and 5.2. □

Theorem 5.4. *If a transaction $\mathcal{T}$ is not authorized, then s_i can not include $\mathcal{T}$ in any authorized checkpoint.*

Proof. If the checkpoint $\beta[s_i, \delta]$ is authorized, then it must be valid according to at least $2f + 1$ nodes, i.e. at least $f + 1$ non-faulty nodes. As per definition 5.8 and 5.11, a checkpoint is structurally invalid if a transaction in its transaction set $\mathfrak{T}^C$ is not authorized, which is validated by the non-faulty server nodes in algorithm 6 (fig 5.5, line 3). Therefore, if $\beta[s_i, \delta]$ is authorized, then it can not contain a transaction $\mathfrak{T}$ which is not authorized. □

5.5.2 Audit Trail

For our analysis of the audit trail model, we consider the following properties:

- *Theorem 5.5:* An audit trail is authorized if an only if the last checkpoint, in the audit trail, is an authorized checkpoint.
- *Theorem 5.6* If the audit trail is authorized, then it must satisfy all the properties required in the audit procedure as per definition 5.1.

For the proof of *Theorem 5.5*, we consider the following lemmas.

- *Lemma 5.3:* If a checkpoint $\beta[s_i, \delta]$, proposed by node s_i, is authorized and it contains β_p as a reference to the previous checkpoint $\beta[s_i, \delta - 1]$, it implies that the checkpoint $\beta[s_i, \delta - 1]$ is authorized.
- *Lemma 5.4:* If a checkpoint $\beta[s_i, \delta - 1]$, proposed by node s_i, is not authorized, it implies the checkpoint $\beta[s_i, \delta]$ can not be authorized given $\beta[s_i, \delta]$ contains β_p as a reference to the previous checkpoint $\beta[s_i, \delta - 1]$.
- *Lemma 5.5:* Given an audit trail $\beta^T_{s_i,\ \delta,\ l}$ for the node s_i, if a checkpoint $\beta[s_i, k] \in \beta^T_{s_i,\delta,l}$ is an authorized checkpoint, it implies that the audit trail $\beta^T_{s_i,\delta,k+1-\delta} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. ($\subseteq_s$ is used as a symbol for sub-sequence)

Lemma 5.3. *If a checkpoint $\beta[s_i, \delta]$, proposed by node s_i, is authorized and it contains β_p as a reference to the previous checkpoint $\beta[s_i, \delta - 1]$, it implies that the checkpoint $\beta[s_i, \delta - 1]$ is authorized.*

Proof. If the checkpoint $\beta[s_i, \delta]$ is authorized, then it must be valid according to at least $2f + 1$ nodes, out of which at least $f + 1$ nodes must be non-faulty. As per definition 5.7 and 5.11, a checkpoint is structurally invalid if the previous checkpoint, for which β_p is the reference, is not authorized; and it is verified by all the non-faulty server nodes in algorithm 6 (fig 5.5, line 3). Therefore, if $\beta[s_i, \delta]$ is authorized and it contains β_p as a reference to the previous checkpoint $\beta[s_i, \delta - 1]$, then the checkpoint $\beta[s_i, \delta - 1]$ must be authorized. □

Lemma 5.4. *If a checkpoint $\beta[s_i, \delta - 1]$, proposed by node s_i, is not authorized, it implies the checkpoint $\beta[s_i, \delta]$ can not be authorized given $\beta[s_i, \delta]$ contains β_p as a reference to the previous checkpoint $\beta[s_i, \delta - 1]$.*

Proof. If follows the same argument as the proof of Lemma 5.3. □

Lemma 5.5. *Given an audit trail $\beta^T_{s_i,\ \delta,\ l} = \langle\ \beta[s_i, \delta],\ \beta[s_i,\ \delta{+}1],\ ...\ \beta[s_i, \delta{+}l{-}1]\ \rangle$ for the node s_i, where $\forall \beta[s_i, k] \in \beta^T_{s_i,\ \delta,\ l}$, $\beta[s_i, k]$ contains a reference β_p to the previous authorized checkpoint $\beta[s_i, k - 1]$, if a checkpoint $\beta[s_i, k] \in \beta^T_{s_i,\delta,l}$ is an authorized checkpoint, it implies that the audit trail $\beta^T_{s_i,\delta,k+1-\delta} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. ($\subseteq_s$ is used as a symbol for sub-sequence)*

Proof. We begin by observing that if the checkpoint $\beta[s_i, k] \in \beta^T_{s_i,\delta,l}$ is authorized, then as per definition 5.5, the audit trail $\beta^T_{s_i,k,1}$ (containing only one checkpoint $\beta[s_i, k]$) is authorized. We restructure the argument as follows: if the audit $\beta^T_{s_i,k,1} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized, it implies that the audit trail $\beta^T_{s_i,\delta,k+1-\delta} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. We prove this result using induction.

Consider 2 base cases. The base case with $x = k$, i.e. $\beta^T_{s_i,k,k+1-k}$, refers to the audit trail $\beta^T_{s_i,k,1}$ which is clearly authorized given $\beta^T_{s_i,k,1}$ is authorized. For the base case $x = k-1$, we show that if $\beta^T_{s_i,k,1}$ is authorized, it implies that $\beta^T_{s_i,x,k+1-x} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. We know that $\beta[s_i,k]$ is an authorized checkpoint and it contains β_p as a reference to the previous checkpoint $\beta[s_i,k-1]$, so using Lemma 5.4, we can say that $\beta[s_i,k-1]$ is an authorized checkpoint, which means that $\beta^T_{s_i,x,k+1-x}$ is authorized.

Let us consider $k-1 \geq x > \delta$ such that the audit trail $\beta^T_{s_i,x,k+1-x} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized given that $\beta^T_{s_i,k,1}$ is authorized. Our inductive hypothesis is that if the audit trail $\beta^T_{s_i,x,k+1-x} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized given that $\beta^T_{s_i,k,1}$ is authorized, then it implies that the audit trail $\beta^T_{s_i,x-1,k+1-(x-1)} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. We know that the checkpoint $\beta[s_i,x] \in \beta^T_{s_i,x,k+1-x}$ is authorized as per definition 5.5 and $\beta[s_i,x]$ contains β_p as a reference to the previous checkpoint $\beta[s_i,x-1]$. So, as per lemma 5.3, we can say that $\beta[s_i,x-1]$ is an authorized checkpoint, which means that $\beta^T_{s_i,x-1,k+1-(x-1)} \subseteq_s \beta^T_{s_i,\delta,l}$ is authorized. Applying this inductive argument for $x = \delta - 1$, we get the required result. □

Theorem 5.5. *An audit trail* $\beta^T_{s_i,\ \delta,\ l} = \langle\ \beta[s_i,\delta],\ \beta[s_i,\ \delta+1],\ \ldots\ \beta[s_i,\delta+l-1]\ \rangle$ *for the node* s_i*, where* $\forall\beta[s_i,k] \in \beta^T_{s_i,\ \delta,\ l}$*,* $\beta[s_i,k]$ *contains* β_p *as a reference to the previous authorized checkpoint* $\beta[s_i,k-1]$*, is authorized if an only if the checkpoint* $\beta[s_i,\delta+l-1]$ *is an authorized checkpoint.*

Proof. Clearly, as per definition 5.5, if the audit trail $\beta^T_{s_i,\delta,l}$ is authorized, it implies that the checkpoint $\beta[s_i,\delta+l-1]$ must be an authorized checkpoint. Conversely, using lemma 5.5, we can say that if the checkpoint $\beta[s_i,\delta+l-1]$ is an authorized checkpoint, it implies that the audit trail $\beta^T_{s_i,\delta,l}$ is authorized. □

Theorem 5.6. *If the audit trail* $\beta^T_{s_i,\ \delta,\ l} = \langle\ \beta[s_i,\delta],\ \beta[s_i,\ \delta+1],\ \ldots\ \beta[s_i,\delta+l-1]\ \rangle$ *for the node* s_i*, where* $\forall\beta[s_i,k] \in \beta^T_{s_i,\ \delta,\ l}$*,* $\beta[s_i,k]$ *contains* β_p *as a reference to the previous*

authorized checkpoint $\beta[s_i, k-1]$, is authorized, then the following 3 properties hold.

1. $\forall \beta \in \beta^T_{s_i,\delta,l}$, $\beta.\sigma = \text{SUMMARIZE}(s_i, \beta.\mathfrak{T}^C)$
2. *The set of transactions* $\mathfrak{T}^A_i = \bigcup_{\beta \in \beta^T_{s_i,\delta,l}} \beta.\mathfrak{T}^C$ *is sound, i.e.*
   (a) *it contains only authorized transactions*
   (b) $\forall \beta, \beta' \in \beta^T_{s_i,\delta,l}$, $\beta.\mathfrak{T}^C \cap \beta'.\mathfrak{T}^C = \emptyset$, *i.e. no transaction is included in multiple checkpoints.*
3. *The set of transactions* $\mathfrak{T}^A_i = \bigcup_{\beta \in \beta^T_{s_i,\delta,l}} \beta.\mathfrak{T}^C$ *is complete, i.e.*
   (a) *it includes all transactions for node s_i that were authorized after the checkpoint $\beta[s_i, \delta-2]$ was authorized and were excluded from the checkpoint $\beta[s_i, \delta-1]$*
   (b) *it includes all transactions for node s_i authorized between the checkpoints $\beta[s_i, \delta-1]$ and $\beta[s_i, \delta+l-3]$*

Proof. As per definition 5.5, we can say that $\forall \beta \in \beta^T_{s_i,\delta,l}$, β is an authorized checkpoint. As per definition 5.6 and 5.11, a checkpoint is structurally invalid if the summary σ is not valid according to the SUMMARIZE function, which is validated by the non-faulty server nodes in algorithm 6 (fig 5.5, line 3). So, if a checkpoint is authorized, it must be structurally valid and hence, property 1 must hold. Properties 2 and 3 follow from Theorems 5.3 and 5.4 for authorized checkpoints. □

5.5.3 Audit Process

Using theorem 5.6, we can redefine the audit process (definition 5.1) using our audit trail model and show that the audit process can be completed in time independent of the number of transactions in the audit trail using theorem 5.7.

Definition 5.13. *(Audit) Given an audit trail* $\beta^T_{s_i,\delta,l} = \langle\ \beta[s_i,\ \delta],\ \beta[s_i,\ \delta+1],\ ...\ \beta[s_i, \delta+l-1]\ \rangle$ *and a summary* σ_i *for the node* s_i*, an audit can be defined as a procedure to verify the following criteria*

1. *the audit trail* $\beta^T_{s_i,\delta,l}$ *is authorized, and*

2. $\sigma_i = \sum_{\beta \in \beta^T_{s_i,\delta,l}} \beta.\sigma$*, i.e. the summary* σ_i *is the aggregate of all the checkpoint summaries in the audit trail.*

Theorem 5.7. *Given an audit trail* $\beta^T_{s_i,\ \delta,\ l} = \langle\ \beta[s_i, \delta],\ \beta[s_i,\ \delta+1],\ ...\ \beta[s_i, \delta+l-1]\ \rangle$ *and a summary* σ_i *for a node* s_i*, the audit procedure can be performed in* $O(l+f)$ *deterministic time.*

Proof. Using Theorem 5.5, we know that the audit trail $\beta^T_{s_i,\delta,l}$ is authorized if (1) $\beta[s_i, \delta+l-1]$ is an authorized checkpoint, and (2) $\forall \beta[s_i,k] \in \beta^T_{s_i,\ \delta,\ l}$, $\beta[s_i,k]$ contains β_p as a reference to the previous authorized checkpoint $\beta[s_i, k-1]$.

The sequence of checkpoint references can be verified in *O(l)* time. Let us consider the check regarding whether $\beta[s_i, \delta+l-1]$ is an authorized checkpoint or not. As per definition 5.3.1, the checkpoint is authorized if it contains $2f+1$ valid signatures from distinct server nodes. Deterministic check for the validity of the set of signatures can be performed in $O(f)$ time. Considering the size of summary values as a model constant, the value σ_i can be verified in *O(l)* time. Hence, the audit procedure can be performed in $O(l+f)$ deterministic time. □

Note that the time complexity of above audit procedure does not depend upon the number of transactions $|\mathcal{T}^C_i|$, which is at least $\Omega(|\mathcal{T}^C_i|)$ factor improvement in the audit procedure using our audit trail model.

5.6 Efficiency Improvement Ideas

As mentioned earlier, the set $\mathfrak{T}^C$, in a checkpoint β, is defined as a set of authorized transactions. For efficient implementation, these sets can be defined with pair $\langle s, id \rangle$ where s is the sender node and id is the sequence number of the corresponding transaction $\mathfrak{T}$. Corresponding validations can be suitably modified to retrieve the authorized transactions, before validation, from the authorized transaction set $\mathfrak{T}^A$ using the $\langle s, id \rangle$ pair. As a result, the size of the set $\mathfrak{T}^C$ can reduced significantly because each pair $\langle s, id \rangle$ has a constant size, whereas the size of corresponding transaction depends upon the number of inputs, outputs and server signatures.

5.6.1 Audit Checkpoint Protocol

We now discuss the efficiency concerns and potential improvements concerning the audit checkpoint protocol depicted in figure 5.4 and figure 5.5.

As discussed in section 5.4, our protocol depends upon the assumption that the set of authorized transactions is sent by the initiator node to all server nodes for validation, irrespective of the representation of the set $\mathfrak{T}^C$, of the proposed checkpoint α. This means that the message complexity of the audit checkpoint protocol is very high and depends upon the number of transactions in the checkpoint, which is clearly very inefficient.

We now discuss some revisions to the protocol that can provide significant efficiency improvements in an implementations, by only requiring an additional round of communication between the checkpoint initiator node and the server nodes.

- The checkpoint initiator node s_i constructs the set $\mathfrak{T}^C$ of $\langle s, id \rangle$ pairs using its new transaction set $\mathfrak{T}^N$ such that all the corresponding transactions involved s_i as either the sender or a receiver, and sends the set $\mathfrak{T}^C$ as the constituent of the

checkpoint instead of the set of corresponding authorized transactions.

- Each server node s_j constructs a deficiency set $\mathcal{T}^D \subseteq \mathcal{T}^C$ consisting of $\langle s, id \rangle$ pairs from the set $\mathcal{T}^C$ which represents a transaction not available, for verification, in node s_j's local copy of the authorized transaction set $\mathcal{T}^A$. Node s_j sends this set $\mathcal{T}^D$ to the initiator node s_i as a request for missing transactions.

- When initiator node s_i receives a request for missing transactions in the form of set $\mathcal{T}^D$, it can reply with the requested transactions and the rest of the protocol proceeds in the same way as defined in figures 5.4 and 5.5.

If the checkpoint initiator is non-faulty, then all the transactions in its checkpoint must be authorized transactions and each of them must have been signed by $2f + 1$ nodes, out of which at most f can be faulty. Therefore, each node can be expected to maintain a significant subset of the transactions included in the checkpoint, and each deficiency set $\mathcal{T}^D$ can be expected to be much smaller than the set $\mathcal{T}^C$. As a result, the number of transactions communicated during this protocol will be significantly lesser than the original protocol.

Another improvement in the size of the set $\mathcal{T}^C$ can be made by simply including the last sequence number of the transactions where the checkpoint initiator was the sender node, instead of the entire list. The previous checkpoint included in the checkpoint can be used to find the lower bound of the list of sequence numbers to be used for the actual transactions set in the checkpoint.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Summary

While the existing cryptocurrency proposals focus on solving distributed consensus problem in order to provide cryptographic security of the system, we have presented a novel alternative non-consensus based approach which utilizes majority signature based replicated storage protocol to process transactions securely, as shown by our analysis. Furthermore, we have presented a self-verifiable audit trail model which facilitates efficient financial auditing for individual financial entities, a desirable property that has not been addressed by the existing cryptocurrency models. Our analysis shows that an audit trail can be verified in time independent of the number of transactions.

Additionally, our model presents an effective solution for the decentralized inter-bank payment processing problem, which can serve as a practical alternative to the existing centralized payment processing and settlement solutions. A system based on our model can potentially enable inter-bank transactions to be completed with lower cost to the customer due to the absence of a central processing entity levying a transaction processing fee.

6.2 Future Work

Our work has introduced a new way of approaching the decentralized financial transaction processing model, and there are a variety of exciting opportunities for future work, summarized as follows.

- Our model assumes a trust relationship between the customers and the server nodes federated by their corresponding banks. Further analysis could be built upon a generalized model wherein customers interact with the network of nodes without any implicit trust assumptions.

- We assume the number of server nodes to be $n = 3f + 1$. Future work can aim to derive an equivalence with the model that allows $n \geq 3f + 1$ using a Byzantine quorum system based approach. Byzantine quorum systems have been studied extensively in the literature [17, 18, 7, 19, 2, 3] and future work can build a quorum system for extending this model.

- Our protocol and analysis assume a static network configuration, where the number of banks and server nodes is n. Additional work is required to analyze the schematics and impact of dynamic reconfigurations in the network. This work can follow a similar approach as discussed by Aguilera et. al [1].

- Smart contracts execution is an exciting avenue that can be studied as an extension of our model to support additional financial applications.

- Finally, a practical study can be performed using a prototypical implementation of the model to analyze its practical effectiveness and comparability to existing protocols.

REFERENCES

[1] M. Aguilera, I. Keidar, D. Malkhi, and A. Shraer, *Dynamic atomic Storage Without Consensus.* PODC, 2009.

[2] A. Aiyer, L. Alvisi, and R. Bazzi, *On the Availability of Non-strict Quorum Systems.* DISC, 2005.

[3] ——, *Byzantine and Multi-writer K-Quorums.* DISC, 2006.

[4] ——, *Bounded Wait-free Implementation of Optimally Resilient Byzantine Storage Without (Unproven) Cryptographic Assumptions.* DISC, 2007.

[5] S. Aiyer, L. Alvisi, R. Bazzi, and A. Clement, *Matrix Signatures: From MACs to Digital Signatures in Distributed Systems.* DISC, 2008.

[6] A. Back, *Hashcash - Amortizable Publicly Auditable Cost-Functions.* http://www.hashcash.org/papers/amortizable.pdf: Whitepaper, 2002.

[7] R. Bazzi, *Access Cost for Asynchronous Byzantine Quorum Systems.* Distributed Computing, 2001.

[8] I. Bentov, A. Gabizon, and A. Mizrahi, *Cryptocurrencies Without Proof of Work.* http://arxiv.org/abs/1406.5694: Cryptography and Security, 2014.

[9] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. Kroll, and E. Felton, *SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies.* Security and Privacy, IEEE, 2015.

[10] K. Chandy and L. Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems.* ACM Transactions on Computer Systems, 3(1):63–75, 1985.

[11] J. Clark, A. Edward, and W. Felten, *Research Perspectives on Bitcoin and Second-generation Cryptocurrencies.* Systematization of Knowledge draft, diyhpl.us, 2015.

[12] FDIC, "Fdic law, regulations, related acts," https://www.fdic.gov/regulations/laws/rules/2000-8500.html, accessed: 2016-05-01.

[13] FFIEC, "IT booklets," http://ithandbook.ffiec.gov/it-booklets.aspx, accessed: 2016-05-01.

[14] M. Fischer, N. Lynch, and M. Paterson, *Impossibility of distributed consensus with one faulty process.* Journal of the ACM (JACM), 32(2):374–382, 1985.

[15] J. Garay, A. Kiayias, and N. Leonardos, *The Bitcoin Backbone Protocol: Analysis and Applications.* Eurocrypt, 2015.

[16] L. Lamport, R. Shostak, and M. Pease, *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3):382–401, 1982.

[17] D. Malkhi and M. Reiter, *Byzantine Quorum Systems.* STOC, 1997.

[18] D. Malkhi, M. Reiter, and A. Wool, *Load and Availability of Byzantine Quorum Systems.* SIAM J COMP, 2000.

[19] D. Malkhi, M. Reiter, A. Wool, and R. Wright, *Probabilistic Quorum Systems.* Information and Computation, 2001.

[20] D. Mazieres, *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus.* https://www.stellar.org/papers/stellar-consensus-protocol.pdf: Whitepaper, 2015.

[21] S. Nakamoto, *A Peer-to-peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf: Whitepaper, 2008.

[22] PCAOB, "Auditing standards," http://pcaobus.org/Standards/Auditing/Pages/ReorgStandards.aspx, accessed: 2016-05-01.

[23] M. Pease, R. Shostak, and L. Lamport, *Reaching agreement in the presence of faults.* Journal of the ACM (JACM), 27(2):228–234, 1980.

[24] D. Schwartz, N. Youngs, and A. Brittos, *The Ripple Protocol Consensus Algorithm.* https://ripple.com/files/ripple_consensus_whitepaper.pdf: Whitepaper, 2014.

[25] SEC, "Sarbanes-Oxley section 404," https://www.sec.gov/info/smallbus/404guide.pdf, accessed: 2016-05-01.

[26] P. Snow, B. Deery, J. Lu, D. Johnston, and P. Kirby, *Factom: Business Processes Secured by Immutable Audit Trails on the Blockchain.* Whitepaper, 2014.

[27] F. Tschorsch and B. Scheuermann, *Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies.* IACR, 2015.

[28] G. Wood, *Ethereum: A Secure Decentralised Generalized Transaction Ledger.* https://gavwood.com/paper.pdf: Whitepaper, 2015.