BLOCKCHAIN DRIVEN SECURE AND TRANSPARENT AUDIT LOGS

by

ASHAR AHMAD
MBA University of Central Florida, Orlando, Florida, 2010
MS University of Central Florida, Orlando, Florida, 2007
BS GIK Institute of Engineering Sciences and Technology, Topi, Pakistan, 1999

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2019

Major Professor: Aziz Mohaisen

ABSTRACT

In enterprise business applications, large volumes of data are generated daily, encoding business logic and transactions. Those applications are governed by various compliance requirements, making it essential to provide audit logs to store, track, and attribute data changes. In traditional audit log systems, logs are collected and stored in a centralized medium, making them prone to various forms of attacks and manipulations, including physical access and remote vulnerability exploitation attacks, and eventually allowing for unauthorized data modification, threatening the guarantees of audit logs. Moreover, such systems, and given their centralized nature, are characterized by a single point of failure. To harden the security of audit logs in enterprise business applications, in this work we explore the design space of blockchain-driven secure and transparent audit logs. We highlight the possibility of ensuring stronger security and functional properties by a generic blockchain system for audit logs, realize this generic design through BlockAudit, which addresses both security and functional requirements, optimize BlockAudit through multi-layered design in BlockTrail, and explore the design space further by assessing the functional and security properties the consensus algorithms through comprehensive evaluations.

The first component of this work is BlockAudit, a design blueprint that enumerates structural, functional, and security requirements for blockchain-based audit logs. BlockAudit uses a consensus-driven approach to replicate audit logs across multiple application peers to prevent the single-point-of-failure. BlockAudit also uses the Practical Byzantine Fault Tolerance (PBFT) protocol to achieve consensus over the state of the audit log data. We evaluate the performance of BlockAudit using event-driven simulations, abstracted from IBM Hyperledger.

Through the performance evaluation of BlockAudit, we pinpoint a need for high scalability and high throughput. We achieve those requirements by exploring various design optimizations to

the flat structure of BlockAudit inspired by real-world application characteristics. Namely, enterprise business applications often operate across non-overlapping geographical hierarchies including cities, counties, states, and federations. Leveraging that, we applied a similar transformation to BlockAudit to fragment the flat blockchain system into layers of codependent hierarchies, capable of processing transactions in parallel. Our hierarchical design, called BlockTrail, reduced the storage and search complexity for blockchains substantially while increasing the throughput and scalability of the audit log system. We prototyped BlockTrail on a custom-built blockchain simulator and analyzed its performance under varying transactions and network sizes demonstrating its advantages over BlockAudit.

A recurring limitation in both BlockAudit and BlockTrail is the use of the PBFT consensus protocol, which has high complexity and low scalability features. Moreover, the performance of our proposed designs was only evaluated in computer simulations, which sidestepped the complexities of the real-world blockchain system. To address those shortcomings, we created a generic cloud-based blockchain testbed capable of executing five well-known consensus algorithms including Proof-of-Work, Proof-of-Stake, Proof-of-Elapsed Time, Clique, and PBFT. For each consensus protocol, we instrumented our auditing system with various benchmarks to measure the latency, throughput, and scalability, highlighting the trade-off between the different protocols.

This work is dedicated to my parents, Zulfiqar and Naheed, grandfather, Muhammad, uncle Salahuddin, and siblings Sarah, Sadaf, and Naveen.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Aziz Mohaisen, for his support, persistence, kindness, encouragement, and patience in guiding me through this journey. Aziz helped me crystallize ideas, execute them, and report them in research articles. Without his guidance, this work would not have been possible, and for that I am thankful to him.

My doctoral studies at the University of Central Florida started more than ten years ago, interrupted by an industrial career, upon which I returned to finish my studies. For my initial work at UCF, I would like to thank Dr. Mostafa Bassiouni, who showed me a great deal of support, and spent countless hours patiently explaining ideas, discussing, and encouraging me to persist. Without his support in my earlier years at UCF, this work would not have been possible.

I would like to thank my dissertation committee members, Dr. Amro Awad , Dr. Clay Posey, and Dr. Wei Zhang for their willingness to serve on my committee, and for being flexible with my defense schedule. I would like to thank them also for all the feedback on my proposal.

I would like to thank my collaborators (in no specific order): Muhammad Saad, Afsah Anwar, Hisham Alasmary, Dr. Murat Yuksel, Dr. Charles Kamhoua, Dr. Laurent Njilla, Dr Kiran Anna, and Dr. Mostafa Bassiouni. Especially, I would like to thank Muhammad Saad who has been a constant collaborator for his help refining my ideas. I would like to also thank my teammates in the Security and Analytics Lab (SEAL) for their encouragement and support.

I would like to thank my colleagues at clear Village inc and GSA inc for their encouragement and support. I would especially like to thank Larry Zerbel and Chet Zema for taking out time to discuss and provide feedback on various aspects of my research.

I am thankful to my Friends Ahsan Mir, Ali baig, Adeel Ali, Adeel Bhutta, Kamran Sadiq, Zain

Masood, Arslan Basharat, Sohaib Hameed, Khawaja Umar Farooq, Shahzad Jumani, Humayun Mukhtar Khanx, Amir Bashir, Swastik Brahma, Kunal Motwani, Tauseef Iqbal, Raheel Rahman, Shazia Alam, Shafaq Chaudhry and others for their time, help and friendship.

Last but not least, I would like to thank my family, here in the US and back home, for their love, help, encouragement, and constant support.

This work was supported in part by KRF grant number NRF-2016K1A1A2912757 and by the Air Force Materiel Command award FA8750-16-0301.

PUBLICATIONS

1. **Ashar Ahmad**, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards Blockchain-Driven, Secure and Transparent Audit Logs. International Workshop on Distributed Ledger of Things, **DLoT 2018** (in conjunction with **MobiQuitous 2018**), New York City, USA. (**Best Paper Award**).

2. Muhammad Saad, Afsah Anwar, **Ashar Ahmad**, Hisam Alasmary, Murat Yukesl, and Aziz Mohaisen. RouteChain: Towards Blockchain-based Secure and Efficient BGP Routing. IEEE International Conference on Blockchain and Cryptocurrency (**IEEE ICBC 2019**), Seoul, South Korea, 14-17 May 2019.

3. **Ashar Ahmad**, Muhammad Saad, Laurent Njilla, Charles A. Kamhoua, Mostafa Bassiouni, and Aziz Mohaisen, BlockTrail: A Scalable Multichain Solution for Blockchain-based Audit Trails, Proceedings of IEEE International Conference on Communications, (**IEEE ICC 2019**), Shanghai, China, May 20-24, 2019

4. **Ashar Ahmad**, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen, "Secure and Transparent Audit Logs with BlockAudit". Elsevier Journal of Network and Computer Applications (**JNCA 2019**).

5. Saad, Muhammad, **Ashar Ahmad**, and Aziz Mohaisen. "Fighting Fake News Propagation with Blockchains." 2019 IEEE Conference on Communications and Network Security (CNS). IEEE, 2019.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Motivation and Background

With the advent of cloud computing, organizations are transitioning their applications to the cloud by using a software-as-a-service (SaaS) model. This model is cost-effective and allows an organization to pay only for the services that it uses, and removes the need for installing, running and maintaining applications inside the organization's data center. In government agencies, these information systems are often used to provide e-government services to citizens, government employees, and other entities interacting with a government agency or department. Some of the services provided by local government agencies are utility billing, business licenses, code enforcement, citizen engagement, building permit, computer-aided mass appraisal, and tax collection. Government agencies and corporate organizations maintain audit trails in their software applications to allow continuous monitoring, transparent auditing, and provenance assurance [87, 67]. Maintaining audit trails is also mandated by Federal laws, regulations, and standards. For example, European Unions' "General Data Protection Regulation" (GDPR) [43] and Canada's "The Personal Information Protection and Electronic Documents Act" (PIPEDA) [1] force the organizations to maintain audit trails for auditing, internal controls, error correction, fraud prevention, and compliance [69].

Audit trails enable organizations to audit the state of the information system, detect security breaches, track users' activities, and ensure user accountability. Due to these properties, audit logs are a critical component of information systems management, often used to track data changes. Changes in data trigger the generation of transactions, which are stored in the audit log and are typically stored in conventional databases, file systems or tape drives. The audit logs are then consulted, where needed, for a historical record of all data changes. As such the audit logs can be used for rollback and recovery (from intentional and unintentional data corruption), as well as for providing fine-grained information for provenance that can guide attribution of the data changes. In most of today's audit log systems, centralization of the audit log storage is a common theme, making them prone to a single point of failure, and exposing them to various forms of attacks [58, 54]. Such an adversary with root access to the database can manipulate critical information both in the database and the corresponding audit log. Once an audit log is compromised, the safety and transparency of the entire system is put to risk. To counter these attacks, blockchains have been used to provide a distributed, tamper-proof, and consensus-driven replication of audit logs across replicas in a peer-to-peer blockchain network [5, 83, 64].

Over the years, blockchains have gained significant attention due to their use in distributed systems [39]. In peer-to-peer settings, blockchains are capable of augmenting trust over an immutable state of system events [59]. The most prominent example of blockchain technology has been realized in Bitcoin [81], a peer-to-peer digital currency that enables secure transfer of digital assets without the need of a trusted intermediary. Since Bitcoin, the use of blockchains has become prevalent in various applications and industries including smart contracts [53, 22],cryptocurrencies [23], communication systems [78], health care [46, 68], Internet of Things [50, 57], and electronic voting [38, 47]. The potential of blockchains is fully utilized in an environment where 1) entities belonging to the same organization have competing interests [40] and/or 2) there is a need for immutable data management whose security increases with time [92, 61]. Audit log applications meet both conditions, and therfore they can benefit from blockchains to improve their security.

Applied to an audit log maintained by Enterprise business application, blockchains can replicate the information contained in audit logs over a set of peers, thereby providing them a consistent and tamper-proof view of the system [4]. Blockchains use an append-only model secured by strong cryptographic hash functions, and therefore, the security of data in the ledger increases while the blockchain grows with time. Furthermore, a malicious party intending to compromise the system will have to change logs maintained by a majority of peers. This increases the cost and complexity of the attack, eventually increasing the overall defense capability of the audit log application. Motivated by this, we propose an end-to-end blockchain-based audit log system called *BlockAudit*. In *BlockAudit*, we capture events from the data access layer of an enterprise application in the form of a transaction and securely replicate them over a set of blockchain nodes.

Despite the promising features that blockchains bring to audit logs, they also compound complexity associated with processing and storage of blockchain-based audit logs. First, a blockchain application requires a copy of the blockchain to be maintained at every peer. Additionally, blockchains employ an append-only model that results in an increasingly growing chain of transactions. These two issues can put enormous strain on the data storage/maintenance capability of peers [51]. Second, in practice, applications may incur thousands of transactions in a short period, generating thousands of audit entries. Government agencies use software systems at federal-, state-, county-, and city-level. The audit logs generated by these systems involve transactions that are solely relevant to the domain of the software system. In light of space and time overhead of blockchain systems, it is counter-intuitive to share data among different domains over a single ledger.

Blockchain systems are known to have low transaction throughput due to serialized transaction processing and the effort involved in obtaining a consensus among a large set of peers. This can be observed in the existing proof-of-work (PoW) blockchain systems such as Bitcoin, which has the maximum processing capacity of 7 transactions/second [7]. To address the throughput limitations of POW, variants of Byzantine Fault Tolerance (PBFT) protocols have been proposed. However, they suffer from low scalability due to high message complexity [74].

In the light of existing challenges faced by audit trail applications and the promising features of blockchains, it is natural to build a distributed and tamper-proof solution using blockchains for maintaining audit trails. As such, developments have been made towards distributed audit trail applications that use blockchains to securely manage data [83]. In theory, this provides an effective design to combat vulnerabilities and limitations of audit trails; however, in practice, these solutions may become infeasible and costly. Typical audit trail applications may incur up to thousands of transactions in a short timespan. With thousands of peers/verifiers present in the system, processing these transactions in a short period can be a major challenge. Some of these transactions may not be related to the peers who are verifying and storing them in their blockchains. This creates a massive overhead in terms of storage space and transaction processing time, making it unfeasible for several enterprise applications.

1.2 Contributions

Motivated by this useful combination of blockchains and audit logs, we present *BlockAudit*: an end-to-end solution that provides the design capabilities of audit logs as well as the security guarantees of blockchain systems. In *BlockAudit*, the audit log application is transformed into a blockchain system actuated by peers in a distributed network. *BlockAudit* captures transactions emitted by peers within the Online Transaction Processing(OLTP) system and aggregates them in a block. Each block is then timestamped and broadcast to all peers within the network. Each peer validates the authenticity and integrity of data within the block and transmits its approval. In *BlockAudit*, we use the PBFT protocol to create consensus among the network peers.

Our efforts are dedicated towards a secure and plug-and-play system that is capable of defending against the well-known attacks on audit logs. Additionally, we intend *BlockAudit* to enable a seamless transition of the legacy OLTP system to the blockchain-based solution. To that end, we expect *BlockAudit* to be application-agnostic, which is capable of masking blockchain layer complexities from the underlying application layer. Finally, we anticipate *BlockAudit* to meet all functional and security requirements with low latency and high throughput.

We envision that the performance and throughput can be increased by using a multi-layered blockchain system that provisions fast processing with minimal space overhead. Leveraging the nature of transactions in an audit trail, we dynamically determine the related peers and only select them to process the transaction. Furthermore, only the related peers may be allowed to keep a copy of the blockchain that records their relevant transactions.

We aim to come up with an end-to-end blockchain-based audit trail system that is secure, efficient and guarantees high throughput with a lower space footprint. We intend to create a model that replicates the state of audit trails across multiple entities that are distributed across multiple levels in the network. For that, we will take the example of information systems (with audit trails) used by cities, counties and states to provide various services. We will construct a layered blockchain model tailored to the needs of audit log application and the nature of its transactions. Based on that, we will dynamically determine the layer to which a new transaction might belong and get approval from peers related to the transaction. Additionally, we also aim to make our system secure against external and internal attacks.

Mindful of the various optimization choices one has to take, especially concerning the employed consensus algorithms, we explore a comparative study to shed light on the latency and throughput trade-off. For that, we first deploy a blockchain testbed consisting of a set of nodes hosted across various geographical locations. At each node, we encode the rules of five consensus protocols and the access control policies. To provide a "level playing field," we generate transactions of a uniform size for each consensus protocol and maintain a consistent network topology. To accurately reflect the real-world application, we use actual transactions from a real business system [32] for which log audit is built, and use its requirement thresholds to assess the feasibility of each protocol. For performance evaluation, we use two metrics including latency in the block confirmation and the transaction throughput. Finally, by contrasting the performance of all the consensus protocols, we present the most optimal solution for blockchain-based audit logs. To the best of our knowledge, this is the first attempt towards the performance evaluation of blockchain-based audit logs under various consensus protocols.

1.3 Organization and Roadmap

The rest of this dissertation is divided into five chapters and encompasses material from three papers [3, 5, 6]. In chapter 2, we introduce the background of this work, including some preliminary review of audit logs, blockchain systems, consensus algorithms, and a general threat model under which our work is analyzed. Chapter 3 uses material from [3, 5], coauthored by Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen, and introduces the design, analysis, and evaluation of *BlockAudit*, our preliminary blockchain-based system for tamper-proof audit logs. Chapter 4 is based on [6], coauthored by Muhammad Saad, Laurent Njilla, Charles A. Kamhoua, Mostafa Bassiouni and Aziz Mohaisen, which present the design, analysis, and evaluation of *BlockTrail*, an optimized hierarchical system for audit logs. In chapter 5, we introduce a testbed-based comparative analysis of various consensus protocols that could be used to optimize the operation of both *BlockAudit* and *BlockTrail* systems. In chapter 6, we present concluding remarks and provide directions for future work. Some material for each of these papers has also been incorporated in chapter 2 and the introductory chapter.

CHAPTER 2: BACKGROUND AND PRELIMINARIES

To contextualize our main contribution in this dissertation, in this chapter we provide a brief overview of various preliminaries and background knowledge useful for understanding the rest of this work. Namely, we will review audit logs and associated vulnerability, high-level introduction to blockchain systems, their features, and types, consensus algorithms, and the threat model used in analyzing the various systems proposed in this work.

2.1 Audit Logs

Online Transaction Processing (OLTP) systems are used for financial transactions, ticketing, billing, customer relationship management (CRM), Enterprise Resource Planning(ERP) and retail sales [84, 65]. These system are designed to process a large number of short transactions, while maintaining concurrency. Audit logs are kept in OLTP system to maintain history of completed transactions, and to monitor users' actions.*e.g.* In a financial system each payment transaction creates an audit record in the audit log. The total payments processed, and the aggregate volume of the transactions can be verified by consulting the audit entries in the audit log and comparing them with the data from the application. The audit logs can also be reviewed to detect anomalies, identify inconsistencies and malicious activities in these financial transactions.

Searchable audit logs are easily accessible from the application, and allow business users to reconstruct the chain of events that resulted in the current state of the system. In Figure 2.1, we provide an overview of audit log creation in an OLTP system. It shows the sequence of events that starts from user making changes using a client, and ends with committing these changes to database. The details of these changes, the previous values, the updated values, time, user Id, etc is stored in the audit log. In future, the entries in audit logs can be compared with the database for auditing , provenance assurance and data recovery.

These e-government applications generally have searchable audit logs, that are stored in the application database or file system. Authorized users can search and view audit logs associated with a object from the application user interface. Some of these applications communicate with other applications in the same city, county, or state. In rare cases, these interactions may also occur across states, at a federal level. As such, this can be viewed as an audit log management at four levela: a country-level federal audit log$(L1)$, a state-level audit log$(L2)$ (for each state), a county level audit log $(L3)$ (for each county), and a city audit log $(L4)$. Within each city, there are applications operated by state agencies that collect information to generate audit logs.

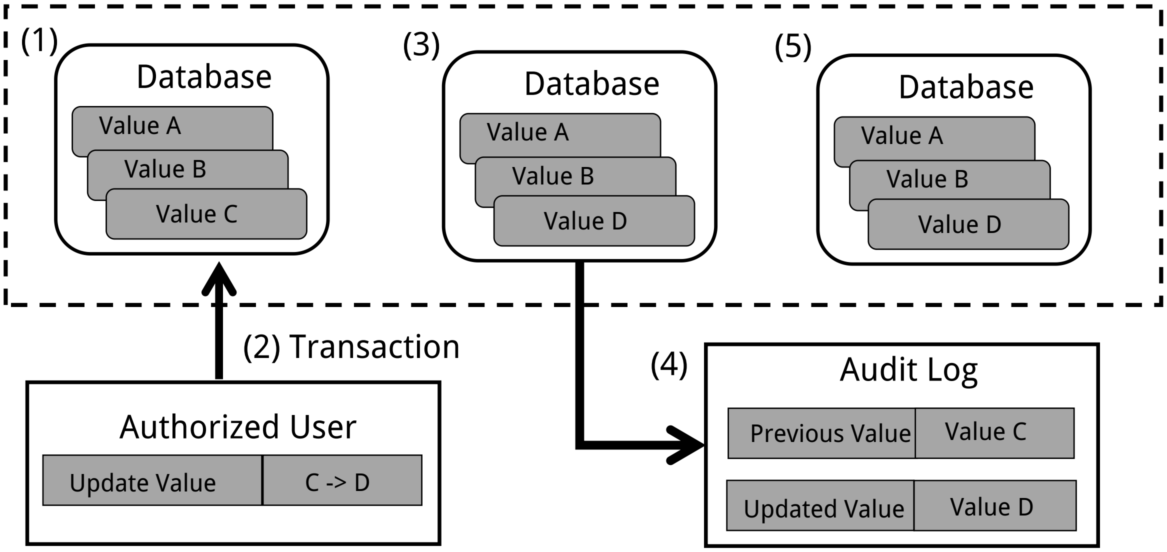


Figure 2.1: Generation of audit log in an OLTP system. Each step is labeled by a number to depict the sequence of event resulting in an audit log entry. The user changes an object value from C to D, and the ORM stores the change in the database and audit log.

2.1.1 Benefits of Audit Logs

With the help of audit logs, organization can keep track of changes in their information systems. Audit logs provide the following benefits to organizations.

- **Accountability and Reconstruction.** Audit logs can be used to associate users with events, and thus trace events to a responsible party. In case of a malicious activity, actions can be taken against the attacker. Additionally, audit logs can be used to chronologically monitor actions, and roll them back to an earlier state.
- **Anomaly Detection.** Audit logs can be reviewed to detect unauthorized or unusual events, and to carry out forensic analysis after an attack. Additionally, this data can also be used to detect bugs and software glitches.

2.1.2 Vulnerabilities in Audit Logs

Although audit logs are useful in tracking changes made to the database, and ensuring system accuracy, they are vulnerable to a series of attacks that could comprise data in an OLTP system. An attacker can exploit the known vulnerabilities in the auditing application to launch attacks and tamper the database and the audit logs. Conventionally, audit logs are protected by using off-site log replication and/or using append-only devices such as *Write Once Read Multiple* (WORM) optical devices or continuous feed printers. These schemes work under a weak security assumption that audit logs cannot be altered or destroyed by compromising the logging site. However, this weak notion of system security at the logging site is insufficient, and the attackers have exploited vulnerabilities at logging sites to tamper data [54, 58]. If an attacker acquires credentials of an authorized user, he can launch corrupt the database and its audit log. Since both the database and audit logs are compromised, the change in database and audit log tells the same story; thus such an attack could remain undetectable.

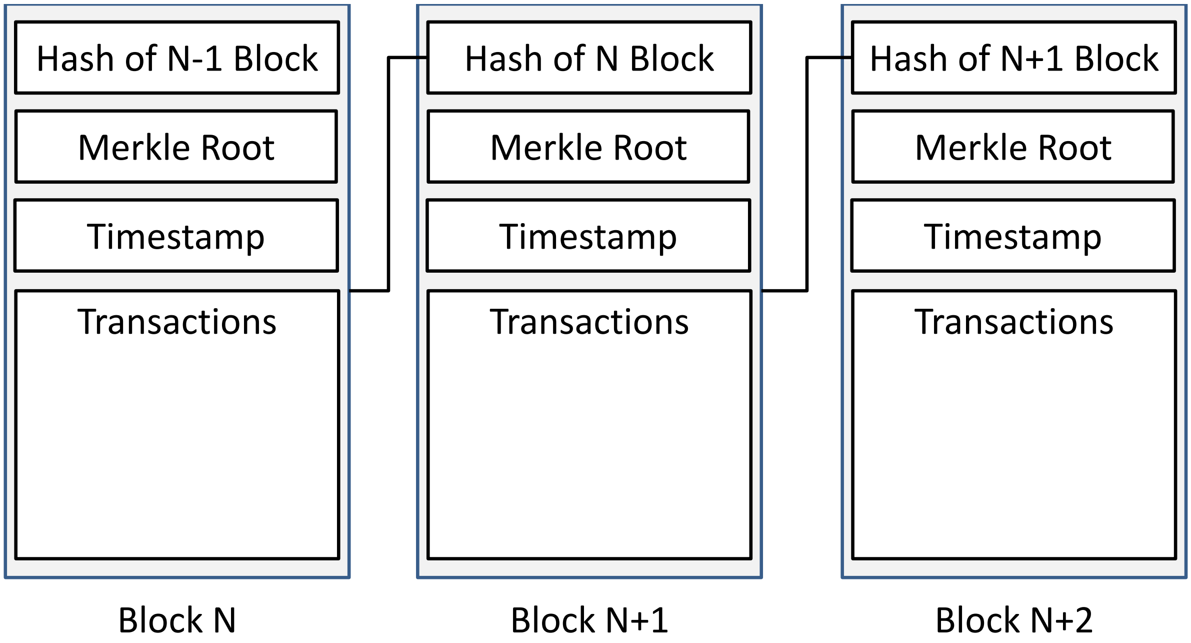


Figure 2.2: An overview of the Blockchain structure consisting of three blocks, containg a set of transactios. Notice that each block header consists of the hash of the previous block. This relationship gives blockchain, the property of an immutable ledger. Also, notice that the Merkle root ensures that the transactions are ordered in a sequence.

2.2 Blockchains

To address some of the issues in the audit logs, blockchains can be used to provide secure and tamper-proof data management without the need for a trusted intermediary. Blockchain systems are known to achieve consensus among peers in a distributed system [26, 75]. A blockchain consists of a sequence of records (transactions), that are linked to the previous block using a cryptographically secure hash function. The hash value of the previous block is used as an input pointer to the next block. A notable property of hash functions, among others, is the one-way property. Roughly speaking, the one-way property ensures that with a given hash value alone, it is hard to obtain the original message corresponding to it. Therefore, as the blockchain extends and more blocks are added, it becomes increasingly difficult to tamper with data stored in the blocks. Therefore, due to high security, a blockchain ledger is assumed to be tamper-resistant. Figure 2.2 provides on overview of the blockchain ledger and the link among the blocks.

2.2.1 Main Features

In the following, we outline some of the key characteristics of blockchains that pivot their utility in distributed systems.

2.2.1.1 Decentralization

In the traditional client-server systems, transactions are verified and approved by a trusted third party. This leads to a risk of a single-point-of-failure if the trusted third party is compromised. However, in blockchains, the root-of-trust is shifted from the trusted party to cryptographically secure blockchain system. The agreement over the state of the chain is brought by a consensus protocol that binds the approval of peers with mathematical constructs of the consensus protocol. Some of the well-known consensus algorithms are the proof-of-work (PoW), proof-of-stake (PoS), proof-of-elapsed time, and Byzantine fault tolerance [41]. Therefore, blockchains operate in a decentralized manner without the need for a trusted third party.

2.2.1.2 Auditability

The blockchain ledger is shared by all participants in the network. Any participant user can trace the history of a transaction and authenticate the exchange of values associated with the transaction. Therefore, it provides complete transparency and auditability to all participants. Moreover, due to the linkability in the blockchain data structure, it can be also be used to provide a high-level history of all the events leading up to any specific transaction. Hence, blockchains are also used to provide provenance in the system. Provenance can be of significant importance in auditing applications where parties can have conflicting interests.

2.2.2 Blockchains Types

Blockchain applications can be categorized based on their openness for user participation and the extent to which they make their data visible to outside world. Based on these traits, blockchains can be classified into three main categorizes namely public, private, and consortium blockchains.

2.2.2.1 Public Blockchain

Public blockchain systems is roughly an open access system in which any user can set up a blockchain node and become part of the system. Naturally, setting up a node requires installing a blockchain application client and often downloading the entire blockchain ledger. The blockchain data is publicly accessible to anyone who may or may not be part of the blockchain network. These blockchains are resource intensive, and considerable amount of computational power is required to add a new block, and there is no privacy of transaction. Popular blockchain applications including cryptocurrencies are public blockchains as exemplified by Bitcoin and Ethereum.

2.2.2.2 Private Blockchain

A private blockchain system is based on a permissioned network in which participants are invited and verified during the joining process. Private blockchains use an access control mechanism in which each node (user) is allowed a certain level of access to the system resources. The identity of the node and its activities are known to the other nodes. Often times, a key management system is used to specify the interaction policies of a node. Unlike public blockchains, the data of the private blockchains is not openly accessible to the users outside the network.

2.2.2.3 Consortium Blockchain

A consortium blockchain is a hybrid of both public and private blockchains. In a consortium, the permission to read and write data on blockchain can be extended to a certain number of peers, thereby distributing the access control to selected participants. These blockchains are generally used in enterprise settings *e.g.* Hyperledger fabric and Hyperleadger Sawtooth.

2.3 Consensus Protocols

A central piece of this work is to evaluate various consensus algorithms and their fitness for our core application. Moreover, in the design of the different blockchain-based audit systems, we utilize various protocols, which we review in the following. Note that additional details of some of those protocols (e.g., PBFT) are listed where they are used in context.

In distributed systems, consensus protocols ensure that the participating nodes (or processes) agree upon the *correct* state of data, typically assuming faulty or Byzantine nodes (behaves arbitrarily), which may halt the execution of processes. To address that, consensus protocols have the property of *fault tolerance*, which means that they can successfully execute the consensus objective notwithstanding a certain number of faulty replicas. There are many protocols developed in the distributed systems community to achieve consensus, including PoW, PoS, PBFT, Clique, PoET, etc. The main objective of those protocols is to replicate ledgers (blockchain) across multiple nodes in the absence of a trusted third party, while ensuring consistency.

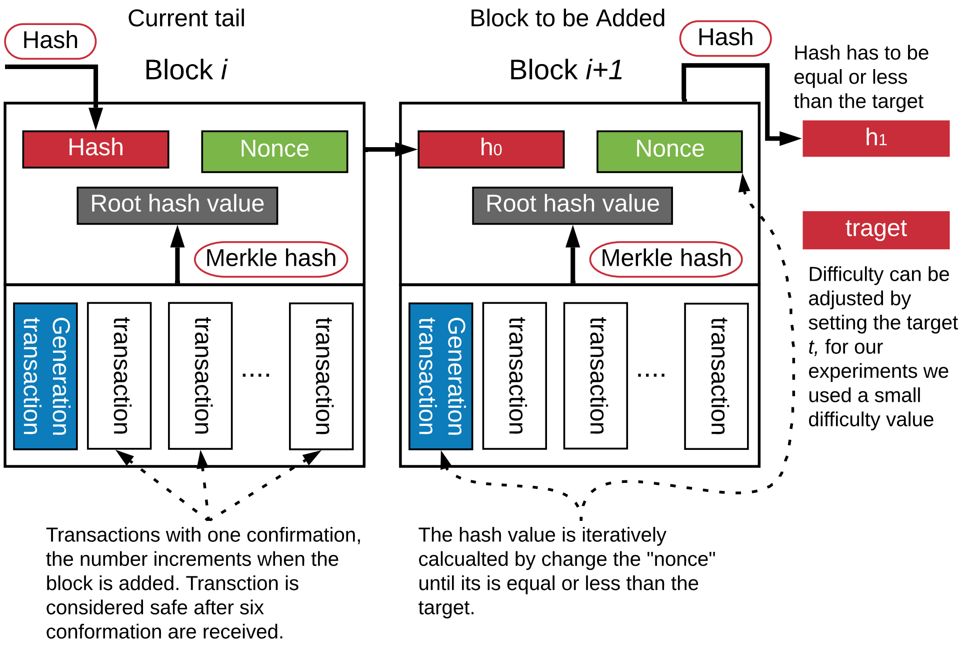

Current tail
Block *i*
Hash
Hash
Nonce
Root hash value
Merkle hash
Generation transaction
transaction
transaction
....
transaction
Block to be Added
Block *i+1*
Hash
h0
Nonce
Root hash value
Merkle hash
Generation transaction
transaction
transaction
....
transaction
Hash has to be equal or less than the target
h1
traget
Difficulty can be adjusted by setting the target *t*, for our experiments we used a small difficulty value
Transactions with one confirmation, the number increments when the block is added. Transction is considered safe after six conformation are received.
The hash value is iteratively calcualted by change the "nonce" until its is equal or less than the target.

Figure 2.3: General workflow in PoW showing two blocks. Block $i+1$ is linked to its parent i by a hash function. The linkage among blocks using hash functions is generic among all blockchain systems. However, in PoW, the block header also consists of a nonce value, which is concatenated with the hash of the previous block and the merkel root of all transactions. All these values are concatenated and hashed. If the resulting hash value is less than the target, the block meets the PoW requirement. Otherwise the nonce value is changed until the desired target threshold is met.

2.3.1 Proof-of-Work

The PoW consensus protocol involves solving a computationally expensive challenge to elect a leader. In PoW-based systems, the leader is also called a miner, who solves the challenge, orders transactions in a block, and broadcasts the block to the network. Upon receiving the block, other network nodes validate the correctness of the PoW solution and append the block to their blockchain. PoW is popularly used in cryptocurrencies such as Bitcoin and Ethereum. In Figure 2.3, we show an abstraction of PoW protocol. Note that the blockchain data structure shown in Figure 2.3 is common among many blockchain systems, where a block is linked to its parent block using a one-way hash function, and the hash function provides immutability and collision resistance to the blockchain ledger. Moreover, Figure 2.3 shows that the hash of the block header is required to be less than or equal to the target value set by the system. The target value can be calibrated to keep the block generation rate under a specified limit. Currently in bitcion a new block is added every ten minutes [74, 73]. PoW is known to be highly scalable since it sidesteps the multi-round message propagation in PBFT. At the same time, PoW is also considered to be energy-inefficient since it leads to a significant waste of critical computational resources.

2.3.2 Proof of Elapsed Time

PoET protocol uses Intel Software Guard Extensions (SGX) to execute the “leader election” code in a secure enclave. Each blockchain node executes the code, generating a random wait time during which the node remains idle. Once the wait time expires, the node is allowed to propose a block [31]. PoET randomizes the leader election process to maintain decentralization. In contrast, PoW and PoS may become centralized if a node acquires an exceptionally high hash rate or has a high balance for auction. Currently, Hyperledger Sawtooth supports the PoET protocol [49].

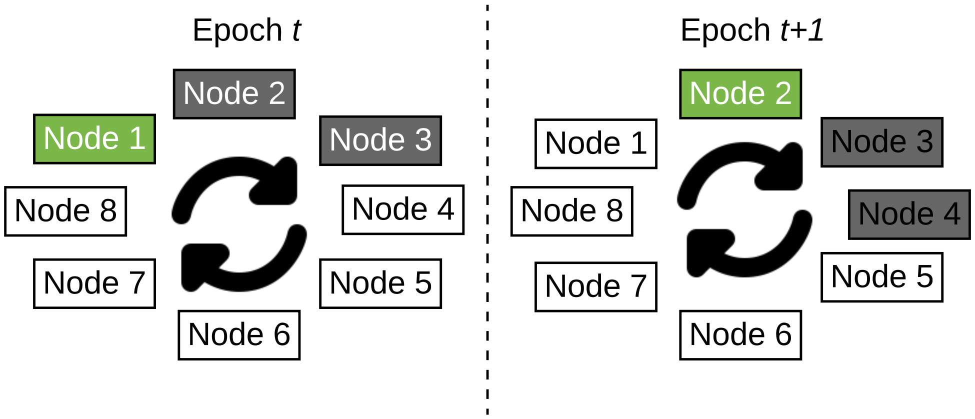


Figure 2.4: An abstraction of the Clique protocol's execution. For simplicity, we show that Clique uses a round-robin scheme to select primary replica. In each round, if the primary fails, the secondary replicas can propose the new block. The total number of replicas in one round are selected using the formula $N - N/2 + 1$, where N is the total number of nodes. Node 1 is the primary and Node 2 and Node 3 are secondary replicas for the first round, for the second round Node 2 is the primary and Node 3 and Node 4 and the secondary.

2.3.3 Clique

Clique belongs to the family of Proof-of-Authority (PoA) consensus protocols, popularly used in the permissioned blockchains [11, 71]. Clique is executed in epochs, and at the start of a new epoch, a transition block is issued which specifies the order of the next primary replicas (authorities). The selected primary replicas propose blocks at their respective turns. To avoid forks, in each round only one node is allowed to propose a block. Since the identity of each node is known prior to the execution of the protocol, a violation or deviation can be easily detected. Currently, Clique is used in the Ethereum Geth client. In Figure 2.4, we provide an abstract execution of Clique.

2.3.4 Practical Byzantine Fault Tolerance

PBFT belongs to the family of BFT protocols, popularly used for the state machine replication. Blockchain systems use PBFT to obtain consensus over the ordering of transactions in a block. PBFT is executed in three phases, namely, pre-prepare, prepare, and commit phase. The general workflow of PBFT is summarized below.

❶ In the *pre-prepare phase*, the primary replica receives transactions from a client, orders them in a block, and broadcasts the block to the rest of the replicas in the blockchain network. Upon receiving the block, each replica validates if the transactions are ordered correctly.

❷ In the *prepare phase*, each replica broadcasts its "approval" to all the other replicas in the network. Each replica waits for at least $2f$ responses from other replicas, where f is the number of faulty replicas.

❸ When $2f$ responses are collected, replicas enter the "commit" phase where they broadcast their final commitment to the client. If the client receives $f+1$ commitments from the network, it adds the block to the blockchain.

2.3.5 Proof-of-Stake

PoS protocol addresses the energy inefficiency of PoW and replaces the computational requirements with the notion of stake in the system [52]. In PoS cryptocurrencies, the stake is the number of coins owned by a user, which are then used to make bids during a block auction process. The auction winner is allowed to propose a block. Due to the growing concerns about the energy consumption in PoW, cryptocurrencies including Ethereum are switching to PoS. Despite the obvious benefits, PoS has its limitations, including the problem of "*the rich gets richer*" [19]. The auction naturally favors rich miners who are rewarded with a transaction fee. As a result, their stake further increases which allows them to win the subsequent auctions as well.

2.4 Threat Model

To sufficiently analyze the vulnerabilities of audit logs and set the security model objectives, we present the threat model for the auditing systems in this section.

Inspired by the limitations found in the prior work [54, 58], our threat model assumes an adversary that is capable of both physically accessing the trusted computing base (TCB) and remotely penetrating the OLTP system by exploiting software bugs. As such, the adversary can be a malicious third party aiming to tamper data to compromise auditing procedures. This would require the adversary to obtain root privileges to the system, or have significant knowledge of the system architecture. Additionally, the adversary can also hack and acquire the credentials of a root user of the system. This can be carried out using various attack procedures available in the conventional attack catalog [55]. However, possessing the knowledge of a private database system or a remotely acquiring credentials of a root user would require exceptional capabilities for the adversary. Therefore, we assume the third party attacker to have strong capabilities.

In a less hostile environment, the adversary can also be someone from within the system with root privileges. For instance, a corrupt auditor, who has tampered data for personal gains, might want to cover his act by changing data values. In contrast to the third party attacker, this adversary will not need sophisticated capabilities since he already has root privileges and the system knowledge.

For the system architecture, we assume an OLTP system similar to a retail sale repository. The system implements the design logic of an application using secure communication protocols such as SSL/TLS. Moreover, the system has a database that keeps records of sales and maintains a remote audit log. The audit log keeps track of the database changes through transactions, as shown in Figure 2.1. In such a design, the attacker can exploit the system by launching two possible attacks, namely the physical access attack and the remote vulnerability attack.

2.4.1 The Physical Access Attack

In one of the notable works on audit logs [79], the authors state "*an intruder (or an insider) who gains physical access to the DBMS server will have full freedom to corrupt any database file, including data, timestamps, and audit logs stored in tuples* ." We use their specification of "physical access" to define the "physical access attack" in which the adversary uses the root privileges to corrupt the database. As shown in Figure 2.1, the adversary will generate a series of transactions to change the values of objects in the database. Once the attacker manipulates the data, the database will automatically generate an audit log, tracking all changes made by the attacker. However, to evade detection, the attacker can either delete the newly generated audit log or modify its values. Furthermore, the attacker will also be able to tamper the history maintained by the audit log in order to corrupt the auditing process. Therefore, in the physical access attack, we assume an adversary inside or outside the system who has access to the key system components.

2.4.2 The Remote Vulnerability Attack

Faithfully embracing the formal specification of the physical access and remote vulnerability in [79], we also define the remote vulnerability attack in which the attacker may only exploit the default vulnerabilities in the OLTP applications such as software malfunctions, malware attacks, buffer overflow attacks etc.. In this attack, the adversary, although not as strong as the physical access attack may still be able to contaminate the database and the audit log with wrong information. Despite these adversarial capabilities, we assume that the OLTP application is secure against the conventional database and network attacks such as SQL injection and weak authentication. Generally, database systems used by corporate organizations are secure against these conventional attacks, and for the application service used in this paper, we ensure this requirement is meet.

CHAPTER 3: BLOCKAUDIT

Audit logs serve as a critical component in enterprise business systems and are used for auditing, storing, and tracking changes made to the data. However, audit logs are vulnerable to a series of attacks enabling adversaries to tamper data and the corresponding audit logs without getting detected. Among them, two well-known attacks are "the physical access attack," which exploits root privileges, and "the remote vulnerability attack," which compromises known vulnerabilities in database systems. In this paper, we present *BlockAudit*: a scalable and tamper-proof system that leverages the design properties of audit logs and security guarantees of blockchain to enable secure and trustworthy audit logs. Towards that, we construct the design schema of *BlockAudit* and outline its functional and operational procedures. We implement our design on a custom-built PBFT blockchain system and evaluate the performance in terms of latency, network size, payload size, and transaction rate. Our results show that conventional audit logs can seamlessly transition into *BlockAudit* to achieve higher security and defend against the known attacks on audit logs.

3.1 Introduction

In this chapter, we present *BlockAudit*: a tamper-proof system that leverages the design properties of audit logs and security guarantees of blockchain to enable secure and trustworthy audit logs. Towards that, we construct the design schema of *BlockAudit* and outline its functional and operational procedures. We implement our design on a custom-built PBFT)blockchain system (see chapter 2) and evaluate the performance in terms of latency, network size, payload size, and transaction rate. Our results show that conventional audit logs can seamlessly transition into *BlockAudit* to achieve higher security and defend against the known attacks on audit logs.

Broadly speaking, in *BlockAudit*, we 1) capture system events generated by the data access layer of an enterprise application, 2) transform the acquired information into blockchain compatible transactions, 3) construct a peer-to-peer network consisting of entities that evaluate and approve the authenticity of transactions by executing a consensus protocol, and 4) lock the transaction in an append-only and immutable blockchain ledger, maintained by each network entity.

In summary, in this work we make the following key contributions:

❶ We outline security vulnerabilities in audit log applications and discuss shortcomings of the prior work in addressing those vulnerabilities.

❷ We present a blockchain-based audit log system called *BlockAudit* which addresses these vulnerabilities and ensures security, transparency, and provenance in the auditing system. Towards that, we review the modular constructs of blockchain systems and discuss suitable design choices that best fit the requirements of an auditing system.

❸ We test the design of *BlockAudit* using a real-world application and analyze its performance using three evaluation metrics, namely the latency, the network size, and the payload size.

❹ Based on observations made from theoretical analysis and experiments, we discuss our proposed solution and provide future directions for research on blockchain-based audit logs.

3.2 Problem Statement and Design Requirements

The prior related research provides the groundwork for securing audit logs with blockchains and represent the foundation of our work. However, our major contribution is seen in our focus on audit logs related to enterprise business applications, focusing on seamless integration with current systems,scalability and performance. As outlined in §3.1, blockchain applications may vary in their access control policies and consensus schemes. Exploring the blockchain model for Enterprise business applications would require an understanding of their requirements, and methods to overcome the domain-specific design challenges, which we explore in this chapter.

Another limitation that can be observed in [37, 79] is the inability to address Byzantine behavior among network peers. In other words, the application assumes all participating entities faithfully execute the consensus protocol without incurring any malicious behavior. However, in distributed systems adversaries can control a subset of replicas who can behave arbitrarily in order to withhold transaction processing and cause conflicting views among other replicas. Tolerance towards Byzantine nodes is a function of consensus schemes to be applied. For instance, permissionless blockchain applications such as Bitcoin can tolerate up to 50% of Byzantine nodes while maintaining operational consistency. On the other hand, PBFT-based private blockchains can tolerate only 30% Byzantine nodes. Therefore, the selection of a consensus algorithm can influence the security model of the application. In *BlockAudit*, we address the aforementioned limitations and present an end-to-end solution constructed by transforming knowledge problems into design problems.

3.2.1 Design Engineering

So far, we have discussed the benefits of audit logs, their key vulnerabilities, and the existing solutions that address those vulnerabilities. We have also presented a threat model to outline adversarial conditions. In this section, we use this knowledge to make design choices to meet the requirements of a practical blockchain-based audit log solution. In the following, we define functional, structural, and security requirements that we expect *BlockAudit* to meet.

3.2.1.1 Functional Requirements

An audit log application is expected to ensure trust in the application data and provide tamper-proof evidence of transaction history when needed. Data tampering has to be prevented for the application data as well as the data stored in the audit logs. However, a priority is given to the audit logs, since they are used to establish provenance. For this purpose, the audit log data should be stored across multiple peers in such a way that it remains consistent at each node, and therefore, hard to corrupt. If tampering happens at any node, the system should be able to detect and correct it. This requirement, however useful, comes with an assumption that a majority of peers behaves honestly, and faithfully executes the system protocols.

For audit data to be added to the blockchain, the participating peers in the audit log network must reach a quick consensus over a newly generated transaction. Since audit logs are generated in real-time and persisted inside the database transaction, therefore, any delay in using distributed audit logs adversely affects the system performance. In order to prevent such delays, the system needs to have low latency while maintaining the capability of processing a large volume of transactions. Additionally, the application should not add any data to the blockchain based audit logs without reaching consensus among a majority of peers.

The audit log system architecture should be modular and service-oriented so that it is possible for various types of applications to participate and benefit from this system. Moreover, audit logs should be data agnostic and must not rely upon the nature of data that is stored in them. The Enterprise business application should be able to provide data in any format as per the requirements of the application, for storage in blockchian-driven audit logs.

Finally, the audit log system should provide searching and retrieval capability to enable the retrieval of any desired transaction or a set of transactions (e.g., audit log entries for the last ten minutes, all audit log entries registered against a specific user ID, etc.). The search needs to be fast and responsive to ensure the end user is able to perform the audit in real-time.

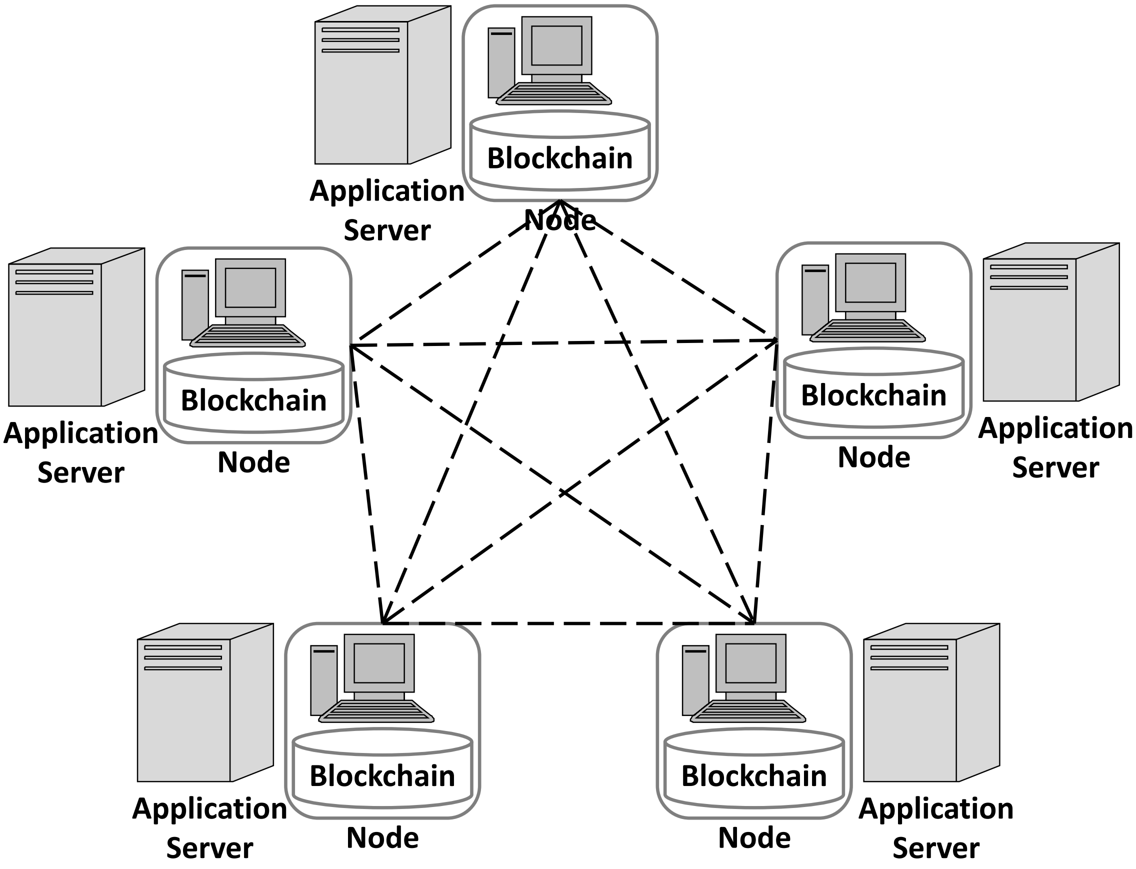


Figure 3.1: The network overview of nodes employing *BlockAudit*. Notice that each node maintains an interface that connects them to the audit log application. They exchange transactions with one another during the application life-cycle.

3.2.1.2 Structural Requirements

Keeping in view of the design the baseline models introduced [37, 79], we envision that *BlockAudit* must operate in a distributed manner with application services running on multiple hosts without a central authority. As such each application peer would require its own blockchain node to become part of the the *BlockAudit* network. The audit log system should have a high throughput and should be able to process a large number of transactions. *BlockAudit* should be able to support transactions of various sizes since the transaction size varies in audit log applications. The audit log system should be easy to integrate with existing system with minimal structural and functional changes in the application. It should also be independent of the underlying application database. Finally, the system auditing should be secure, transparent, and visible to all peers within the network.

3.2.1.3 Security Requirements

In the light of our threat model §2.4, we require *BlockAudit* to be secure in adversarial conditions. To that end, if the adversary launches a physical access attack, *BlockAudit* should be able to neutralize it and prevent data tampering at the source. If the adversary launches the remote vulnerability attack, *BlockAudit* should stop the attack propagation across the network peers. In other words, if the adversary exploits a bug in the audit log of one peer, *BlockAudit* should immediately recognize the attack and notify the victim peer. Furthermore, the infection should be curtailed at the target zone, preventing its spread through the network.

In addition to the baseline attack model, we also expect *BlockAudit* to remain secure in the presence of Byzantine nodes. Therefore, if a strong adversary controls a subset of nodes in the network, he should not be able to corrupt audit logs or delay transaction verification. This can be achieved by either raising the attack cost *i.e.,* constructing a large network or relaxing anonymity so that the adversary risks identity exposure by misbehaving.

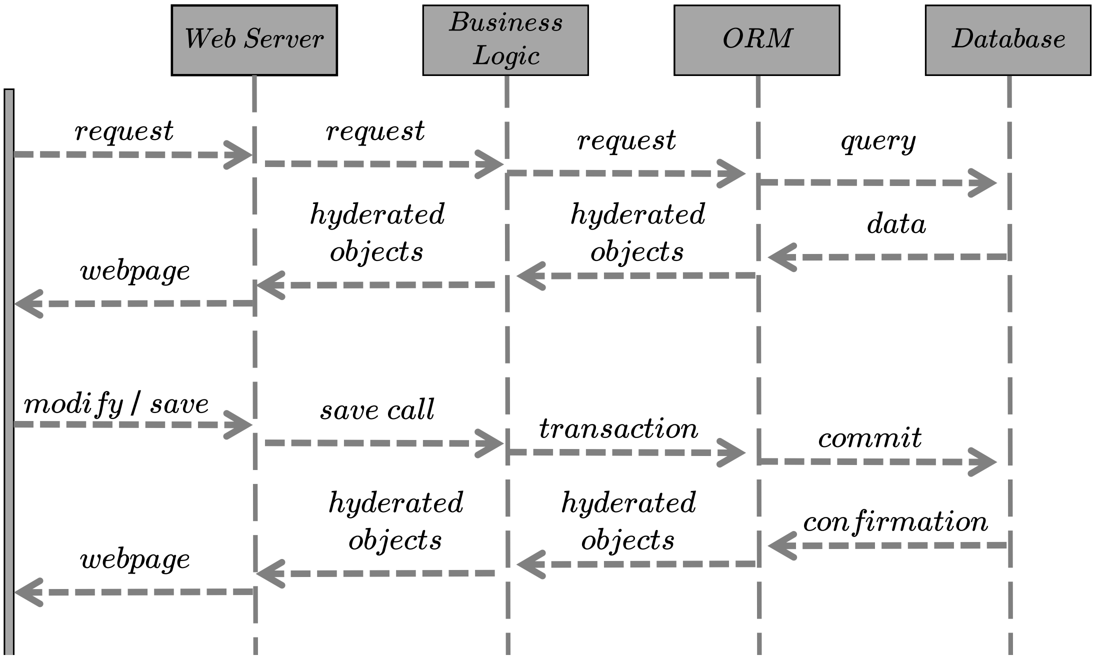


Figure 3.2: The information flow between various components of the application. Notice that the transaction is generated at the business logic layer, and once the database commits to the transaction it is rendered on the web page.

3.3 BlockAudit Design and Implementation

In this section, we show the implementation of *BlockAudit*. First, we describe the eGovernment application that we used to generate audit logs. Next, we show how a blockchain network is constructed to integrate audit logs. In that, we describe the methods of generating transactions, creating a distributed network, managing the access control, and developing consensus among peers over the state of the audit logs.

3.3.1 Application Architecture

We used an e-government application, built using Service Oriented Architecture(SOA), to implement *BlockAudit*. This application has web and mobile clients, that interact with a business logic layer using REST services. Business logic layer interacts with the database using a data access layer, which primarily uses Object-relational mapping (ORM) to communicate with the database. Figure 3.6 shows the layered application architecture of our e-government application along with its blockchain component. In Figure 3.2 we provide an overview of each layer and the sequence of data exchange that generates an audit trail which is then passed to the blockchain system [6]. Some of the key components of our e-government application system are discussed below.

3.3.1.1 Clients

Our application has mobile and web-based clients. The web applications are built using *asp.net* and users access the application services through a web browser. Additionally, native clients are provided for Android and iOS, built using their respective development frameworks. The web application and web services are hosted on Microsoft's Internet Information Services (IIS) web server and authorized users can access the application 24/7 via a web browser and mobile client. The public side portal is available on the Internet and gives public users access to information without authentication. Atop this, a staff portal is provided to the organization staff which is only accessible from within the organization, thereby providing another security layer.

3.3.1.2 Business Logic Layer

The business logic layer is an interface between clients and the database layer, responsible for implementing business rules. Among other functions, the business logic layer also manages data creation, data storage, and changes to the data with the help of object-relational mapping (ORM). Upon receiving a request from the client, the web server instantiates the relevant objects in the business logic layer, which uses the ORM to send the processed object to the client. The ORM writes changes to the objects in the relational database management system (RDBMS) tables.

3.3.1.3 Data Access Layer

The data access layer is built using *NHibernate* ORM [33] (ORM solution for Microsoft .NET framework), which provides a framework for mapping classes in the business logic layer to the database tables using an object-oriented paradigm [8, 13]. Modern web applications are well suited for this technique since they are multi-threaded and are rapidly evolving. ORM automatically create the SQL statements to hydrate data object and also executes SQL statements to flush and commit changes to the database. ORM reduces the code complexity of database interaction and allows software engineers to focus on writing business logic. Our current application instances are either running or the Oracle and or SQL Server relational database management systems. In our application, we have extended the ORM to generate the audit trail entry for each transaction format that can be stored in the application database or an external audit provider *e.g.Blocktrail*. In Figure 3.2, we provide the information flow between various components of the application and in [3], we provide the details of generating audit trails from this layer.

3.3.2 Generating Audit Logs

In this section, we show how the application generates an audit log once the user commits a transaction. To implement auditing, three events provided by *nHibernate* are used, namely *IPostInsertEventListener*, *IPostUpdateEventListener*, and *IPostDeleteEventListener*.

IPostInsertEventListener event is triggered once a transient entity is persisted for the first time. Each class that requires auditing is marked with *Auditable* attribute, which is then used to create audit logs for classes containing this attribute. All mapped properties are then audited by default and a suppress audit attribute is added to suppress auditing of a target property. Usually, and by default, all properties are audited. However, in special cases where auditing is not required, the *SuppressAudit* attribute is added to the property. In Algorithm 1, we show the process of generating the audit log when *IPostInsertEventListener* event is triggered.

When an audit entry is created, it contains a session ID (transaction ID), a class name, an event type (Insert, Update, or Delete), audit ID, creation date, user ID, URL, and a collection of values for all properties. The collection of values consists of the old value before the update and the new

Algorithm 1: Creation of the audit log entry for persisting new objects to the database

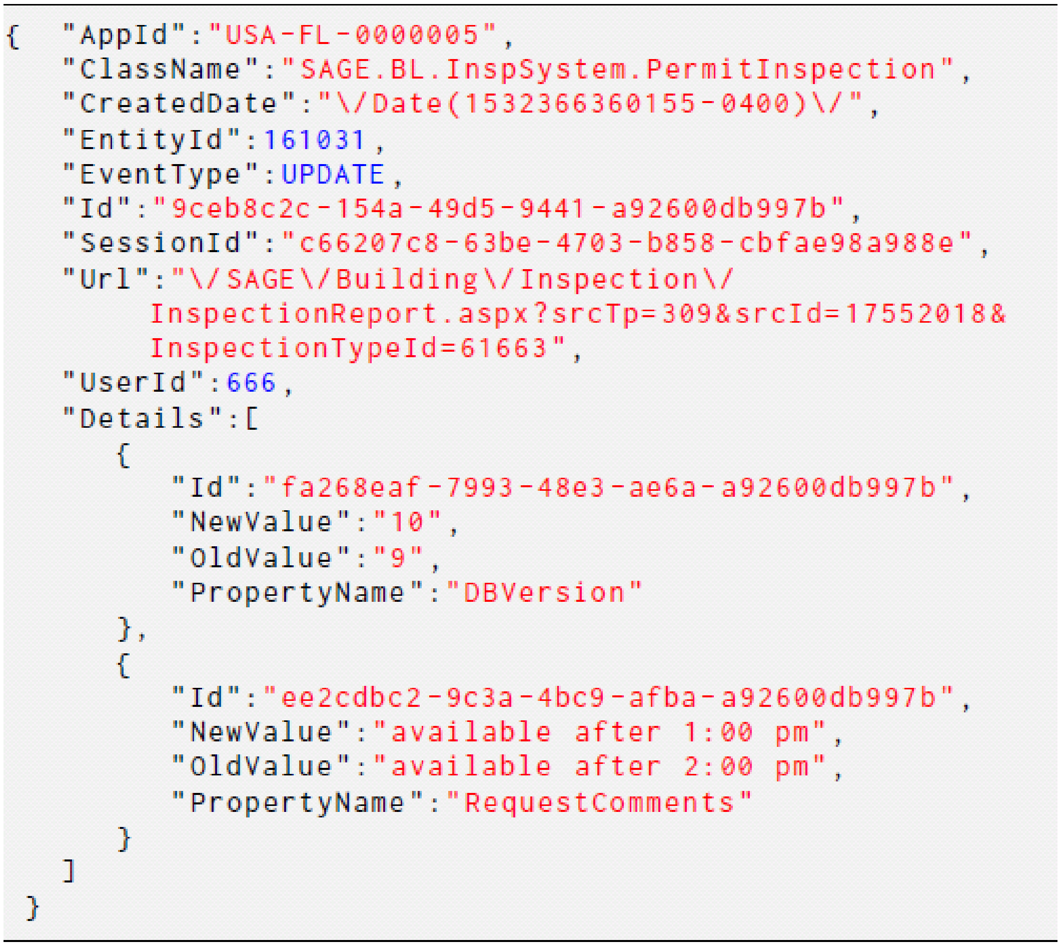
```
Function OnPostInsert(PostInsertEvent e)
    if (e.Entity = AuditLog) then
        return;
            ▷ Do not create log entry for AuditLog
    if (e.Entity = AuditLogDetail) then
        return;
        ▷ Do not create log entry for AuditLogDetail
    if e.HasAttribute(AuditableAttribute) then
        var new AuditLog(SessionId,
          AuditEventType.INSERT,
          EntityName, EntityId, UserId, Url);
    for $i = 0; i <$
      $e.Persister.PropertyNames.Length - 1$ do
        if (suppressedProp.Contains(propName))
          then
            continue;
        auditLog.AddDetail(propName, oldValue,
          newValue);
    if (auditLog.Details.Any() then
        SaveToBlockchain(auditLog);
```

Algorithm 2: Creation of the audit log entry for persisting existing objects to the database

```
1  Function OnPostUpdate(PostUpdateEvent e)
2                      ▷ Do not create log entry for AuditLog
3      if (e.Entity = AuditLog) then
4          return;
5          ▷ Do not create log entry for AuditLogDetail
6      if (e.Entity = AuditLogDetail) then
7          return;
8      if e.HasAttribute(AuditableAttribute) then
9          var new AuditLog(SessionId,
            AuditEventType.UPDATE,
            EntityName, EntityId, UserId, Url);
10     for
        $i = 0; i < e.Persister.PropNames.Length - 1$
        do
11         if (suppressedProp.Contains(propName))
            then
12             continue;
13         if (oldldValue <>newValue)) then
14             auditLog.AddDetail(propName,
                oldValue, newValue);
15     if (auditLog.Details.Any() then
16         SaveToBlockchain(auditLog);
```

value resulting from the update. Moreover, during an update, old and new values are compared. Only if the two values are different from one another, the change is committed to the audit log. In Algorithm 2, we outline this procedure of updating audit logs. Currently, these audit logs are saved inside an RDBMS using two tables, the AuditLog table, and the AuditLogDetail table. Furthermore, Globally Unique Identifiers(GUID) are used as primary keys in auditlog tables.

Once a change is observed in a class, the ORM's event handler is invoked. Similarly, the event handler is also invoked when the change is observed in the "AuditLog" and the "AuditLogDetail" classes. Lines 2–5 in Algorithm 1 and Algorithm 2, prevent the creation of logs for Audit Classes.



```
{   "AppId":"USA-FL-0000005",
    "ClassName":"SAGE.BL.InspSystem.PermitInspection",
    "CreatedDate":"\/Date(1532366360155-0400)\/",
    "EntityId":161031,
    "EventType":UPDATE,
    "Id":"9ceb8c2c-154a-49d5-9441-a92600db997b",
    "SessionId":"c66207c8-63be-4703-b858-cbfae98a988e",
    "Url":"\/SAGE\/Building\/Inspection\/
         InspectionReport.aspx?srcTp=309&srcId=17552018&
         InspectionTypeId=61663",
    "UserId":666,
    "Details":[
       {
          "Id":"fa268eaf-7993-48e3-ae6a-a92600db997b",
          "NewValue":"10",
          "OldValue":"9",
          "PropertyName":"DBVersion"
       },
       {
          "Id":"ee2cdbc2-9c3a-4bc9-afba-a92600db997b",
          "NewValue":"available after 1:00 pm",
          "OldValue":"available after 2:00 pm",
          "PropertyName":"RequestComments"
       }
    ]
 }
```

Figure 3.3: Blockchain transaction generated after serializing data from the audit log. This transaction is exchanged among the peers during the application runtime.

In the absence of this condition, the event logger would fall into an infinite event loop.The infinite loop can also be prevented by removing the “AuditableAttribute” from the audit classes. However, we use lines 2–5 as a check to avoid the loop in case a developer adds the attribute by mistake.

Once an audit log is generated, users have various ways of accessing the audit logs. Such using the audit search screen and searching based on users, date range, business object type etc. The application also provides a link to the audit log page from the primary object. The link allows end users to look at the object history and track any discrepancy caused by a bug or malicious activity.

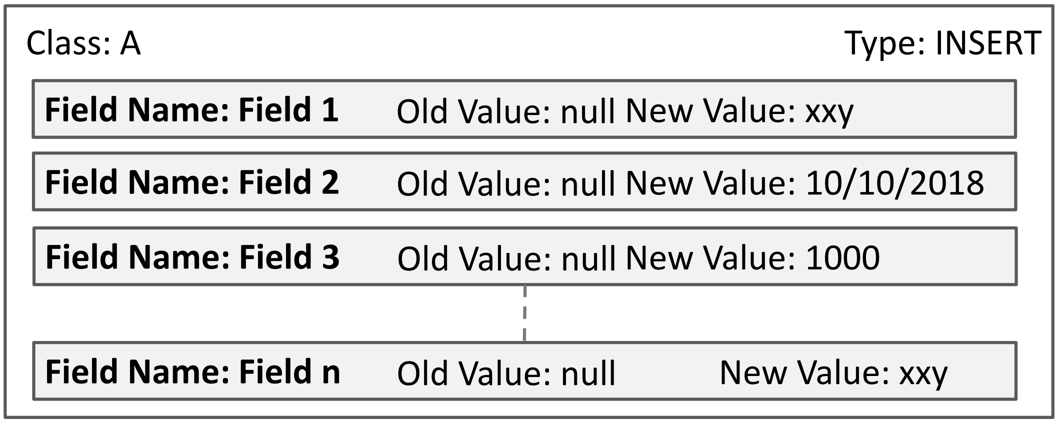
3.3.3 Blockchain Integration to Audit Logs

In this section, we will show how audit logs, obtained from our application, are integrated with the blockchain. So far in our design, we have an application that stores audit logs upon receiving a transaction. Now, we need to convert the audit log data into a blockchain-compatible format (blockchain transactions) and construct a distributed peer-to-peer network to replicate the state of the blockchain over multiple nodes. In our current implementation the audit log is generated using the ORM, which calls a Representational State Transfer(REST) Application Programming Interface(API) to store the audit log entry.

We used the ORM to create audit logs because the ORM acts as the gateway to capture all database transactions. Therefore, it is efficient to take advantage of ORM events to capture all the database changes and convert them into a JSON packet for the REST API. Our design is flexible and generic, and can also be used by other applications that do not use the ORM. Other than the ORM, the application layer or the data access layer can also be extended to capture the database changes in a JSON format and invoke the REST API. Moreover, the REST API can also be used by applications built using a serverless architecture.

3.3.3.1 Creating Blockchain Network

In *BlockAudit*, the network consists of peers that all have the privilege of accessing the application and creating an audit log. This network is connected in peer-to-peer model [42] and each peer can connect to all the other peers in the network. Connecting to a bigger subset of peers is beneficial, because it can avoid unnecessary delays in receiving critical information.



Class: A

Type: INSERT

Field Name: Field 1 Old Value: null New Value: xxy

Field Name: Field 2 Old Value: null New Value: 10/10/2018

Field Name: Field 3 Old Value: null New Value: 1000

Field Name: Field n Old Value: null New Value: xxy

Figure 3.4: Audit Entry generation for an object. Object A is a new object that is being inserted into the database for the first time. Note that an audit log entry only contains one object.

Access Control. As mentioned in section 3.1, access controls may vary across blockchain application. These applications can be permissionless (open access) or permissioned (selective access). In permissionless applications, such as Bitcoin, an arbitrary user can download the Bitcoin Core software and join the network. However, in the private and permissioned blockchains, an access control mechanism is applied that restricts the participation to only approved users. Since audit logs consist of sensitive data, therefore, in *BlockAudit* we use a permissioned blockchain with access control provisioned to selected users. In permissioned blockchains, adjusting access control is trivial since any custom membership service can be used for the access control [17]. To avoid runtime complexities, we do peer screening prior to network creation. The peer screening is done based on the IP addresses in which we curate a list of IP addresses, compile them in executable code, and provide the code to each peer. Upon executing it, the peer gets connected to the network.

Additionally, each node is required to keep a copy of the blockchain at their servers and maintain a persistent connection with their corresponding application server. Persistent connections are necessary to maintain an up-to-date view of the blockchain in order to process, validate, and forward transactions, as well as to avoid unwanted forks and partitioning attacks that may result from an outdated blockchain view.

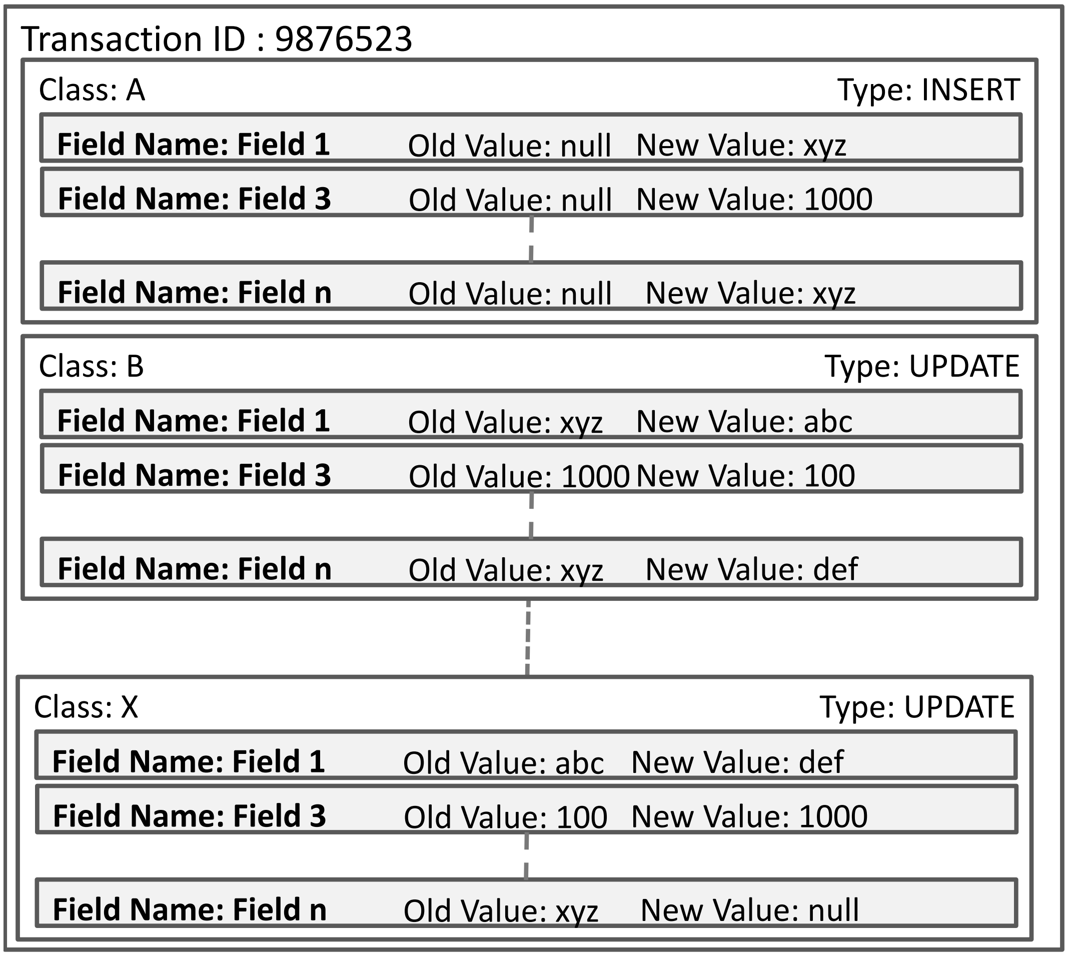


Figure 3.5: Audit generation for a transaction spanning across multiple objects. Object A is inserted and Object B and X are updated.

Table 3.1: The description of fields of audit log JSON packet. The packet has a header and a detail record for each updated property. The detail records can have $(0-n)$ records depending upon the class properties that are being updated.

Field Name		**Description**
AppId		Unique identifier for the application
ClassName		The name of updated application class
CreatedDate		Creation date and time for the audit record
EntityId		Unique key for business object
EventType		This would be Update, Insert or Delete
Id		Unique id of the audit record, GUID
SessionId		Unique Id for a transaction
Url		Application page creating the audit
UserId		User id, for the user making the change
Details $(0-n)$	**Id**	Unique Id for the detail record
	NewValue	Current property value
	OldValue	Old Property value
	Name	Name of property e.g owner's name

Table 3.2: *Clear Village's* actual transaction sizes in (bytes) for the three transaction schemes, based on the transactions from October 2018. The average size is between 32 bytes to 9,302 bytes.

Type	**Max**	**Min**	**Average**
Per Transaction	501,760	321	9,302.81
Per Record	31,104	319	2,617.80
Fixed Length	32	32	32.00

3.3.3.2 Creating Blockchain Transactions

Once the network architecture is laid out, the next step is to create blockchain-compatible transactions from the audit log data. For that, we convert the audit log data to a JavaScript Object Notation(JSON) format [34]. We preferred JSON over other standard data storage formats such as XML, due to its data structure compactness and storage flexibility. To obtain a blockchain transaction, we first pass the audit log data to a function that serializes it to JSON and calls *createAudit* REST [48] web service to create the audit log transaction. Each JSON packet is then treated as a blockchain transaction, and as soon as a node in the network receives a transaction, it broadcasts the packet to the rest of the network. Nodes can connect to multiple peers to avoid the risk of delayed transactions due to malicious peer behavior or network latency. In Table 3.2, we show the average transaction size from our sample system for October 2018. In Table 3.1 we describe the purpose of each field in the audit JSON packet and in Figure 3.3, we show the data structure of the blockchain transaction that is obtained after serializing data from the audit log.

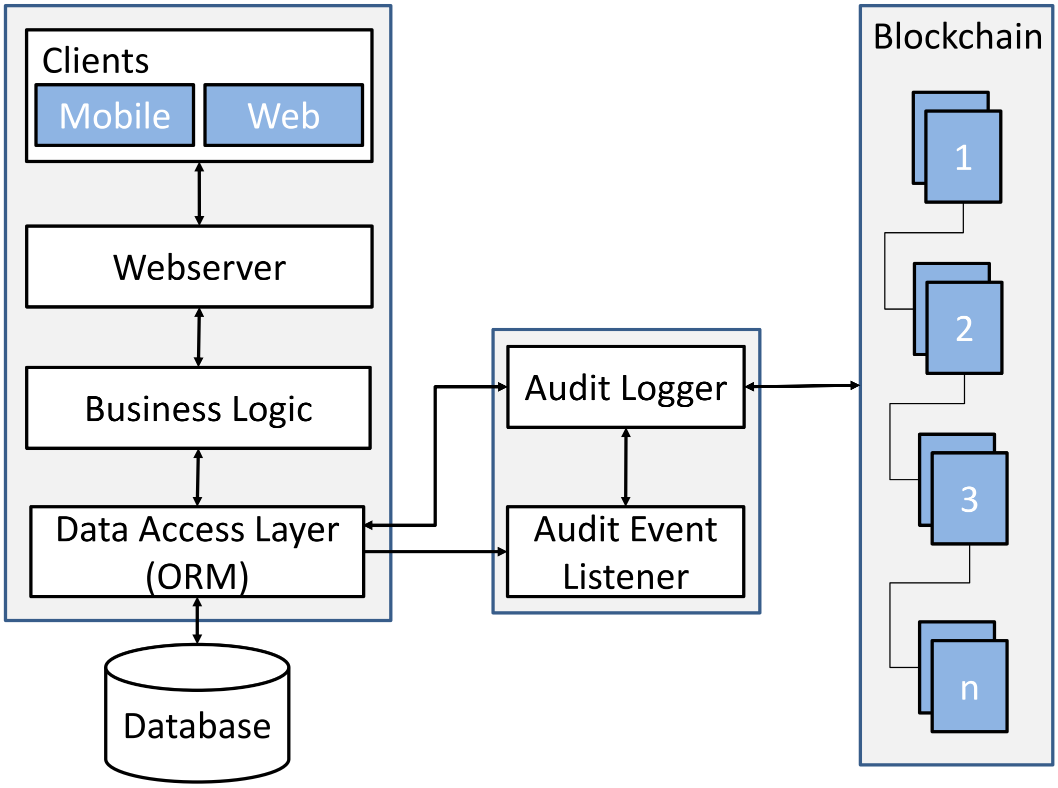


Figure 3.6: The *BlockAudit* architecture after blockchain integration in the JSON output.

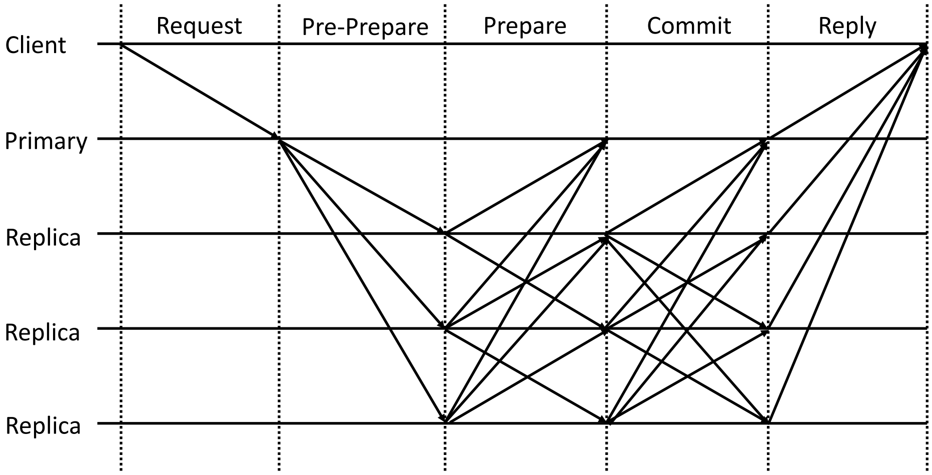


Figure 3.7: An overview of PBFT protocol with client issues a request to the primary replica. The primary then broadcasts the transaction to all the other replicas. The replicas validate the order of the transaction and share their view with each other. Once the client receives the desired number of responses, the transaction is considered validated. The process of transaction verification follows four stages, namely Pre-Prepare, Prepare, Commit, and Reply.

Log for table/class. The audit event logger can also create a packet for each object in a transaction. We used this method in the prior work [4] and found that the packet size was small, however, the number of web service calls for each application transaction was high. For instance, if a transaction contains 10 classes, it will create 10 web service calls. While 10 calls can be handled by ORM-based audit logs, however, they are not optimal for blockchain-based audit logs.

Log for transaction. The audit event logger creates a packet containing all insertions, updates, and deletions, which span across one or more objects, and sends the packet to *BlockAudit* as shown in Figure 3.5. Since the audit log data is consolidated, it is hard to search for updates of a specific class; a typical use case. Creating an audit log for a transaction reduces the number of web service calls and improves efficiency, and this design is more suited to blockchain based audit logs.

3.3.4 Consensus Protocol

The next phase in the *BlockAudit* design is the use of a consensus scheme among the peers to develop their agreement over the sequence of transactions and the state of the blockchain. There are various consensus algorithms used in blockchains, such as proof-of-work (PoW), proof-of-stake (PoS), proof-of-knowledge (PoK), Byzantine fault tolerance (BFT), etc. [70, 63]. In Table 3.3, we compare the popular blockchain consensus algorithms. Notice that PoW and PoS have high scalability and fault tolerance. More specifically, they can scale beyond 10,000 nodes and can tolerate up to 50% malicious replicas. On the downside, they have low throughput and high confirmation time [36, 85]. In contrast, PBFT has high throughput and low confirmation time. However, PBFT has low fault tolerance which makes it less suitable for permisionless settings.

For *BlockAudit*, we use PBFT consensus algorithm [82, 25], which was originally designed to facilitate the decision-making process in a distributed environment. *BlockAudit* uses a permissioned blockchain system [10], in which all network participants are known to one another, and there is a weaker notion of anonymity. Since our system is primarily a private and permissioned blockchain, therefore, we are not constrained by high scalability challenges. Although in the future, we aim to extend our design to a bigger network, however, at the prototype stage, we are less than 100 peers. Due to high throughput and low latency, naturally, PBFT is more suited for our design.

In PBFT, the system comprises of a client that issues a request (transaction), and a group of replicas that execute the request. The primary replica orders transactions and relays them to other replicas. The transaction is processed in four stages, namely pre-prepare, prepare, commit, and reply. When the client receives a minimum of $3f+1$ responses, f being the number of faulty replicas, the transaction processed. In **??**, we provide an illustration of PBFT, which we later use to design and calibrate *BlockAudit*. In Figure 3.6, we show the complete design of *BlockAudit*, where the blockchain is integrated with the serialized JSON output of the business application.

Table 3.3: An overview of popular consensus algorithms used in blockchains. Notice that PBFT has high throughput and low confirmation time.

Properties	**PoW**	**PoS**	**PBFT**
Blockchain Type	Permisssionless	Permissionless	Permissioned
Participation Cost	Yes	Yes	No
Trust Model	Untrusted	Untrusted	Semi-trusted
Scalability	High	High	Low
Throughput	<10	<1,000	<10,000
Byzantine Fault Tolerance	50%	50%	33%
Crash Fault Tolerance	50%	50%	33%
Confirmation Time	>100s	<100s	<10s

3.4 Analysis of *BlockAudit*

In this section, we analyze various aspects of *BlockAudit*, including its design (with key takeways), complexity (time and space), and security (with respect to the aforementioned threat model).

3.4.1 Design Analysis

In *BlockAudit*, each peer uses the ORM-based audit log application that is connected to a database. Once the ORM observes a change, it updates the database and issues a transaction, and sends it to the primary replica. The primary orders the transaction and broadcasts them to all the other replicas. Upon receiving the transaction, each replica checks if the transaction is valid and follows the correct order. The order of the transaction is ensured by the timestamp, and the ordering rule involves the chronological sequencing of each transaction. In *BlockAudit*, the primary preforms transaction sequencing based on the time at which it receives transactions from the application replica. We use this approach as a security design choice to prevent malicious replicas from arbitrarily modifying their transaction timestamps. In the following, we show how transaction sequencing is performed in *BlockAudit*:

❶ An application generates a transaction at time t_i and the primary receives the transaction at time t_j.

❷ First, the primary checks if the transaction respects the temporal ordering ($i < j, \forall\, i, j$). This assumption is valid for any real-world system, since each transaction experiences a non-zero delay during transmission.

❸ If the primary observes a violation *i.e.,* $i > j$, it assumes that the application replica is misbehaving. Therefore, the primary discards the transaction.

❹ In the transaction confirmation phase, the active replicas also compare the time at which they receive a transaction to the time of the transaction generation. This serves as an added security measure to ensure that the policy precedence is honored, even when ignored by the primary.

In *BlockAudit*, we enforce the ordering of transaction since it is critical in audit log applications. For instance, consider a case in which tx_a involves a change made to a class. The next transaction tx_b reverses the change made by tx_a, then it is critical to process tx_a before tx_b. Otherwise, the order will be violated and the audit log will reflect a different state of the database than the actual.

In summary, *BlockAudit* constitutes of a client (audit log application) that generates blockchain compatible transaction, a primary replica that receives and orders transactions, and a group of active replicas that execute PBFT to generate a blockchain-based audit log. In conventional PBFT, the client is independent of the active replicas that execute the consensus protocol. In *BlockAudit*, the client is one of the active replicas that issues the transaction. In the verification process, the issuer becomes the client and all other replicas act as validators.

Key Takeaways. From the design implementation, we had the following takeaways:

❶ PBFT-based permissioned blockchains are more suitable for audit log applications.

❷ Extending ORM provides an efficient mechanism of converting database transaction to blockchain compatible transactions.

❸ Existing application can seamlessly integrate with blockchain based audit logs by extending the ORM in their data-access layer.

❹ REST services can be easily extended to support applications that do not use ORM.

❺ JSON format is the de facto standard for REST API's and is simple, hierarchical, lightweight and fast, and therefore efficient and suitable for an audit log transaction.

3.4.2 Complexity Analysis

A key aspect of PBFT-based blockchain systems is the time and space complexity associated with the network and the blockchain size. The time complexity partakes the time taken by replicas to develop consensus on a transaction or a block. The space complexity involves the storage and the search overhead that compounds due to append-only distributed blockchain design. In the following, we analyze these aspects of complexity in *BlockAudit*.

3.4.2.1 Time Complexity

To achieve consensus over the state of blockchain with n replicas, $n^2 - n$ messages are exchanged, as shown in **??**. Therefore, for each transaction generated within the system, the overall complexity becomes $O(n^2)$. Compared to PoW-based blockchains, in which the consensus complexity is $O(n)$, PBFT has a high message complexity which can lead to system overheads and delays. However, we argue that in PoW-based blockchains systems such as Bitcoin, the total number of active nodes are over 6-8k [12]. In comparison, *BlockAudit* constitutes less than 100 peers. Therefore, it can tolerate this complexity overhead, keeping in view the other benefits associated with PBFT such as high throughput.

3.4.2.2 Space Complexity

The space complexity of the system can be ascribed to the overhead associated with the storage of blockchains at each peer. One major limitation of replacing the client-server model with a peer-to-peer blockchains system is that each peer is required to maintain a copy of the blockchain. This leads to a high storage footprint since blockchains are always growing in size. The size footprint also increases the search complexity for transaction verification. For instance, when a newly generated transaction is sent to peers for verification, they validate its authenticity by consulting its history in the blockchain. If the blockchain size is large, the verification time increases. As such, if the rate of the incoming transaction is high, then high verification time may lead to processing overhead, thereby increasing latency and reducing the throughput. In *BlockAudit*, the space complexity of a system, complementary to any other blockchain system is $O(n)$.

Key Takeaways. From the complexity analysis, we had the following takeaways:

❶ PBFT-based based blockchains have high message complexity. Therefore, if the network scales beyond a few hundred nodes, the application may become inefficient. Therefore, we observe a tradeoff between the message complexity and the network scalability.

❷ Generally, the space complexity of blockchain is high, due to the append-only model. In *BlockAudit*, the space complexity is similar to any other blockchain application.

3.4.3 Security Analysis

An essential component of our work is the defense against the attacks outlined in the threat model §2.4. In this section, we discuss how *BlockAudit* defends against the physical access attack and the remote vulnerability attack.

3.4.3.1 Physical Access Attack

In the physical access attack, if the attacker acquires the credentials of a user, he can make changes to the application data using the application graphical user interface. In this case, his activity will be logged in *BlockAudit*. Since the log is kept in the blockchain by the user, the attacker will not be able to remove the traces of his activity. Therefore, when the attacker's activity is exposed, auditors will be able to track the tampered records and take corrective measures to restore data to the correct state. Moreover, if the attacker can get write access to the database, he will be able to change data in various tables. Since the audit log generation is at the ORM level, therefore, these changes will not be reflected in the audit log. This discrepancy will enable the auditors to detect malicious activity and take preventive actions.

3.4.3.2 Remote Vulnerability Attack

In case of a remote vulnerability attack in which the attacker exploits a bug or vulnerability in the application, the audit log will show the effect of the changes or errors resulting from the attack. Additionally, the blockchain will also preserve the tamper-proof state of the audit log before the launch of the attack. As a result, the auditor will be able to compare the audit log and the current data to detect changes made during the attack. In the absence of the blockchain, if the attacker corrupts the prior state of the audit log, there is no way auditors can recover from it. However, with

BlockAudit, not only the attacks are detected, but the system state is also recovered. Furthermore, for an malicious party to launch a successful attack in the presence of *BlockAudit*, the attacker will need to corrupt the blockchain maintained by each node in the blockchain network. Based on the design constructs and security guarantees of blockchains, corrupting blockchain repositories of a majority of nodes is costly, and therefore infeasible.

After realizing that *BlockAudit* can defend against the attacks outlined in our threat model §2.4, there are however few considerations to be made while using the PBFT-based blockchain model. The prior work in this direction does not consider Byzantine behavior among the participating nodes. In *BlockAudit*, we consider that peers may behave arbitrarily and try to create confusion in the view of other honest peers. Therefore, we want *BlockAudit* to be robust against malicious replicas. While other consensus mechanisms such as PoW may withstand up to 50% of faulty replicas in the system, PBFT, in contrast, has low fault tolerance. In a situation where there are f faulty replicas, a PBFT-based blockchain system needs to have $3f+1$ honest replicas to function smoothly. Roughly speaking, PBFT-based blockchains require 70% nodes to behave honestly to avoid disagreements. However, in *BlockAudit*, we try to raise the threshold of fault tolerance by making minor adjustments to the security design.

3.4.3.3 Increasing Fault Tolerance

In a situation where there are r honest replicas in a blockchain and the attacker is able to position f faulty replicas such that $4f+1 > f+r$, then the attacker will be able to stop transaction verification and may even cause forks. To counter this, we propose an expected verification time window W_t which will be set by the primary replica before passing the transaction to the verifying replicas. The primary replica knows the total number of active replicas in the system and can calculate the total number of messages to be exchanged until the transaction gets verified. In this case, the total number of messages will be in the order of $(f+r)^2 - (f+r)$. Let $c \times t_b$ be the time taken for the transaction confirmation, where c is an arbitrary constant set by the primary replica. Based on these values, the primary replica can set an expected time window $W_t \geq c \times t_b$ in which it expects all peers to validate the transaction and submit their response. Let t_{start} be the start time at which the primary replica initiates the transaction. If by W_t the primary does not receive the expected number of responses from the replicas, it will abort the verification process and notify the auditor.

Depending on the application's sensitivity, the primary replica can either set another optimistic value of W_t', where $W_t' \geq W_t$, and repeat the process or it can simply abort the process and notify the application auditors regarding the malicious activity. We leave that decision to the audit log application and its sensitivity to malicious activities. However, in our experiments, we relax the condition of sensitivity and re-submit the transaction for another round of verification. We set a new expected verification time window W_t' and wait for the response. Our choice of relaxing the condition of sensitivity is owing to the unexpected delays in the message propagation; given that our system would run over the Internet. However, if the primary replica does not receive the approval for the second time, it aborts the process and notifies the application.

3.4.3.4 Detecting Malicious Nodes

In *BlockAudit*, we also enable detection of the malicious nodes that corrupt the process of transaction verification. For that, we store the identity of the replica in each iteration of the response. For instance, in the first iteration of W_t, we note the identity of replicas that send their digitally signed approval for the transaction. Let h be the subset of replicas that send their response in the first iteration, where $h \leq (f+r)$. The primary replica stores the identities of replicas in h and initiates the second iteration at t_{start}' and waits for a response till W_t'. Upon receiving the response in the second iteration, the primary replica updates h and removes the duplicates. By comparing h with the identity of all the replicas, the primary replica can find the malicious replicas and request their removal from the verification process.

It is possible that an adversary, aware of the two-phased approval process, may attempt to trick the system by sending a response from a subset of malicious peers in each phase of approval. The adversary can split his set of malicious replicas in f_1 and f_2, where $f_1 + f_2 = f$. In the first phase of approval, the adversary can send a response from f_1 replicas. However, the adversary ensures that $3f_1 + 1 \geq r$, so that the transaction does not get enough approvals to be accepted by the primary replica. The primary replica will append f_1 to its set of h. In the second iteration, the adversary will incorporate signatures from f_2, and the primary replica will also add them to h. As a result, the primary replica will not be able to detect the actual number of malicious replicas in the system.

To counter that, we randomize the two-phase approval process to v-phase approval process, v may take any value of the primary replica's choice. When the transaction fails the first attempt, the primary replica can either abort or continue the approval process. Continuing from the above-outlined scenario, if $v = 3$, then the attacker will either have to include one of f_1 or f_2 replicas in the third phase. And if the primary replica iterates one more time, the adversary will be bounded to include the set of replicas that he did not include in the previous iteration. As such, the primary replica will notice the incoherence in the response of a few replicas in each iteration of the approval process, and the adversary will risk exposing his malicious replicas. Although this procedure ensures high security and the ultimate exposure of adversary in the process of verification, it is, however, time-consuming and may lead to a transaction stall. Again, we leave this choice to the primary replica, which can make decisions that best suit the requirements of the application.

Key Takeaways. From the security analysis, we had the following takeaways:

❶ *BlockAudit* counters the conventional audit log attacks namely the physical access attack and the remote vulnerability attack.

❷ Additionally, *BlockAudit* also makes audit logs secure against Byzantine behavior, tolerating up to 30% malicious replicas.

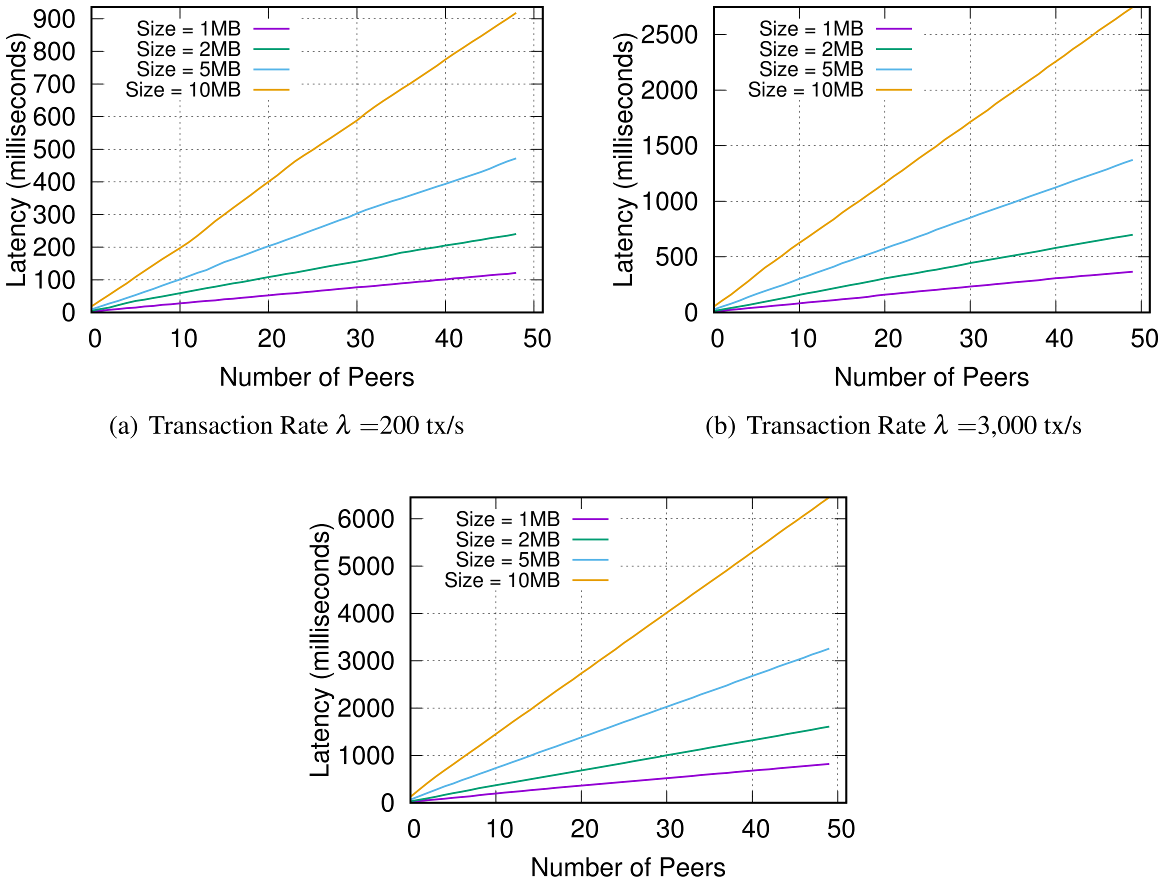
❸ Leveraging the design policies in permission settings, *BlockAudit* is able to detect malicious replicas and take preventive measures.

3.5 Experiment and Evaluation

In this section, we present experiments carried out to evaluate the performance of *BlockAudit*. First, we extended the *nHibernate* ORM to generate a serialized JSON output in the form of transactions as shown in Figure 3.3. The transactions are broadcast to the network where a *BlockAudit* blockchain instance is configured at each node. For experiments, we used sockets to set up the network and a NodeJS client to receive JSON transactions.

3.5.1 Simulation Environment

We simulated our blockchain network using a LAN setup at our research lab. We used 20 machines, each running the Linux OS with Intel Core i5 processor and a 16MB RAM. Next, we set up a virtual environment at each node to construct a multi-host network. We assigned port numbers and sockets to each host that acted as a peer. The socket connections were used to exchange data with peers using IP addresses and port numbers. Each peer was equipped with a JSON master list that contained the information of all the other nodes. Data packets of the desired size were generated and broadcast over the network. We encoded the PBFT protocol in NodeJS and executed it over all the peers. The selection of the primary replica can be done using any method suitable for the application. In *BlockAudit*, in each iteration, we selected the primary in a round-robin manner. To reflect the real-world delays in our simulation, we added a round-trip delay of 100ms in each transaction broadcast over the network. Finally, after the transaction obtained sufficient approvals, it was added to the blockchain of the primary replica, and subsequently, all the other replicas.



(a) Transaction Rate λ =200 tx/s

(b) Transaction Rate λ =3,000 tx/s

(c) Transaction Rate λ =6,000 tx/s

Figure 3.8: Time taken to reach consensus at different types of audit transaction with varying transaction rate λ (200-6,000 tx/second). Notice that as the network size and the payload size increases, the confirmation time for a transaction increases. Also, it can be seen that the as λ increases, confirmation time increases. Naturally, this can be associated with high verification delays with the bulk of the incoming transactions.

We evaluate the performance of our system by measuring the latency over the consensus achieved by peers in our blockchain network. We increase the transaction payload size from 2MB to 20MB and the rate of transaction λ from 200 transactions per second to 6,000 transactions per second. By adjusting these parameters, we monitor the time taken by peers to approve the transaction. Let t_g be the transaction generation time, and t_c be the time at which it gets approval from all active peers. In that case, the latency l_t is calculated as the difference between t_c and t_g ($l_t = t_c - t_c$, where $t_c > t_g$). We report the simulation results in Figure 3.8.

3.5.2 Simulation Results

Our simulation results show that irrespective of the payload size, the latency margins remain negligible as long as the number of peers is less than 30. As the size of the network grows beyond 30 nodes, the latency factor increases considerably. Furthermore, we also notice that a sharp increase in latency when the payload size changes from 5–10MB and a negligible change in latency when the payload size changes from 15–20MB.

We also noticed that as the rate of transaction λ increases from 200 transactions per second to 6,000 transactions per second, the confirmation time for transaction also increases. Intuitively, this can be attributed to the processing overhead caused by the increasing rate of λ at each replica. However, it can be observed from 3.8(c) that within a network size of 50 peers, *BlockAudit* has the capability of processing 1,000 transactions per second, with the payload size of 10 MB. This payload size is equivalent to 10 blocks in Bitcoin. For the payload size of 1MB, *BlockAudit* achieves a throughput of 6,000 transactions per second. Considering low throughput of conventional blockchains (3–7 transactions/second in Bitcoin), *BlockAudit* achieves high throughput. This also justifies our choice of using PBFT as consensus scheme for our system.

Evaluation parameters obtained from our experiments can be used to define the block size and the network size, specific to the needs of the application. We will also be able to use these parameters along with other consensus schemes to find optimum block size and the average block time for the audit log application. By varying consensus schemes, we could compare and contrast the performance of various design choices and select the best that can be used for *BlockAudit*.

3.6 Discussion

With *BlockAudit*, we were able to meet our overall objective of securing audit logs using blockchains. We show with theoretical analysis and simulations that our system is secure and efficient, and it achieves high throughput(§3.5) by using the PBFT consensus protocol. In *BlockAudit*, audit log transactions were seamlessly generated with minor changes to the existing system. Moreover, *BlockAudit* can be plugged into any enterprise business application, that consumes a REST API to send audit log data as a transaction. In summary, we successfully extended our application into the blockchain paradigm to harden its security and increases the overall trust in the application. Our system is robust against the physical access attack and the remote vulnerability attack.

3.6.1 Limitations

Despite all the promising outcomes, there are, however, two major limitations in *BlockAudit*. The first constraint is the high message complexity due to PBFT, and the second is a high storage footprint due to data redundancy in the blockchain design. Since in PBFT, the message complexity is high ($O(n^2)$), therefore, in adverse network conditions, PBFT may perform poorly, compared to other consensus protocol [2]. In spite of these limitations *BlockAudit* performs within the requirements of our application, and could support PayPal [44] which processes 170 transactions/second, however, our solution would not be feasible for Visa which has a transaction rate of 2000 transactions/second [35]. Secondly, audit logs by design have a high storage footprint, as each transaction in the system has a corresponding entry in the audit logs. In *BlockAudit*, the problem is further increased since transactions are replicated on multiple peers, resulting in high storage overhead.

Keeping in view these limitations, we propose that high message complexity can be resolved by using other newly proposed consensus algorithms such as Clique [11], that belongs to the family of Proof-of-Authority consensus protocols. Clique has a message complexity of $O(n)$, which is considerably lower than PBFT and PoW. Using Clique may allow us to support a larger number of peers, achieve high throughput, and reduce confirmation delays of transactions in *BlockAudit*. However, in Clique, peers run into the risk of multiple views at the same time. In blockchains, this inconsistency is called a blockchain fork. These forks can lead to temporary or permanent partitioning in the network. Currently, we are exploring methods of fork resolution in Clique, and therefore applying it in *BlockAudit* is part of our future work.

The space complexity can be reduced by adding data retention policy and purging data after its fixed retention time. This would optimize the overall size of the blockchain, and lead to less storage and search complexity. In addition to these two schemes, we also propose two other optimization strategies to meet the design limitations in *BlockAudit*.

Another limitation in *BlockAudit* is the weak link between the application and the audit log. In the current implementation, if the application itself is compromised, and subsequently the audit log generation fails, then *BlockAudit* will not be able to detect the fault at the application. At present, *BlockAudit* enables applications to seamlessly integrate with blockchain system and benefit from it. Therefore, *BlockAudit* remains agnostic to the application itself and the data being produced by it. As a result, we observe a trade off between the seamless integration of audit logs with the application and the enhanced security of the audit log generation interface. Currently, *BlockAudit* is designed to facilitate the integration of audit logs with eGovernment application. In future, we also aim to focus on detection application-level faults in *BlockAudit*.

The latency is a critical problem in distributed systems, which can be 1) latency due to the consensus scheme operation, and 2) latency due to network conditions. To minimize latency due to consensus, we select consensus algorithms, such as PBFT, which is known to provide low latency and high throughput compared to other popular schemes such as PoW. We note that such a choice comes at a certain cost: PoW is known to have better security, since it tolerates up to 50% Byzantine nodes while PBFT tolerates only 30% [15]. Acknowledging that, and giving latency a higher priority over security, in *BlockAudit*, we made the consensus choice to minimize the latency.

The other component of latency is due to the network, which includes transmission and propagation delays under a certain payload size. In *BlockAudit*, and as shown in Figure 3.8, with a payload of 10MB and a network of 50 replicas, the transaction confirmation experiences a delay of 6 seconds. In *BlockAudit*, this is an upper bound on the end-to-end latency, which is considerably low compared to 600 seconds of delay in Bitcoin. For our Enterprise application, this delay is tolerable. However, if *BlockAudit* is to be extended for applications with larger payloads, we suggest two improvements as the latency increases. First, the communication medium between applications can be enhanced to support high bandwidth. Second, localities could be exploited to host applications within the same autonomous system to reduce propagation delays.

3.6.2 Optimization

To increase the performance and to keep the audit log tamper-proof, we propose having two sets of blockchains, namely the *recovery blockchain*, and the *detection blockchain*. In Figure 3.9 we provide a system overview of this two blockchain system. The recovery blockchain stores the complete audit log transaction, including the details of all data changes in an application-level transaction. The *recovery blockchain* can be used to restore data to its prior state, which would be the state of data before an attack. The *recovery blockchain* would require more space, and longer consensus time due to large transaction audit data packets. The number of peers k in the *recovery blockchain* can be kept small to ensure immediate consensus and avoid delays. Since the security of PBFT relies on the faithful execution of the protocol by at least 70% replicas, therefore for a baseline, the minimum size of the *recovery blockchain* must be four nodes considering one malicious replica ($k \geq 4$).

The *detection blockchain* can be used to detect audit log tampering only. It will not have the information to recover the audit log to a correct state before the attack. The business application will generate a cryptographic hash using the audit transaction. The hash and a unique transaction identifier will be stored in the blockchain. In the case of data tampering in the audit log, the newly computed hash will not match the hash stored in an audit log. This will indicate that the audit log has tampered. Once tampering is detected, application administrators could use corrective measures to fix the security breach. Atop that, data can could be restored to the previous state by using database backups. The recovery blockchain which will have K_d peers, where $K_d > 2K$. Therefore, the adversary will have to compromise twice as many nodes to tamper the system without being

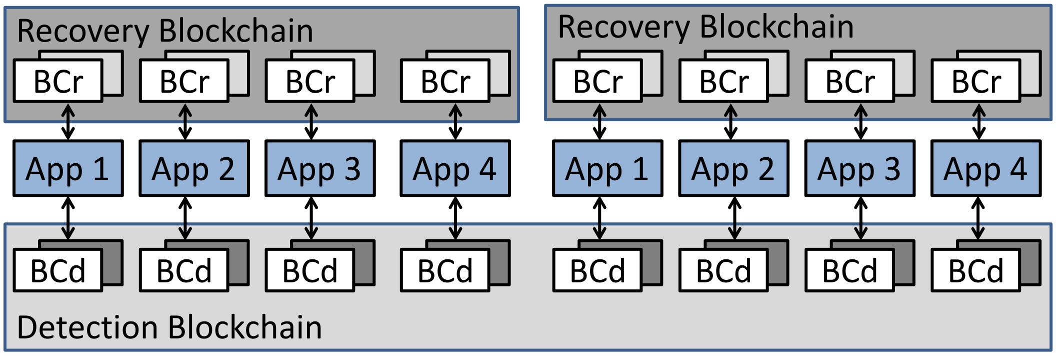


Figure 3.9: Audit log block chain detection vs recovery, there are Recovery blockchain and detection blockchain, the recovery blockchain has less nodes and stored complete log, and the detection blockchain stores a hash value of block and has more nodes.

detected. This optimization increases security and provides a second layer of defense.

Despite the existing challenges, *BlockAudit* is a feasible approach towards blockchain-based secure audit logs. Extending the capabilities of the prior work, *BlockAudit* brings the theoretical foundations into practice and as shown in section 3.5, it has been deployed and instrumented in a real blockchain network. Moreover, *BlockAudit* is also capable of ensuring operational consistency even in the presence of Byzantine replicas. Therefore, it is a better candidate for the audit log security and can be applied to existing eGovernment solutions.

3.7 Summary

In this chapter, we present a blockchain-based audit log system called *BlockAudit*, that leverages the security features of blockchain technology to create distributed, append-only, and tamper-proof audit logs. We highlight the security vulnerabilities in existing audit log applications and propose a new design that extends NHibernate ORM to create blockchain-driven audit logs. For our experiment, we used an application provided by ClearVillage inc to generate transactions from audit logs, and record them in our custom built blockchain. By design, *BlockAudit* is agile, plug and play, and secure against internal and external attacks. In the future, we will extend the capabilities of *BlockAudit* by deploying it in a production environment and explore various performance bottlenecks and optimization techniques.

CHAPTER 4: BLOCKTRAIL

Audit trails are typically used for storing, tracking, and auditing data in information systems. Entities in the audit log applications have weak trust boundaries which expose them to various security risks and attacks. To harden the security and develop secure by design applications, blockchain technology has been recently introduced in the audit logs. Blockchains take a consensus-driven clean slate approach to equip audit logs with secure and transparent data processing, without a trusted intermediary. On the downside, blockchains significantly increase the space-time complexity of the audit logs, leading to high storage costs and low transaction throughput. In this chapter, we introduce *BlockTrail*, a novel blockchain architecture that fragments the legacy blockchain systems into layers of codependent hierarchies, thereby reducing the space-time complexity and increasing the throughput. *BlockTrail* is prototyped on the PBFT protocol with a custom-built blockchain. Experiments with *BlockTrail* show that compared to the conventional schemes, *BlockTrail* is secure and efficient, with a low storage footprint.

4.1 Introduction and Problem Statement

Despite the promising features of blockchains, their use in the auditing applications, including the application in the previous chapter, has some key limitations. Blockchain-based audit logs significantly increase the storage complexity and processing overhead of an auditing application. A blockchain application mandates a copy of the ledger to be maintained by every peer who is part of the network. Additionally, blockchains follow an append-only model that results in an increasingly growing chain length. Combined, these two issues can put enormous strain on the data storage/maintenance capability of peers [51]. Furthermore, a vast majority of government agencies use software systems at federal-, state-, county-, and city-level. The audit logs generated by these systems involve transactions that are solely relevant to the domain of the software system. Considering the storage and processing overheads of blockchain systems, it is therefore counter-intuitive to share “non-overlapping” transactions of various domains within a single ledger.

Moreover, blockchain systems are known to have low transaction throughput due to serialized transaction processing and the effort involved in obtaining a consensus among a large set of peers. This can be observed in the existing PoW blockchain systems such as Bitcoin, which has the maximum processing capacity of 7 transactions/second [7]. To address the throughput limitations of POW, variants of PBFT have been proposed. However, they suffer from low scalability due to high message complexity [74] of the PBFT protocol.

To address all these challenges, in this chapter, we present *BlockTrail*; an end-to-end solution that combines audit logs with blockchains, and provides design capabilities of audit logs as well as the security guarantees of blockchains. We tailor *BlockTrail* to address the challenges associated with throughput and scalability. Towards that, we leverage the nature of audit log transactions to construct a multi-chain blockchain system that allows concurrent transaction processing at various layers of the system and achieves faster consensus by grouping the network peers. The multilayer blockchain structure is composed of federal-, state-, county-, and city-level blockchains, each being organized by a primary replica and corresponding replicas that approve the transactions. While the multilayer structure facilitates parallel processing of transactions, the grouping of replicas enables faster consensus. *BlockTrail* uses the PBFT consensus algorithm to create consensus among the peers over the blockchain state. *BlockTrail* is an application-agnostic system that provides plug and plays services for the audit log applications.

Contributions. Using the above-mentioned approach, this chapter makes the following key contributions towards blockchain-driven audit logs.

❶ We propose a novel design called *BlockTrail* for blockchain-based audit logs that facilitates parallel processing of transactions at multiple layers, thereby increasing scalability and throughput while optimizing the storage and search complexity.

❷ We outline the multilayer hierarchical structure of *BlockTrail* and show how our design choices advocate for its usefulness in audit logs.

❸ We provide theoretical primitives for our proposed design, and use a real world e-government application to implement the proposed model.

❹ We analyze performance of *BlockTrail* using three evaluation parameters, namely the transaction latency, the network size, and the payload size.

4.2 Background and Preliminaries

In this section, we provide the background of audit logs, blockchains, and their limitations. Consolidating those limitations, we present key challenges for an optimized design for blockchain-based audit logs and discuss our approach.

4.2.1 Audit Logs

In this system, we use the same audit log structure used in the previous chapter. The interested reader should refer to section 2.1 for further details on the audit log system.

4.2.2 Application-specific Scope of Audit Logs

Government agencies at the federal-, state-, county- and city-level use various software applications to manage their work. Property appraisers, for example, use the computer-aided mass appraisal systems(CAMA) to compute the value of a property and generate a tax bill. The tax bill is then used by the tax collector to collect taxes from property owners. City authorities interact with citizens using a citizen engagement module and take actions based on the complaints and feedback provided by the residents. Building permitting systems are used to issue building permits that are mandatory to construct new buildings or carry out work on the existing structures. State-level agencies have their own computer systems, such as a contractor licensing systems, that are used to issue a vocation or skill license to individuals. For instance, a technician interested in installing or repairing an Air-conditioning system needs a license from the state to be able to practice.

For this work, we assume that the e-government applications generally have searchable audit logs, that are stored in the application database. Authorized users can search and view audit logs associated with an object from the application user interface. Some of these applications communicate with other applications in the same city, county, or state. In rare cases, these interactions may also occur across states, at a federal level. Therefore, this can be viewed as an audit log management at four level: a country-level federal audit log$(L1)$, a state-level audit log$(L2)$ (for each state), a county-level audit log $(L3)$ (for each county), and a city audit log $(L4)$. Within each city, there are applications operated by state agencies that collect information to generate audit logs.

4.2.3 Challenges

In the light of existing challenges faced by audit log applications and the promising features of blockchains, it seems intuitive that blockchains can be used to address many challenges of the audit logs. To that end, there are some existing solutions for distributed audit log applications that use blockchains to securely manage data [83, 3]. In theory, blockchain-based audit logs provide an effective design to address vulnerabilities and limitations of audit logs, however, in practice, these solutions may become infeasible and costly. Typical audit log applications may incur up to thousands of transactions in a short time span. Processing such a large volume of transactions, in a short time span, can be a major challenge. As outlined in §4.1, some of these transactions may not be related to the peers who are verifying and storing them in their blockchains. This creates a massive overhead in terms of storage space and transaction processing time, making it infeasible for a number of enterprise applications. Considering these challenges, the problem is to create a blockchain design that is capable of achieving high transaction throughput while significantly optimizing the storage and space complexity. More specifically, the research challenge in this case is to propose a blockchain system for which we can 1) perform logical sharding of the blockchain ledger to support concurrent transaction processing and favor parallelization, 2) select a suitable consensus algorithm to meet the high throughput requirements of the audit application, and 3) show by contrast, an improvement in the performance of the resulting system compared to the conventional design in prior work [3].

4.2.4 Design Approach

We envision that the aforementioned problems can be solved using a multi-layered blockchain system that provisions fast processing with minimal space overhead. A multi-layered solution, by default, provides a notion of sharding for the blockchain ledger. Additionally, it supports the nature of our enterprise blockchain where transactions are exchanged among entities that reside in various application hierarchies (cities, counties, and states). Therefore, a multi-layered blockchain solution will create a natural segregation among those entities, and further support concurrency, parallelization, and storage and search optimization.

Using this approach of a multi-layered design, we aim to construct an end-to-end blockchain-based audit log system that is secure, efficient and guarantees high throughput with low storage overhead. We intend to create a model that replicates the state of audit logs across multiple entities that are distributed across multiple levels in the network. For this purpose, we make use of an e-government system, provided by Clear Village inc, to generates audit logs. The access to the real-world application provided clear visibility to its operational features, which allowed for accurate modeling of *BlockTrail*'s design. The key objective was to identify the hierarchical nature of transactions in the system and leverage that to construct a multi-layered blockchain solution. As such, any application that showcases a hierarchical design can be easily modeled with *BlockTrail*. Another example of such an application can be ATMs operating across multiple regions (*e.g.* Midwest US, Nortwest US etc.). Moreover, *BlockTrail*'s design is not restricted to the geographical construction of hierarchies only. Instead, hierarchies can be constructed using any property of the application in which multiple entities can function independently. One such example can be the supply chain application, where suppliers can be modeled across various hierarchies based on their unique roles in the application. However, as stated earlier, for the current *BlockTrail* prototype, we used the audit application provided to us by Clear Village inc, which is used by city, counties and states to provide various services, such as property appraisal, driving license,building permits ,business licenses and transfer of exemptions across the state lines. We will construct a layered blockchain model tailored to the needs of audit log application and the nature of its transactions. Based on the that, we will dynamically determine the layer to which a new transaction might belong and get approval from peers related to the transaction. We will use the PBFT protocol for fast and efficient consensus. Additionally, we also aim to make our system secure against external and internal attacks. For that, we assume that if an attacker manages to infiltrate the database by exploiting an inherent vulnerability, then the distribution of audit logs across multiple peers must enable the detection of such an attack, and switch the system back to a stable state.

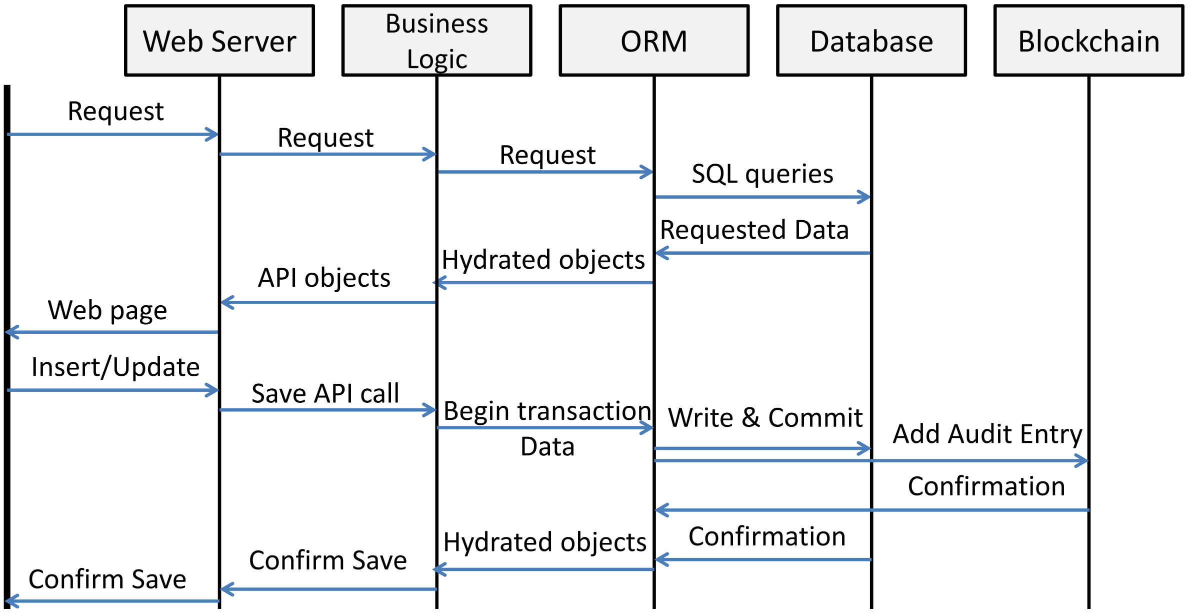


Figure 4.1: The information flow between various components of the application. The transaction is generated at the business logic layer, and once the database commits to the transaction it is rendered on the web page.

4.3 BlockTrail Design

In the following, we present our multilayer, hierarchical blockchain solution called *BlockTrail*. We begin by providing an overview of the audit log application that we use for the design, development and instrumentation of *BlockTrail*.

4.3.1 Application Architecture

In this chapter, we use an extended system from that of *BlockAudit*. In particular, we used an e-government application, built using Service Oriented Architecture (SOA), to implement *BlockTrail*. This application has web and mobile clients, that interact with a business logic layer using REST services. Business logic layer interacts with the database using a data access layer, which primarily uses Object-relational mapping (ORM) to communicate with the database. Figure 3.6 shows the layered application architecture of our e-government application along with its blockchain component. In Figure 4.1 we provide an overview of each layer and the sequence of data exchange that generates an audit log which is then passed to the blockchain system [6]. Some of the key components of our e-government application system are discussed below. The rest of the application architecture follows the same structure as in *BlockAudit*.

4.3.2 Transaction Examples

In this section, we show examples of various transaction types associated with events related to cities, counties, and state, which are used as the key design characteristic for optimization. The examples that we provide are taken from real-life events that can be observed in our auditing application. Since these events can be segregated at different layers (cities, counties, or states), therefore, they can be concurrently processed at different blockchain levels. In other words, we can partition the blockchain system into various layers where each layer corresponds to a transaction type in the audit application. As a result, transactions that are mutually independent at different layers, can be processed in parallel. Prior to that, we show how real-life examples of such events exist in our auditing application.

4.3.2.1 City-level Events

Cities use various software systems (such as finance, solid waste management, utility billing planning etc.) to carry out government functions and provide services to citizens. For instance, a citizen may file a complaint to the complaint management system about an unauthorized addition to a building. The city employees will review the complaint and go through a workflow to take actions on the reported issue, and report the results back to the citizen. The staff could find the complaint valid and decide that the code enforcement department should take action on the request. This creates a city-wide transaction among the code enforcement system and the complaint management system. This transaction is between the applications within a city. Our e-government application system has components where transactions of this nature take place frequently.

4.3.2.2 County-level Events

The property appraiser in a county is responsible for discovering the marketing value of the property and appraising it. The building department in the county is responsible for issuing building permits and managing the changes and construction of a new building. When a building contractor applies for a building permit, the building department fetches the property information (owner, property area, existing building, zoning, etc.) from the appraiser. Once the permit process is completed, the building permit information is sent to the property appraiser. A building permit could result in a change in the overall appraised valued of the property. The property appraiser application and the building permit create a county-level transaction among the two systems.

4.3.2.3 State-level Events

A homestead exemption provides legal protection to a primary residence that is owned and occupied by a person or a family. If the occupants change their primary residence from one county to the other, to a property they own, the homestead exemption would be ported to the new county of residence. This results in transactions between the two counties, and the audit log for this transaction is stored in the state-level blockchain.

4.3.2.4 States within a country

If an individual changes their primary residence from one state to another, they could port the homestead exemption from their old state to the new state. This process creates a transaction between two states and can be considered a state-level transaction.

4.3.3 System Architecture

Leveraging the multi-layered nature of transactions, we create a multi-layered blockchain solution for our application. In this section, we provide the details of our proposed model. As defined earlier, the audit logs generated by the application are broadly associated with the exchange of property information among multiple entities at different hierarchies, as a working example (note: same hierarchical structure can be exploited in other kinds of applications and transactions). This exchange of information occurs among:

❶ peers within the same city,

❷ cities within counties,

❸ counties within states, or

❹ states within a country.

In conventional schemes of generating blockchain-based audit logs, a global blockchain is used to incorporate all the transactions [3]. The global blockchain does not distinguish between transactions that are generated at different levels. As a result, all transactions are treated at the same level and processed serially. Therefore, the conventional system in [3] is not efficient and scalable. Each transaction has to traverse the entire network and get approval from all the peers. In particular,

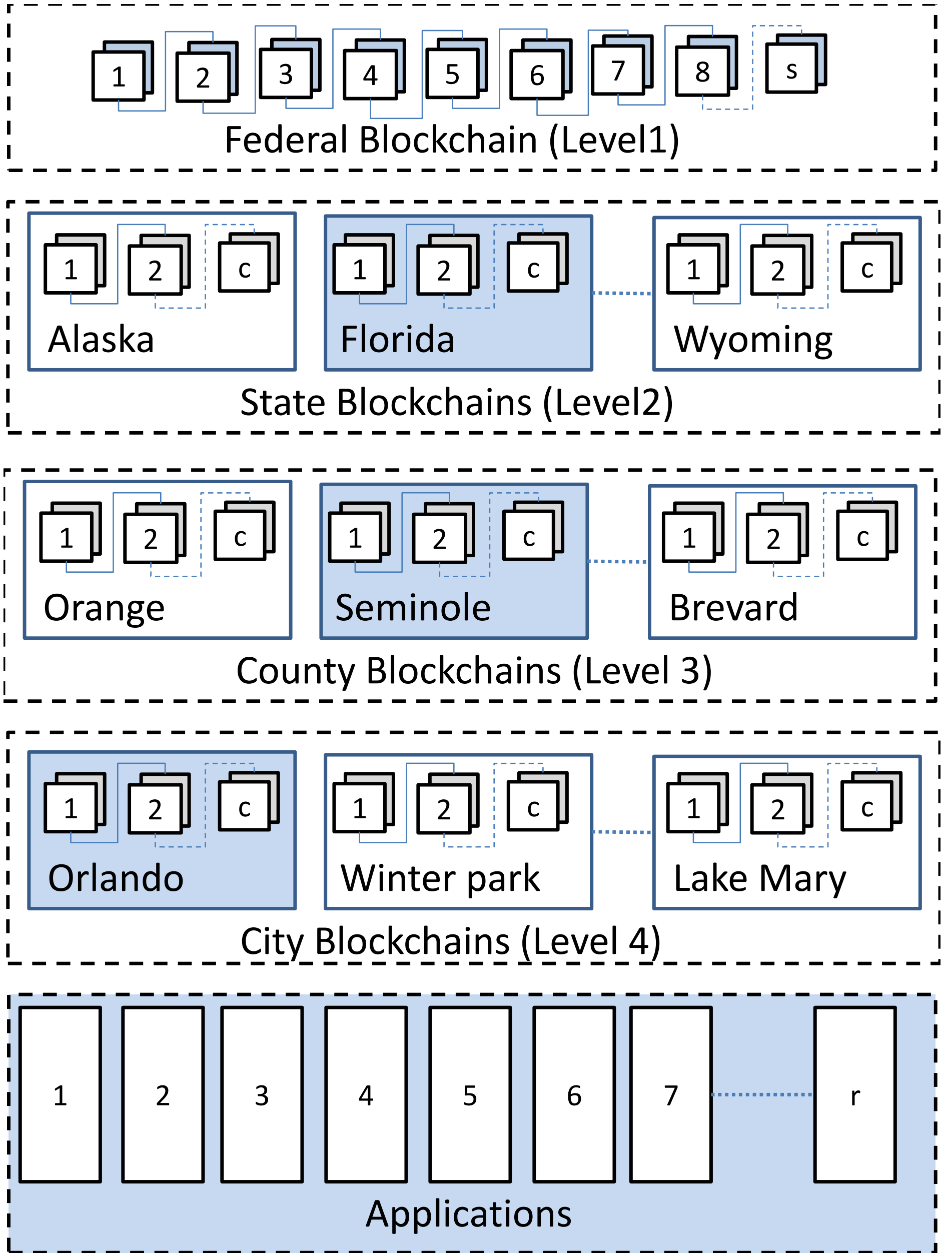


Figure 4.2: Design of our multi-chain blockchain system that is tailored to the specifications of *BlockTrail*. Levels denote the hierarchies that keep blockchains. At the lowest level, there are applications connected to a city that emanate transactions from audit logs.

a local transaction related to an ownership change at the city level will require approval from all other parties in other cities that might not be relevant to that transaction. In addition to causing delay overhead by obtaining approval from irrelevant peers, this also limits the throughput of the system since the PBFT protocol serializes the transaction processing.

We argue that efficiency and throughput challenges faced by conventional systems [3, 10] can be resolved by partitioning the network into multiple hierarchies. As such, transactions that are specific to a group of organizations within a city must be processed locally, while the transactions related to cities within a county can be processed at the county level and stored in county's blockchain. Taking this bottom-up approach from peers within the cities to the states within a country, we obtain a hierarchical tree of blockchain system that incorporates multiple blockchains, each holding data of its corresponding set of peers. The transactions will be generated by the organizations within the cities that act as a root in the system. Each transaction will have an identifier that will determine its destination blockchain. Using this structure, our system will be able to achieve the following features:

➊ Transactions within the same branch can be processed in parallel, thereby enabling parallel processing and increasing throughput.

➋ For a transaction within the same branch, the approval will be required from the leaf nodes within that branch that is relevant to the transaction. This will reduce the processing overhead incurred by transactions in the conventional scheme.

➌ Other than transaction generation and processing, this scheme is highly efficient in blockchain queries since it reduces the search complexity, which we show later.

In Figure 4.2, we show the topology of our hierarchical blockchain paradigm, and in the following, we provide the notations that capture the abstraction of our model. Let $\mathscr{L}_f = \{L_1, L_2, \dots, L_s\}$

denote the country-level (federal) hierarchy that incorporates a set of all states within the country. This hierarchical model can be extended from more than four levels (if needed), to increase the scalability and reduce the time and space complexity.

Keeping in mind the baseline fault tolerance of PBFT, we assert that the minimum number of replicas in blockchain, at each level is $s \geq 4$. For each state in $\mathscr{L}_f$, let $L_i = \{l_1, l_2, \ldots, l_c\}$ be a set of counties in state. For each county in L_i, let $l_j = \{p_1, p_2, \ldots, p_d\}$ denote the number of cities that are associated with each county. Finally, for each city in l_j, let $a_q = \{n_1, n_2, \ldots, n_r\}$, be the set of peers (audit log applications), operating within the city. Given this topology, the overall size of the network S, determined by the number of audit-log applications, can be computed using.

$$S = \sum_{i=1}^{s} \sum_{j=1}^{c} \sum_{q=1}^{d} X_{qji} \tag{4.1}$$

Here, X_{qji} represents the position of a node within the system (identified by the city, county, and state indexes). For each system level, we have a primary replica that orders the transactions and executes the transaction verification process. Specific to the design outlined in this chapter, we have a primary replica for each city, county, and state in the system. Therefore, the total number of primary replicas in our blockchain system is $d + c + s$.

4.3.4 Consensus Protocol and Access Control

In *BlockTrail*, we use PBFT consensus protocol due to its low latency and high throughput [28, 90]. In PBFT, the transaction execution follows a multi-round phase, in which a primary replica orders transactions in the block. Next, in the pre-prepare phase, it broadcasts the block to all the other replicas who verify the ordering of transactions in the block. The pre-prepare phase is followed by prepare, commit, and reply phase in which replicas follow the specified protocol of PBFT and the primary obtains “approvals” from the honest replicas [82]. An overview of the PBFT protocol execution is shown in Figure 3.7. The figure shows ❶ a client sending transaction requests to the primary replica who assembles the them in a block, ❷ the primary broadcasting the block to all nodes in the network, ❸ each node approving all the transaction in the block and sending a reply to the primary and the client.

Another key component of *BlockTrail* is the trusted environment for the network where nodes can exchange transactions and blocks with limited visibility to the outside world. Therefore, we use an access control mechanism in *BlockTrail* to specify permissions and privileges for various application replicas. *BlockTrail* uses a Membership Service Provider (MSP), similar to the Hyperledger Fabric [10] which consists of certificate authorities that issue and validate certificates for nodes that join the blockchain network. Rules for the access control are specified within the certificates issued to each node. In addition to that, in *BlockTrail*, we use REST API that allows each application node to connect to the application server to retrieve audit data if needed. Finally, to communicate payloads (transactions and blocks) within the network, we use WebSocket connections. We use the secure WebSocket connection which uses Hypertext Transfer Protocol Secure (HTTPS) to communicate the payload.

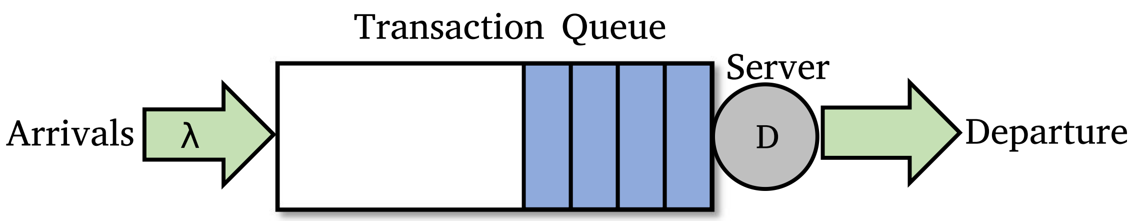


Figure 4.3: *M/D/1* queue where transactions are arriving with rate λ, and there is one server that process the transactions at the average rate *D*

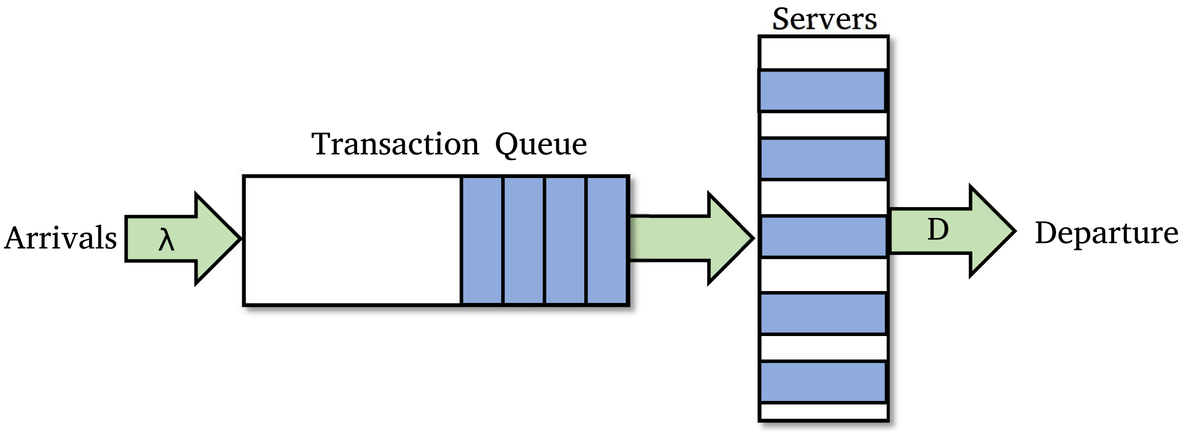


Figure 4.4: *M/D/c* queue with transactions arriving at mean rate λ, and a group of servers are processing transactions with rate *D*.

4.4 Analysis of *Blocktrail*

In this section, we perform a theoretical analysis of our proposed model *BlockTrail*. We begin by analyzing the transaction processing and throughput. Next, we show how the *BlockTrail* optimizes the time and space complexity, followed by its security analysis.

4.4.1 Transaction Processing and Throughput

To understand the efficiency *BlockTrail* with respect to the transaction processing, we use a Markovian model that broadly formulates the PBFT-based blockchain systems. To that end, we envision that the system can be viewed as a Poisson process characterized as an *M/D/1* queue at the primary replica [24]. Here, *M* denotes the arrivals determined by a Poisson process, *D* denotes the deterministic mean service time, and 1 shows that there is one server in the system. In *M/D/1* queue, as shown in Figure 4.3, λ denotes the mean arrival rate of the transactions at the primary replica and *D* denotes the mean service rate of the active replicas that collectively act as a server. From this, we can derive $\rho = \lambda / D$, which denotes the utilization of the server. If the arrival rate is less than the service rate $\lambda \leq D$, there is no queuing at the primary replica, and each transaction is processed before the arrival of the next.

However, in practice, the rate of incoming transactions is usually greater than the rate of transaction confirmation [75]. This leads to the formation of a queue at the primary replica. In PBFT, if there are a number of active replicas in the system, then the maximum number messages exchanged, in a phase, are $a(a-1)$. With a nodes in the system, there are $a-1$ messages in the pre-prepare phase, $a(a-1)$ in the prepare phase and $a(a-1)$ in the commit phase. Assuming each phase to be an independent component of the transaction confirmation process, the bottleneck can be expected in the prepare and commit phase. For the analysis in this section, we use the worst case to model the system's performance.

Next, assume the time taken in each phase to be a complementary to the message exchange. For instance, let t_a be the time taken for the execution of pre-prepare phase, and t_b and t_c be the time for the execution of the prepapre and commit phase. Since, prepare and commit phase experience same message complexity, therefore, for simplicity, we assume $t_b \approx t_c$. Since prepare and commit phase incur high message complexity, therefore $t_b = t_c > t_a$. Again, for simplicity, and without losing generality, we specify t to be the time taken to execute prepare or commit phase.

The time t is related to the number of messages exchanged in the prepapre and commit phase. And the phase execution depends upon the number of messages exchanged among network nodes. Some key performance indicators of *M/D/1* queue are the mean number of transactions in the system and the average wait time for each transaction. The mean number of transactions L in the system can be calculated as:

$$L = \rho + \frac{1}{2}\frac{\rho^2}{1-\rho} = \lambda t + \frac{1}{2}\frac{(\lambda t)^2}{1-\lambda t}$$

In (4.2), ρ is the server's utilization, λ is the arrival rate, and t is the time taken to exchange one message. Moreover, the average wait time for a transaction in the system w is:

$$w = t + \frac{1}{2}\frac{\lambda(t)^2}{1-\lambda t}$$

Applied to our multi-chain model, the number of servers are partitioned into multiple groups at each layer. Transactions that belong to a specific group are only sent to the servers in that group. Therefore, this can be used to exploit parallelism which in turn reduces the service time for each transaction. As a result, the system reflects an *M/D/c* queue [9]. Here, c are the total number of primary replicas associated that are working in parallel. For instance, assume a transaction tx_1 that is initiated between two cities C_a and C_b at time $t1$. The total replicas involved in the verification process are $a+b$. On the other hand, another transaction tx_2 is initiated at the same time t_1 among two different cities C_e and C_f, having total active replicas $e+f$. Now, these two transactions can be processed in parallel if the following condition is met:

Condition 1 *Two transactions can be considered to be non-overlapping if their associated active replicas are unique and have no intersection.* $(C_a \cup C_b) \cap (C_e \cup C_f) = \emptyset$.

Depending on the size of replicas, each transaction will be processed accordingly. Under the assumption that at a given moment, there is a set of size c replicas that satisfy the aforementioned criteria, the system will behave as a *M/D/c* queue for transactions destined for each server.

As the size of server may vary, depending on the number of verifiers involved with the transaction, the verification time and the throughput of the system may also vary accordingly. However, our design relies on the assumption that at any given time, there are more than one transaction in the system that are independent from each other, and therefore may partition the system into two or more sets of active replicas. In the worst case, if all the transactions are related to the global blockchain at Layer 1, involving all parties, then the system will behave as the conventional blockchain model with the primary replica at the top layer behaving as an *M/D/1* queue.

4.4.2 Time and Space Complexity Analysis

A key challenge with blockchain-based audit logs is the time and space complexity associated with the growing size of the chain. The time complexity can be ascribed to the time taken by peers to achieve consensus over the blockchain state, and the number of queries required to retrieve a particular transaction. The space complexity, on the other hand, is related to the storage cost that comes with the growing size of blockchain. Intuitively, as the chain size grows, its storage footprint increases linearly, which causes strain on the storage capacity of peers. However, the multilayer architecture of *BlockTrail* is helpful in reducing the time and space complexity to achieve faster consensus and optimize the storage overhead.

4.4.2.1 Consensus Complexity

To achieve consensus with n replicas, at maximum, $n^2 - n$ messages are exchanged. Therefore, the cost of consensus in the conventional blockchain model becomes $\mathcal{O}(n^2)$. However, in *BlockTrail*, the system is partitioned into sublayers, each comprising of different number of replicas (§4.3). This partitioning of the system, as shown in Figure 4.2, reflects a tree structure with branches depicting multiple layers. As such, if there are k layers in the system, then each layer reduces the complexity of system by a factor of k. Since the federal blockchain is similar to the conventional global blockchain, therefore, it has the same message complexity of $\mathcal{O}(n^2)$. However, the state level blockchain reduces the complexity to $\mathcal{O}((n/k)^2)$, where $k < n$. Accordingly, the complexity of county and city blockchains is $\mathcal{O}((n/k^2)^2)$ and $\mathcal{O}((n/k^3)^2)$ respectively. Another feature associated with blockchains is the cost of querying data which depends upon the number of peers in the system. The cost of query in the conventional blockchain (presented in [3]) and country level blockchain is $\mathcal{O}(n)$. Likewise, the complexity in *BlockTrail*, at each subsequent level, the complexity becomes $\mathcal{O}((n/k))$, $\mathcal{O}((n/k^2))$, and $\mathcal{O}((n/k^3))$, respectively.

4.4.2.2 Space Complexity

BlockTrail reduces the space complexity of system by optimizing the transaction overhead at each layer. For instance, in conventional scheme [3], all the transactions are added to the blockchain that is stored at each replica. If there are n replicas in the system, and t transactions in the blockchain, where $(t > n)$, then the space complexity of system will be $\mathcal{O}(t)$. However, a major downside of this method is that every peer is required to maintain a log of transactions that may not be related to its application. Acknowledging this challenge, and benefiting from the hierarchical structure of *BlockTrail*, the space overhead can be considerably reduced. In *BlockTrail*, we allow replicas in a hierarchy to only store the transactions that are local within their branch. For instance, all the transactions exchanged between applications within the city should only be recorded in corresponding city blockchain. All other cities and their replicas should neither participate in the verification nor store the transaction in their blockchain.

As a result of this scheme, the space complexity reduces at each layer of blockchain. Similar to the conventional blockchain, the space complexity at federal blockchain in *BlockTrail* remains $\mathcal{O}(t)$. However, at the state level, due to fewer and non-overlapping transactions, the complexity reduces

Table 4.1: Complexity analysis of *BlockTrail*. Federal blockchain takes the most time to search or add data to the blockchain. *BlockTrail*, for common case takes less time for data query or consensus. In worst case, when all audits are global, the search cost will be equal to federal blockchain.

Blockchain	**Query**	**Consensus**	**Space Complexity**
Conventional	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(t)$
Federal	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(t)$
State	$\mathcal{O}(n/k)$	$\mathcal{O}((n/k)^2)$	$\mathcal{O}(t/k)$
County	$\mathcal{O}(n/k^2)$	$\mathcal{O}((n/k^2)^2)$	$\mathcal{O}(t/k^2)$
City	$\mathcal{O}(n/k^3)$	$\mathcal{O}((n/k^3)^2)$	$\mathcal{O}(t/k^3)$

to $\mathcal{O}(t/k)$. Accordingly, the complexity at county and city level becomes $\mathcal{O}(t/k^2)$, and $\mathcal{O}(t/k^3)$, respectively. In Table 4.1, we summarize how to *BlockTrail* reduces space-time complexity in blockchain-based audit logs. Notice that the top layer behaves similar to any conventional design [3] while each subsequent layer optimizes the overhead.

4.4.3 Security Analysis

In this section, we present the security analysis of *BlockTrail*. We begin by outlining our trust model and threat model to evaluate the security guarantees of *BlockTrail* against various attacks. We use our analysis to suggest possible advancements that can be made to enhance the security of PBFT-based blockchain systems.

4.4.3.1 Trust Model

In *BlockTrail*, we assume that at any level of the blockchain, there are four or more replicas to process a transaction. This criterion is critical for developing consensus in PBFT blockchains, that requires approval from at least $3f+1$ active replicas in the presence of f faulty replicas. PBFT-based systems generally have low fault tolerance compared to PoW and PoS-based blockchain systems [74]. However, since our design makes use of the permissioned blockchain systems, we can assume a more trustworthy environment where peers have mutual interests and limited incentives to misbehave. Therefore, in contrast to PoW-based applications such as Bitcoin, permissioned blockchain systems such as Hyperledger, are likely to encounter fewer attacks.

We further assume that *BlockTrail* is equipped with all essential blockchain guarantees such as immutability, distributed trust, and availability. The underlying cryptographic primitives of *BlockTrail* including hash functions and digital signatures are assumed to be secure. The messages exchanged among peers within a layer are secured by transport layer security mechanisms and the network is secure against eavesdropping and man-in-the-middle attacks.

4.4.3.2 Threat Model

For the threat model, we assume a computationally-bounded adversary that controls a set of malicious replicas in the system. We assume that the adversary attains the trust of other peers and positions himself among the active replicas. If the network has n replicas and the adversary controls f replicas, then in conventional blockchain design, the value of f has to be large enough ($n-f \leq 2f+1$) to enable the adversary to attain control over the system. If the value of f is sufficiently large, then the adversary can compromise the system by asking the faulty nodes to withhold their signatures on a given transaction in order to halt the verification.

However, in *BlockTrail*, the number of replicas can be small as we traverse down the layered structure. For instance, at the city level, the number of replicas are much smaller than the federal blockchain. This provides a more targeted and less costly attack opportunity for an adversary with a smaller number of active replicas. Therefore, while the layered design of *BlockTrail* provides high efficiency, it has a weaker notion of security under PBFT. Additionally, in a layered design, a major challenge for the adversary is to position his replicas in a way to obtain maximum benefits with minimum effort. Positioning of replicas mean that the adversary needs to decide at which layer he should deploy his malicious nodes f in order to attack the system. In the following, we discuss and analyze the possible options for the adversary.

4.4.3.3 Positioning Malicious Replicas

The attacker with f malicious replicas can either randomly position them in the network at different layers of blockchain or select a targeted layer with fewer replicas to launch a targeted attack. In this direction, we will evaluate both these design choices and analyze the system under attack. First, we observe the possibility when the attacker randomly allocates f malicious peers in a layered blockchain system with b number of blockchains.

The random allocation of f replicas in b blockchains can be modeled as the classical *balls-into-bins* probability problem [21]. Provided that there are b blockchains and f malicious replicas, the probability that a replica gets allocated to any random blockchain is $\frac{1}{b}$. Using this premise, we are interested in answering the following questions:

❶ Probability that two malicious replicas are allocated to a specific layer of blockchain,

❷ Probability that a specific blockchain has exactly p malicious replicas, where $p \leq f$,

❸ Probability a specific blockchain has no malicious replica.

To answer the first question, let Alloc_i^k denote the event that i-th replica gets allocated to blockchain k, and let $\text{M}_{i,j}$ be the event that replica i and j get allocated to the same blockchain. By using Bayes' rule, the probability of such an event is:

$$\begin{aligned}\mathbf{Pr}[\text{M}_{i,j}] &= \sum_{k=1}^{b} \mathbf{Pr}[\text{Alloc}_2^k | \text{Alloc}_1^k] \mathbf{Pr}[\text{Alloc}_1^k] \\ &= \sum_{k=1}^{b} \frac{1}{b} \mathbf{Pr}[\text{Alloc}_1^k] = \frac{1}{b}\end{aligned} \tag{4.2}$$

While doing the random allocation, there is a possibility that a blockchain with more sensitive information may get exactly the number of peers that may compromise it. Assume that a specific blockchain b_s with p number of honest replicas cannot accommodate more than q malicious replicas. This leads to a problem raised in the second question which estimates that the target blockchain gets exactly p malicious replicas ($p \leq f$), as show in the following model.

$$\begin{aligned}\mathbf{Pr}[b_s \text{ has } p \text{ replicas}] &= \binom{b}{p} \left(\frac{1}{b}\right)^p \left(1 - \frac{1}{b}\right)^{b-p} \\ &\leq \frac{b^p}{p!} \frac{1}{b^p} = \frac{1}{p!}\end{aligned} \tag{4.3}$$

There is possibility that the attacker may not be able to position any malicious replica at any layer of the blockchain. This eventually adds to our trust assumptions of the system and may require less effort to defend against attacks. In the following, we show the probability that a specific blockchain in our system exhibits this property and contains no malicious replica belonging to the adversary, thereby addressing the third question raised above.

$$\mathbf{Pr}[\text{ blockchain gets no replicas}] = 1 - \left(\frac{1}{n}\right) \tag{4.4}$$

As the hierarchy of blockchain increases from city to county and eventually the federal blockchain, the security of the system also increases due to more active replicas being involved in the transaction confirmation. To enhance the security features of *BlockTrail* at lower levels of blockchain, we propose the following countermeasures.

4.4.3.4 Countering Targeted Attacks

In a situation where there are r number of honest replicas in a city blockchain and the attacker is able to position f faulty replicas such that $3f+1 > f+r$, then the attacker will be able to: 1) delay or stop transaction verification by withholding the required number of approvals, 2) mislead consensus by propagating bogus approvals, and 3) launch denial-of-service attacks against honest replicas in the system. In the following, we outline how *BlockTrail* defends against each of these attacks while maintaining system guarantees and operational consistency.

4.4.3.5 Approval Withholding

To counter approval-withholding by malicious replicas, we propose using the expected verification window W_t, set by the primary replica. The primary, with the knowledge of the total number of replicas, can compute the number of approvals required for verification of a given transaction. For this example, the total approvals will be in the order of $(f+r)^2-(f+r)$. If a message exchanged among $f+r$ replicas, takes t' time, then the total time of verification will be $c \times t'$, where c is an arbitrary constant set by the primary. Using this knowledge, the primary can set an expected time window $W_t \geq c \times t'$ in which it will expect to receive all the approvals. Once the window expires, it will match the received approvals with the access control list to find the faulty replicas.

4.4.3.6 Denial-of-Service

An adversary may also halt the transaction verification process by launching a denial-of-service attack on the honest replicas. In the approval phase, faulty replicas can send a sequence of transactions to honest replicas and choke their network link. As a result, the expected time window W_t will expire and the primary will not be able to process the transaction. To counter this, in *BlockTrail*, we restrict the number of received approvals per replica. Since each approval contains the senders' identity, therefore, when the faulty replicas send a high volume of messages, the honest replicas discard them without verification.

4.5 Implementation and Evaluation

The deployment of *BlockTrail* requires:

❶ A blockchain network consisting of participants at each level as we specified,

❷ An auditing application that generates transactions when the audit log is updated,

❸ a consensus protocol to ensure an agreement among peers over the blockchain state,

❹ a layered architecture to emulate our blockchain structure, and

❺ a system storage for blockchain ledger at each node.

In this direction, we discuss the end-to-end deployment of *BlockTrail*, from transaction generation to the broadcast of information, and the final commitment to the blockchain. In particular, we highlight our design choices for the aforementioned deployment goals, and justify their usability in the operational context of *BlockTrail*. So far in our design of *BlockTrail*, we show the presence

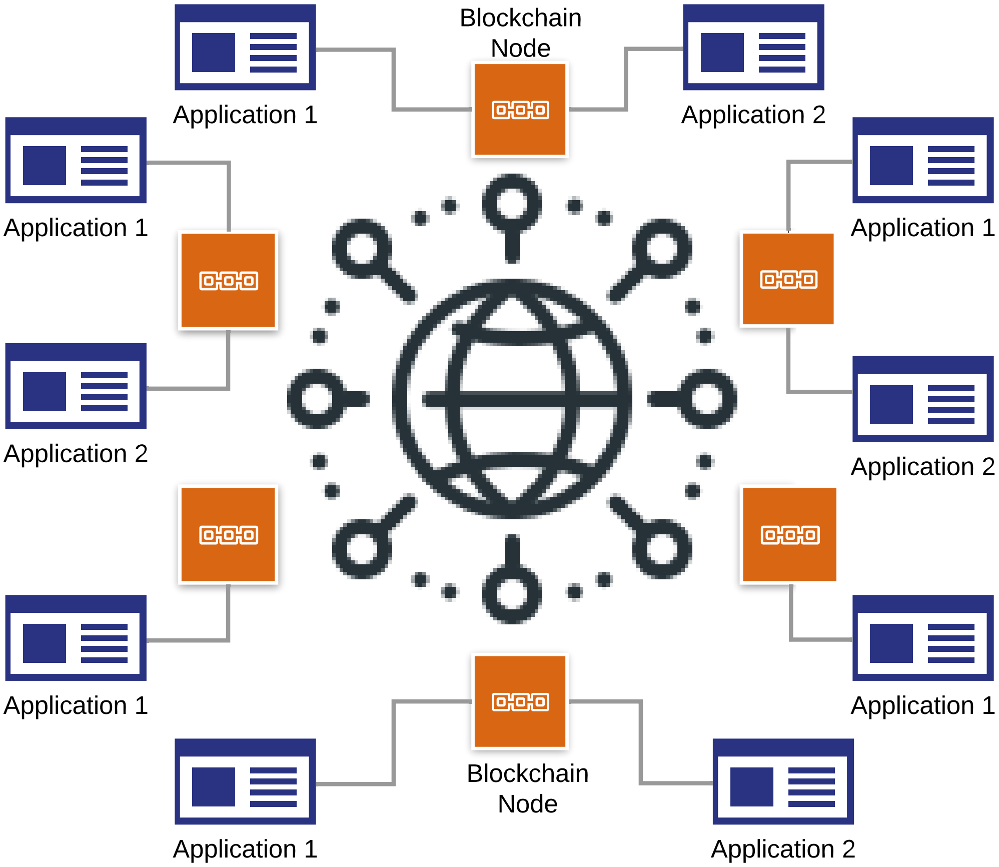
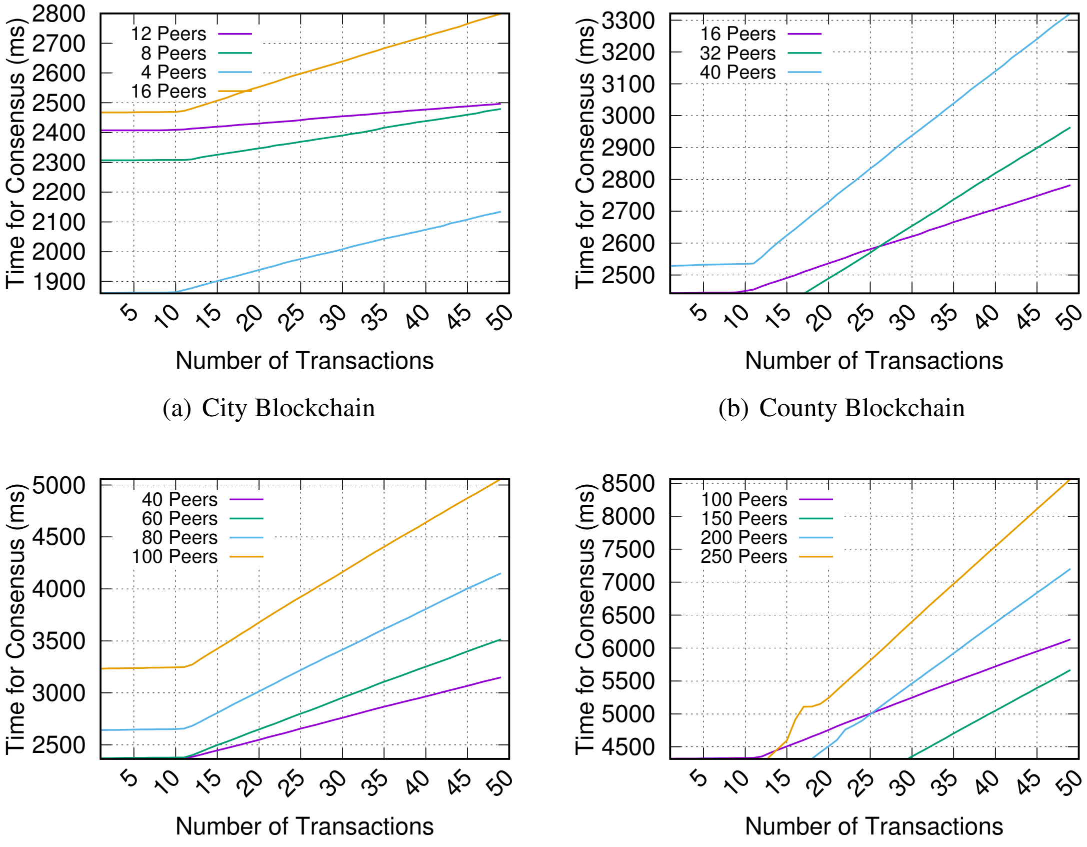


Figure 4.5: Overview of *BlockTrail* network containing application servers and blockchain nodes.Notice that each blockchain node connects to an application server node, and maintains a persistent connection to exchange audit transactions during the application life-cycle

of an audit log application that serves as the backbone to our system. The audit log application is sensitive to the changes made in the database by a user and, by design, records those changes in the form of an audit log. The audit log is then broadcast to the network comprising of peers/nodes. In the following, we outline the steps taken to deploy our blockchain system.

Creating Blockchain Network. The first step in our system deployment is setting up a distributed



(a) City Blockchain

(b) County Blockchain

(c) State Blockchain

(d) Federal Blockchain

Figure 4.6: Time taken to reach consensus at different layers of blockchains with varying transaction rate (λ). As the hierarchy of the blockchain increases, the time of consensus increases accordingly. Additionally, as the rate of transaction λ increases, the consensus time increases.

network of multiple nodes that process the blockchain transactions and create new blocks. In *BlockTrail*, the network consists of a blockchain peer for each application instance, such that each instance has the permission to create and access the audit log records. Each application node maintains a persistent connection with its blockchain node to 1) create new audit log entries, 2) search and retrieve audit log entries, and 3) after validation, forward transactions to the other peers in the network. In Figure 4.5, we illustrate the distributed architecture of nodes that reflect the peer-to-peer model of a blockchain application.

Creating Blockchain Transactions. The next step is to transform audit log data generated by the ORM in a format that can be consumed by blockchains. To achieve this, we convert the audit log data to a JSON format. We prefered the JSON format due to its compactness and storage flexibility. To generate the blockchain transaction, we pass the audit log data to a function that serializes data to a JSON payload. The payload is then used to create a blockchain transaction and the blockchain node broadcasts this transaction for all other nodes for confirmation.

Table 4.2: Results obtained by running *BlockTrail* simulations with five iterations over each experiment. Here, peers denote the number of peers in the layered blockchain, T shows the number of transactions, Exp$_1$, Exp$_2$, Exp$_3$, and Exp$_4$ are the number of experiments performed over the values of T, and μ is the mean of all the experiments.

Level	**Peers**	**T**	**Exp$_1$(ms)**	**Exp$_2$(ms)**	**Exp$_3$(ms)**	**Exp$_4$(ms)**	**Exp$_5$(ms)**	**μ(ms)**
Flat/Legacy	100	20	8927.00	2292.00	2171.00	2831.00	7564.00	4757.00
		30	9409.00	2788.00	2681.00	3312.00	8042.00	5246.40
		40	9883.00	3266.00	3154.00	3782.00	8512.00	5719.40
	250	20	8495.00	4017.00	5652.00	3970.00	4080.00	5242.80
		30	9639.00	5174.00	6826.00	5136.00	5227.00	6400.40
		40	10800.00	6314.00	7967.00	6287.00	6348.00	7543.20
City	4	20	3753.00	1604.00	2021.00	2476.00	2298.00	2428.40
		30	3777.00	1628.00	2045.00	2501.00	2322.00	2452.40
		40	3801.00	1650.00	2067.00	2524.00	2344.00	2475.00
	8	20	4951.00	1839.00	2083.00	1367.00	1495.00	2343.00
		30	4994.00	1882.00	2126.00	1410.00	1538.00	2386.00
		40	5043.00	1930.00	2174.00	1461.00	1585.00	2434.60
County	16	20	2501.00	5101.00	1923.00	1338.00	1817.00	2536.00
		30	2585.00	5184.00	2009.00	1422.00	1904.00	2620.80
		40	2672.00	5270.00	2096.00	1507.00	1987.00	2706.40
	24	20	2269.00	7295.00	2019.00	2367.00	2554.00	3300.80
		30	2393.00	7422.00	2152.00	2498.00	2679.00	3428.80
		40	2519.00	7545.00	2274.00	2632.00	2806.00	3555.20
State	40	20	4356.00	1965.00	2189.00	1754.00	2480.00	2548.80
		30	4563.00	2182.00	2412.00	1957.00	2692.00	2761.20
		40	4768.00	2380.00	2638.00	2153.00	2892.00	2966.20
	60	20	5533.00	2040.00	1695.00	2097.00	1882.00	2649.40
		30	5833.00	2337.00	2024.00	2393.00	2181.00	2953.60
		40	6124.00	2631.00	2329.00	2696.00	2484.00	3252.80
Federal	100	20	8927.00	2292.00	2171.00	2831.00	7564.00	4757.00
		30	9409.00	2788.00	2681.00	3312.00	8042.00	5246.40
		40	9883.00	3266.00	3154.00	3782.00	8512.00	5719.40
	250	20	8495.00	4017.00	5652.00	3970.00	4080.00	5242.80
		30	9639.00	5174.00	6826.00	5136.00	5227.00	6400.40
		40	10800.00	6314.00	7967.00	6287.00	6348.00	7543.20

4.5.1 Experiment and Results

In this section, we show the experiments carried out to evaluate the operation and performance of *BlockTrail*. For the experiment, we used existing audit logs stored in the database to generate audit transactions as a JSON packet and send them to the first layer primary replica. The primary replica looks into the transaction and determines its correct destination. It then notifies all the concerned replicas that are associated with the transaction and requests them for transaction validation.

For our experiments, we generate a series of transactions for each layer of the blockchain. We vary the transaction rate by increasing λ, and note the time taken by all the peers to reach consensus. Additionally, for each layer, we vary the number of peers and the size of the transaction to see the overhead in consensus time. λ was increased from 25 to 50, and the city peers were set to 10,20,30 and 50, the county peers were set to 50, 100, 150 and 200, and the state level peers were set to 80, 160, 240 and 320. Finally, the federal level peers were set from 100, 200, 300, and 400. We prototyped *BlockTrail* using `.net` framework, and deployed replicas over virtual machines.

We evaluate the performance of *BlockTrail* in terms of the time taken for all the nodes to reach a consensus over the transaction sent by the primary replica. Let t_g be the transaction generation time, and t_c be the time at which it gets approval from all active peers and gets confirmed in the blockchain. In that case, the latency l_t is calculated as the difference between t_c and t_g ($l_t = t_c - t_c$, where $t_c > t_g$). Results are reported in in Figure 4.6 and Table 4.2. In Figure 4.6 , it can be observed that as the number of peers increase at each layer, the consensus time increases considerably. Also, as the rate of incoming transactions increases, the consensus time increases. As expected, the time for consensus at the city level was less compared to the county and the state level. Therefore, this validated the utility of the *BlockTrail* model whereby sharding the blockchain system into multiple layers can reduce consensus delays and increase the throughput.

4.5.2 Discussion and Limitations

Since the results obtained from our simulations validate the theoretical results, we see the hierarchical structure of blockchain as a practical solution to its scalability and complexity issues. Although, there are limitations of this work as well. Considering the high volume of transactions being generated by the audit log applications, an average consensus time of 2000 milliseconds 4.6(a), even at the lowest level of blockchain is considerably high. As we proceed with towards the upper layers in the hierarchy, the problem increases with a federal blockchain of 200 peers taking up to 9 seconds to verify transactions. This problem results from the number of active replicas involved in the verification and a block size of 1 Mb used during the simulation.

An easier solution to this problem is by reducing the total active replicas involved in the verification process. These replicas can be divided into active and passive replicas, where active replicas sign and verify the transactions, while the passive replicas merely approve the process. This can reduce the consensus overhead and improve system efficiency. However, as outlined in the security analysis §4.4.3, reducing the number of active replicas may increase the security threat, since the adversary can position his active replicas and compromise the validation process. As such, this situation presents a tradeoff between security and efficiency. A system that mostly comprises of trusted nodes, may afford to split replicas between active and passive, with an assumption that there are fewer malicious replicas in the system. In contrast, a system that can tolerate higher delays and is sensitive to malicious activities, may not prefer the reduction of active replicas.

4.6 Related Work

In this section, we review prominent work on creating tamper resistant audit logs. We highlight our contribution in comparison with the existing work.

Audit logs. Schneier and Kelsey [76, 77] enhanced audit logs to detect tampering in audit logs. The proposed scheme can detect log tempering even if there is a security breach at the logging server. Their system can only detect audit log entries that were created prior to the attack and does not provide a mechanism to stop the attacker for deleting or inserting audit records. However *BlockTrail* can detect these unwanted modifications to audit logs. Waters*et al.* [86] proposed a searchable encrypted audit log, which uses identity-based encryption keys to encrypt audit logs and allow search by certain keywords. Yavuz and Ning [91] developed a forward secure and aggregate audit logging system for distributed systems, without using a trusted third party. Snodgrass *et al.* [79] used trusted notary and a check field stored in each tuple for detecting tempering in audit logs. Zawoad *et al.* presented Secure-Logging-as-a-Service (SecLaaS) for storing virtual machine audit logs in a secure manner, SecLaaS ensures confidentiality of users and protects the integrity of logs by preserving proofs of past logs. Ma and Tsudik [56] looked into temper-evident logs that use forward-secure aggregating signature schemes.

Blockchain and Audit Logs. Chi and Yai [29] proposed an ISO/IEC 15408-2 Compliant Security Auditing system using Ethereum that creates encrypted audit logs for IOT devices. Chen *et al.* [30] proposed a Blockchains based system to address shortcomings in log-based misbehavior monitoring schemes used to monitor Certificate Authorities (CA). In contrast to prior work, *BlockTrail* is implemented by extending the ORM, which does not require any significant change to the underlying application. Xu *et al.* [89] proposed to use game theory and blockchain to reduce latency by moving applications to edge servers. Similarly, we are using geographical proximity to store audit logs in servers that are close. to reduce latency. Sutton and Samvi [83] proposed solution stores integrity proof digest to the Bitcoin network. Castaldo *et al.* [27] designed a centralized logging system that used blockchain for unforgeable record keeping. This system is used for facilitating health data exchange across multiple countries in the European Union.

4.7 Summary

BlockTrail uses a hierarchical blockchain architecture to reduce the space and time complexity of conventional blockchain systems. In *BlockTrail*, we leverage the nature of transactions to partition the system into multiple layers that can process transactions in parallel. *BlockTrail* reduces the space overhead and accelerates the validation process by reducing the number of active replicas. Additionally, we also introduce various security measures that enhance the defense capabilities of *BlockTrail*, and enable the detection of faulty replicas in the system.

CHAPTER 5: PERFORMANCE EVALUATION OF CONSENSUS PROTOCOLS

Blockchain-based audit systems suffer from low scalability and high message complexity. The root cause of these shortcomings is the use of PBFT consensus protocol in those systems. Alternatives to PBFT have not been used in blockchain-based audit systems due to the limited knowledge about their functional and operational requirements. Currently, no blockchain testbed supports the execution and benchmarking of different consensus protocols in a unified testing environment. In this paper, we build a blockchain testbed that supports the execution of five state-of-the-art consensus protocols in a blockchain network; namely PBFT, Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Elapsed Time (PoET), and Clique. We carry out performance evaluation of those consensus algorithms using data from a real-world audit system. Our results show that the Clique protocol is best suited for blockchain-based audit systems, based on scalability features.

5.1 Introduction and Motivation

As we have seen so far, audit systems use blockchains to harden the security of audit logs and provide a fine-grained provenance model for data flows [3]. From a security standpoint, blockchains prevent the audit logs from two well-known attacks, the "the physical access attack" and "the remote vulnerability attack", which both allow an adversary to corrupt data in the audit logs [58, 54]. For provenance, the blockchain ledger maintains a high-level history of changes made in the audit logs, which can be used for fault detection if needed. However, despite such benefits, blockchain-based audit systems suffer from low scalability and throughput. The root cause of the poor scalability is the use of the "Practical Byzantine Fault Tolerance" (PBFT) consensus protocol, which has a high message complexity and, as a result, low scalability. To amortize the complexity cost, a design optimization has been recently proposed which fragments blockchains into hierarchical layers to support parallel processing as the system scales up [6]. However, the proposed design also uses the PBFT consensus protocol. Therefore, beyond nominal optimizations, the existing audit systems still suffer from low scalability.

Moreover, in all notable efforts on blockchain-based audit logs, including our prior work, the performance evaluation has been done using an abstract simulation of the proposed frameworks [3, 6]. Intuitively, simulations convey a general idea about the system performance. However, they provide limited information about the design feasibility in the real world. Some frameworks that may appear to be operational in a simulated environment may not be fully relevant in real-world settings. Acknowledging that, in [3], the authors mention that their proposed system *BlockAudit* is yet to be evaluated in the production environment to study its suitability to the actual audit system.

With these two challenges in mind, an intuitive solution would be to apply alternative consensus protocols in the blockchain-based audit systems for optimization. Some of the notable consensus protocols include Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Elapsed Time (PoET), and Clique [16]. The benchmarking and evaluation of these protocols require realistic testing environments that can accurately model the varying dynamics of the audit systems and the distributed blockchain. Such a comparative analysis can provide new and more accurate insights to improve the performance of the blockchain-based audit logs.

5.1.1 Challenges

The intuitive approach to tackle these problems has its own inherent challenges. The existing benchmarks for the aforementioned consensus protocols are usually the latency in end-to-end transaction processing and the transaction throughput. Audit systems cannot tolerate latency in transaction processing beyond a few milliseconds [32] (§5.3.1). In PBFT-based systems, the processing latency increases with the number of network nodes. Therefore, the performance is affected as the system scales up. However, in the PoW-based systems, the latency margins show minimal growth as the number of nodes increases [74]. As a result, they are considered highly scalable and more useful for large distributed networks. The throughput of a blockchain system is measured by the number of transactions processed per second. In the literature, and due to varying system features the reported throughput of consensus protocols is often not comparable. For example, in [14], the authors compare the throughput of Bitcoin with Visa, stating that Bitcoin has a low throughput of 7 transactions per second, while Visa can process up to 24,000 transactions per second. However, the two systems are not evaluated under the same settings. The transaction size of Bitcoin and Visa vary significantly, and therefore the total number of transactions processed in a second are not easily comparable.

5.1.2 Approach

Motivated by these limitations, we take a clean-slate approach towards the performance evaluation of various consensus protocols in blockchain-based audit logs. Towards that, we create a comprehensive testbed to model a real-world blockchain-based audit system. We select five blockchain consensus protocols and benchmark their performance in a "level playing field." Finally, by contrasting their performance, we reason about the most suitable protocols that meet the requirements of the audit systems with respect to various parameters, including end-to-end latency. The notable features of our work are summarized below as the key contributions.

5.1.3 Contribution

First, we deploy a blockchain testbed consisting of a set of nodes hosted across various geographical locations. At each node, we encode the rules of five consensus protocols and the access control policies. To provide a "level playing field," we generate transactions of a uniform size for each consensus protocol and maintain a consistent network topology. To accurately reflect the real world application, we use actual transactions from a real business system [32] for which log audit is built, and use its requirement thresholds to assess the feasibility of each protocol. For performance evaluation, we use two metrics including latency in the block confirmation and the transaction throughput. Finally, by contrasting the performance of all the consensus protocols, we present the most optimal solution for blockchain-based audit logs. To the best of our knowledge, this is the first attempt towards the performance evaluation of blockchain-based audit logs under various consensus protocols. Additionally, the rest of the chapter includes background and preliminaries in §5.2, testbed design and deployment in §5.3, results and evaluation in §4.5.1, related work and discussion in §5.5, and summary and concluding remarks in §5.6.

5.2 Preliminaries

In this section, we provide the background of blockchain-based audit logs, mostly based on the prior work in [6], and discuss their limitations that motivate our work and provide a brief overview of five consensus protocols that we evaluate.

Today's blockchain systems for audit logs have various shortcomings, including the following. First and foremost, the state-of-the-art blockchain-based audit systems mostly use the PBFT consensus protocol [5, 6], which suffers from high message complexity and low scalability features. Second, the existing systems suffer from a high-latency, even with optimizations. For example, even with optimizations that rely on fragmenting the blockchain system into multiple layers, as in [6], the transaction confirmation time ranged from 2 to 20 seconds. On the other hand, audit systems would typically mandate a confirmation time in the order of milliseconds. Third, existing evaluations and assessments for the performance of blockchain systems heavily relied on simulations that abstract (or even ignore) various parameters that are hard to measure, such as the network latency, and compute capabilities, especially on shared infrastructure, making it difficult to reason about the relevance and practicality of those results; e.g., [5, 6]. In [6], for example, *BlockTrail* was tested across 250 nodes with varying transaction rates. However, we note that the same system uses PBFT, which has a high message complexity and is unlikely to scale to 250 nodes. Indeed, using our testbed, we demonstrate in §3.5 that PBFT is impractical beyond 50 nodes. There is a need for performance evaluation of blockchain-based audit systems, taking into account the actual characteristics of networks and various blockchain consensus protocols.

Comparative Analysis. In Table 5.1, we show some statistics about the consensus protocols evaluated in this work. It can be observed that PBFT has the lowest fault tolerance (33%) and the highest throughput (10,000 transactions per second). The fault tolerance of each protocol is derived from their theoretical analysis, while the throughput measurements are obtained from the experiments and simulations. As mentioned in §5.1, the experimental results of these protocols are not comparable since they were evaluated in different settings with varying transaction and network sizes. Our testbed resolves this issue by creating a unified testing environment for each consensus protocol. As a result, the values we report are comparable.

Table 5.1: Comparison of the consensus protocols considered in this study. The values mentioned in the table are collected from various sources including [62, 88, 80, 18, 20]. From the values reported below, PBFT seems to achieve the highest throughput with over 10,000 transactions per second.

Properties	**PoW**	**PoS**	**PBFT**	**Clique**	**PoET**
Blockchain Type	Permisssionless	Permissionless	Permissioned	Permissioned	Permissionless
Trust Model	Untrusted	Untrusted	Semi-trusted	Semi-trusted	Untrusted
Scalability	High	High	Low	High	High
Throughput	<10	<1,000	<10,000		1,000
Fault Tolerance	50%	50%	33%	50%	50%
Energy Efficient	No	Yes	Yes	Yes	Yes

5.3 Testbed Design and Deployment

In the following, we present details of our blockchain testbed. We outline the functional requirements of our audit system, which provides the baseline thresholds for scalability and throughput.

5.3.1 Audit System Requirements

For our testbed instrumentation, we used a real-world audit system provided by an eGovernment enterprise system (application) [32, 45]. Such systems typically serves 20–200 clients at any time, depending upon the nature of the application. As such, we expected the blockchain network size to vary between 20–200 nodes based on the system requirement. The audit log transaction size varied between 1MB–4MB, with an expected confirmation time of 50 milliseconds. In other words, with a maximum network size of 200 nodes and a maximum transaction size of 4MB, the application required the audit log transaction to be processed under 50 milliseconds. We used these thresholds as our baseline benchmarks to evaluate the performance of each of the five consensus protocol. The protocols that exceeded the baseline thresholds were considered practically feasible for our blockchain-based audit logging system.

5.3.2 Blockchain Nodes

We set up the blockchain nodes using Docker containers. Docker uses OS-level virtualization to create lightweight, isolated, and standalone software packagers called containers [60]. A Docker container contains the application code, runtime system tools, and system libraries by default. This allows the container application to seamlessly run in any environment that supports Docker, sidestepping the complexities of the host server configurations. Specific to the requirements of our testbed, Docker enables us to swiftly spawn multiple blockchain nodes across various data centers.

Moreover, Docker also supports dynamic adjustment of the network size, which allowed us to test the performance of each consensus protocol at varying network size. However, for this experiment, we maintain a fixed topology in which the network is a completely connected graph.

We deployed the testbed nodes on the *digital ocean data center*, which hosts its data centers globally. All servers in data centers were assigned Public IP addresses, and 50 Docker containers were set up at each server, each identified by a unique port number. The port numbers ranged between 42421–42470, and the port mapping scheme was specified in the Docker configuration file. Each server has an Intel Xeon processor with 4 cores, 16GB RAM, and 500GB hard drive.

5.3.3 Communication Model

To enable communication among the blockchain nodes, we used NodeJS [66] , an open-source JavaScript framework that allows asynchronous execution of the JavaScript code. At each container, we also installed a blockchain node middleware, built using the express minimalistic web framework. The middleware contained the rules of the consensus protocols and communication model. At the application layer, each node communicated using HTTP protocol with the GET and POST methods to exchange data (*i.e.,* transactions and blocks) with other nodes. In Figure 5.1, we provide an overview of our testbed design. The Docker containers were hosted in five cities namely San Francisco, New York, Singapore, London, and Frankfurt. The broad distribution of nodes helped us to closely model real-world blockchain systems (globally distributed), where nodes are hosted across multiple cities. As a result, the latency observed in the transaction confirmation were more realistic, adding reliability to our methodology and evaluation.

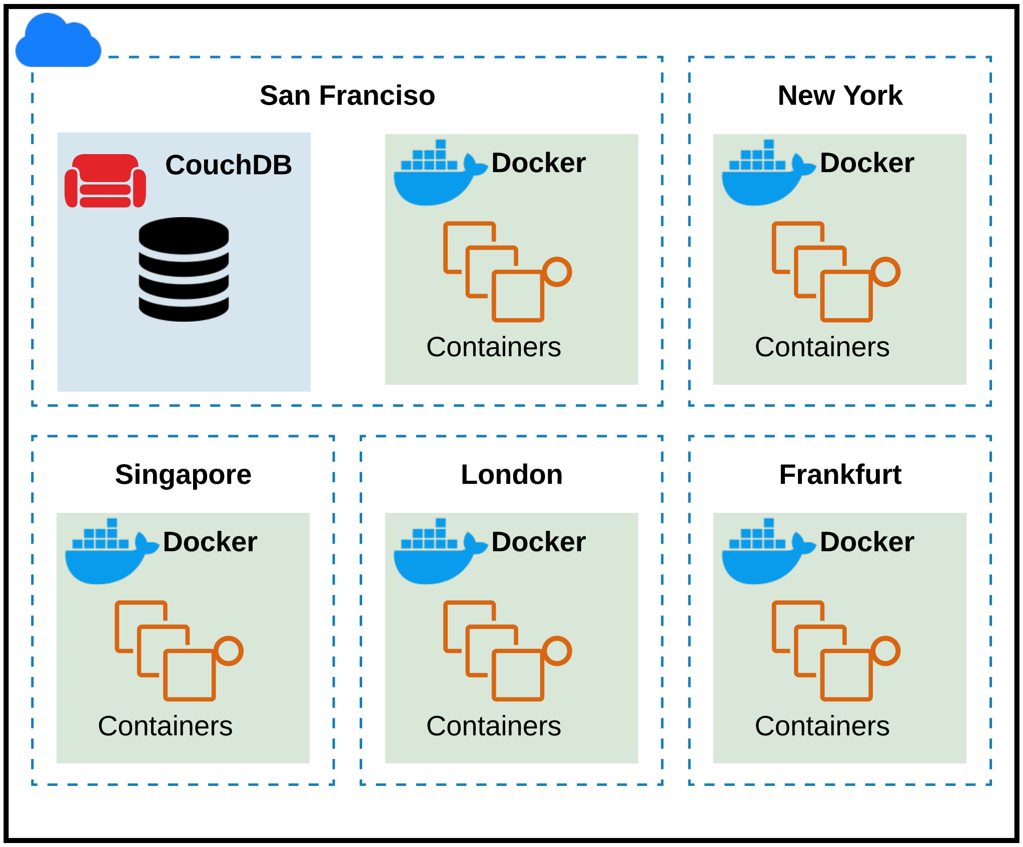
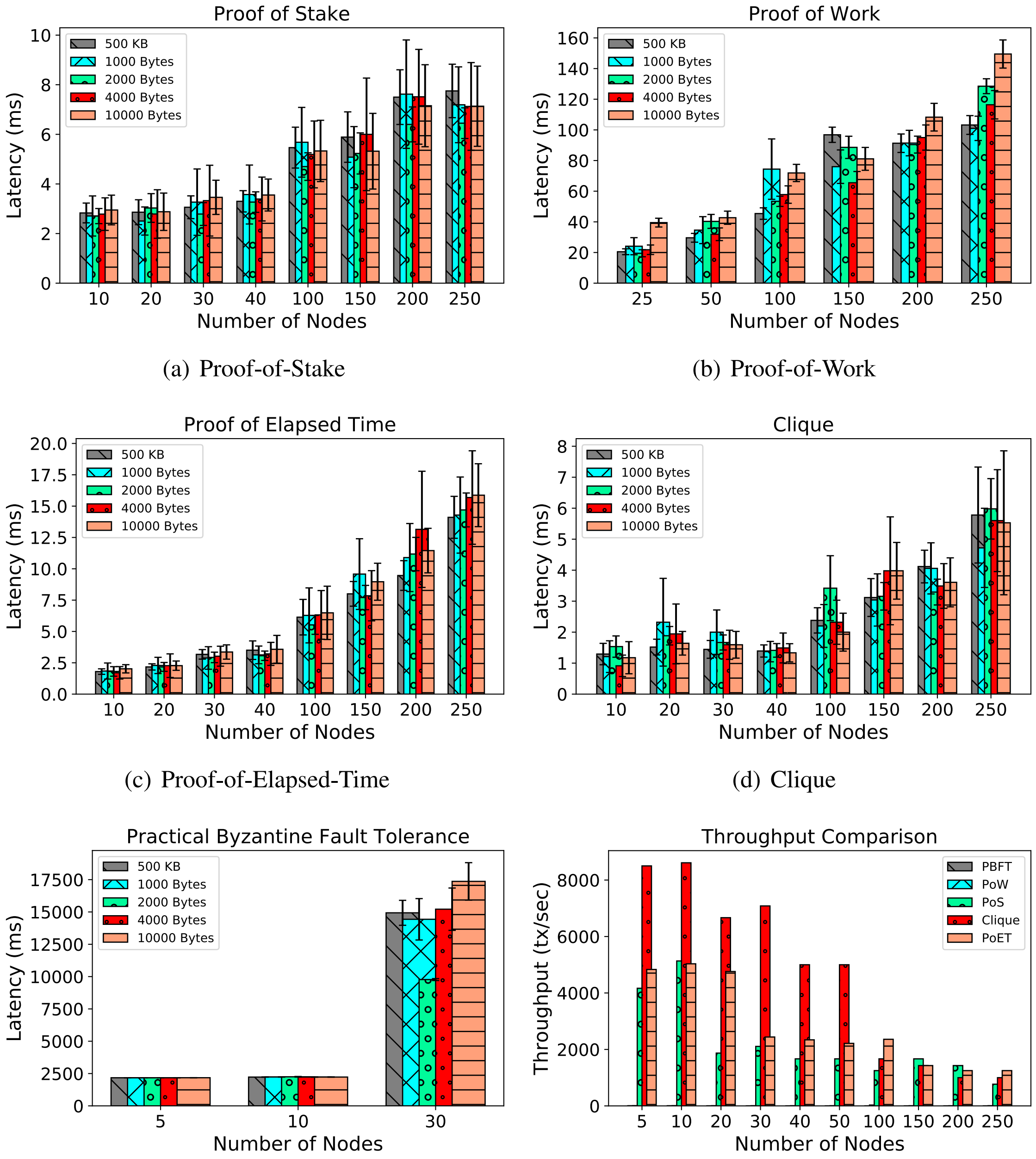


Figure 5.1: Overview of the testbed setup. We used five servers based on three continents each hosting up to 50 blockchain nodes. The CouchDB database is used to store statistics generated from the other nodes.

5.3.4 System Adjustments

Since each consensus protocol has unique rules, we tailored our system to correctly implement them. For PoW, we used the ‘work-token’ library in NodeJS that allows us to modularly adjust the difficulty (see Figure 2.3) for the target threshold. We selected a difficulty limit that allowed block generation after every 10 milliseconds. When the PoW protocol was executed, each node used its processing power to solve the challenge. Since the network was completely connected, a block is received by all the other nodes directly from the winner who produced the block.



(a) Proof-of-Stake

(b) Proof-of-Work

(c) Proof-of-Elapsed-Time

(d) Clique

(e) PBFT

(f) Throughput

Figure 5.2: Experimental results. In 5.2(a)–5.2(e), we report the latency in transaction confirmation for each consensus protocol. In 5.2(f), we report the transaction throughput all the protocols, measured at a transaction size of 4MB. Notice that in PoW and PBFT, transactions incur high latency, exceeding 50 milliseonds threshold set by our audit system (§5.3.1). In contrast, PoS, PoET, and Clique meet the latency requirements of the audit system. Among them, Clique has the most desirable performance, achieving a low latency and high throughput.

For PoS, we randomly assigned a stake tokens to each node. The stake token had a value between 100-10,000. To simplify the auction process, we embedded stakes within the block header. In each round, every node generated its own block with a randomly selected stake and broadcast the block to all other nodes. It also received blocks from the other nodes with their specified stakes. Only the block with the highest stake was selected as the winner.

To implement PoET, we sidestepped the cost of installing the Intel SGX at each node. Instead, we relaxed the trust model by replacing the SGX code with a trusted code that issued a random waiting time for each node. We acknowledge that replacing SGX relaxed (i.e., weakened) the security guarantees of our testbed. However, within the scope of this work, we are primarily concerned with the performance evaluation rather than the security guarantees. Therefore, for PoET, as outlined in §5.2, random waiting times were allocated to each node and once the waiting time expired, the node released its block. The waiting time calibration prevented the issuance of multiple blocks within one round. As a result, we avoided forks.

To implement Clique and PBFT, we followed a round-robin scheme for the selection of primary replicas. For Clique, we encoded the primary selection procedure in the *Clique.js* file at each node. The protocol followed the same workflow as outlined in Figure 2.4. For PBFT, we also switched the primary replica in each round. The primary ordered transactions in a block and executed a multi-phase protocol in which it obtained approvals from all the other replicas. Each phase was implemented using REST API.

5.4 Results and Evaluation

As mentioned in §5.3.1, transaction size, and the network size varied between 1MB–4MB and 20–200 clients, respectively. Following that, and keeping into account the safety margins, we tested our system with transaction size between 0.5MB and 4MB, and 10–250 nodes. For simplicity, and without losing generality, we treated one transaction as a unique block. For each protocol, we evaluated the transaction throughput as the number of transactions processed in one second. In Figure 5.2, we present the results obtained from our experiments. In 5.2(a)– 5.2(e), we report the latency in transaction confirmation for each protocol, and in 5.2(f), we report their throughput.

Figure 5.2 shows that, on average, transactions experienced minimum latency in PoET and Clique. For Clique, the maximum latency was $\approx$ 14 milliseconds with a network size of 250 nodes and a transaction size of 4MB. For PoET, the maximum latency was $\approx$ 20 milliseconds with a network size of 200 nodes and a transaction size of 1MB. For PoW and PoS, the maximum latency was 140 milliseconds and 30 milliseconds at the network size of 250 and 200 nodes, respectively. An increase in the latency in PoW was observed due to the distribution of the computing power as the number of nodes has increased. As mentioned in §5.3.1, our audit system cannot tolerate latency beyond 50 milliseconds. Therefore, PoW becomes infeasible after 50 nodes (see 5.2(b)). Surprisingly, latency in PBFT increased significantly as the number of nodes increased beyond 10 nodes. The maximum latency was 35 seconds with transaction and network size of 10MB and 30 nodes respectively. Beyond 30 nodes, the latency became more significant, and we omit results from our plot in 5.2(e). In all the consensus protocols, the latency in transaction confirmation increased with the network size due to propagation latency among nodes hosted in different cities.

For transaction throughput reported in 5.2(f), we use the transaction size of 4MB and vary the the network size from 5–250 nodes. We observed that up to a network size of 50 nodes, Clique achieved the maximum throughput, followed by PoET and PoS. With a network size of 5 nodes, Clique achieved a throughput of 8,000 transactions per second. After 50 nodes, the transaction throughput of Clique decreased significantly and PoET and PoS became more dominant. In contrast, PBFT had a very low throughput, showing that the models proposed by the prior work in blockchain-based audit logs [6, 5] may not be feasible in a real-world implementation, and calling for a different choice of the consensus algorithm.

In summary, our results show that for the general purpose, PoS, PoET, and Clique can be used for audit systems. Among them, Clique is the most suitable protocol when the network size is below 50 nodes. Once the network size exceeds 50 nodes, the audit system can benefit by utilizing one of the three protocols. We also noticed that PoW is not feasible beyond 50 nodes since the latency margins exceed the threshold requirements of our audit system. Even at a network size below 50 nodes, the latency is much higher than PoS, PoET, and Clique. In all settings, PBFT's performance was significantly lower than the other protocols, making it the least suitable solution for the blockchain-based audit systems.

5.5 Related Work and Discussion

In this section, we briefly discuss the related work to blockchain-based audit systems. Although, notable efforts have been done to integrate blockchains with audit logs, this chapter—to the best of our knowledge—is the first attempt to evaluate the performance of audit systems under different consensus protocols.

Towards blockchain-based audit systems, Chi and Yai [29] used the Ethereum blockchain to create encrypted audit logs for IoT devices. Sutton and Samavi [83] developed a mechanism to store integrity proof digests of audit logs in bitcoin blockchains. Castaldo *et al.* [27] proposed a logging system which uses blockchains to maintain unforgeable records of health care data exchanges across multiple countries. Cucrull *et al.* [37] used blockchains to improves the security of the immutable logs by publishing integrity proofs on the blockchain network. Their proposed system is resilient to log truncation and log regeneration.

Ahmad *et al.* [3] proposed a framework that integrated Online Transaction Processing System

(OLTP) system's audit logs with blockchains. Their system used nHibernate (Object Relation Mapping) events to generate audit log transactions, and PBFT consensus protocol to obtain the agreement from the network. In [6], a design optimization of [3] was proposed using a hierarchical design to reduce the system complexity and increase the throughput. However, their proposed system only supports audit applications that follow a hierarchical system model. Moreover, even in the optimized version, PBFT is used for consensus. In our work, we show that PBFT is incapable of meeting the requirements of the audit systems. Instead, through a comparative analysis, we show that Clique and PoET are better replacements of PBFT and are highly suitable for audit systems.

5.6 Summary

In this chapter, we evaluate the performance of a blockchain-based audit system under five consensus protocols, namely PoW, PoS, PBFT, PoET, and Clique. We developed a testbed to set up a blockchain network of 250 nodes and implemented the consensus rules of each protocol and evaluate their performance at a unified transaction size and network. Our results show that specific to the requirements of audit systems, PoET, PoS, and Clique are the most useful consensus protocols, with Clique achieving the maximum transaction throughput among them. Moreover, our results show that PBFT, used in notable prior works, may not be an optimum choice due to its high latency. Our work opens a new direction in the performance evaluation of blockchain systems by showing trade-offs in the real-world performance of key blockchain consensus protocols.

CHAPTER 6: CONCLUSION AND FUTURE WORK

The main goal of this work is to harness blockchain technology to create tamper-proof, distributed append-only audit logs that can seamlessly integrate into existing enterprise business applications. For that purpose, first, we presented *BlockAudit* which extended the nHibernate ORM to generate blockchain-compatible audit logs, that are persisted in a blockchain network. For our experiments, we used a custom blockchain simulator (abstracted from Hyperledger fabric) to validate our design. *BlockAudit* is secure against known audit log attacks and can be easily integrated into existing audit applications. However, this system has two major limitations, 1) high storage footprint due to data replication over all network peers, and 2) low scalability due to high message complexity ($O(n^2)$) of the PBFT consensus protocol.

To overcome the limitations of *BlockAudit* we created *BlockTrail* which exploits the hierarchical distribution of replicas across multiple layers to reduce space and time complexity of the system. We leveraged the nature of transactions generated by the audit applications to partition the system into multiple layers to enable parallel transaction processing. *BlockTrail* amortizes the space overhead of storing data into a single ledger and increases the overall system throughput. Additionally, in *BlockTrail*, we hardened the security of the system to detect malicious or faulty replicas. We tested our design using our custom-built blockchain system. Our results validate the hypothesis by showing a considerable reduction in space and time overhead in the layered blockchain system.

BlockTrail addressed several limitations of *BlockAudit*, but a key bottleneck in both designs was the use of PBFT consensus protocol. PBFT is commonly used in permissioned blockchains and is known to provide a high transaction throughput. However, it also suffers from a high message complexity, which limits scalability. As such, we realized a need for a different consensus protocol that would overcome the limitations of PBFT. For that purpose, in the third and final component of this work, we evaluated the performance of a blockchain-based audit system under five consensus protocols, namely PoW, PoS, PBFT, PoET, and Clique. We developed a blockchain testbed to implement consensus protocols and evaluate their performance at a level playing field (*i.e.,* uniform transaction and network size). Our results show that PoET, PoS, and Clique are highly suited for audit systems, with Clique achieving the maximum transaction throughput among them. Moreover, our results show that PBFT, used in prior works, may not be an optimum choice due to high latency.

Our work opens new directions towards the design and development of blockchain-based audit logs using various consensus protocols. In the future, we plan to extend the usability of our design by incorporating other consensus protocols including Proof-of-Stake, Avalanche, Snowflake etc., and analyze their performance for audit applications. Additionally, we will generalize the testbed utility to allow the instrumentation of other applications under these consensus protocols.

Understanding the performance of the various systems under different types of workloads pertaining to audit logs is also an open question. While we have evaluated our systems under load, we did not consider adversarial behaviors of the peers in the underlying network, which is yet another future direction: How do the guarantees of those blockchain systems change by changing the behavior of the underlying peers; e.g., under prefix hijacking [72].

For *BlockAudit* and *BlockTrail*, we used nHibernate ORM, and SQL-based database to generate the audit logs. These features allow easy compatibility of applications with blockchains. A major challenge in the future is to create blockchain-based audit logs for applications that do not use nHibernate ORM and SQL databases. This might require tailoring the application structure for a blockchain-compatible format. This is also part of our future work. Moreover, for this work, we have explored the benefits of blockchains for e-government applications. In the future, we will extend our work beyond the e-government applications and explore possible methods of creating a hierarchical blockchain solution, specifically tailored to their requirements. The possible applications could be IoT and supply chain systems. We will also design a framework to optimize the number of nodes in the system to increase security while minimizing the storage footprint.

We carried out all our experiments using a uniformly distributed blockchain network (*i.e.,* all data centers have the same number of nodes). However, the blockchain nodes may not be uniformly distributed in a real blockchain network. Therefore studying the effects of network topology on the performance of the consensus protocol could be another future direction.

LIST OF REFERENCES

[1] The personal information protection and electronic documents act, 2004.

[2] Ittai Abraham and Dahlia Malkhi. The blockchain consensus layer and BFT. *Bulletin of the EATCS*, 123, 2017.

[3] Ashar Ahmad, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards blockchain-driven, secure and transparent audit logs. In *International Workshop on Distributed Ledger of Things (DLoT) in conjunction with MobiQuitous*, 2018.

[4] Ashar Ahmad, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards blockchain-driven, secure and transparent audit logs. In *ACM International Workshop on Distributed Ledger of Things(DLoT), in conjunction with International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous, New York City, NY, USA*, pages 443–448, Nov 2018.

[5] Ashar Ahmad, Muhammad Saad, and Aziz Mohaisen. Secure and transparent audit logs with blockaudit. *Journal of Network and Computer Applications*, 145:102406, 2019.

[6] Ashar Ahmad, Muhammad Saad, Laurent Njilla, Charles A. Kamhoua, Mostafa Bassiouni, and Aziz Mohaisen. Blocktrail: A scalable multichain solution for blockchain-based audit trails. In *IEEE International Conference on Communications, ICC Shanghai, China*, pages 1–6, May 2019.

[7] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. In *Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA*, Feb 2018.

[8] Abdullah Alghamdi, Majdi Sabe Owda, and Keeley A. Crockett. Natural language interface to relational database (NLI-RDB) through object relational mapping (ORM). In Plamen Angelov, Alexander E. Gegov, Chrisina Jayne, and Qiang Shen, editors, *Advances in Computational Intelligence Systems - Contributions Presented at the 16th UK Workshop on Computational Intelligence, Lancaster, UK*, volume 513 of *Advances in Intelligent Systems and Computing*, pages 449–464. Springer, Sept 2016.

[9] Arnold O. Allen. *Probability, statistics and queueing theory - with computer science applications (2. ed.).* Academic Press, 1990.

[10] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys Conference, Porto, Portugal*, pages 30:1–30:15, April 2018.

[11] Stefano De Angelis. Assessing security and performances of consensus algorithms for permissioned blockchains. *CoRR*, abs/1805.03490, 2018.

[12] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. pages 375–392. IEEE, 2017.

[13] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. Automated synthesis and dynamic analysis of tradeoff spaces for object-relational mapping. *IEEE Trans. Software Eng.*, 43(2):145–163, 2017.

[14] Shehar Bano, Mustafa Al-Bassam, and George Danezis. The road to scalable blockchain designs. *USENIX:Login*, 42(4), 2017.

[15] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[16] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[17] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. SoK: Consensus in the Age of Blockchains, 2017. `https://arxiv.org/abs/1711.03936`.

[18] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *arXiv preprint arXiv:1406.5694*, 2014.

[19] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *International Workshop on Financial Cryptography and Data Security , Christ Church, Barbados*, pages 142–157, Feb 2016.

[20] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake. *IACR Cryptology ePrint Archive*, 2014:452, 2014.

[21] Petra Berenbrink, Tom Friedetzky, Peter Kling, Frederik Mallmann-Trenn, Lars Nagel, and Chris Wastell. Self-stabilizing balls & bins in batches. *CoRR*, abs/1603.02188, 2016.

[22] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella Béguelin. Formal verification of smart contracts: Short paper. pages 91–96, 2016.

[23] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research perspectives and challenges for bitcoin and cryptocurrencies. *IACR Cryptology ePrint Archive*, 2015:261, 2015.

[24] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[25] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

[26] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *ACM SIGSAC Conference on Computer and Communications Security, CCS Dallas, USA*, pages 683–699, Oct 2017.

[27] Luigi Castaldo and Vincenzo Cinque. Blockchain-based logging for the cross-border exchange of ehealth data in europe. In Erol Gelenbe, Paolo Campegiani, Tadeusz Czachórski, Sokratis K. Katsikas, Ioannis Komnios, Luigi Romano, and Dimitrios Tzovaras, editors, *Security in Computer and Information Sciences*, pages 46–56, Cham, 2018. Springer International Publishing.

[28] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186, 1999.

[29] Shi-Cho Cha and Kuo-Hui Yeh. An ISO/IEC 15408-2 compliant security auditing system with blockchain technology. In *2018 IEEE Conference on Communications and Network Security, CNS 2018, Beijing, China, May 30 - June 1, 2018*, pages 1–2, 2018.

[30] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for TLS connections. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 2060–2068, 2018.

[31] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[32] ClearVillage. Clearvillage, 2018.

[33] Nhibernate Community. Nhibernate, 2018.

[34] Douglas Crockford. The application/json media type for javascript object notation (JSON). *RFC*, 4627:1–10, 2006.

[35] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[36] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In *International Workshop on Financial Cryptography and Data Security, Christ Church, Barbados*, pages 106–125, Feb 2016.

[37] Jordi Cucurull and Jordi Puiggali. Distributed immutabilization of secure logs. In *International Workshop on Security and Trust Management STM Heraklion, Greece*, pages 122–137, Sept 2016.

[38] Gaby G. Dagher, Praneeth Babu Marella, Matea Milojkovic, and Jordan Mohler. Broncovote: Secure voting system using ethereum's blockchain. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP, Funchal, Madeira - Portugal*, pages 96–107, Jan 2018.

[39] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. 2016.

[40] M. M. Eljazzar, M. A. Amr, S. S. Kassem, and M. Ezzat. Merging supply chain and blockchain technologies. *Computing Research Repository (CoRR)*, abs/1804.04149, 2018.

[41] P Froystad and Jarle Holm. Blockchain: powering the internet of value. *EVRY Labs*, 2016.

[42] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Computers*, 52(2):139–149, 2003.

[43] GDPR. General data protection regulation gdpr, 2019.

[44] Johannes Göbel and Anthony E Krzesinski. Increased block size and bitcoin blockchain dynamics. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE, 2017.

[45] GSA. Government software assurance, 2018.

[46] Rui Guo, Huixian Shi, Qinglan Zhao, and Dong Zheng. Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems. *IEEE Access*, 6:11676–11686, 2018.

[47] Freya Sheer Hardwick, Raja Naeem Akram, and Konstantinos Markantonakis. E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy. *CoRR*, abs/1805.10258, 2018.

[48] Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors. *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*. ACM, 2008.

[49] Hyperledger. Hyperledger sawtooth, 2018.

[50] Emanuel Ferreira Jesus, Vanessa R. L. Chicarino, Célio V. N. de Albuquerque, and Antônio A. de A. Rocha. A survey of how to use blockchain to secure internet of things and the stalker attack. *Security and Communication Networks*, 2018:9675050:1–9675050:27, 2018.

[51] Ghassan Karame. On the security and scalability of bitcoin's blockchain. In *ACM Conference on Computer and Communications Security SIGSAC, Vienna*, pages 1861–1862, Oct 2016.

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

[53] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. pages 839–858, 2016.

[54] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Loggc: garbage collecting audit log. In *ACM SIGSAC Conference on Computer and Communications Security CCS, Berlin, Germany*, pages 1005–1016, Nov 2013.

[55] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. *J. Computer Virology and Hacking Techniques*, 13(1):47–85, 2017.

[56] Di Ma and Gene Tsudik. A new approach to secure logging. *TOS*, 5(1):2:1–2:21, 2009.

[57] Imran Makhdoom, Mehran Abolhasan, Haider Abbas, and Wei Ni. Blockchain's adoption in iot: The challenges, and a way forward. *Journal of Network and Computer Applications*, 125:251 – 279, 2019.

[58] Jonathan Margulies. A developer's guide to audit logging. *IEEE Security & Privacy*, 13(3):84–86, 2015.

[59] Lara Mauri, Stelvio Cimato, and Ernesto Damiani. A comparative analysis of current cryptocurrencies. pages 127–138, Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP , Funchal, Madeira - Portugal, January 2018.

[60] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[61] Matthias Mettler. Blockchain technology in healthcare: The revolution starts here. In *18th IEEE International Conference on e-Health Networking, Applications and Services, Munich, Germany*, pages 1–3, Sep 2016.

[62] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Online, `https://bitcoin.org/bitcoin.pdf`, 2008.

[63] Giang-Truong Nguyen and Kyungbaek Kim. A survey about consensus algorithms used in blockchain. *JIPS*, 14(1):101–128, 2018.

[64] Hoang-Long Nguyen, Claudia-Lavinia Ignat, and Olivier Perrin. Trusternity: Auditing transparent log server with blockchain. In *Companion of the The Web Conference, Lyon , France*, pages 79–80, April 2018.

[65] Christos Nikolaou, Manolis Marazakis, and G. Georgiannakis. Transaction routing for distributed OLTP systems: Survey and recent results. *Inf. Sci.*, (1&2):45–82, 1997.

[66] Node.js. Hyperledger sawtooth, 2019.

[67] Werner Olivier and Rossouw von Solms. The effective utilization of audit logs in information security management. In *Annual Working Conference on Information Security Management & Small Systems Security*, pages 51–62, Sept 1999.

[68] Dean Rakic. Blockchain technology in healthcare. In *Proceedings of the 4th International Conference on Information and Communication Technologies for Ageing Well and e-Health, Funchal, Madeira, Portugal, March 2018.*, pages 13–20, 2018.

[69] Christoph Ringelstein and Steffen Staab. DIALOG: distributed auditing logs. In *IEEE International Conference on Web Services, ICWS, Los Angeles, USA*, pages 429–436, July 2009.

[70] M. Saad and A. Mohaisen. Towards characterizing blockchain-based cryptocurrencies for highly-accurate predictions. In *IEEE Conference on Computer Communications Workshops*, April 2018.

[71] Muhammad Saad, Afsah Anwar, Ashar Ahmad, Hisham Alasmary, Murat Yuksel, and Aziz Mohaisen. Routechain: Towards blockchain-based secure and efficient bgp routing. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 210–218. IEEE, 2019.

[72] Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, and Aziz Mohaisen. Paritioning attacks on bitcoin: Colliding space, time, and logic. In *IEEE International Conference on Distributed Computing Systems ICDCS, Dellas, Texas, US*, July 2019.

[73] Muhammad Saad, Aminollah Khormali, and Aziz Mohaisen. End-to-end analysis of in-browser cryptojacking. *arXiv preprint arXiv:1809.02152*, 2018.

[74] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles A. Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. Exploring the attack surface of blockchain: A systematic overview. *CoRR*, abs/1904.03487, 2019.

[75] Muhammad Saad, My T. Thai, and Aziz Mohaisen. POSTER: deterring ddos attacks on blockchain-based cryptocurrencies through mempool optimization. In *Proceedings of Asia Conference on Computer and Communications Security, ASIACCS, Incheon, Republic of Korea*, pages 809–811, Jun 2018.

[76] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur*, 2(2):159–176, 1999.

[77] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium, San Antonio, USA*, Jan 1998.

[78] Pradip Kumar Sharma, Shailendra Rathore, and Jong Hyuk Park. Distarch-scnet: Blockchain-based distributed architecture with li-fi communication for a scalable smart city network. *IEEE Consumer Electronics Magazine*, 7(4):55–64, 2018.

[79] Richard Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper detection in audit logs. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada*, pages 504–515, Aug 2004.

[80] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013(881), 2013.

[81] Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Alexei Zamyatin, and Edgar R. Weippl. Agreement with satoshi - on the formalization of nakamoto consensus. *IACR Cryptology ePrint Archive*, 2018:400, 2018.

[82] Harish Sukhwani, José M. Martínez, Xiaolin Chang, Kishor S. Trivedi, and Andy Rindos. Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric). In *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*, pages 253–255, 2017.

[83] Andrew Sutton and Reza Samavi. Blockchain enabled privacy audit logs. In *International Semantic Web Conference ISWC, Vienna, Austria*, pages 645–660, Oct 2017.

[84] Marco Vieira and Henrique Madeira. Benchmarking the dependability of different OLTP systems. In *International Conference on Dependable Systems and Networks (DSN)*, pages 305–310, June 2003.

[85] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In Jan Camenisch and Dogan Kesdogan, editors, *International Workshop on Open Problems in Network Security - IFIP WG 11.4 , iNetSec, Zurich, Switzerland*, volume 9591 of *Lecture Notes in Computer Science*, pages 112–125. Springer, Nov 2015.

[86] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004, San Diego, California, USA*, 2004.

[87] Christopher Wee. Audit logs: to keep or not to keep? In *Recent Advances in Intrusion Detection*, 1999.

[88] Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. ACM, 2016.

[89] Dongjin Xu, Liang Xiao, Limin Sun, and Min Lei. Game theoretic study on blockchain based secure edge networks. In *International Conference on Communications in China, ICCC, Qingdao, China*, pages 1–5, Oct 2017.

[90] Yin Yang. Linbft: Linear-communication byzantine fault tolerance for public blockchains. *CoRR*, abs/1807.01829, 2018.

[91] Attila Altay Yavuz and Peng Ning. BAF: an efficient publicly verifiable secure audit logging scheme for distributed systems. In *Twenty-Fifth Annual Computer Security Applications Conference, ACSAC 2009, Honolulu, Hawaii, USA, 7-11 December 2009*, pages 219–228, 2009.

[92] Mian Zhang and Yuhong Ji. Blockchain for healthcare records: A data perspective. *PeerJ PrePrints*, 6:e26942, 2018.