Scaling Blockchain Applications with Pub/Sub

by

Yuxi Zhang

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science and Engineering
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Scaling Blockchain Applications with Pub/Sub

Yuxi Zhang

Master of Applied Science and Engineering

Graduate Department of Electrical and Computer Engineering

University of Toronto

2019

The world of distributed ledger technology(DLT) has been getting significant attention. Many new applications want to utilize the trustless model and the high consistency guarantees this technology grants. This thesis aims to identify the needs of various business use cases and explores how DLT technology can be adapted to suit these applications. We recognize existing initiatives to adopt certain DLTs with some applications, however, there still exists gaps between these initiatives and production grade solutions. We provide a list of applications that can utilize the features of DLT technology and propose an encompassing structure to facilitate the needs of these applications. Specifically, we focus on a DLT-based publish and subscribe system for a subset of the interested applications. We provide the design and rationale behind the technology selection and analyze the relationship between DLT technology and Pub/Sub.

Acknowledgements

I would like to thank the support from all my friends and family. I would like to thank Bogdan Scaunasu and Kaiwen Zhang for their valuable discussions and feedback on key ideas presented in this thesis. I would like to thank my lab mates Geoff Elliott and Fei Pan for their continued support during the writing of this thesis. My final thanks goes to Professor Hans-Arno Jacobsen, whose guidance has been invaluable throughout my time with the Middleware Systems Research Group.

Contents

Chapter 1

Introduction

1.1 Motivation

Blockchain technologies have recently captured considerable attention from academic and industrial communities [41, 33]. Emerging systems, also known as *distributed ledger technology*, are designed to provide strong consistency guarantee over time. Furthermore, these systems also display impressive resistance to data tampering. The tamper resiliency results from how data is stored in the system. Data is stored in the form of chained blocks which are also replicated and maintained by a network of peers [31]. Each block from those chained blocks serves as a fundamental storage unit that holds a certain amount of transaction data and additional encryption information. By combining the technology of cryptography and distributed systems, blockchain technology allows transaction processing without having a trusted third party among mutually mistrusting peers [52]. Therefore, blockchain holds potential in enabling large-scale applications involving easily accessible peers while a trusted third party plays a 'debasement' role, e.g., in an anonymous financial transaction system or a highly decentralized system where a trusted third party becomes a dominating factor of centralization. Furthermore, though the data in blockchain system can be anonymous and encrypted, they are publicly traceable by all the peers of the system, meaning that blockchain provides some level of *transparency* on the user data.

In order to process user requests in a correct and efficient way, many blockchain systems employ a monetary validation protocol. All the peers involved in the validation of a user request (a transaction in terms of blockchain) have the potential to receive some amount of financial reward if they meet the condition of the protocol. This intricate payment scheme is designed to encourage correct processing by non-malicious peers to achieve high levels of consistency and finality in the system. This is especially important since the validation protocol involves many peers and messages requires time to be transmitted throughout those peers, allowing for many potential failures to occur. The payment system is designed to not only make attacks expensive and make non-correct processing to be unprofitable. New systems based on this technology have not only been tested by businesses but also been proposed as test-beds for new applications ranging from health-care to distributed computing platforms [3, 8]. Therefore, the blockchain system can appear as on-demand processing power, allowing systems to scale using its computing resources.

Furthermore, recent blockchain system, such as [39] [73] show promise in extending the functionality of blockchain systems. These improvements not only include system throughput but also include additional types of processing that can be done on the blockchain. The prime example here is Ethereum, a blockchain-based platform that processes smart contracts based on a Turing complete language [26]. Applications that make use of this platform can execute more complicated operations and track progress through the public blockchain that backs this platform.

In summary, current blockchain technology allows us to build a large scale system with non-trusting peers. It also allows for the implementation of an incentive scheme to attract more processing power. During the study of features in blockchain systems, connections were drawn between blockchain and pub/sub. Pub/sub is a widely used distributed message delivering paradigm. Pub/sub, short for publish/subscribe, let the publishers, who produce messages, to send messages to subscribers, who consume messages, via a network of middlemen abstracted as brokers. The presence of brokers allows for decoupling in the message sending system. For example, subscribers and publishers do not needs to know each other in order for messages to be sent between the two, making the system easier to scale. However, pub/sub miss some key features that can cause concern for the users. The user often does not understand how their data is processed and used. We call this the *transparency* issue. Another way of describing the transparency problem is how data processing and message delivery is seen as a black box from the client's perspective. For most types of communication, this is acceptable. However, in cases where user data is involved, this can be a problem.

Some common examples of pub/sub include Facebook Wormhole, Yahoo Pulsar, and Apache Kafka[65, 43, 30]. While these applications are very robust and successful in providing fault tolerant content delivery, none of them allow users to track the processing of their data. This can bring about privacy concerns; a growing controversial topic since the release of the European General Data Protection Regulation (GDPR) [23]. In addition, introduction for monetizing of broker processing can also prove to be valuable. Using monetary incentives to ensure non-faulty execution of broker logic is a novel area that we believe can prove to be useful in achieving robust content delivery.

Therefore, we wanted a pub/sub system that provides transparency to the user so that they can track the usage and delivery of their own data. In order to achieve this, we decided to combine blockchain and pub/sub in an attempt to build a new system that achieves the goals mentioned above.

1.2 Problem Statement

In a system where there are a number of data producers, who generate data, and consumers, who are interested in a certain type of data, system scalability and the flexibility of network topology can be an issue since many message exchanges within the network can be concurrent and result in network congestion. This can be resolved by using the pub/sub paradigm, where the producers and consumers are decoupled and all the information are delivered through a group of brokers. However, current pub/sub system has several limitations in preserving the safety of user data. Safety is interpreted as the ability to ensure the user data is not modified, misused or discarded by the brokers. One possible solution is to provide a trusted third party. The problem then repeats itself for the third party in question. Therefore, we need to implement a mechanism that can introduce trust among mutual un-trusting peers. Ideally, This mechanism needs to provide the following features:

- *Transparency.* The owner of the data should be able to trace all the operations that have been executed by any broker on the data. Here, the ownership of a certain piece of data (an event in terms of pub/sub) is held by both the publisher who generate the event and all the subscribers that are interested in the event. For example, a journalist wants to make sure all the articles are delivered to the target clients, and a news subscriber needs to ensure all the interesting news are not filtered by the brokers. Since the brokers are decentralized, we also have to choose a decentralized storage strategy to hold the tracing of delivery history.

- *Immutability of tracing history.* While we have a storage strategy of the tracing history, we have to ensure that the history is trustworthy. No malicious brokers, publishers or subscribers should be able to change the history after the history is generated and validated.

- *Financial incentive mechanism.* To encourage third party processing power joining the system as brokers, we want a system that can accurately trace the compensation for the delivery of messages. This mechanism can also encourage brokers to work correctly. Furthermore, some events may be processed with a higher fee to achieve faster delivery.

After we studied blockchain, we believe we can meet our design goal by building a pub/-sub system on top of a blockchain system. We are particularly interested in Ethereum as it offers the ability to process more complicated logic. The blockchain system provides the transparency, immutability and financial encouragement mentioned in our goals while maintaining the decoupling properties of pub/sub. However, we can not directly store all the delivered data on the blockchain as blockchain systems have a prohibitively high storage cost. Ethereum is no differnt and storing data in Ethereum is one of the most expensive operations. A typical message in Ethereum only contains a few bytes such as a payment transaction. Therefore, we have to design a storage strategy that reduce the reliance on Ethereum storage.

1.3 Approach

We examine some common applications on blockchain systems to see how we can better improve our prototype. We also look to see potential applications that can leverage the properties described in our problem statement.

In order to demonstrate transparency and trust in a pub/sub system and address the challenges described above, we need to clarify context and the underlying assumptions. There are two parts to trusted execution, one is the execution logic and the other one is the execution environment [18].

In blockchain systems, the level of confidence after the threshold time is very close to that required of trusted execution. We will design around these constraints while designing how pub/sub can be facilitated in the blockchain.

To bring down the cost of running storage and complicated network logic, we look for alternative message passing and storage solutions.

1.4 Contribution

In this work, blockchain properties are analyzed from several blockchain use cases. We identify several features that can be valuable in pub/sub and examine how they can be achieved. We also present several use cases that can leverage these features on top of the pub/sub functionality.

This thesis outlines the design and evaluation of BlockPub, a system designed to facilitate information dissemination between mutually un-trusting peers that alleviate the traditional pains of Blockchain based applications while preserving the benefits of the pub/sub paradigm. BlockPub comes with an intrinsic scaling scheme that encourages new content generators to join the network, making the system self-sustaining. This method of scaling is new to pub/sub, and opens up a new design pattern for future pub/sub systems.

1.5 Organization

The next chapter covers the nomenclature used in this thesis. Chapter 3 describes the current state of the art in blockchain research. Chapter 4 analyzes a list of blockchain use cases while motivating the design of BlockPub. Chapter 5 describes BlockPub design and implementation.

Chapter 6 presents our the evaluation of the BlockPub system and lessons learned. Future work will be outlined in Chapter 7 and we conclude in Chapter 8.

Chapter 2

Background

In this chapter, we review core concepts related to blockchain technologies and publish and subscribe (pub/sub) paradigm. In particular, we define key terms which will be employed throughout the rest of the thesis.

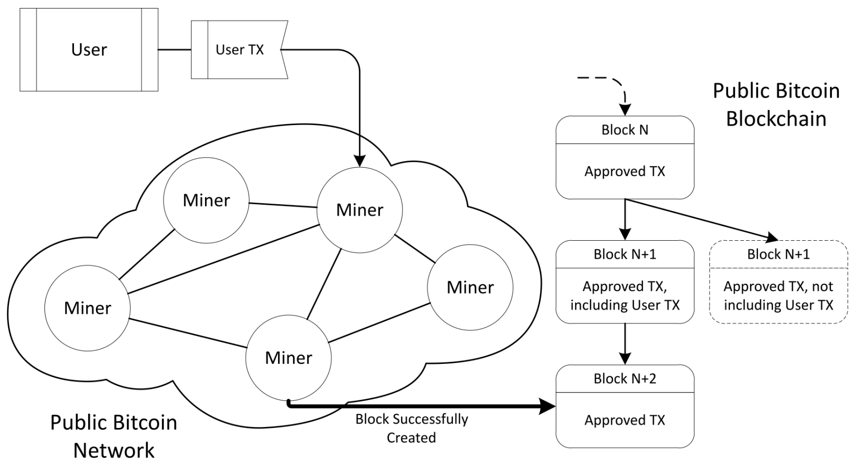
User
User TX
Public Bitcoin Blockchain
Block N
Approved TX
Miner
Miner
Miner
Miner
Miner
Block N+1
Approved TX, including User TX
Block N+1
Approved TX, not including User TX
Block N+2
Approved TX
Block Successfully Created
Public Bitcoin Network

Figure 2.1: Bitcoin Architecture

2.1 Key Terms in Blockchain

A *distributed ledger* is a collection of records (commonly financial transactions) which is recorded as a log. Incoming data is appended at the end of the ledger. The ledger is replicated across multiple nodes, where each maintains a consistent copy of the data.

A distributed ledger consists of three major components: the *data structure*, the *network*, and the *consensus protocol*. Various alternatives are possible for each component. These components depend heavily on the DLT used.Therefore matching applications with the right DLT is often the first step in designing a DLT application.

Broadly speaking, we can categorize distributed ledgers as either *public* or *private*. In a public ledger, anyone is free to join the network as a peer and manipulate the ledger. Conversely, a private ledger restricts write access to a subset of nodes (validators). From a technical point of view, the major difference between both types of ledgers concerns the assumption made about validators. In a public ledger with no trust amongst peers, each node is capable of behaving arbitrarily, hence additional measures are required to tolerate malicious attackers. On the other hand, a private ledger assumes some level of trust between validators, which means the underlying failure model is weaker.

2.2 Base Components

We now describe in detail two out of the three components comprising distributed ledgers: the *data structure*, the *network*. The last component, which is the *consensus protocol*, is described in greater detail in the following chapters.

Data structure - The most commonly used data structure for distributed ledgers is the *blockchain*. Each block contains a set of records added to the ledger and is immutable once generated. A newly created block is inserted in the blockchain by linking it to the last block in the chain. To prevent tampering of information found in existing blocks, the integrity of each block can be verified using a hash function which takes into consideration all preceding blocks. Therefore, in order to successfully alter an older block, one must also modify all subsequent blocks, which is designed to be unfeasible or unlikely. This implies that the amount of trust in the information contained in a block depends on the block age (i.e., the number of blocks following it).

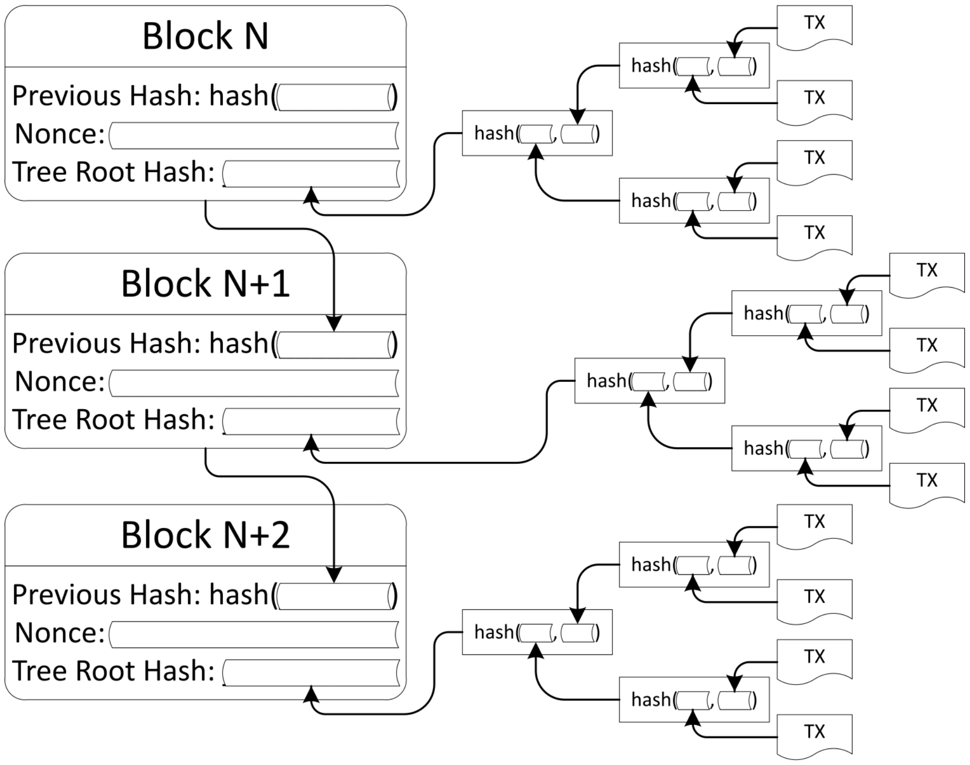

Figure 2.2: Blockchain Data Structure

Due to its strong guarantees, blockchains are the most prevalent form of storage for ledgers. For the remainder of the thesis, we will focus exclusively on blockchain-based ledgers. Figure 2.2 shows an example blockchain data structure as used by bitcoin. Records within each block are arranged to form a Merkle tree (i.e., the hash value of a parent node is based on the hash value of its children), with the tree root hash value included in the block information. Each block also contains the hash of the previous block in order to preserve immutability. Finally, the nonce value is generated depending on the consensus algorithm used to validate the block (e.g., Proof-of-Work, see below). The nonce can be thought of as the correctness guarantee for the creation of the block.

Network - In order to exchange information about the ledger, peers communicate using a network. In general, an unstructured mesh network is employed, such that messages are disseminated through multiple hops. For instance, bitcoin employs a *peer-to-peer* P2P network similar to Gnutella [22].

2.2.1 Consensus

The consensus problem is one that allow for multiple processes to come to an agreement under the presence of faulty programs. A consensus algorithm is required to maintain the ledger consistent across all the peers. It is precisely this component which is the most varied and studied [58, 10]. We can categorize the consensus algorithm as being *proof-based* or *non-proof-based.*

The original blockchain consensus algorithm, pioneered by Satoshi Nakamoto for the bitcoin system, employs a mechanism called *Proof-of-Work* (PoW) to enforce data immutability [53]. In order to insert a new block to the chain, a computational puzzle must be solved which takes as input the entire blockchain. In addition, PoW is coupled with the longest chain policy (also known as *Nakamoto consensus*), where non-faulty peers are always operating on the longest known branch. These two mechanisms together enforce the immutability of the blockchain data, as it takes an attacker a large volume of computational resources (i.e., more than 51% of the entire network) in order to alter existing data and rewrite all blocks following it to generate consistent proofs. Figure 2.1 illustrates the bitcoin system. Transactions are first submitted by users, which are then pooled into blocks. The blocks are appended to the blockchain one by one after they have been validated. Due to concurrency and network delays, multiple blocks with a valid PoW (i.e., two blocks $N+1$) may be generated based on the current blockchain (i.e., a fork starting at block N). However, the longest chain policy will eventually force all peers to converge on the state of the blockchain and choose one block and reject all other candidates.

Due to the immense computational costs, which are economically and environmentally prohibitive, many alternatives have been proposed to PoW. One example is Proof-of-Stake (PoS), e.g., PeerCoin [40]), which requires participants to commit a share of the digital currency in order to forge new blocks. This approach substantially reduces the computational efforts required to preserve safety.

Public ledgers also require an incentive system to encourage peers to join the network and to keep the blockchain up-to-date with incoming transactions. In bitcoin, this is accomplished by rewarding the peer (also called miner) who successfully adds a block to the blockchain. This block is used to collect transactions. In addition to this block reward, a miner is also allowed to collect all fees associated to the transactions in the new block as the block is appended to the main ledger.

In private ledgers, where a certain level of trust is maintained amongst peers, non-proof-based consensus algorithms are commonly employed. For instance, Hyperledger employs a practical Byzantine fault tolerance method (PBFT) to determine the order of incoming transactions [5]. Peers are able to elect a trustworthy leader which can determine the sequence of transactions to be applied to the blockchain. This does not require any computational puzzles to be solved and has the advantage of allowing modifications in the existing blockchain if nec-

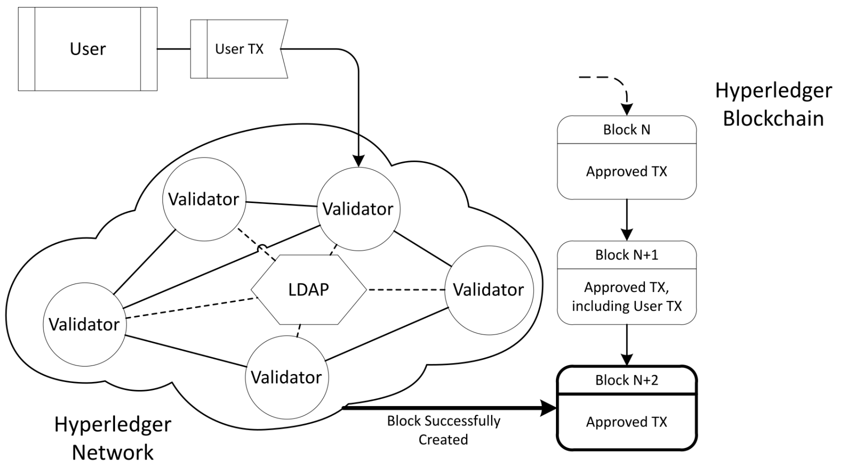

Figure 2.3: Hyperledger Structure

	Public Ledger	Private Ledger
Proof-of-Work	Bitcoin	—
Proof-of-Stake	Peercoin	—
Non-Proof	Ripple	Hyperledger

Table 2.1: Taxonomy of blockchain-based services

essary. Figure 2.3 shows the architecture of Hyperledger, an open-source project intended for private blockchains. The verifiers insert incoming transactions by running a PBFT consensus protocol to decide their order in the blockchain.

Also note that non-proof-based public ledgers do exist, such as the Ripple system [12]. Table 2.1 shows example systems as categorized by their consensus mechanism and their visibility scope. To the best of our knowledge, there does not exist private PoW- or PoS-based systems, since there is no incentive to employ such heavy-handed measures when trust is assumed amongst peers.

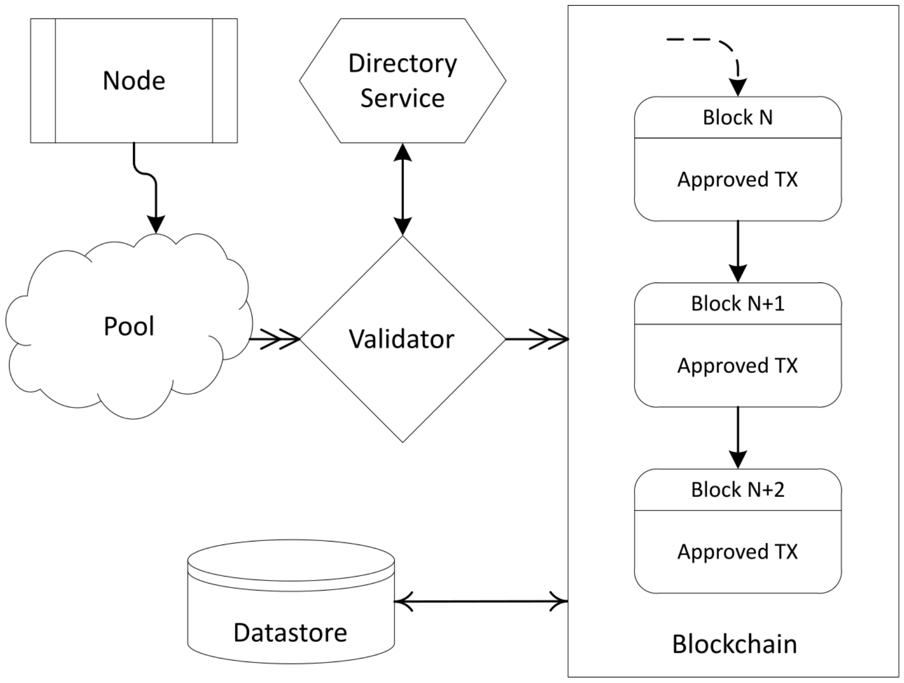

Figure 2.4: Proposed Middleware Structure

2.2.2 User Nodes

The users participate by generating transactions within the context of the application. These transactions are defined by the purpose of the application in question; in bitcoin, for example, the nodes produce transactions that are logged in the public blockchain after the validation and consensus process.

2.2.3 Transaction Pool

In existing distributed ledger applications, not all transactions are processed the moment they are submitted. In fact, in public blockchains, there exists competition between different transactions as the reward for processing them may be different. It is not always the case that a transaction will be processed and stored in the current block after a submission. The transaction may take several minutes before it manifests on the main chain; this is why the transaction pool exists. In private ledgers, a mechanism can be implemented to prioritize old transactions in the pool so a certain service level agreement is reached. For example, the application can ensure that no transaction remains outstanding in the transaction pool for more than 2 minutes. In public ledgers/blockchains, old transactions that do not offer sufficient incentive for the validators may experience significant delay before they are manifested in the main chain. This is a property not commonly seen in conventional middleware systems. This challenge is instead left to the developer of dencentralized applications(Dapps) to solve. One guarantee that is maintained in all the applications mentioned above is the ordering of transactions and the guarantee that no *double-spending* occurs [60].

2.2.4 Validators

Validators ensure that the transactions are correct and run consensus protocols amongst each other to achieve eventual consistency [15]. This process is independent from the transaction generation process. Regular nodes could also run the validation process in parallel if permitted to do so (some times validation requires permissions). In this model, validators could be either trusted or untrusted depending on the type of distributed ledgers in question. By trusted, we mean that the validator cannot commit Byzantine failures. Byzantine failure refers to arbitrary behavior and not the generalized crash failure model [1, 56]. The type of ledger also dictates the failure model for these validators.

PoW Public Ledgers - Nodes are not trusted in public blockchain applications. Instead they are incentivized via some form of payment to validate properly. By validating, they get paid in the form of digital currency. The correctness of this system is analyzed in past works [51]. Without proper validation, the transactions logged in the ledger cannot be trusted and the application would be unable to provide trust and generate value. How to assign monetary incentives to validators is left to the developers. Compared to the non-PoW ledger applications, implementing an incentive scheme to discourage malicious behavior is much more difficult. This is not a simple problem and conventional systems like bitcoin and Ethereum both have state of the art solutions. In PoW ledgers, digital currency is awarded per validation and is sent to the validation peer after correctly performing a validation. To further promote correct validations, Ethereum applies a mechanism that takes away the peer's *ether* (crypto currency) when malicious behavior is detected. The social economic model is incomplete without knowing the cost of the validation process. To make use of these mechanisms, one needs to make sure that the validator is given sufficient incentive to validate accurately. Another requirement is that the validator is properly discouraged from arbitrarily releasing this information via a punishment scheme. The lack of a well defined attacker model in this case is concerning as the validation scheme is application dependent. If an application is to make use of the fact that validation can be done by arbitrary nodes in a PoW ledger, then the implementation needs to ensure that the validator has sufficient stake in executing properly. A monitoring process also needs to be designed such that the execution is monitored in some fashion. For instance, given a set of instructions that validators need to execute, two validators can run this set of code with a third party supervising the execution and performing conflict resolution between two different results. This is similar to what bitcoin does for transaction processing.

Non-PoW Ledgers - Typically, validators are trusted in non-PoW ledgers. Validators in the application are either run by a trusted entity in the case of private ledgers, or identified by a service in public ledgers.

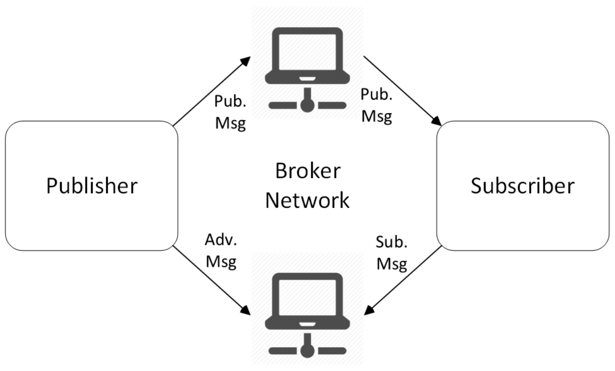

Figure 2.5: Publish and Subscribe Paradigm

2.3 Publish and Subscribe

Conventional topic based Pub/Sub paradigm consists of three main types of entities: publishers, subscribers and brokers as shown in 2.5. The role of the publisher is to provide content while the subscriber consumes the content. The intermediate matching and point to point message delivery process is logically centralized and executed by brokers. The publishers notify the brokers through advertisement messages about the topics they want to publish on. The subscribers notify the brokers the topics they are interested in through subscription messages. The delivery of content is depicted as a publication message that gets propagated from the publisher to the subscriber. An example application of a topic based Pub/Sub system is Facebook [65]. In this case the publishers are users of the Facebook service and the publications the events generated by the user. The subscribers are friends with the publisher within the Facebook app. Each event is categorized by means of a topic, represented by a string or int. Whenever a message is sent out, the brokers deliver the message identified by the serialized topic identifier to the subscribers. Typically the brokers form into a topology to deliver messages more efficiently. For the rest of the paper, conventional Pub/Sub systems refers to systems adopting this paradigm and running the brokers on the assumption that they can only commit crash failures and not byzantine failures.

2.3.1 Fault Tolerance

Types of failures in distributed system can be categorized into two groups.

General Omission Failure - General omission failure occurs when one node in the network fails to send or receive a message [61]. It also include crash failures. Crash failures occur when a node crashes and stops sending and receiving messages altogether. These types of failures are generally considered to be benign in distributed systems.

Byzantine Failure - Byzantine failures occur in a distributed system when a node behaves differently when sending and receiving messages. For example, selectively choosing to respond to certain nodes' messages in the network. This type of behavior doesn't necessarily result from malicious intent and can also occur as a result of network instability [1].

In conventional Pub/Sub systems, broker byzantine faults are not tolerated[21]. This means that the broker isn't allowed to manipulate information and send conflicting information to different brokers.

Chapter 3

Related Work

This chapter outlines the current state of the art in blockchain and pub/sub research.

3.1

3.2 Applications Using Blockchain

We leverage the properties offered by a public blockchain for our BlockPub implementation. This idea was heavily motivated by other applications leveraging similar properties. Prominent examples include decentralized applications implemented on the Ethereum platform. Similar business logic has been implemented in the DAO initiative [4]. Although it was later found that logical errors in the implementation caused huge financial repercussions, showcasing some of the drawbacks of this system. Thus, such applications often requires advanced analytic of attacker and failure models.

3.3 Trusted Execution

Users need to be able to trust the execution environment in order to achieve transparency. Without trusting the underlying processing environment and processing pipeline, validating the processing results comes back to trusting the processed output. A robust way transparency can be achieved is by allowing users the ability to check the code logic and to validate the execution environment. The two conditions respectively correspond to open source code and verified execution environments like Intel SGX [13, 35].

3.4 Identity Protection

As entioned in the vision works, one of the main challenges that is not addressed is the identity pilfering problem and achieving pseudoanonymity [59]. Protocols still need to be developed for applications to use in order to preserve the anonymity of the users. This is especially true for business logic and sensitive information processing.

3.4.1 Pub/Sub

For a long time, large scale publish and subscribe (pub/sub) systems have been supported by middleware that runs on proprietary hardware while allowing for crash failures. Examples include Facebook Wormhole, Yahoo Pulsar, and Apache Kafka[65, 43, 30].

In addition, many works focus on optimization and processing of message sending within the topic based publish and subscribe model. The topic based model stem from a long history of messaging applications and information systems [48]. Our design of BlockPub adheres to this paradigm and reference designs like PADRES and Kafka [37, 43].

In processing logic design and implementation we heavily borrow from large scale applications reliant on a logically centralized processing unit such as Wormhole. The logically centralized processing unit that allow for high consistency in our design reference data structure and storage format in applications such as Zookeeper [34].

3.4.2 DHT

Plenty of research have gone into the implementation and optimization of distributed hash tables (DHT). We modified the DHT for our to cater to our needs. To minimize the look-up messages required for our modified implementation we heavily borrow from optimizations done on the Cassandra DHT [45].

Chapter 4

Use Cases

One of the biggest concerns of blockchain technology is the use cases for it under existing performance conditions. This chapter of the thesis aims to systematically analyze a number of blockchain applications and their key points. Several works have presented their views on the list of applications that blockchain technology can support. We look over 20 individual use cases and identify the work-flows they require in association with the business logic backing the use case.

4.1 Classifying Use Cases

There are many different ways of categorizing different types of blockchain applications. The standard described in the Swan book categorizes the different types of applications as follows [69].

4.1.1 Blockchain 1.0

In spite of the flurry of other applications leveraging blockchain technology for a variety of purposes, the largest market for blockchain-based services is still centered around crypto-currencies.

Blockchain 1.0 represents the use of PoW distributed ledger in crypto-currency applications. As of now, bitcoin still holds the highest market net worth amongst all the other crypto-currencies. The following describes alternative crypto-currency systems considered part of Blockchain 1.0.

Ripple - Another major crypto-currency is the one used in the Ripple application. Ripple is used to facilitate high volumes of transaction processing. In financial applications, it is often desirable to be able to verify at a later time that a transaction has been processed. This is why Ripple uses a non-PoW, publicly available distributed ledger to store processed transactions. Without PoW, validator nodes are assumed to always execute correctly. Dependency between transaction blocks is maintained by the Merkle tree data structure [70]. The Ripple application makes its ledger publicly available, where transactions are displayed using digitized signatures instead of the actual IP addresses. This scheme allows for validators to run their own validation process to ensure that the main ledger is correct and can also outsource validation processing to third parties. A registration service is used within Ripple to maintain a list of trusted validators [12].

4.1.2 Blockchain 2.0

The next step in the blockchain vision is executing Turing-complete code and logging the state of the execution on the distributed ledger. The ability to do this is known as Blockchain 2.0. Unlike Blockchain 1.0, where we observe standalone applications, Blockchain 2.0 is more akin to a platform with the ability to execute arbitrary code; for example, a platform capable of processing and validating business processing languages like BPEL [55]. There are several platforms aiming to support these operations, such as Ethereum. We use the term *smart contracts* to refer to the processing and logging of arbitrary operations. We introduce two main platforms that support smart contracts and outline their features below.

Hyperledger - Hyperledger takes the approach of abolishing PoW altogether and using only trusted nodes during the validation process [5]. Making this change results in the creation of non-PoW, private ledgers. Unlike bitcoin, the distributed ledger is not openly available and is only shared within a trusted network identified by a registration service. Private ledgers have a very similar application structure as conventional transaction processing applications. Much like how transactions on the VISA network are validated in a logically centralized manner, Hyperledger transactions are validated the same way [11]. This validation process takes advantage of distributed consensus to maintain strict ordering of transactions within the ledger. The ledger not only maintains the strict ordering of transactions, it is also an immutable object shared amongst multiple validator nodes. Another challenge Hyperledger addresses is tackling environment heterogeneity. To solve this problem, Hyperledger runs validation nodes in docker containers [72].

Ethereum - Ethereum is built off of a similar application structure as bitcoin and also functions on a PoW-based public ledger [6]. The idea is to allow any decentralized application to be able to execute on the Ethereum platform. The implementation of such a system is also more complicated in comparison to the original bitcoin framework. To handle arbitrary operations, Ethereum uses a Turing-complete scripting language called Solidity [26]. Solidity is used to write smart contracts that are in turn executed on the Ethereum Virtual Machine (EVM). One novel addition that is made by the Ethereum project is the use of the Patricia trie, a tightly linked data structure coupled to the Merkle tree [6]. However, instead of storing hashes, the trie stores transaction states of current transactions. The value of this adaptation is shown in allowing for partial validations to take place, as the root of the trie represents a past state in time with a high level of guarantee. This means that instead of performing validation to the start of the ledger, nodes only need to validate until the root of the trie. Validation within Ethereum is less memory intensive compared to bitcoin, as the full history of the ledger is not needed during this process. Like bitcoin, new nodes can join the validation process and earn *ether*, Ethereum's own crypto-currency, in return. Ethereum is also the main proponent of an alternative to Proof-of-Work called Proof-of-Stake (PoS). PoS is a less computationally expensive procedure for block creation (mining). The correctness of this alternative is still being analyzed.

4.1.3 Blockchain 3.0

Despite the versatility of smart contracts, applications sometimes require more than just economic incentives to ensure the functionality of the application. These applications aim to provide services beyond currency and markets. They focus on government services and other national interests. The implementation of these services using blockchain technology, is known as Blockchain 3.0.

4.2 Use Case Analysis

However as applications become more complicated, the categories do not sufficiently distinguish between the types of use cases. To get a better understanding of the types of applications we came up with 20 use cases and analyze them. We notice that many of these applications share common components that fulfill similar roles.

We categorize the applications by their application types. We begin by pointing out several financial applications that can leverage the benefits of a private blockchain. One of the recurring themes in financial applications of blockchain is using it for settlement purposes. The problem of settlement exists in several different scenarios. Of these, one of the longest settlement times exists in international money transfers. This process takes longer than the message sending in the Swift network due to the additional validity checks that takes place after the transaction has been received. Here the benefit of moving this procedure onto a distributed ledger has two distinct benefits. One is the traceability of the settlement transactions. The distributed nature of the system allows for multi-party verification of the settlements. The second major benefit results from the many processes that lack automation in the banking industry. In the banking industry, availability and stability are paramount. Given such, old systems still play a core role in their services. As such, new methods of automation that guarantee data consistency closely fits banking philosophy and methodology. Furthermore, financial institutions have a plenitude of different types of transactions, allowing for a variety of blockchain adaptations based on the current technological progress. In this regard the financial institutions, especially banks, are unique. The classification of these use cases are primarily settlement based between different stakeholders. Below we will discuss the different uses cases for financial institutions.

4.2.1 Tax Credit Tracking

Settlements exist on different levels when it comes to financial institutions. In this particular case there are many complications during the process associated with filing taxes. One crucial aspect of this is settling the taxes and paying them. This is where the blockchain comes into play and can offer value. The tax filing process does not have to happen on chain. For this use case, the blockchain component can play several different roles. One of them is settlement and tracking of the payment process; another is the payment channel. The details associated to registering and the intrinsic logic to the tax filing process can be a bonus to the settlement process. However, this requires integration of the software used to file taxes into the blockchain system. Filing and paying for taxes requires identity protection and as a method of payment, could be easily be put into a private blockchain with private accounts. This is particularly useful for government tax collection as settlements do not have to be immediate while also saving a lot of paperwork and manpower during the process. However there are numerous considerations for this use case.

Interface - The interface to such a system is important. Each user who keeps track of their own taxes can have a client with it's own public key. With taxes being tied to an individual or business in real life, this system relies on an identity service to keep track of the individual accounts. Thus the interface includes a translation process of the person's real life identity to that of the blockchain system.

Tax Token - The concept of using blockchain for tax purposes is not new. In fact there is a tax token aimed at tackling this very problem [7]. However, there are several distinctions between their use case and the settlement one described above. In the first case, the tax token focuses on filing taxes for crypto-currency gains. They do this by logging the transactions of a particular digital wallet.

In the case of tax collection and settlement, the system has a higher focus on keeping track of the tax remittance and payment. To do this, a token can be introduced for the tracking process. The transactions can document the various interest payments while allowing for individuals to check their own tax payment history.

Benefits - Several benefits can be associated with a blockchain based tax settlement approach. One of the obvious ones is the tracking and transparency of the tax system. Another is the intrinsic automation of the tax collection process. For government systems, there is often a centralized server for the access control for personal accounts.

4.2.2 International Money Transfer

International money transfers historically require intermediaries and a lot of paperwork. Settlement details can be very complex and take several weeks. Automation and transparency of such systems are hard to maintain due to the complicated multi-party involvement during the settlement process. Similar to the tax settlement case, this use case does not require real-time settlement. However the access pattern is different in this use case. Instead of individuals accessing their own accounts, international bank transfers requires specific read/write access implementation. One of the key factors in doing this is running a service for writing access. This brings about more changes in the implementation process.

Throughput - Some existing services already handle this use case. One of the most well known is Ripple. In recent years there are several competing platforms that cater to other clients in a similar fashion. Of these are emerging names like Red Belly and Hash Graph. These platforms offer trusted private chain services to businesses for settlement.

Digital Identity - For international settlements, the peers are composed of trusted entities with real life stakes in the settlement process. In this regard it is worth it to initially validate the real life identity and correlate to the digital identity used in the settlement process. This is the approach used by Ripple when allowing for validators to join their network [12]. However, it is worth mentioning that there are systems that employed this approach in the past. Bitcoin green addresses ended on a notorious note, but the idea was to associate digital identity with real identity. In a fully decentralized system with no central authority, it is difficult to punish misconduct. This became the downfall of green addresses in bitcoin. This concept is leveraged extensively in private chains such as Hyperledger.

Bank Accounts vs. Business - There are a couple of distinctions to be made between small amounts of international money transfer between bank accounts versus businesses transferring large sums of money. One of the main differences resides in the verification time period. Large transaction amounts have a tendency to go through more paperwork and validation to prevent fraud. One example of this is international checks for large portion of exports. In some cases the businesses requiring the good do not necessarily have the cash flow to pay up front and have to go through a lot of the paperwork required to make the deal happen. Such settlements involves more paperwork and is a lot harder to automate.

4.2.3 Cross Bank Settlements

This use case has been explored by multiple banks in various countries such as the Bank of Canada and the the National Monetary Association of Singapore[54]. End of day settlements involve multiple parties settling their temporary loans to each other based on the fluidity of their current assets and central bank's loaning policy. In the use case of central bank settlement, the central bank coordinates the settlement process and processes the IOU (I owe you) [38]. The IOUs are placeholders for each bank in the network to facilitate end of day settlements. There are several competing platforms for the cross settlement use case. Two of which we mentioned already in the initial introduction to categorizing blockchain use cases, namely Ripple and Hyperledger. The last one is Corda, a pseudo blockchain platform.

Corda - Corda caters to businesses as a distributed database. The way it works is similar to Hyperledger in the sense it has a central access control component. Corda handles settlements and resolve any conflicts. Like Ripple it can retroactively reconcile any inconsistencies for past transactions. However it is interesting to note that Corda does not use a blockchain; the transactions are not kept in blocks. It is closer to a distributed log for business applications.

4.2.4 Blockchain Market Place

A novel idea that bounced around since the initial days of Decentralized Autonomous Organizations is the Blockchain Market Place. Using blockchain to facilitate an open marketplace with micro transactions from peer to peer get rids of the hindrance of paper checks when it comes to settling transactions. The hopes of automating this process using blockchain and smart contracts still exists today. In practice this becomes a lot more complicated since the inclusion of the brokers in a blockchain system requires a lot of additional functionality to be added to support a blockchain based market place. The focus here is still on the settlement problem. In a lot of marketplaces, common pain spots of pain include the matching between seller and buyer, facilitating settlement between the peers.

Matching Service - For a market place application, the matching process is key and it can be a pain point for many existing solutions. There can be a wide variety of approaches to match the goods and services to the consumer. One example of this can be employing a broker facilitated transactions scheme between seller and buyer. The broker in this case exists as a physical entity that associate interested buyers with sellers. Given the nature of this interaction, it can be benefitial if the matching and settlement occur digitally to optimize and automate the process.

Comparison to Conventional Market Place Applications - One of the biggest questions is whether or not a blockchain is a viable platform compared to existing tyrants like Amazon and Alibaba. These existing platforms embrace centralized business logic that also include the delivery and distribution of the sold goods. In a blockchain based market place with no central service provider, this scheme will have to be adjusted. Providing the distribution model within the decentralized model can be very complicated because the fulfillment and distribution services need to have a decentralized method of tracking for a service that has historically been run on proprietary processing platforms.

Settlement Disputes - Another issue is the problem of dealing with malicious behavior in the decentralized marketplace. One example that we can draw parallel to is bad Airbnb tenants, wherein the complaints would be directed towards the service provider. In the market place, how misbehavior is discovered, punished or even verified can be complicated. In a blockchain system, one possible solution is having a voting protocol. This will be covered in later chapters.

As blockchain systems advances in performance and popularity, more applications started to consider it as a viable alternative method to provision services that include even storage services. This idea can be counter intuitive, because storage is one of the most expensive operations in POW blockchain based systems. However, with the high consistency guarantee, it could prove to be invaluable as an access control database for large scale applications.

4.2.5 Healthcare Document Storage

Common agreement is that sensitive information should not be stored on the public blockchain. In the health care storage use case, the information is often proprietary and should not be accessible by third parties. Patient related information such as health records, prescriptions, and diagnosis history are all part of the information that needs to readily accessible by parties with the proper permissions. Different from normal file storage, these files require strict access control while varying drastically in size. The size differential for files results from the different types of files on storage, ranging from high definition scans to issued prescriptions. As a caveat to this problem, there may be multiple storage systems catering to the different types of files.

Access Control - In comparison to the financial settlement use cases mentioned above, access control for patient related data is paramount. Misuse of medical information directly impacts the patient's daily life. In this regard none of the information should be able to be retrieved from the various data storage systems by other parties. Having this component is comparable to any typical database. However one challenge for a massive-scale medical data storage is the availability of the access control component. The consistency guarantee of this component is also very important so as to not allow parties who no longer have permission get access to the sensitive data. Zookeeper with proper scaling is an example of how this can be done. Alternative solutions like ETCD and LDAP can also achieve this.

Availability - Another crucial element of this system is availability. Accessing health records by clients is a part of the main functionality. Based on the constraints however, access control can often times become the bottleneck. Replication being intrinsic to blockchain peers, availability of the service should be high. One of the more interesting problems is how availability can be maintained with a logically centralized access control component.

4.3 Pub/Sub Use Cases

Far from settlements and access control are applications that truly leverage the guarantee blockchain offers. We see in these use cases about how publish and subscribe models can come into play.

Take the common example of Facebook. In this case the publishers are users of the Facebook service and the publications are the events generated by the users. The subscribers are friends with the publisher within the Facebook app. Each event is categorized by means of a topic, represented by a string or integer. Whenever a message is sent out, the brokers deliver the message identified by the serialized topic identifier to the subscribers. Typically the brokers form into a topology to deliver messages more efficiently. For the rest of the paper, conventional Pub/Sub systems refers to systems adopting this paradigm and running the brokers on the assumption that they can only commit crash failures and not byzantine failures. In conventional

Pub/Sub systems, broker byzantine faults are not tolerated[21]. This means that the broker isn't allowed to manipulate information and send conflicting information to different brokers. As most of these conventional systems run on privately owned machines, this is not a concern. However, a new business model has proven to be successful over the years, which relies less on offering services running on privately own machines and more on third party computing power. Cloud computing embraces this very philosophy. This same business model also applies to other applications that have proven to be profitable. Uber and Airbnb adopted this business philosophy and offer services indirectly by connecting service providers to users through their own system while enforcing a specific set of regulations to penalize misbehavior of users and third party service providers.

We noticed that many messages in Pub/Sub systems can be associated with monetary compensation and could be adopted to the aforementioned business model. For example, Reuters news subscription service delivers news to subscribers for a registration fee. Another example is YouTube views. Video creators get paid based on the number of views their video gets. These systems run on proprietary servers and are scaled using conventional scaling techniques, such as horizontal and vertical scaling (replication and resource allocation) [14]. While scaling these systems has the benefit of having low computation overhead in comparison to the blockchain-based approach, the drawbacks are also apparent. First, processing of these transactions isn't necessarily transparent from the user's perspective. Second, having a logically centralized monitoring service for scaling allows for data gathering of the users with or without consent of the user [34]. Third, conventional scaling techniques require owning or leasing physical assets, limiting the rate of scaling the application to that of the purchasing power of the proprietary entity. An additional advantage to using a non-logically centralized system is that the system is resilient to censorship from the service provider.

4.3.1 News Dissemination Network

One prominent example of how blockchain can be applied is the news network. Reuters deliver news to subscribers with content relevant to their interests. The system functions with traditional pub/sub entities. More importantly, the delivery and accuracy of the news warrants tracking of the source and date of the report. The consistency and immutability aspects of blockchain come into play when vouching for the source and originality of the content. News dissemination as a distributed application also requires high availability and network partition tolerance. The partition tolerance is important because of the geological location differences of the news receiving clients. Classically, localized gossip protocols deal with network partition[2]. Having a blockchain based system intrinsically offers availability and gossip protocols. Coupled with the added trust and tracking of the blockchain system, the prospects are very good for adapatation.

4.3.2 Data and Information Marketplace

As data becomes more and more integral to daily convenience, we envision data exchanges to happen more frequently. Consequently, data integrity during and after the transfer become matters of paramount importance. Recent blockchain use cases often focus on how data is stored in the blockchain. In this use case, the focus is on the sensitivity and reliability of the data exchanged. In handling data, trusted execution and secure computation environment are required. Having a private blockchain process data as a service allows the data processor to take direct validation schemes from the data provider. Having this service is very beneficial in ways more than one. The first is having a mutually trusted information transfer channel. The second is having specified requirements and validation scheme based on real world restrictions defined in code that gets followed through during the process. In this fashion, the transparency and correct execution of code logic in the system offered by the underlying blockchain structure can be leveraged to great effect.

Chapter 5

System Design and Components

Within the current constraints of blockchain technology, applications can utilize the technology as a logically centralized coordination service. One limitation that is imposed on conventional public blockchains is the time required to reach certainty [27]. This limitation has been reduced by increasing the block generation rate, but still deters applications requiring high levels of consistency.

The second limitation is throughput. As previously mentioned, throughput is still very limiting to many applications. It remains one of the main problems in both public and private ledger systems alike.

The third concern for applications is the existence of secure processing enviornments. In many cases, having black-box processing can be an advantage in building distributed applications [42]. A method of achieving this lies in using the Intel SGX secure execution environment during the validation process [25, 35]. We considered the use of SGX for data protection but rejected its use in our model for the following reasons:

- It is cheaper to leverage current existing hardware owned by government organizations when implementing applications for Blockchain 3.0 [18]

- SGX is still a new technology with limited public adoption, limiting its application within public ledger systems
- The use of SGX adds additional computational overhead on top of existing performance issues [13]

We believe that in a permissioned blockchain system, the verification is sufficiently secure and the data stored could be isolated from context for processing purposes. In the use cases where storage is even more sensitive, private communication channels and careful permission management could be used.

5.1 Integration with a Distributed Data Store

One of the most expensive operations when working with blockchains is the persistence of data. Data persisted on the ledger is present in every local copy the ledger, and thus consumes large amounts of memory when viewed from a global perspective. Given such, logic follows that the bulk of messaging and storage should not be done directly through the ledger. While there is merit in simplifying the application structure by storing data in the chain itself, for most applications it is not desired due to the induced high memory overhead.

Logically, there is a need to integrate the distributed ledger with a storage or message sending service, such as a pub/sub messaging application [20, 37, 44, 66]. Messages are passed between peers through the network, generating transactions, and the record is stored in the chain itself in the form of hashed values. Although trivial on the surface, there are multiple considerations in implementing such a “translation process”, from the transaction to the hash stored within the block itself. One interesting aspect of this problem is converting the content generated by the users to actual blockchain verifiable transactions. An example of this is storing the hash of the message that is sent. A validator signs off on this transaction by checking the corresponding hash values. Another interesting problem to be solved by the programmer is the consensus scheme. After the validation step has been completed, the validators need to come to consensus amongst each other to generate a block. Depending on the application and platform, this step varies significantly.

For PoW ledgers, the proper reward needs to given for the validation process. For non-PoW ledgers, the consensus algorithm depends on the expected failure rate of the hardware, difficulty of the validation process, and other factors. Depending on the specific model used, the overall performance can vary greatly. These problems should be known to the developer and the solution optimized based on the characteristics of the specific application.

Optimizing Around Constraints - Applications impose certain intrinsic constraints depending on the use case. Transaction-based systems have a requirement for the level of consistency guaranteed within the system. There may also be additional requirements in the form of throughput and availability. For applications with a high message throughput, multiple validation processes can be consolidated into one single entry on the blockchain. This technique can be useful in both private and public blockchain implementations. A good translation process will encompass the limitations of the application while optimizing for minimal storage within the blockchain itself. One example of this would be sacrificing the time it takes to perform a longer validation process and sign off on multiple transactions at a time pertaining to this validation process.

Sometimes it is not required that all nodes within the system keep a complete track record of the transaction history. Namely, the history of messages are not stored within the messaging system or the storage system. For instance, some nodes do not require that the entire Merkle tree is present in the validation process. Partial validation has been explored by Ethereum and this allows for more efficient processing of transactions. Thus, another component to file storage integration is the storage of the original content. In typical applications, it is not required that the entire history is stored. Here there can exist a concept of *revision*. Where a view of the storage system is stored and represents a state in the blockchain counterpart. This allows flexibility in making use of the storage system. This mimics a *repository* of sorts for information that is guaranteed to be immutable with versioning.

5.2 Access Control Component

For applications dealing with sensitive information or validators requiring non-Byzantine behavior patterns, an access control component is usually added. Such a component would allow for only a subset of trusted nodes to access relevant information. This component is usually used to ensure that sensitive information is not leaked. In the translation process, validators need access to the actual information. In private blockchains, this is easily doable since access control could be implemented at the participation level, i.e. all participants in the validation process are trusted. In the scenario where access is tiered and only a subset of all participating nodes have access to the history of the system, authentication can be added using public key encryption. This implementation resembles that of Active Directory [36].

5.3 Pub/Sub Components in Blockchain Setting

In our set of analyzed use cases, most of the above components are required for the application. In a way, the core functionality isn't unlike traditional database systems. In fact, several applications we mentioned above is associated with storage. However, with blockchain as the backbone consistency guarantee in a storage system, the focus is on making the system available . This is because in many of the use cases, the user base is often times relatively large. Using the example of the medicare, many users could access their files at the same time. Given the rules of CAP theorem, in this example we want to maximum consistency and availability [19]. The same can be said for a lot of use cases that cater to a large user base such as Amazon and Facebook. Given the similarities, there is much to explore on similar services that might consider blockchain as an acceptable technology to leverage. On a very low level, blockchain based systems and distributed system share the storage and consistency problems. Observing this, they aren't that different when both systems are viewed from the outside as black boxes. Each system focus on two types of actions, to store and retrieve. Scaling these actions for availability in regards to the user base require us to consider how the actions are executed. In particular we are interested in how we maintain consistency between the auxiliary storage system and the blockchain system.

In Figure 1 we outlined the the main procedures and components of pub/sub. In the new model, each component works slightly differently, which changes how message passing and handling is realized. Specifically, the correctness of the system requires a financial incentive scheme, such that each entity is bound by a specific payment scheme when processing data.

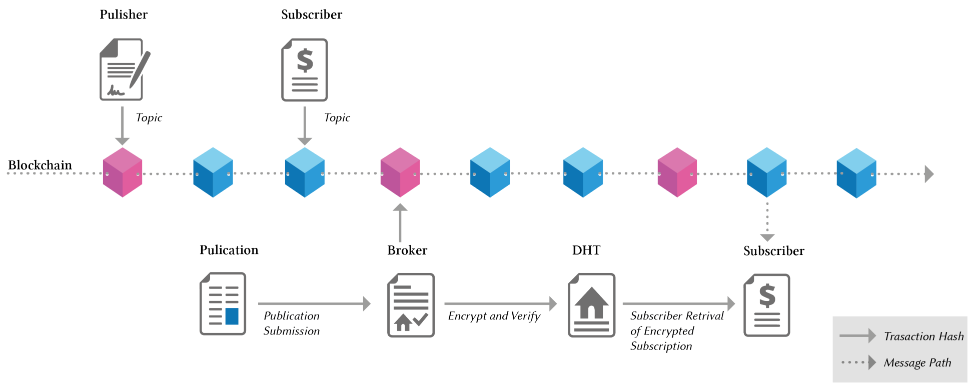

Figure 5.1: Pub/Sub Protocol

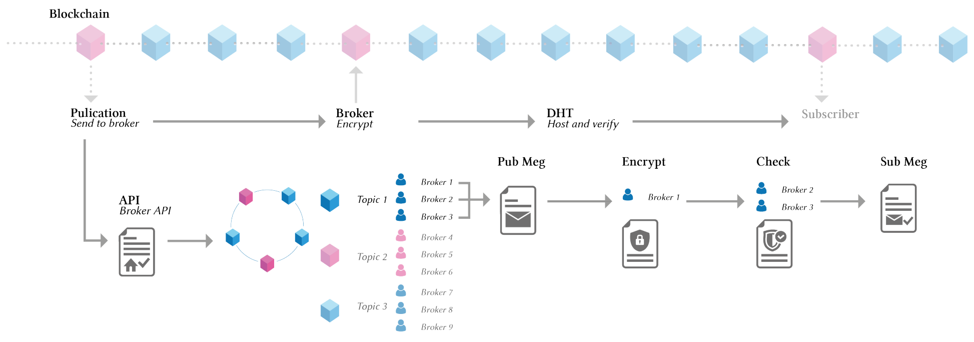

Figure 5.2: Broker Functionality

5.3.1 Blockchain Benefits

The benefits of using blockchain as a consistency guarantee is not immediately apparent. We found the fully distributed nature of the blockchain as a good fit for transparent processing. The idea of leveraging the blockchain as a processing bus is not new. Many applications aim to leverage Ethereum as a distributed log service. In our model however, the blockchain is used so that users can trust the execution while providing a tracking method for transactions. We chose the Ethereum blockchain for decentralized processing and tracking over other platforms. Ethereum is a public blockchain platform designed to offer fully decentralized processing. It also one of the earliest platforms to offer a Turing complete language set, allowing us to express complicated logic in the contract itself. Thus when we refer to blockchain in the figures and models, we are referencing the Ethereum blockchain.

5.3.2 Auxiliary Storage

Despite the recent improvements on the processing speed on public blockchain implementations, there remains a difference in orders of magnitude between the throughput achieved in the conventional VISA network and the current blockchain technologies [6, 28, 69]. We anticipate blockchain performance bottlenecks and design auxiliary storage systems in conjunction with the blockchain to achieve higher performance. In the proof of concept implementation, the brokers are organized into a distributed hash table (DHT) architecture. This does not have to be the case. We discuss several different auxiliary storage systems and their properties in Section 5. For simplicity purposes, we will refer to the storage system as a DHT implementation to describe how we handled message passing in a topic based pub/sub application.

5.3.3 Broker

To facilitate hosting of publication messages for each subscriber, BlockPub uses a network of brokers to host the publication messages. The brokers are organized into a DHT within the network [64]. To join the DHT, the brokers first register in the blockchain by invoking the join contract. They are then assigned a pseudorandom value that maps to a hash value in the DHT, achieving randomness in the DHT. Brokers find each other via the discovery protocols in the DHT and cross referencing the identity in the blockchain. The brokers forming this DHT are then split into several smaller DHTs, each handling a different topic. Brokers oversee each others work inside the topic DHT and have the option of starting a voting procedure to penalize the broker who they think has committed a fraud. The broker DHT forms the communication channels between publishers and subscribers. Brokers are registered with the blockchain by depositing some collateral to start forwarding pub/sub content.

5.3.4 Publishers and Subscribers

Publishers register their publisher status and publishing content by using the smart contracts and provide some collateral to start publishing content. Subscribers status is established in a similar manner via another set of contracts except they do not need collateral. Instead, the subscriber indicates its interest and deposits funds for subscriptions.

5.3.5 Message Passing

Once a new publication message is issued, the broker network takes this message and encrypts it with the subscriber's public key [62] as shown in Figure 3. The brokers from within the topic DHT validate the encryption by recomputing the hash and endorsing the result by providing their digital signature with the correct hash. For example, one publication message gets received by brokers in a certain topic DHT as depicted in Figure 4. One broker will encrypt the data and the other brokers responsible for the topic will take up the roles of validators. The validated and encrypted data is then hosted in the DHT. The subscriber contract is notified once this is done via updating the the stored hash value in the blockchain. Simultaneously after other brokers verify that the encryption is correct, the payment deposited for the publication by the subscriber will be released. The money is distributed between the participating brokers and the publisher as per the contract specifics. The new encrypted publication is hosted between the brokers from the same topic DHT. Subscribers expect the message to be delivered to it from the broker once it receives an update from its smart contract. After receiving the content, the subscriber can compare the hash value to the one that is stored in the blockchain to validate the correctness.

This section presents the overall implementation of BlockPub. Then, we will focus on attacker model analysis, smart contracts, and distributed hash table.

5.4 Architecture

The model diagram outlines the main procedures and components in BlockPub 5.3.

Broker - To facilitate hosting of publication messages for each subscriber, BlockPub uses a network of brokers to host the publication messages. The brokers are organized into a dis-

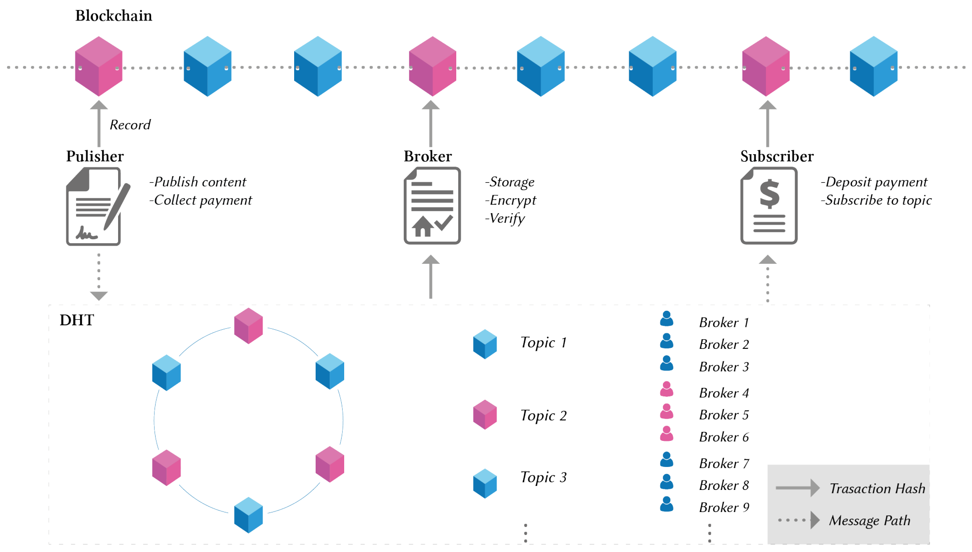

Figure 5.3: Structure Diagram

tributed hash table(DHT) within the network [64]. To join the DHT, the brokers first register in the blockchain by invoking the join contract. They are then assigned a random value that maps to a hash value in the DHT, achieving randomness in the DHT. Brokers find each other via the discovery protocols in the DHT and cross referencing the identity in the blockchain. The DHT then groups brokers into topics and they form a smaller DHT. Within each topic, Brokers oversee each others work and has the option of starting a voting procedure to penalize the broker who they think has committed a fraud. The broker DHT forms the communication channels between publishers and subscribers. Brokers are registered with the blockchain by depositing some collateral to start forwarding Pub/Sub content.

Publishers and Subscribers - Publishers register their publisher status via the smart contracts and provide some collateral to start publishing content. Subscribers status is established in a similar manner via another set of contracts except they do not need the collateral. Instead the subscriber indicates its interest along side the deposited funds for subscriptions. Once a new publication message is issued, the broker network takes this message and encrypts

Table 5.1: Comparison of Attacker Models

Attacks	**Types of Attacks**		
Attacker Model	***General Omission Failure***	***Broker Broker Collusion***	***Broker Subscriber Collusion***
Honest but Curious	✓		
Individually Byzantine	✓		
Coordinated Byzantine	✓	✓	✓

it with the subscribers public key [62]. Brokers from within the topic validate the encryption by recomputing the hash and endorse the result by providing their digital signature with the correct hash. The subscriber contract is notified once this is done via updating the the stored hash value in the blockchain. Simultaneously after other brokers verify that the encryption is correct, the payment deposited for the publication by the subscriber will be released. The money is distributed between the participating brokers and the publisher as per the contract. The new encrypted message is hosted between the brokers from the same topic. Subscribers expect the message to be delivered to it from the broker once it receives an update from its smart contract. After receiving the content, the subscriber can compare the hash value to the one that is stored in the blockchain to validate the correctness.

5.5 Possible attacker models

5.5.1 Attacker Model Analysis

Historically, Pub/Sub systems analyzed the correctness of the system based on non-byzantine failures. In identifying the attacker model for our application, we looked at colluding brokers. As the broker network is composed of third party machines, our attacker model is more complicated and this directly impacts how the smart contracts are written. The following table outlines the types of attacker models we have considered.

In blockchain systems, the number of correct nodes in the network is expected to exceed

51% of all nodes [57]. Our system relies on the same assumption within the network of all brokers. During the design phase of our application, we noticed that conventional attacker models are insufficient when analyzing the various attacks that we want our system to tolerate. We break down conventional byzantine fault to represent different attacker models, specifically the cases in which subscribers and brokers can freely collude with each other. In analyzing these attacker models we realize that the some of the limitations of our system and explore solutions to the more complicated attacks. In the sections below we analyze different attacker models and identify the one that BlockPub is capable of handling.

Honest but Curious

Most conventional Pub/Sub systems use this attacker model. Analysis of this attacker model is rather trivial as the brokers can only commit general omission failures (omission failure citation from csc2221). As both the DHT and the blockchain consensus correctness analysis tolerates general omission failures. The analysis of the system's correctness is simplified to the individual correctness analysis of the DHT and the execution of the smart contracts [57, 49]. Crash failures introduce delays in the consensus process. The extent of crash failure influence is heavily dependent on the timeout values that are used. Specifically the design of our DHT handles this through replication and several design parameters. Brokers are organized in hash rings regarding a topic. Subscriber and publisher crashing does not affect the correctness of the system, as the content is still delivered and payment accredited according to the smart contracts.

Individually Byzantine

This attacker model is significantly more complicated to handle in a fully distributed system. In fact, the FLP impossibility theorem tells us that consensus is impossible when a participating node is malicious [29]. However, consensus under such conditions can still be achieved by enforcing synchronous rounds [47, 24].

Coordinated Byzantine

This attack model is hardly ever talked about in conventional Pub/Sub systems. This attacker model introduces very complicated attacks on Pub/Sub systems. These attacks are very hard to stop as the basic assumption of trying to make a profit may not apply when there exists a conflict of interest within the Pub/Sub system. An example of this is external entities aiming to bring down the system at the expense of the external entities. Another attack that can occur is the scenario of the broker who also double acts as a publisher. Digital watermarks and shuffling of brokers from topic to topic are possible ways to prevent this; discussions on these are reserved for future works.

One sample solution to the coordinated byzantine attacker problems rely heavily on cryptography methods that split the publication into several different pieces and delivery channels described in secret sharing [74]. This approach doesn't use blockchain for synchronization, but its correctness heavily dependent on the routing algorithm used during the formation of the broker network.

Handling Coordinated Byzantine Attacks

To the best of our knowledge, non-blockchain based solutions are still insufficient at addressing these conflict of interest attacks. Intuitively, conducting all the transaction logic and storage in the blockchain system itself will ensure high consistency in all operations. Doing this ensures that all the stored files can be validated for correctness. This solution is too expensive and our solution leverages randomness and leader election in the blockchain to preemptively prevent coordination between the malicious entities. First we analyze how BlockPub handles individually byzantine brokers. Next we will analyze some examples of coordinate attacks that BlockPub is capable of handling. Finally we give some insight to attacks that BlockPub does not handle currently handle. We will also discuss some potential solutions to deal these attacks.

Individually Byzantine - At a higher level, the individual entities in our system are analyzed for conducting attacks for personal gain. The assumption is that a broker, subscriber and publisher can and will maliciously manipulate the messages it sends out for personal gain. In the case of the broker, there are multiple types of attacks that can be attempted. To start, brokers can disagree with other brokers in order to discredit other brokers. This attack is aimed at dividing the incentive with less brokers in the future. The broker run the risk of having its collateral taken away should correct brokers make up the majority and over rule this broker's decision during the validation process. Another attack is lazy verification. Broker can do lazy verification and not validate the work of other brokers. This runs the risk of endorsing the work of another malicious broker. In this case both brokers are ruled to be malicious and will lose their collateral. Both of these attacks are not profitable in the long run.

For publishers, this opens the avenue for providing garbage content to the subscriber. The subscriber has a means of invoking a voting protocol to notify the broker DHT to blacklist the publisher. At this point, the publisher no longer receives payment from the subscribers. The inherent sign up costs of joining the network prevent malicious publishers from joining the network for free.

For subscribers, there is no good attack, as subscriptions include locking some money amount aside for the publications. If this amount is insufficient then no publication will be delivered.

Handling Colluding Brokers - To break coordination of malicious brokers, the brokers are assigned randomly by being assigned randomly in the DHT and is registered in the blockchain by invoking the join contract. Each topic is handled by a smaller group of brokers. Brokers oversee each others work and has the option of starting a voting procedure to penalize the broker who they think has commit a fraud.

The leader election process is done through the blockchain, by invoking a smart contract [50, 63]. This allow for verifiable leader election results, contrast to the modified versions of paxos and raft [47, 24]. These modified protocols have similar fault tolerance to the leader election process to stop byzantine brokers from tampering with the election process. The implementation is low cost and based on previously analyzed implementation of this protocol on Ethereum [50].

Conflict of Interest Attacks - These attacks are very hard to imitate even in generating the attacker model analysis. BlockPub does not handle these attacks. Paying the penalty to disrupt a topic is easily doable as long as the attacker does not care about monetary loss. An example of this could be coordinated subscribers discrediting publishers only to disrupt the topic. One of the solutions could be a credit score associated with the publisher that gets updated with time. This will make it harder and more expensive for the attacker to discredit the publisher. Another potential solution is having some trusted third party being involved in the verification process to render the coordinated attack even harder to execute.

5.6 Smart Contract Design

Smart contracts is a set of logic based on the basic set of functions given in the Solidity language [9]. The reason why we chose solidity is because many of the solutions to the different attacks described above involve more complicated processing and demand a more powerful base instruction set.

Collateral

In writing the smart contracts, we realized that to discourage attacks, we needed collateral in the system in addition to the monetary incentives offered by doing more advanced processing. This design is inspired by proof of stake protocols [17]. The collateral is taken away when a user is detected to be malicious.

Reducing Contract Complexity

Although solidity offers Turing completeness, not all the computation can be done in the smart contract due to how expensive the computations are in the blockchain network. The most expensive operations include storing data in the network and account creation. Specifically, the DHT is used to store information of the topics and the list of brokers in the network. The information that is stored in the blockchain is the encrypted key to the encrypted content for the subscriber. The topic information and the subscriber contract can be executed. The hashing and verification happens in the broker DHT and only the core comparison logic is in the contract itself.

Determining Core Consistent Information

In conventional Pub/Sub systems, a logically centralized controller is often used to coordinate the storage metadata as well as delegation of storage responsibilities. In the wormhole example given in the motivation section, it uses Zookeeper as the coordinator.

We considered many alternatives to implement this central manager when designing our system. To begin with, we considered the naive solution of doing all the calculation within the blockchain network. This means that the content processed is also stored and replicated between the nodes in the blockchain network. The benefits of this system is apparent. Even the strongest adversary can not modify and take advantage of becoming brokers as any attack would be exceedingly expensive and is equivalent to conducting an attack on the entire blockchain network. The drawbacks of such a system are obvious. These high consistent guarantees come at too great a cost for the system to be scalable and use-able compared to its competitors.

To replace the heavy workload in the blockchain network, we designed several key consistency checks to represent the bulk of the processing required by the broker network and expressed them in the smart contracts.

In our case, the key guarantee of content delivery happens in the transaction states that are stored throughout the various stages of contract execution. To initiate the transaction, the subscriber and publisher are registered in the blockchain network by paying a node to kick off the stored smart contract. After the sign up process, Pub/Sub transaction can take place via the broker DHT. The Pub/Sub transaction contracts is more interesting than the registration contract. The overall transaction represents a delivery of the content from a publisher to a subscriber. To start the transaction, the publication gets sent to the broker DHT. The content is then encrypted and checked by all brokers in the topic. The balance is checked before the broker encryption. This is processed as a check in the smart contract. The next check happens once the brokers finish the encryption and check each others work. After this is done the contract is completed and the subscriber notified. Aggregated methods can be explored to even further reduce the cost of processing Pub/Sub transactions by combining multiple delivery transactions into one. The details will be discussed in the future work section.

5.7 DHT

Table 5.2: Distributed Storage Selection

Storage Candidates	**Actions**		
	Delete	***Topic Segregation***	***Leader Election***
IPFS		✓	
Kademlia DHT	✓		
Modified DHT	✓	✓	✓

Scaling the processing of Pub/Sub transactions while maintaining consistency is tricky given our attacker model. Storage in the blockchain is simply too expensive to hold the amount of information required to facilitate the end to end Pub/Sub message delivery within just the blockchain. These are the reason why we designed a DHT to handle the computational heavy processing and the message passing requirements of a Pub/Sub system.

IPFS

One of the first candidates that we considered when designing our system was IPFS [16]. IPFS organizes information in a file system manner. The system also use Having a file system to facilitate storage fits the needs of topic based Pub/Sub very well. It also saves the need of subscribers querying for the responsible brokers to retrieve the content. The drawback of IPFS is the lack of the delete function. In our system, published messages are individually encrypted for each subscriber. Given this design, we wanted a system that can delete the processed messages quickly and avoid storing deprecated information. Although IPFS offer great convenience in providing organization, this draw back led to us choosing with another storage system.

Kademlia DHT

The next candidate we considered was an open source DHT implementation [49]. Kademlia organizes the peers by XOR distance, allowing us to group brokers by the nearest neighbors to a topic. The XOR distance plays a key role in differentiating Kademlia from other open source DHT implementations such as Chord [68]. Organizing the brokers by XOR distance makes it easy to look up brokers in a similar hash range in the DHT, this works well with the topic based Pub/Sub model that we wish to support. Using a DHT handles deletion of content very well in comparison to IPFS and is easier to modify as well. The main functionality that we noticed was missing from Kademlia was the ability to dynamically allocate more brokers to each topic based on the number of subscriptions. We identified these shortcomings and added our functions on top of the stock implementation. Our modifications are described below.

Modified DHT

The modified DHT in our system organizes topics in a hash ring. The closest brokers are responsible for hosting the message for subscribers to that topic. We based our modification off of the open-source Kademlia DHT implementation.

Topic Organization - The topics are hashed into the range of the DHT. In our application we designed to accommodate only topic-based Pub/Sub. This design allow us to implement the logic of dynamically organizing the brokers according to the number of subscribers to each topic. The number of subscribers and the active brokers are kept in the blockchain and maintained by the master contract. For each topic, the content replicated and hosted by all brokers responsible for the topic. The replication is limited to per-topic to conserve storage space on participating servers in the entire DHT.

Leader Election - Having a leader for each topic serves two purposes. First, the leader handles general omission failures of brokers. Once failure is detected, the leader election process starts and synchronization is restored after a new leader has been elected [46]. Second, the leader is responsible for peer discovery, forwarding messages to the proper nodes in the DHT. The elected leader does not have elevated privileges compared to the other brokers in terms of executing the smart contracts. It can still be penalized if the majority of the brokers detect that the elected leader fails to execute contracts properly. The leader election is treated as a form of voting [50]. Having access to a blockchain based system make the leader election process trivial. The high consistency guarantee of this algorithm comes at higher costs of operation and longer wait times. However, leader election is not expected to happen often in the system due to the expensive join costs to the blockchain network and the risk of losing the deposit.

Look-up Modification - One of the benefits of using the DHT was that we could leverage the orginal join function in the DHT implementation. In our experiments we noticed some of the draw backs in Kademlia DHT while increasing the number of active brokers to facilitate Pub/Sub. This resulted in unwanted overhead increase as we populated the Pub/Sub system with more subscribers and brokers. We added additional cache for each broker to to reduce the number of look-up messages as more and more nodes joined the DHT network.

Broker Allocation - In topic based Pub/Sub it is very common for different topics to have discrepancy between the number of subscribers. To adhere better to this model, we organize the brokers according to the ratio of subscribers in a topic over the total number of subscribers. This information is stored in the blockchain and and organization of brokers are transparent as the organization algorithm is simple and can be executed upon joining the the network.

Chapter 6

Evaluation

We implemented a prototype of our system and ran it on a local cluster. The cluster consists of 12 nodes. Each cluster node runs Ubuntu 14.04 LTS and the exact same computing hardware, with 2 dual core Intel Xeon 5120 processors (4 cores in total), 8GB of DDR2 memory and 3Gbit/s SAS-1 hard drives. We simulated the publishers and subscriber as threads interfacing with a lightweight Ethereum wallet to invoke the functions in the smart contracts. These threads are distributed evenly between the nodes. We ran a full Ethereum node on a separate machine the for the light wallets to connect to. This machine runs Intel Xeon E5-2620 (6 cores total), 12GB of DDR4 memory with 3Gbit/s SAS-1 hard drive. The full node is used to run a private Ethereum network within the local network. Our evaluation are focused on the following:

- Evaluating the performance of the modified DHT and validating its functionality

- Evaluating the improvements we did to the DHT

- Evaluating the functionality of the overall end to end Pub/Sub system

6.1 DHT

Although we considered many alternatives to DHT to enable message processing and forwarding, we are unsure about the performance of our modified DHT. We wanted to better understand the message overhead that our system has when running Pub/Sub.

6.1.1 Look-up Messages

To test this, we run Kademlia in active mode and allow brokers to connect to the network through the blockchain. We create a DHT bootstrap with one of our own processes and let other brokers join the DHT network using the bootstrap. To observe the overhead, we run the experiments while varying the subscriber count and the number of brokers servicing topics in the DHT network while keeping the number of topics the same. For the payload, we used 160 bit SHA-1 hash values as publication content [67]. We issue 100 publications distributed evenly between the topics and measure the number of look-up messages, specifically 'find_node' messages issued by Kademlia, that are issued to properly deliver and verify the content of each publication. For each different configuration of subscribers and brokers, we run the experiment 5 times and take the average.

6.1.2 Trend Analysis

As seen in 6.1, the y-axis represents the number of look-up messages per publication while the x-axis represents the number of subscriber per topic. We notice that there is a near linear increase in messages per publication as the number of subscribers per topic increases. We believe that the linear growth is due to how buckets, each holding up to 20 contacts, are populated in

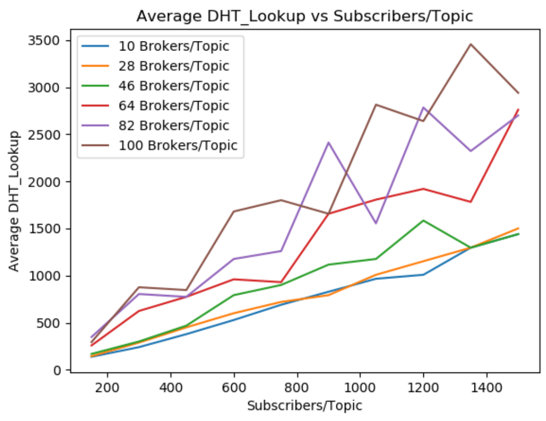

Figure 6.1: Average DHT Look-up Messages

Kademlia using XOR distance. Every publication verification depends on the verifier finding the node responsible for publishing the content. Thus as the number of subscribers increases, the number of publication verifications rise as well. This is what causes the near linear relation seen in Figure 6.1.

Things start to become more interesting as we vary the number of brokers given a set number of subscribers for that topic. We notice that with more brokers, the number of look-up messages generally increases per publication as expected (from 5 to 35 as depicted) However, when the number of brokers per topic exceeds 80, the number of look-up messages are significantly more than those under 80. We believe that this is due to the bucket size, $k = 20$, used in Kademlia. When the number of brokers exceed 80/topic, the buckets start being full and the look-up needs to happen recursively to locate the next bucket.

6.2 Improvements

Realizing that the bucket size was causing the look-up messages to increase dramatically when the number of brokers increased, we decided to add additional look-up space for each broker so that they can cache more brokers within their topic and store more information about other brokers in the network.

6.2.1 Caching

To confirm that our modifications reduced the number of look-up messages during our experiment we ran a second set of experiments on the same cluster with our modified DHT.

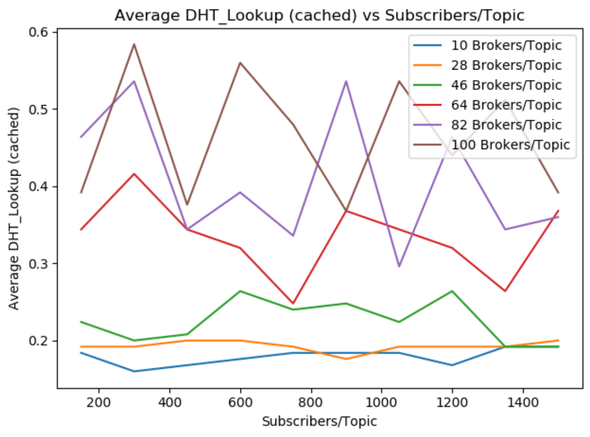

Figure 6.2: Look-ups Events Per Delivery

6.2.2 Discussion of Results

As can be seen, after the modification, the average look-up message per delivery dropped significantly. This is because the brokers are aware of many more brokers in the network. After all its neighbours, the cache is filled up and through out the publication handling process it almost never needs to issue a look-up request due to the high hit rate in the cache. The brokers should be incentivized to listen to events from Ethereum dealing with new brokers entering the dht, old brokers voted out, and even work reassignments. The cache can be kept updated as long as the brokers are handling all events properly.

6.3 Functionality Evaluation

We did all of our evaluations within the test network provided by the full Ethreum node. This environment does not simulate real world delays that exist in the actual Ethereum network. The information stored would have similar levels of guarantee as in our test environment with the exception of waiting on the confirmation of individual transactions that are being processed. We can confirm the issuing and reception of publication messages through the smart contracts. Analysis is done on expected delay times in the network. Additional tests and functionality validation are discussed in the Future Work section.

6.3.1 Storage Contract

The master contract in this case is very much a storage contract. In the experimental stage, the master contract keeps track of the topics and associated public keys of the brokers in the network. One of the limitations of this approach is the initial cost of storage of this contract. Storage code snippet below for topic and broker storage. One of the considerations here is using a different data structure for better gas efficiency. A simple storage solution for the initial prototype was chosen and shown below.

In designing the the contracts we realized that there are several design patterns we should watch for as to prevent logical errors from occurring during the transaction processing stages. One of the common problems is the re-entrancy attack[32]. Contract evaluation tools such as Mythil were used during the implementation stage to validate the quality of the smart contracts.

```
pragma solidity ^0.4.8;


contract Storage {
  // This shows Topics storing Brokers, separate contract for publishers and
      subscribers
  struct Broker{
    bytes32 addr;
    bool isLeader;
  }

  // store topics as public so other contracts can read from topic storage
  mapping(bytes32 => Topic) public topic;

  // Set values in storage, key is the IP in bytes
  // addr is the address in the test network
  function StoreBroker(bytes32 key, bytes32 addr, bool isLeader) returns (bool
      success) {
   topic[key].name = addr;
   topic[key].isLeader = isLeader;
   return true;
  }

}
```

6.3.2 Storage Cost Considerations

Another problem that was considered when designing the main contract is the gas limit of the block. Currently the block gas limit sits at 8 million. This can become a problem when more data is stored using the storage contract. Since each storage call is approximately 20000 gas per 32 byte word, in actual storage costs per word is approximately $0.01 USD. We tested this using a simple storage contract on the main network to evaluate the costs associated with running our system in this fashion. For a topic with 100 brokers, the storage costs amounts to \$2.00. The initial deployment cost of the contract is 100,000 Gas (\$0.50).

6.3.3 Comparison to Naive Implementation

Without the use of DHT to offset the computation costs in the Ethereum blockchain we would have to rely on storage and computation logic implemented within the smart contract it self. We show how the presence of the DHT significantly decreases the costs associated with processing a publication message.

6.3.4 Evaluating Voting Performance

We simulated the results in both the local network as well as the ethereum test network. We found that the gas cost for casting a vote locally is 30000 Wei(8 cents). We mimic the voting process in the local network. One thing we observed is the trend of confirmation times versus the number of voters casting votes.

It is noticeable that confirmation times grows near linearly with the amount of concurrent voters in the system. In a single node network this is understandable. In the test network the transactions would be submitted to a full node by the light clients, which would be close to the scenario shown here. The difference between the simulated network and the test network is the delay and confirmation times. Below we analyze the wait times for voting and transaction confirmation.

6.3.5 Voting Wait Time

Confirmation wait times in the network could seriously impact the performance of the overall system. Especially in the Ethereum test network and main network. In fact we can see that

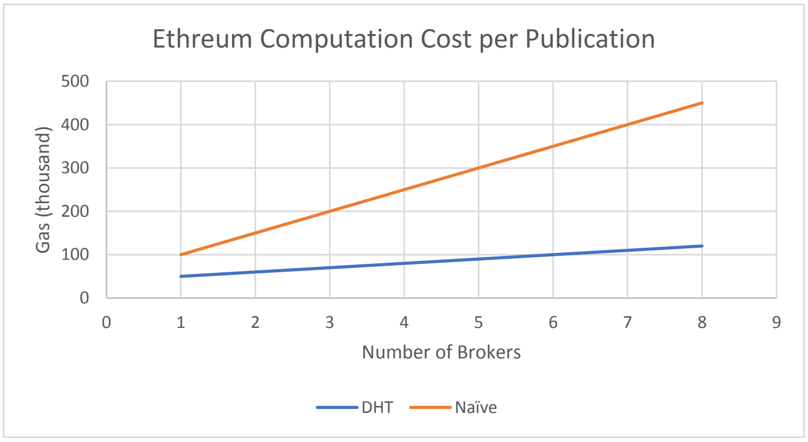

Figure 6.3: Gas Cost Comparison VS Naive Implementation

the majority of wait time can be due to waiting for confirmation of transaction results.

Here we can see that the typical wait times for transaction processing range from 2.4 minutes up to 1 hour depending on the transaction fees paid to the miner. Confirmation of the transaction follows 6 blocks later, ranging from 6-9 minutes.

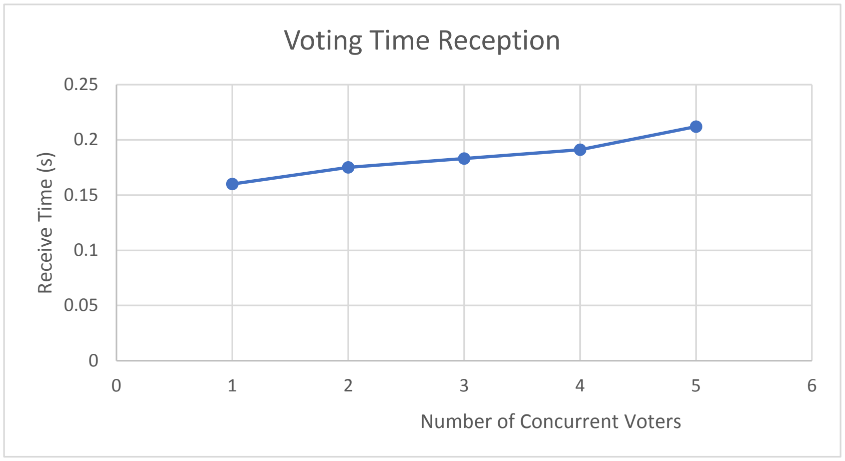

Figure 6.4: Gas Price VS Confirmation

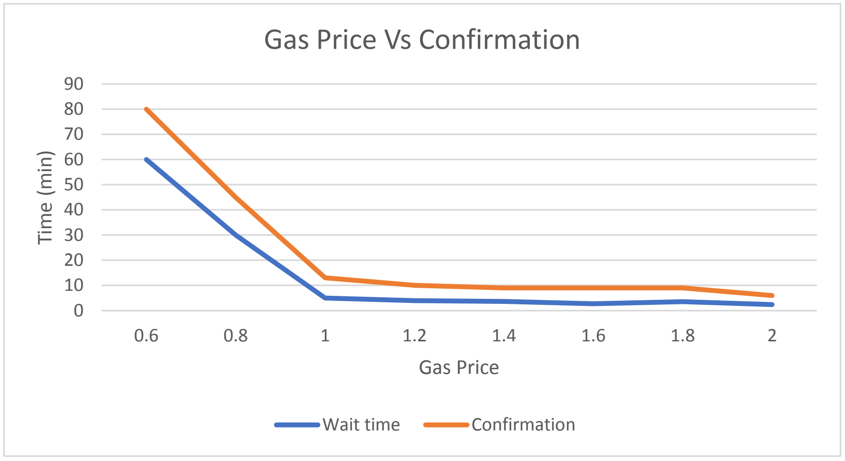

Figure 6.5: Gas Price VS Confirmation

6.4 Limitations

We can see from the evaluations that the wait times are a hard limitation to any DLT application. This is no different for our prototype. The best case scenario is waiting for approximately 3 minutes before the confirmation of the transaction at best. On top of this more costs are associated with the processing of the deliveries of messages. Each transaction would involve a voting process for delivery checks, adding an over head of $0.08 * number of brokers validating the result. Although we reduced the number of messages passed through the DHT and minimized the messages stored on the blockchain, the limitations of the DLT is still the most prominent. From these results we can generate a few constraints for applications adopting this method of processing transactions.

6.4.1 Transaction Value

Transactions are expensive in blockchain systems. In our motivating case, each publication should be worth significantly more than the transaction costs. Contract initialization is a sunk cost and isn't really taken into account here.

6.4.2 Transaction Time Dependency

As can be seen in from the confirmation wait times, it is very difficult for time-sensitive applications to adopt to Ethereum. In this case performance is also sacrificed for high level of guarantee the prototype offers.

Chapter 7

Future Work

From the start of During the design and implementation stages we learned a lot about designing and writing smart contracts. However, the validation of these contracts within the context of our design still need to be verified through more rigorous simulations and harsher attacker models. These steps can be summarized as follows.

7.1 Attacker Model Simulation

We addressed the various attacker model definitions and considered several example attacks based on these definitions. Simulating these attacks are a lot more difficult and to validate in practice. We wrote the contracts based on the different types of attacks. The next steps involve simulating the attackers and deploying these malicious brokers into the system and observe the behaviour.

7.2 Additional Modifications to the DHT

We could improve the DHT to better fit our use case by taking into account the number of Publishers per topic to better allocate brokers to topics. Additional consideration involves tracking the number of publications in each topic and allocating brokers dynamically.

7.3 Deploying to Ethereum Network

Deploying the application to the actual Ethereum network would better simulate the delays in the network and real work network conditions. This would also give us a more accurate comparison of the throughput performance of BlockPub versus conventional Pub/Sub systems and the performance penalties from using a blockchain based coordination solution.

7.4 Verification of Contract Correctness

Verifying that the contract behaves in the expected manner is very difficult using the current development tools. Many problems and vulnerabilities still exist in smart contracts that are deployed to the actual Ethereum network. These vulnerabilities are very hard to find out during the functional testing stage.

7.5 Finding Alternative Platforms to Ethereum

From our evaluations we found that the biggest limitation to our prototype was the confirmation time delays. If Ethereum manages to upgrade its throughput or if an alternative blockchain based platform can be utilized to generate trust then we would expect to see better adaption of similar models.

Chapter 8

Conclusion

Recent efforts in improving and adopting blockchain technology have lead to many interesting scenarios for middleware systems. This in turn indicates that decentralized applications will be more popular in the future [71]. The benefits of blockchain technology lies in the decentralized architecture and the high levels of guarantees it can offer. These properties are very desirable in several application categories which have been previously identified.

In this thesis, a list of motivating use cases were presented for the design and implementation of BlockPub, a Pub/Sub message delivery system that scale in processing power using third party computation resources. BlockPub utilize the high consistency guarantees offered by blockchain technology to achieve fault tolerance and transparency in data handling. Moreover, it introduces a novel, self-sufficient model that can scale without having a trusted third-party to handle the coordination between non-trusting entities. We define and analyze various constraints in this model and propose a solution to this problem. We identified the limitations of using existing blockchain platforms and identify key future developments required in order for the system to be applicable to more applications.

Bibliography

[1] Byzantine fault tolerance in a distributed system. online, http://sce.uhcl.edu/goodwin/Ceng5334/downLoads/byzantine.pdf.

[2] Gossip protocols. http://www.inf.u-szeged.hu/ jelasity/ddm/gossip.pdf.

[3] The golem project, 2016. https://golem.network/doc/Golemwhitepaper.pdf.

[4] Understanding the dao attack, Jun 2016. https://www.coindesk.com/understanding-dao-hack-journalists.

[5] *Hyperledger Whitepaper*. http://www.the-blockchain.com/docs/Hyperledger%20Whitepaper.pdf, may 2017.

[6] *A Next-Generation Smart Contract and Decentralized Application Platform*. https://github.com/ethereum/wiki/wiki/White-Paper, may 2017.

[7] Tax token, 2017. https://taxtoken.io/.

[8] Gridcoin white paper, 2018. https://www.gridcoin.us/assets/img/whitepaper.pdf.

[9] Introduction to smart contracts, 2018.

[10] Ittai Abraham and Dahlia Malkhi. Bvp: Byzantine vertical paxos. 2016.

[11] C.A. Adams. Transaction approval system, January 5 1993. US Patent 5,177,342.

[12] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In *International Conference on Trust and Trustworthy Computing*, pages 163–180. Springer, 2015.

[13] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Daniel O'Keeffe, Mark L Stillwell, et al. Scone: Secure linux containers with intel sgx. In *12th USENIX Symp. Operating Systems Design and Implementation*, 2016.

[14] Tom Atwood. Vertical and horizontal computing architectures - trends and attributes. Technical report, Sun Microsystems SUPerG Berlin, May 2003.

[15] Peter Bailis and Ali Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *Communications of the ACM*, 56(5):55–63, 2013.

[16] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *CoRR*, 2014.

[17] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Perform. Eval. Rev.*, 42(3):34–37, December 2014.

[18] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. Securekeeper: Confidential zookeeper using intel sgx. In *Proceedings of the 16th Annual Middleware Conference (Middleware)*, 2016.

[19] Eric Brewer. A certain freedom: Thoughts on the cap theorem. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 335–335, New York, NY, USA, 2010. ACM.

[20] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, January 2000.

[21] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[22] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM.

[23] European Commission. Eugdpr, Nov 2018. https://eugdpr.org/.

[24] Christopher N. Copeland and Hongxia Zhong. Tangaroa: a byzantine fault tolerant raft. 2014.

[25] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[26] Chris Dannen. *Introducing Ethereum and Solidity.* Springer.

[27] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[28] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59. USENIX Association, 2016.

[29] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[30] Apache Foundation. Pulsar: open-source distributed pub-sub messaging system, 2016. https://pulsar.incubator.apache.org/.

[31] Vincent Gramoli. On the danger of private blockchains. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[32] Gustavo Guimaraes. Reentrancy attack on smart contracts: How to identify the exploitable and an example of an attack contract, 2017.

[33] Richard Hull, Vishal S Batra, Yi-Min Chen, Alin Deutsch, Fenno F Terry Heath III, and Victor Vianu. Towards a shared ledger business collaboration language based on data-aware processes. In *International Conference on Service-Oriented Computing*, pages 18–36. Springer, 2016.

[34] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[35] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: a distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 533–549. USENIX Association, 2016.

[36] David Iseminger. *Active Directory Services for Microsoft Windows 2000.* Microsoft Press, 1999.

[37] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh. The padres publish/subscribe system. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[38] Scott Hendry Andrew McCormack Wade McMahon James Chapman, Rodney Garratt. Project jasper:are distributed wholesale payment systems feasible yet?

[39] T. Jin, X. Zhang, Y. Liu, and K. Lei. Blockndn: A bitcoin blockchain decentralized system over named data networking. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 75–80, July 2017.

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

[41] Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13:1–13:21, March 2016.

[42] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE, 2016.

[43] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: a distributed messaging system for log processing. 2011.

[44] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[45] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[46] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[47] Leslie Lamport. Byzantizing paxos by refinement. In *Distributed Computing*, pages 211–224, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[48] Ying Liu and Beth Plale. Survey of publish subscribe event systems. 06 2003.

[49] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[50] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. Cryptology ePrint Archive, Report 2017/110, 2017. https://eprint.iacr.org/2017/110.

[51] Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. *Available on line: http://nakamotoinstitute.org/research/anonymous-byzantine-consensus*, 2014.

[52] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. *URL: http://www.bitcoin.org/bitcoin.pdf*, 2012.

[53] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. *URL: http://www.bitcoin.org/bitcoin.pdf*, 2012.

[54] Monetary Authority of Singapore. Project ubin: Central bank digital money using distributed ledger technology.

[55] Oracle. Business process management and ws-bpel 2.0, May 2017. http://www.oracle.com/technetwork/topics/bpel-130653.pdf.

[56] Philippe Raipin Parvédy and Michel Raynal. Uniform agreement despite process omission failures. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 7–pp. IEEE, 2003.

[57] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673, Cham, 2017. Springer International Publishing.

[58] Rafael Pass and Elaine Shi. Hybrid consensus: Scalable permissionless consensus, 2016.

[59] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity-a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.

[60] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016. *URl: https://lightning.network/lightningnetwork-paper.pdf (visited on 2016-04-19)*, 2015.

[61] Philippe Raipin Parvedy and Michel Raynal. Uniform agreement despite process omission failures. page 212, 01 2003.

[62] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[63] P. McCorry S. Azouvi and S. Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. Unpublished.

[64] Siamak Sarmady. A survey on peer-to-peer and DHT. *CoRR*, abs/1006.4708, 2010.

[65] Yogeshwer Sharma, Philippe Ajoux, Petchean Ang, David Callies, Abhishek Choudhary, Laurent Demailly, Thomas Fersch, Liat Atsmon Guz, Andrzej Kotulski, Sachin Kulkarni,

Sanjeev Kumar, Harry Li, Jun Li, Evgeniy Makeev, Kowshik Prakasam, Robbert Van Renesse, Sabyasachi Roy, Pratyush Seth, Yee Jiun Song, Kaushik Veeraraghavan, Benjamin Wester, and Peter Xie. Wormhole: Reliable pub-sub to support geo-replicated internet services. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 351–366, Berkeley, CA, USA, 2015. USENIX Association.

[66] Yogeshwer Sharma, Philippe Ajoux, Petchean Ang, David Callies, Abhishek Choudhary, Laurent Demailly, Thomas Fersch, Liat Atsmon Guz, Andrzej Kotulski, Sachin Kulkarni, Sanjeev Kumar, Harry Li, Jun Li, Evgeniy Makeev, Kowshik Prakasam, Robbert Van Renesse, Sabyasachi Roy, Pratyush Seth, Yee Jiun Song, Kaushik Veeraraghavan, Benjamin Wester, and Peter Xie. Wormhole: Reliable pub-sub to support geo-replicated internet services. NSDI'15, pages 351–366, Berkeley, CA, USA, 2015. USENIX Association.

[67] Marc Stevens, Pierre Karpman, and Thomas Peyrin. Freestart collision for full sha-1. In *Proceedings, Part I, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665*, pages 459–483, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[68] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[69] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[70] Michael Szydlo. *Merkle Tree Traversal in Log Space and Time*, pages 541–554. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[71] Don Tapscott and Alex Tapscott. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin, 2016.

[72] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[73] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2017.

[74] Young Yoon and Beom Heyn Kim. Secret forwarding of events over distributed publish/-subscribe overlay network. *PLOS ONE*, 11(7):1–23, 07 2016.