2018

# PEER TO PEER DIGITAL RIGHTS MANAGEMENT USING BLOCKCHAIN

James Rinaldi
*University of the Pacific*, jamesrinaldi22@gmail.com

# PEER TO PEER DIGITAL RIGHTS MANAGEMENT USING BLOCKCHAIN

by

James Rinaldi

A Thesis Submitted to the

Graduate School

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE ENGINEERING SCIENCE

School of Engineering and Computer Science
Computer Science

University of the Pacific
Stockton, California

2018

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Jeff Shafer for his advice during this project. I would like to thank my mother and my brother for their patience and support.

Peer to Peer Digital Rights Management using Blockchain

Abstract

by James Rinaldi
University of the Pacific
2018

Content distribution networks deliver content like videos, apps, and music to users through servers deployed in multiple datacenters to increase availability and delivery speed of content. The motivation of this work is to create a content distribution network that maintains a consumer's rights and access to works they have purchased indefinitely. If a user purchases content from a traditional content distribution network, they lose access to the content when the service is no longer available. The system uses a peer to peer network for content distribution along with a blockchain for digital rights management. This combination may give users indefinite access to purchased works. The system benefits content rights owners because they can sell their content in a lower cost manner by distributing costs among the community of peers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1: Introduction

The motivation for this work is to create a content distribution network (CDN) that maintains the consumer's rights and access to works they have purchased like music, movies, books, and software indefinitely. The problem with current systems is that when a user purchases an electronic copy of a work from a traditional CDN they lose access when the CDN service is permanently or temporarily unavailable. The benefit of using a peer to peer (P2P) file distribution network for content delivery and a blockchain for digital rights management (DRM) is that it can give users indefinite access to purchased works.

CDNs use DRM systems to manage customers' digital content rights. DRM protects a content creator's revenues by letting only authorized consumers access a work. This paper considers a DRM system failed when the servers on which it relies are no longer operated. Servers are shut down by their operators if they are legally compelled or operating the servers is unprofitable. DRM systems can be unprofitable from server cost or from having to rewrite DRM client software to keep up with advances in operating system design [4]. DRM services are considered failed from the consumer's perspective and from an economic perspective. Consumers lose current and expected value when DRM services are shutdown. Unprofitable servers produce less economic value than they consume and have failed economically.

DRM system failures have negatively impacted both consumers and businesses. Failed DRM systems have hurt consumers by reducing availability of purchased content. A failed

DRM system that hurt consumers is the shutdown of Yahoo! Music. On September 30, 2008 Yahoo! Music shut down their license key servers [5]. This shutdown stopped consumers from downloading purchased media. To transfer content consumers had to burn the content to CDs and then rip the content to a new computer [5]. Another failed DRM system was Diesel eBooks which in March 2014 announced that their servers would shut down at the end of the month [6]. Ubisoft's DRM services were temporarily interrupted by a denial of service (DOS) attack. Ubisoft's games require communication with a server to be played and the DOS attack prevented some player's games from communicating with the server. These players were unable to play their games during the attack [7].

Failed DRM systems hurt businesses who operate them while they are unprofitable. Walmart announced on September 26, 2008 they were shutting down their DRM servers. But after customer complaints Walmart reversed the decision [8]. Microsoft announced that on August 31, 2008 that MSN music servers would be shutdown preventing users from authorizing new devices to play music. But after customer complaints Microsoft supported adding new devices to MSN music until the end of 2011 [9]. The DRM systems failed because the companies were financially better off by ending services. Announcing the end of services hurts the company's reputation so they decide to continue service and reduce profits.

The solution to prevent failure of a DRM system is to decentralize its operation. Decentralization should be both technological and financial with a community of peers supporting the DRM system. Centralized DRM systems have the operating company as a single point of failure. Failure occurs when the company goes out of business or because the DRM system becomes unprofitable. Short term unavailability occurs during power

failure, natural disaster, or cyber attack. The decentralized system can have a partial or a complete failure. Partial failure occurs in a decentralized system if the last copy of a file is unavailable. Total failure occurs in a decentralized system when every peer that participates in the system shuts down. Decentralized operation will prevent financial failure by distributing costs among the community operating the system. A decentralized system mitigates the causes of short term failure since power failures and natural disasters won't cause correlated peer failures assuming peers are diverse in physical location.

The decentralized DRM system proposed here will utilize P2P filesharing and blockchain technology. P2P filesharing creates a CDN that is decentralized and has each client as a potential server. Blockchain creates a distributed database to contain rights information. This thesis will present a review of the literature in Chapter 2, a background of the systems used to implement the distributed DRM system in Chapter 3, the design of the DRM system in Chapter 4, its implementation in Chapter 5, and its testing and evaluation in Chapter 6.

# Chapter 2: Literature Review

## 2.1 Peer to Peer File Sharing

Studies of P2P network architectures place emphasis on improving network efficiency and reliability. The work of Tang et al. [10] reviews the history of traditional, P2P, and cloud based CDNs. Tang found that first generation P2P CDNs are centralized and unstructured, with a central server used to coordinate peers. An example of a first generation network is Napster. The advantage of this type of system is that peers do not have to store complex routing data. A significant disadvantage of this type of system is that a shutdown or failure of the central server will collapse the entire P2P network [10]. Tang found that second generation P2P networks do not use centralized servers and are decentralized and unstructured [10]. Second generation P2P networks, such as Gnutella, relay queries among all peers until the desired file is located. Redundant messages in this type of P2P network inefficiently waste bandwidth, causing poor scalability [10]. Other second generation P2P networks, such as Chord, use a distributed hash table (DHT) for locating files. Large DHT networks do not work well due to routing data storage [10]. Third generation P2P networks combine properties of the first and second generations. An example of a third generation is BitTorrent (BT). BT's trackers are first generation central servers except they track specific files instead of all files. BT can use a DHT instead of a tracker. BT splits files into multiple chunks, increasing replication speed in the P2P network by allowing downloaders to begin

uploading part way through the download [10]. A summary of the generations identified by Tang et al are shown in Table 1.

Table 1: Generations of P2P Networks

| | Centralized | Structured | Reliable | Scalable | Year |
|---|---|---|---|---|---|
| $1^{st}$ Generation | Yes | No | No | Yes | 1999 (Napster) |
| $2^{nd}$ Generation | No | Yes | Yes | No | 2000 (Gnutella) |
| $3^{rd}$ Generation | Yes/No | No/Yes | Yes | Yes | 2002 (BT) |

Stoica et al. [1] introduce a P2P network lookup protocol called Chord. The Chord protocol maps a key to a P2P node, so peers can find the node with a desired file. Chord divides routing information among nodes so they only store $log(N)$ entries and can query other nodes when missing information is required. Figure 1 shows example routing information. It contains a key in the start column, the node that is holding the key in the successor column, and the number of keys this node tracks. Chord finds desired files in $log(N)$ messages. Chord can add and remove nodes from the routing table in at most $log^2(N)$ messages with a high probability of success. Chord evenly distributes files among the nodes to balance network load.

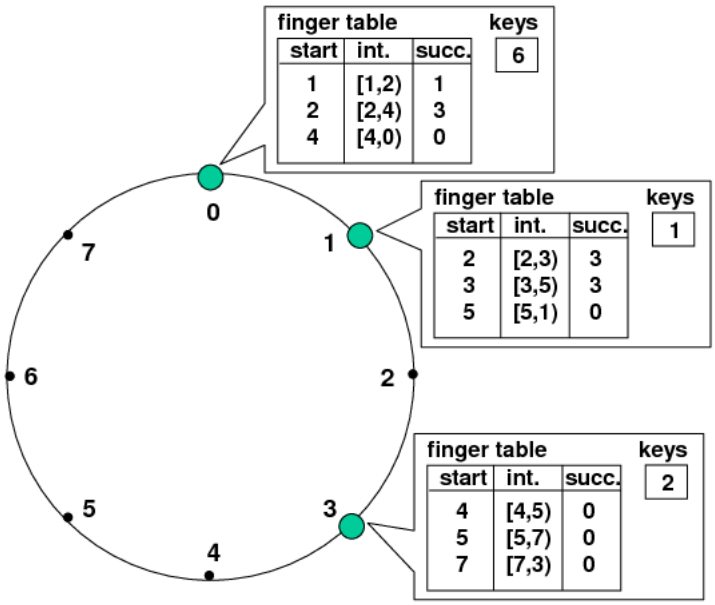


Figure 1: Chord Architecture [1]

Jagadish et al. [2] introduce a P2P lookup protocol tree structure called Balanced Tree Overlay Network (Baton), shown in Figure 2. Baton maps a key to a P2P node and uses the tree structure to perform range queries adding additional functionality. Baton can add and remove nodes in $log(N)$ messages improving upon Chord which could do so in $log^2(N)$ messages. Baton can find a file or perform a range query in $log(N)$ operations. Baton avoids bottlenecking of communication caused in trees when nodes from one side of the root must communicate with nodes on the other side of the root. Baton avoids this bottlenecking by adding a link between nodes on the left side of the tree with their analogous node on the right side of the tree. Baton nodes also have links to the two nodes to their left and two nodes to their right at the same level. These additional links reduce bottlenecks and increase robustness as nodes join and leave the network. Using figure 2 if node *m* needs a

file held by node $i$ it can immediately communicate with node $i$. But if node $m$ wants a file held by node $h$ then node $m$ will pass the request to node $l$ since node $m$ knows that node $l$ knows the location of node $h$.

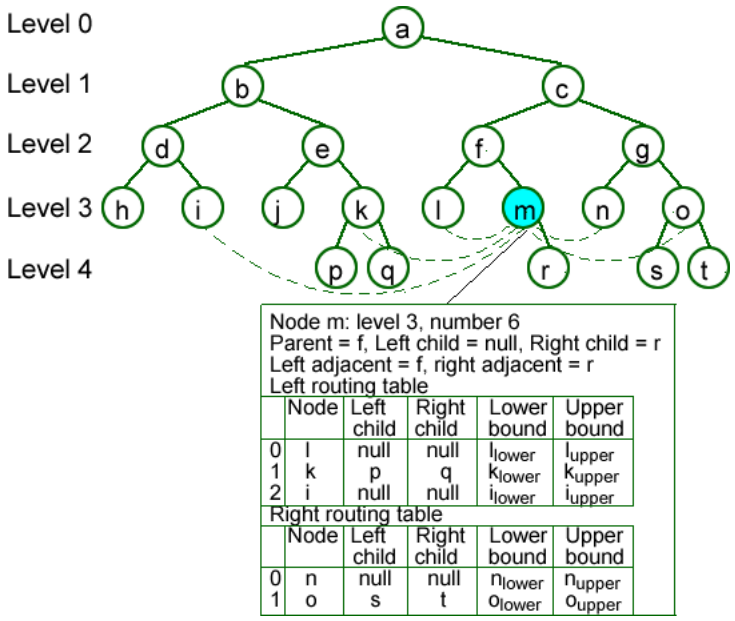


Figure 2: Baton Architecture [2]

Tsolis et al. [3] introduce the Autonomous Range Tree (ART), shown in Figure 3. ART nodes store a binary tree with links between each node at the same level. ART nodes also store a directory called a Random Spine Index (RSI). The RSI contains links to random nodes at every level of the ART. The ART is more efficient than many popular P2P architectures [3]. ART finds keys in $log_b^2 log(N)$ messages. The maximum routing table size for ART is $N^{1/4}/log^c N$. ART updates routing information in $log(log(N))$ messages [3].

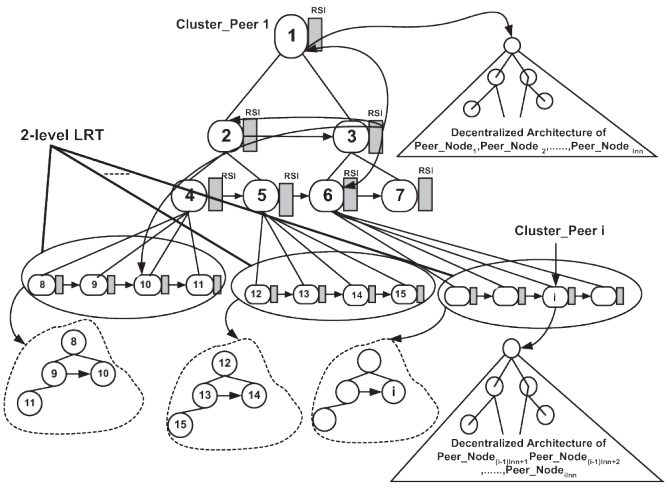


Figure 3: ART Architecture [3]

Peltotalo et al. [11] combine multicasting with P2P CDNs. A multicast server sends data to multiple clients by sending the data with special instructions telling the network to send it to all the clients. Peercasting is the combination of P2P filesharing and multicasting [11]. Peltotalo found that peercasting optimizes network usage if used in large scale CDNs. Delco is their prototype of peercasting and is based on BT. Delco can be run in peercasting and non-peercasting mode allowing users whose ISP does not allow multicasting to participate in the network. Multicast support is most common in networks to deliver IPTV but is not commonly supported on the public internet.

## 2.2 DRM

DRM uses multiple approaches to protect content [4]. Controlling access to content is the first line of defense. Allowing only authorized users to download content reduces the number of places that leaks could originate and requires a leaker to purchase the content.

Cryptography is the second line of defense. Cryptography prevents content from being stored on disk or RAM in plain text. Clients request decryption keys if they do not have them. If the content is rented the keys expire when the rental period ends. Content is decrypted in small portions and is stored in RAM unencrypted for a limited amount of time, to protect against attackers reading the content from RAM. Once decrypted, content is sent through a secure channel, like High-bandwidth Digital Content Protection (HDCP), to the display. The last line of defense is used after content has been leaked. Forensic marking uniquely identifies the purchaser of a content, this helps track down the leakers of the content and is used to discourage leakers. Digital watermarking is a form of forensic marking that changes color or sound values slightly and is only detectable with the detector and secret key.

DRM systems operate in untrusted environments requiring methods to secure operation [4]. Tamper resistant hardware offers protection against physical or logical attacks. The processor, RAM, and storage in tamper resistant hardware is not physically accessible and detects intrusion attempts. Tamper resistant hardware is used for encryption and decryption to protect the secret keys and the content. Tamper resistant software uses programming practices, obfuscation, and encryption to prevent analysis and manipulation of software. To prevent static analysis the software is obfuscated and can be encrypted with a driver program that unencrypts the software to run it. To prevent run time analysis the software detects the use of debuggers or if it is in a virtual machine. If it is being analyzed, it will stop execution at a random point. Programing practices like the principle of least privilege and compartmentalization are used to minimize the effect of attacks. Multiple versions of functionally equivalent software can be distributed to prevent universal attack

tools.

## 2.3 P2P with DRM

Current work on integrating DRM into P2P CDNs varies in DRM method and P2P network architecture. DRM methods found in literature encrypt content, encrypt keys, watermark content, require hardware, or require an operating system interface [12] [3] [13]. Examples of P2P DRM include fully P2P systems and others that have central key or rights management servers.

Shi et al. [14] implement a P2P research network called vuCRN. The vuCRN network is for distributing PDF files. Adobe's Extensible Metadata Platform embeds rights information into PDF files. The types of rights supported by vuCRN include unrestricted, reproducible, and personal use only. Juxtapose P2P networking software coordinates peer communication in vuCRN. The work does not specify how the rights meta-data added by the scheme is enforced.

Tsolis et al. [3] combine P2P image sharing and digital watermarking. Whenever a node receives an image it watermarks the image with its key. The digital watermarking is invisible to viewers but can be found by a detector. The detector can find each key in the image establishing rights enforcement and assisting in transaction tracking [3]. The P2P network used is the ART described in section 2.1.

Chen et al. [12] increase privacy while combining DRM and P2P file sharing. First the content provider (CP) generates a master key to encrypt the file. The master key is hidden using Shamir's threshold algorithm, splitting the master key into multiple parts called shadow keys. No information is revealed about the master key until it is recovered.

The shadow keys are distributed among the P2P network nodes. The peers encrypt the shadow keys using a group key. Users downloading a file use the group key to request the shadow keys from the other nodes. The user recovers the master key with the shadow keys and decrypts the file. The encryption process hides the file content and decryption keys. The P2P nodes do not know with which user they are communicating [12]. Third parties watching network traffic do not know which file is being transferred and do not learn any portion of the decryption key.

Chang et al. [15] describe methods of key distribution for P2P Internet Protocol Television (IPTV). Two key distribution methods are presented, one uses asymetric-cryptogram-family key distribution (ACF) and the other uses private-key encryption (PKE) [15]. ACF protects by requiring peers to decrypt content and encrypt it with their private key before forwarding the content. PKE offers protection by encrypting the content issuer's private key. ACF has better DRM protection than PKE, but PKE is more cost effective than ACF [15]. Both ACF and PKE rely on a central key management and rights issuer server. PKE is more efficient than ACF because the peers decrypt a smaller amount of data less often. PKE reduces both cryptographic processing and key request traffic [15]. The authors suggest increasing the security of PKE through periodic distribution of a new master key.

Kumar et al. [13] propose DRM middleware (DMW). DMW is designed to be used in a P2P network. DMW utilizes a hardware component and the device's operating system (OS) to enforce copyright. DMW communicates with the OS to gain information about the use of copyrighted content. DMW then instructs the OS to stop actions that violate copyright. The hardware component decrypts content just in time for the user. DMW proposes to use a content information exchange not described by Kumar et al. [13]. The

work of Kumar et al. is a research proposal that does not appear to have been developed further.

## 2.4 Blockchain with DRM

Kishigami et al. [16] describe The Blockchain Based Content Distribution System. This system manages access rights to 4k video using a blockchain. This allows copyright owners to manage the permissions of the content including removing permission to use content from a user. This system does not allow off-line viewing of content since the client requires a transaction ID from miners on the network to decode the content. The design proposed in this thesis differs from the work of Kishigami et al. since it has a built in currency which allows for decentralized purchasing.

Herbert and Litchfield [17] use a blockchain for software license validation. The authors identify two blockchain forms in license validation. In the Master Bitcoin Model, a consumer proves ownership by showing that they own a Bitcoin that originated from the software vendor. The Bespoke Model is the Master Bitcoin Model with additional data fields available. This allows the software maker to store license information like time until license expiration. The Bespoke Model was implemented for a single user owning a single license. Future work mentioned is allowing clients to hold hundreds of copies of a license.

McConaghy and Holtzman [18] use the Bitcoin blockchain to record image ownership. The legal registry stores the terms of service and a timestamp. The registry is stored on the blockchain with ownership information. A web crawler uses machine learning to detect images that have been placed on a website without the owner's permission.

Ujo uses Ethereum to enable music publishing [19]. Ujo demonstrated proof of

concept by publishing Imogen Heap's 'Tiny Human' using Ethereum. The Ethereum blockchain is used by Ujo for payments, rights management, and identity storage for artists. Ujo uses Interplanetary File System (IPFS) for data transfer.

PeerTracks is being developed to allow for streaming and retail music sales using blockchain [20]. PeerTracks uses the Muse blockchain, which is specifically for PeerTracks. PeerTracks includes a tipping and patronage system. Muse "monetizes attention" by allocating new cryptocurrency to each active listener during a day and then distributing the cryptocurrency to content creators based on user consumption [21].

DECENT is a blockchain based CDN for any kind of file [22]. DECENT uses IPFS for distribution. DECENT uses a custom blockchain with delegative proof of stake (DPoS) for mining. DPoS essentially has peers vote for miners using their cryptocurrency and these elected miners create blocks. DECENT creates a rating system for content that cannot be manipulated like normal online reviews. DECENT allows for custom tokens allowing content creators to create new methods of monetization.

# Chapter 3: Background

## 3.1 Blockchain

A blockchain is a database that is securely stored and updated by members of a P2P network. The first example of a blockchain is the Bitcoin blockchain which was described by Satoshi Nakamoto in 2008 [23]. The purpose of the Bitcoin blockchain is to manage the Bitcoin cryptocurrency. A cryptocurrency is a digital currency that uses cryptography for security [24].

Blockchains are made of blocks of transactions which are chained together through a cryptographic hash of the previous block. The blocks contain a cryptographic hash, a nonce, a timestamp, and a set of transactions. The nonce is a random number used in the mining process. The cryptographic hash of the previous block establishes a sequential order. A diagram of a blockchain is shown in Figure 4.

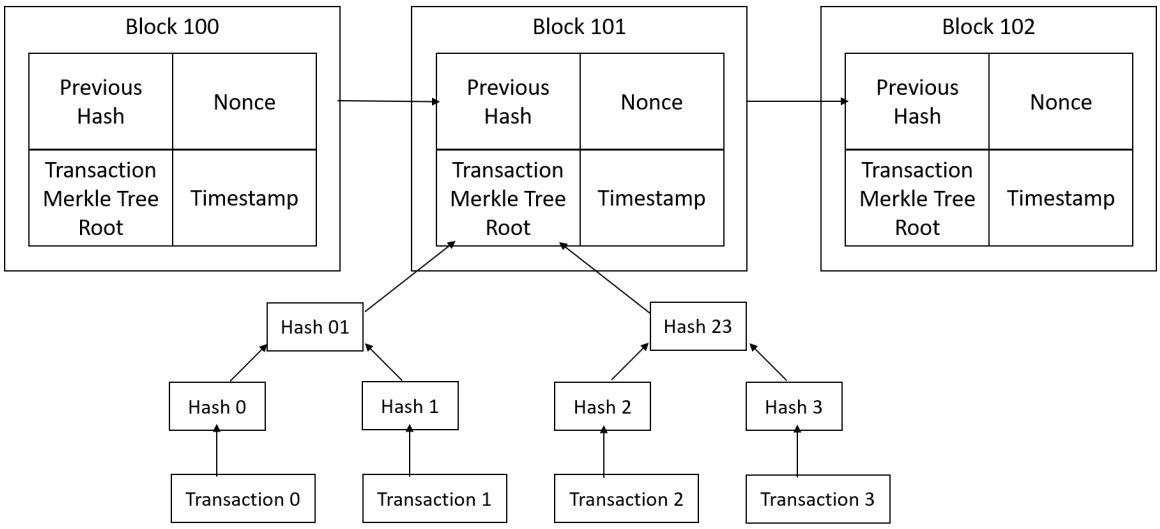


Figure 4: Bitcoin Blockchain

The accounts on the blockchain are called wallets which are a public/private key pair. The private key signs transactions to prove that the owner of the sending wallet is making the transaction. All the transactions are public, but anonymity is considered established through not knowing the owner of each wallet [23]. The Bitcoin peer network is an unstructured P2P network run by all nodes. Nodes use a list of previous nodes for peer discovery, and if the previous nodes do not lead to peers they contact one of multiple ”DNS servers” for peer discovery. The DNS servers are like the BitTorrent tracker system.

The process of generating the blocks of the blockchain is called mining and a peer who mines a block is known as a miner. Before mining a block, the miner receives and verifies transactions. If it receives too many transactions to fit in one block, the miner decides which transactions will be in the block it tries to mine. To mine a block of transactions a miner uses the proof of work scheme. The miner guesses a value for the block's nonce so that when the block is hashed it will be less than a threshold value [23]. The threshold

value is adjusted to change the difficulty if blocks are being mined either too slowly or too quickly compared to a targeted time between each block [23].

When a miner creates a valid block whose hash is less than the threshold value it is broadcasted to the peer network. The peers check the validity of the transactions and the hash of the block. The mining process is expensive to create blocks and cheap to verify blocks, requiring many hash operations to create a block but only one to verify the block. Mining is incentivized by giving miners Bitcoin for each block mined, by generating new Bitcoin and paying transaction fees [23]. The maximum number of Bitcoin generated is 21 million and the amount of new bitcoin decreases as the number of blocks increases [23]. Peers in the blockchain network assume the longest blockchain is the correct blockchain, which is important for security [23].

The security of the blockchain relies on multiple assumptions. The first assumption is that public key cryptography is secure. An attacker that can calculate others' private keys can create fake transactions. The second assumption is that the cryptographic hash algorithm is secure. An attacker that can generate a new block whose hash matches an existing block can replace the block compromising the integrity of the blockchain. The third assumption is that an attacker will not possess more processing power than all the honest miners, which is known as a 51% attack. Attackers could censor or delay transactions by not placing them in their blocks or generate an alternative blockchain that tries to replace the honest blockchain. The fourth assumption is that a miner must use brute force methods to mine a block [25]. If a miner can use a more efficient method they could perform a 51% attack more easily.

The risk of a malicious blockchain replacing the legitimate blockchain is mitigated by

economic incentive [23]. If a fake blockchain replaces the honest blockchain users will abandon the network. Once the network is abandoned the economic value associated with the accounts in the blockchain will be significantly reduced. This will reduce the reward an attacker will receive. This economic disincentive will deter economically rational attackers. Attackers who are economically irrational and desire to destroy the legitimate blockchain will not be deterred.

## 3.2 Ethereum

Ethereum is an open source blockchain being developed by the Ethereum Foundation [26]. The Ethereum blockchain builds upon the concepts of the Bitcoin blockchain by creating an environment for executing code stored in the blockchain. The Ethereum blockchain database stores values representing the Ether cryptocurrency and programs called contracts which have been compiled to bytecode. Ethereum uses public/private key wallets. The blocks of the Ethereum blockchain store a hash of the previous block, a nonce, a timestamp, transactions, the bytecode of contracts created in the block, and other fields [27].

The contracts stored on blockchain are like classes in object oriented programming. Contracts contain state variables, which are stored on the blockchain and functions which have access to the state variables. State variables can only be modified by a function inside the same contract but can be viewed by anyone with access to the blockchain [26]. Functions modify state variables, call functions of other contracts, or send Ether [26].

Many contracts currently on the public Ethereum network create tokens. A token is a digital currency whose account balances are managed by the contract. Another use of

contracts on the public Ethereum network is Cryptokitties. Cryptokitties are virtual cats stored on the blockchain with unique genomes that can be bought, sold, and bred.

Contracts in the blockchain are executed in the Ethereum Virtual Machine (EVM) [26]. The EVM creates a consistent execution environment among network peers. The EVM does not have access to networking, storage, or other processes, increasing security [26]. Contracts are written in Solidity, a Turing complete programming language. The Solidity source is compiled into EVM bytecode. The compilation into EVM bytecode minimizes the size of each contract to avoid bloating the blockchain [26].

Contracts can be programmed, deployed, and executed by anyone. A contract is deployed to an address on the blockchain, allowing it to receive transactions. Ether is paid to deploy a contract to the blockchain and the amount depends on the size of the bytecode. The Ether fee maximum, which can change over time, limits the size of bytecode which can be deployed. Contracts are executed when they receive a transaction called a message call. Owned contracts are a common security feature allowing a single account to execute some or all its functions. All full Ethereum nodes validate the state of the blockchain. So, when contracts are executed on the blockchain all full nodes run the code in the EVM.

The EVM uses the data portion of the message call to execute a function. Ether in the message call pays for the execution of the contract [26]. The transaction limit for each block in Ethereum is based on computational complexity of the transactions in the EVM. The computational complexity is measured in units called gas. Each computation during the execution of the contract is charged, to prevent denial of service attacks using infinite loops to stall miners. The attack could prevent the miner from making a block or prevent other contracts from being executed in the current block. The charge assigns a price to the

attack, and execution halts when the message call runs out of funds. Once halted, changes to state variables will be reverted [26].

The EVM has a maximum call stack height of 1024 to make it deterministic and independent of the implementation language. Each miner needs the same answer when executing the contract, requiring the EVM to be deterministic. Contract execution fails when stack height is exceeded [26]. The stack height introduces a security vulnerability. An attacker can build the call stack before calling another contract causing it or a contract it calls to fail [26]. The effect of the failure depends on the contract attacked. The call stack vulnerability can be prevented through proper contract programming.

The mining process of Ethereum uses proof of work. Proof of work in Ethereum differs from Bitcoin proof of work and is ASIC resistant [26]. ASIC mining leads to centralization of a decentralized system and weakens the benefits of building a distributed system [26]. The Ethereum proof of work requires a large, currently 1 GB, directed acyclic graph during the mining process. The nonce needs a small part of the DAG and miners either generate a portion of the DAG during each mining attempt or store the entire DAG in RAM [26]. Generating pieces of the DAG for each nonce is too expensive to effectively mine, requiring storage of the DAG in RAM [26]. RAM I/O limits benefits of ASICs because the nonces for mining blocks are randomly distributed in the DAG and RAM is the most efficient way to randomly access data.

Verification is cheap for clients, requiring a small portion of the DAG which can be generated. The DAG changes every 30000 blocks and is based on the block height allowing pregeneration of the DAG. The DAG change reduces the speed up from shared memory architecture. The DAG is pregeneratable, so generation does not stop mining temporarily

[26].

Mining is incentivized with new Ether and fees paid for transactions and contract execution. There is no limit to the total number of Ether. A block that is mined, and could have been a valid block, but was mined after another valid block is called an uncle. Figure 5 shows uncles in Ethereum. Miners who generate uncle blocks are given a partial reward in Ethereum, unlike in Bitcoin. The uncle reward prevents centralization risks with a short time between blocks [28].

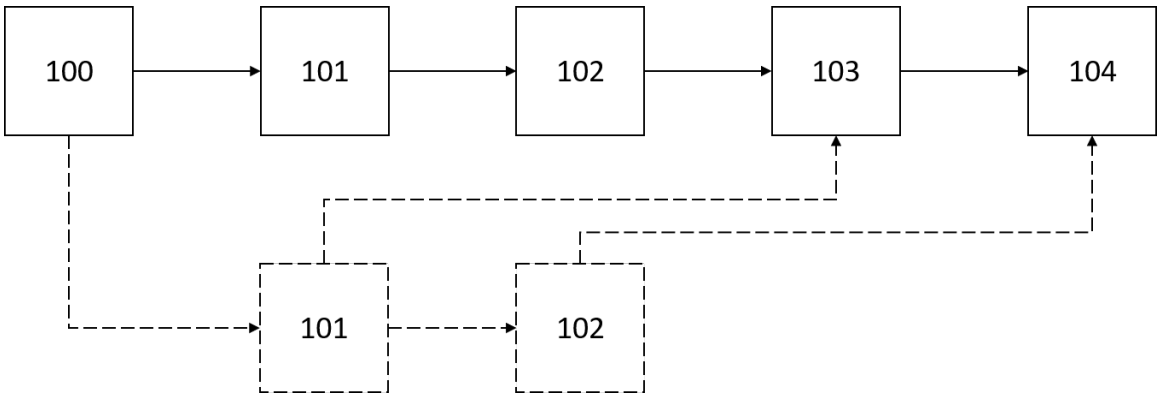


Figure 5: Uncles in Ethereum

### 3.3 CDN

CDNs for delivering DRM protected content are made of three services: the merchant service, the rights management service, and the content delivery service [4]. Some examples of large CDNs with DRM are Netflix, Amazon Video, and iTunes. The merchant service sells digital rights. First the merchant service confirms payment and then it tells the rights management service to update the user's database entry. The rights management

service maintains the digital rights database. The rights management service sends license keys to clients, so they can request and decrypt content. The content delivery service delivers content to authorized users. Content delivery in a CDN relies on geographically diverse servers to minimize delivery time. CDN services can be operated by multiple parties. For example, the content delivery service can be outsourced. Many ISPs operate CDNs with servers inside their network so companies can quickly deliver content to their customers. If all services, merchant, rights management, and content delivery are P2P the CDN will not contain a single point of failure.

# Chapter 4: Design

## 4.1 System Design

The design goal of this thesis is to make a P2P CDN that benefits both consumers and content producers. Consumers want to purchase content knowing that they will not have their ownership revoked in the future. Content producers want to sell their works without maintaining servers. The CDN uses a blockchain to maintain the consumer's rights to works indefinitely. Through P2P filesharing the CDN distributes content creators works in a low cost manner and gives consumers indefinite access. An overview of the system is shown in Figure 6.

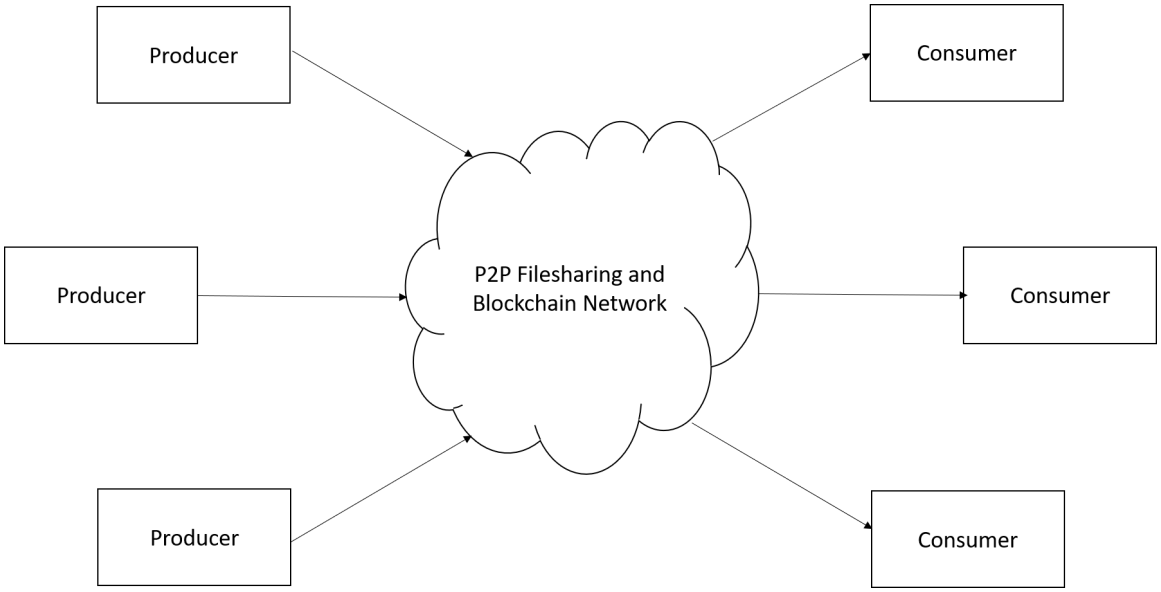


Figure 6: System Overview

The content distribution network will use an Ethereum blockchain to operate the merchant and rights management services. The content distribution network will use a P2P network like the BitTorrent tracker system as the content delivery service. An Ethereum blockchain will securely store digital rights and allow programs to execute securely. This makes the merchant and rights management services completely P2P. It is impractical and insecure for the content delivery service to store the content in the blockchain. Blockchain size will increase to an impractical level, leading to blockchain storage centralization. And any peer who has the blockchain will have every piece of content. Instead, the content will be stored in a P2P file sharing network. The blockchain will be used by network peers to restrict content access so peers cannot store files that they do not own. The content creator will have to serve the file until it is purchased and downloaded by peers. In this system content creators use a content packager to publish their content to the blockchain and the P2P filesharing network. Consumers pay the content's contract to gain access rights. Consumers then request the content from the P2P filesharing network. When content has been received and verified the user can view the content.

## 4.2 The Blockchain

The merchant service and rights management service will be implemented using contracts on a private Ethereum network so the blockchain will only be for this system. Calling contract functions requires the contract's address and interface. A header and hub system was implemented to assist in contract function calls. The header contract stores the filename, hub address, and hub interface. The header is standardized so the interface does not change for each content, allowing the interface to be known in advance. The header

information is used to interact with the hub contract. The hub stores addresses and interfaces of the license mint contract, the verification contract, and the index server directory contract. The license mint contract stores the digital rights, the price to gain rights, and is used to purchase digital rights. The verification contract stores the SHA-256 hash to verify received files. The index server directory contract stores information required to communicate with index servers to find P2P filesharing peers, allowing for index servers to be added or removed.

There were multiple iterations of the systems deployed on the blockchain during development. The first system was insecure. The second system was a secure system. The third and final system is a more efficient and secure system.

**4.2.1 Insecure System** The initial system allows content creators to publish content and deploy contracts. The content creators create the content and write the contracts. The system is described in Figure 7. This system is insecure, as content creators can write flawed or malicious contracts. Using insecure contracts causes unintended results, such as the consumer paying too much or not gaining access rights.

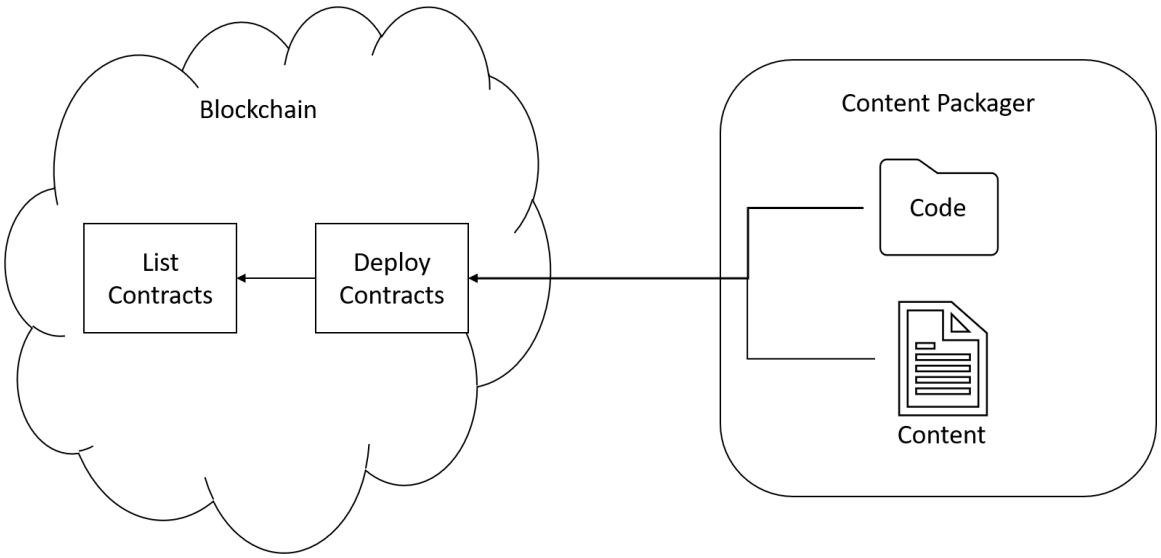


Figure 7: Insecure System

**4.2.2 Secure System** The secure system does not allow content creators to write their own contracts, to prevent flawed or malicious contracts. This requires a trusted third party to create the contracts to ensure consistency of contract implementations across content creators. The third party creates factory contracts for use by content creators, shown in Figure 8. A factory is a type of contract whose job is to create other contracts. To reduce the trust required, the factory stores its source code on the blockchain. The content creators call a factory function to create new contracts for their content, shown in Figure 9. The factory addresses are listed on a factory lookup contract, helping content creators locate every factory. Factory produced contract addresses are placed on a contract lookup, helping consumers locate every content's contracts. The interface of each contract created by the same factory will be identical. To improve space efficiency, interfaces are stored in the interface lookup contract and are given an interface ID. The header and hub store the

interface ID, using the interface lookup to retrieve the interface.

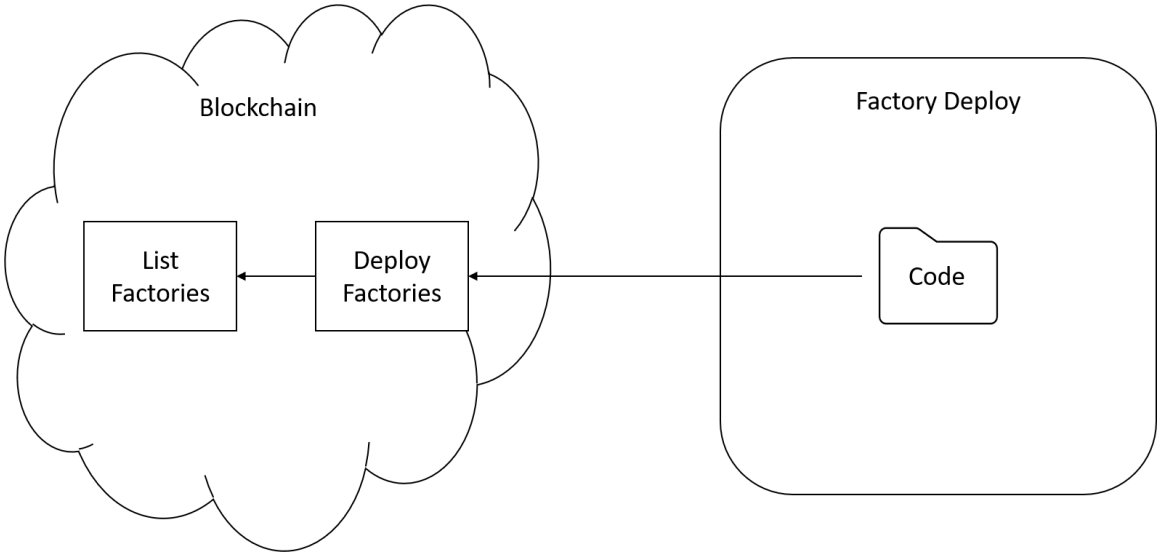


Figure 8: Secure System - Factory Creation

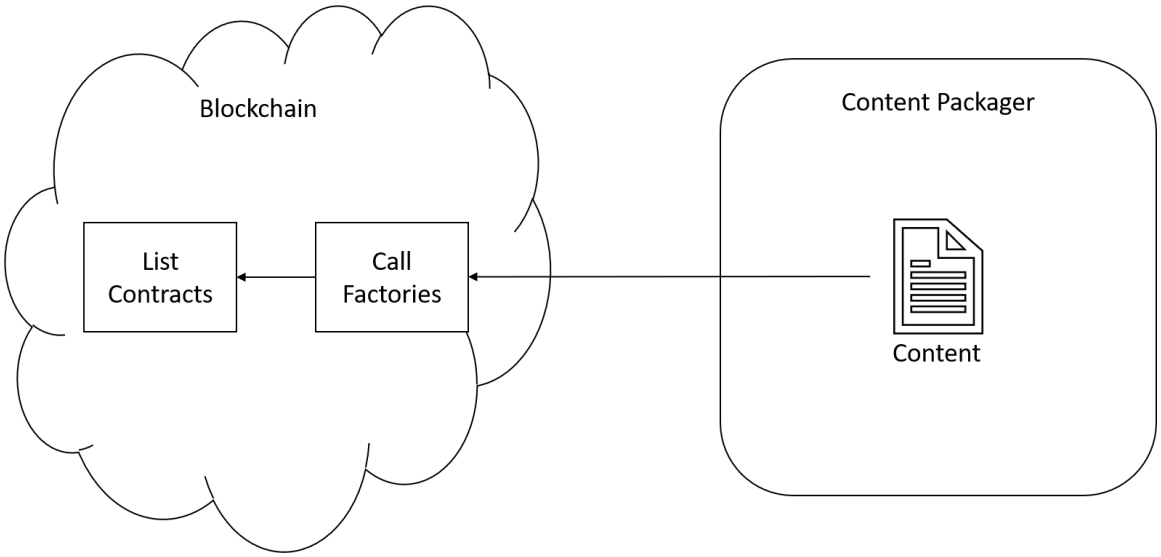


Figure 9: Secure System - Content Publishing

**4.2.3 Efficient System** The efficient system is secure and more space efficient. The secure system wastes storage by creating identical bytecode each time a factory creates a contract, illustrated in Figure 10. The efficient system removes the replication of bytecode, shown in Figure 11. Content license contracts, which store data for multiple contents, are used instead of factories. The trusted third party creates the content license, shown in Figure 12. The content license stores its own source code. The content license combines the functionality of the license mint, verification, and index server directory contracts. Content producers call a content license function to add the required data for their content, shown in Figure 13. The amount of storage saved on each node with the efficient design is shown in equation 4.1. The calculations to get equation 4.1 are in the appendix.

$$kb_f + kd_f + nb - kb \tag{4.1}$$

with

$n$ = number of content

$k$ = number of license standards

$b$ = size of bytecode

$b_f$ = size of factory bytecode

$d_f$ = size of factory state variables

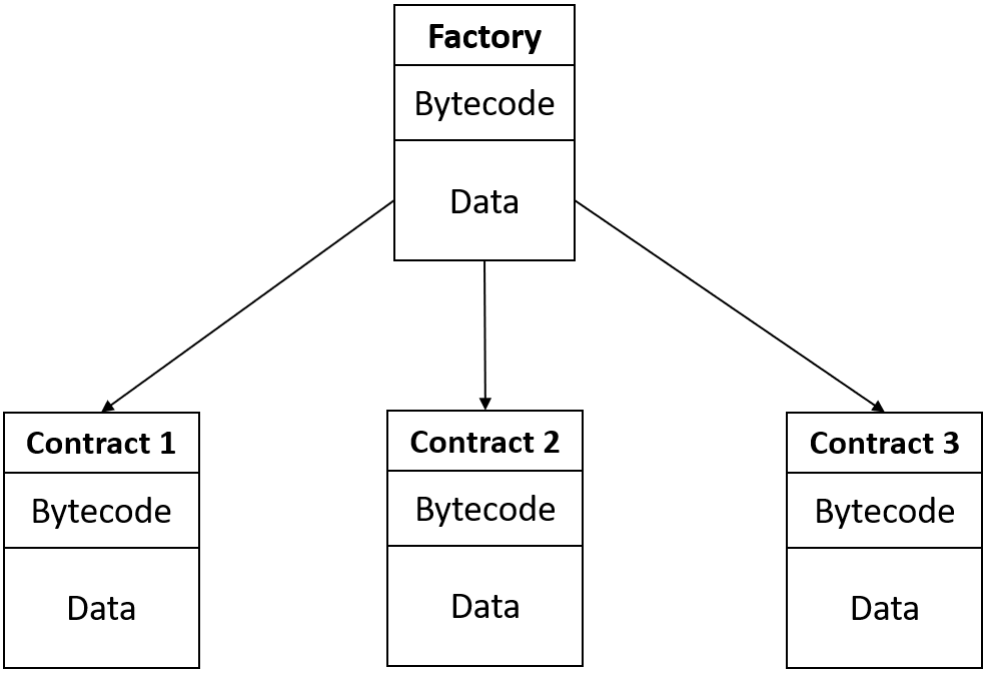


Figure 10: Naive Design with Redundant Bytecode

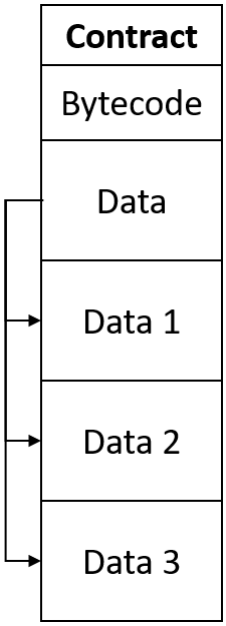


Figure 11: Efficient Design Eliminating Redundancy

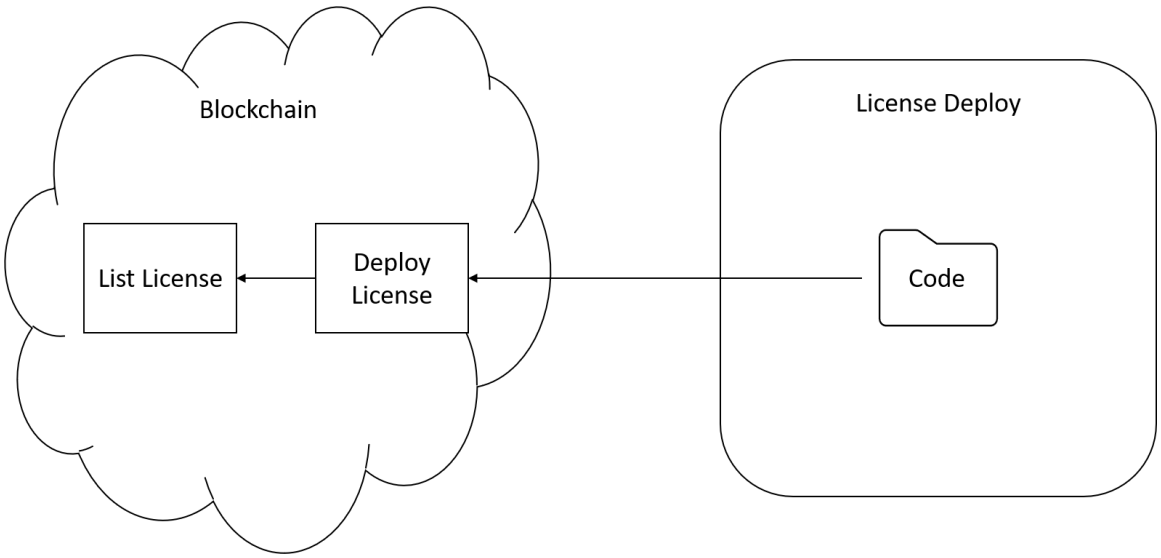


Figure 12: Efficient System - License Creation

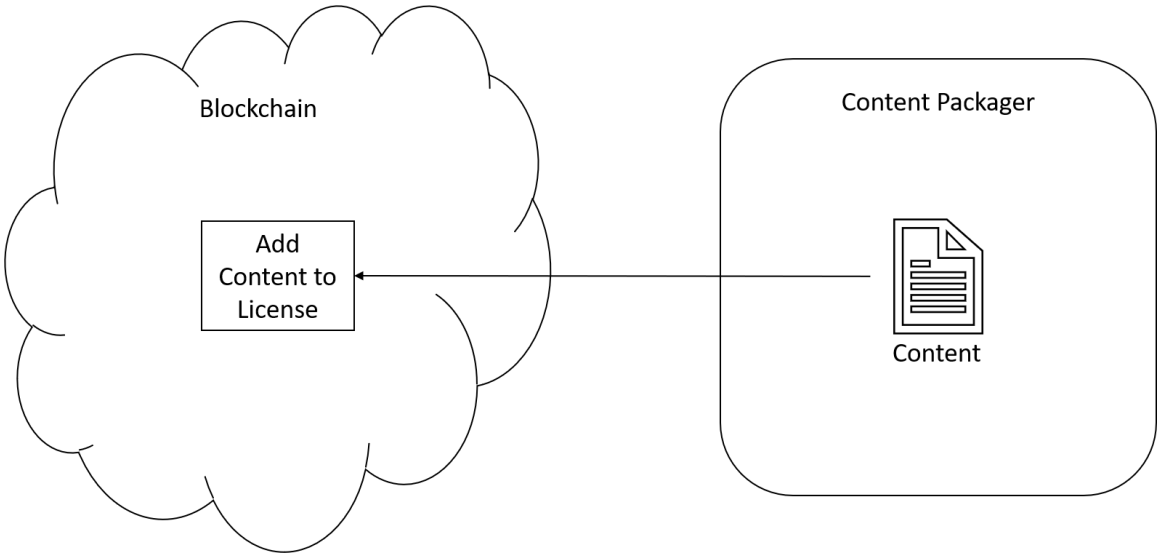


Figure 13: Efficient System - Content Publishing

### 4.3 P2P Filesharing Network

The content delivery service's P2P filesharing network is like the BitTorrent tracker system. Peers in the network rely on index servers for peer discovery. Any peer can run an index server, listing it individually on the blockchain for any number of contents. There is not a limit on the number of index servers per content. Serving peers list themselves on the index server. Downloading peers get a list of peers serving the file from the index server. The sequence of activities involving the index server is shown in Table 2.

Table 2: Index Server Activity

| | |
|---|---|
| 1 | Index Server lists itself for the file on the blockchain |
| 2 | Index Server starts index for the file |
| 3 | Producer lists themselves on the file's index |
| 4 | Consumer asks Index Server for servers |

The peer list is used to request content from peers. Files are transferred if the requester has rights on the blockchain. When requesting content, peers include their public key from the blockchain. Peers prove they own the public key by signing a nonce chosen by the serving peer. The serving peer validates the signature and checks if the public key has rights on the blockchain. The serving peer then sends the file to the requester. The file is not chunked like in BitTorrent in this prototype. The content request process is shown in Figure 14.

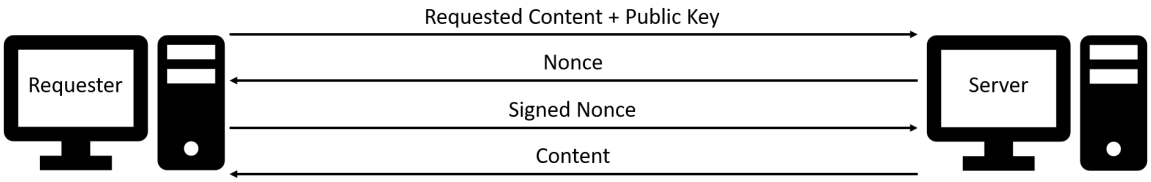


Figure 14: Content Request from P2P Network

Received files are validated by comparing the SHA-256 hash to the stored hash on the blockchain. The P2P filesharing network uses hypertext transfer protocol (HTTP) to transmit information and files. HTTP is used for its wide adoption and available libraries. The peer list is transmitted in JavaScript object notation format (JSON). JSON libraries enable easy packing and unpacking of transferred data. The sequence of events for a consumer to get the file from a producer is shown in Table 3.

Table 3: Sequence of Consumer getting content from Producer

| | |
|---|---|
| 1 | Producer places content data on the blockchain |
| 2 | Index Server lists itself for the file on the blockchain |
| 3 | Index Server starts index for the file |
| 4 | Producer lists themselves on the file's index |
| 5 | Producer begins serving file |
| 6 | Consumer pays for access rights (blockchain) |
| 7 | Consumer asks Index Server for servers |
| 8 | Consumer asks Producer for file |
| 9 | Producer serves to Consumer |
| 10 | Consumer lists themselves on the files index |
| 11 | Consumer begins serving file |
| 12 | Consumer2 pays for access rights (blockchain) |
| 13 | Consumer2 asks Index Server for servers |
| 14 | Consumer2 asks Producer or Consumer for file |
| 15 | Producer or Consumer serves to Consumer2 |

## 4.4 Content Protection

Content protection relies on encryption to prevent content from being accessible to attackers and uses watermarks to locate the source of a leak. Secret keys are not amenable to a fully P2P CDN. Key servers create a central failure point, which could prevent content from being viewed by peers with rights. A key stored on the blockchain is insecure, as it is accessible to every peer who has the blockchain. Giving authorized peers the key will result in a leak allowing anyone to decrypt the content. Individualized watermarks change the cryptographic hash of a file for each peer preventing content validation.

**4.4.1 Whitebox** Whitebox cryptography is used to protect keys in software applications that are executed on untrusted devices [29]. Whitebox cryptography combines the secret key into the cipher in an obfuscated manner. Whitebox ciphers allow the content to be stored in an encrypted form. Ciphers should be generated for each content or a limited number of contents. Individual ciphers can make key theft economically irrational compared to purchasing content. If all contents share the same cipher, key theft will have a greater reward and may be economically rational. The cipher may delay key theft, protecting content creators' revenue streams. The SHA-256 hash of the cipher should be stored on the blockchain and the ciphers stored in the P2P filesharing network to prevent blockchain centralization. Cipher access should only be given to rights holders, so key thieves must purchase the content, to further reduce incentive for key theft. The use of whitebox ciphers may not be beneficial. Even when decrypted in small portions the content will be vulnerable to leak. Using systems like HDCP to protect the content in transit to the display may prevent indefinite access to content. When HDCP is no longer supported by computers the

system to display the content will no longer operate unless it is updated. To update the system the secret key would be required to rebuild the whitebox, leading to the initial problem the whitebox was trying to fix. Because of the long term support issues of whitebox cryptography it is not included in this design.

**4.4.2 Non-Encryption Protection** Service superior to pirated networks protects content by incentivizing consumers to use legitimate services. Service quality can be increased through economic incentive. Paying index and peer servers incentivizes uptime and serving peers, increasing quality and quantity of service. Consumers paying for peer discovery and file delivery will not increase the overall cost. Server costs are factored into the content prices when purchasing from regular CDNs like Netflix or iTunes. Since content producers do not have to run servers the cost to purchase content will decrease. Server cost efficiency may increase for some content and decrease for others. Centralized CDNs use economies of scale to reduce costs and increase efficiency, the loss of economies of scale will decrease efficiency. Competition drives innovation and reduces cost, and the competition among peers may reduce costs and increase efficiency. Content served by large distributers may have the cost of delivery increase as competition's benefit is outweighed by the loss of economies of scale. Content served by small distributers may see the cost of delivery decrease as the loss of economies of scale is outweighed by the effect of competition.

# Chapter 5: Implementation

## 5.1 Implementation

The system design previously described was implemented to validate it and measure its performance in a realistic scenario. The version of Ethereum used in the implementation is Geth 1.7.2, with Solidity 0.4.11 used to write contracts. These are not the most recent versions of Geth or Solidity. Ethereum is currently under development with new versions being released frequently. The version was pinned to stop updates from breaking software. The version of Python used in the implementation is 3.6.3 so the secrets module could be used for secure random numbers. Flask is used for the HTTP servers. The version of Flask used is 0.12.2. The testbed interacts with Amazon Web Services, and the API module used is Boto3 1.4.7. An overview of the software used is in Table 4.

Table 4: Software Versions Used

| | |
|---|---|
| Geth | 1.7.2 |
| Solidity | 0.4.11 |
| Python | 3.6.3 |
| Flask | 0.12.2 |
| Boto3 | 1.4.7 |

## 5.2 Test Environment

**5.2.1 Simulated Environment** The environment being simulated is consumers using extra resources on their computer (or old computers) to serve files. Peers have 2 core processors with moderate clock speeds. Peers have small hard drives to represent extra storage or smaller disk machines. Index servers are dedicated home computers with more resources than consumer peers. Miners are dedicated computers with larger amounts of RAM and processing than other peers. Peers are assumed to be running in private residences with low to moderate speed network connections. Contracts are on the blockchain prior to testing because the creation of the contracts would not affect most consumers buying content.

**5.2.2 Testbed** The testbed is on AWS because of the free credits they give to students and previous experience with their cloud. The testbed has a dedicated machine for the miner and bootnode. The other machines are producers, consumers, or index servers. The machines are networked in an Amazon Virtual Private Cloud and are not open to the public internet, aside from SSH connections. An overview of the testbed is shown in Figure 15.

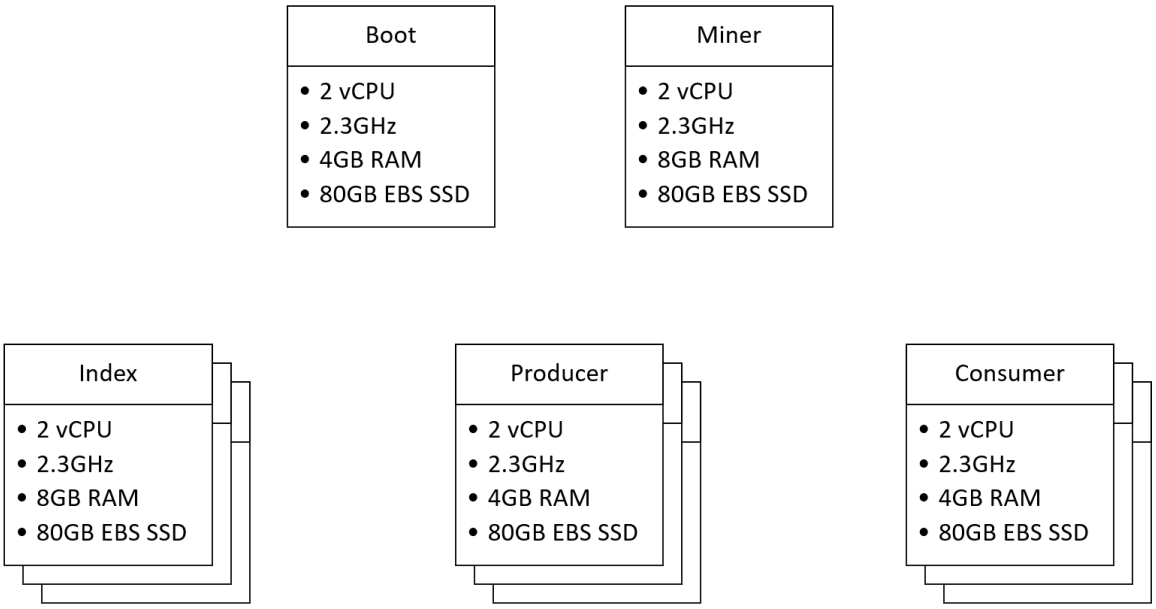


Figure 15: AWS Testbed

Each machine runs a program that simulates the behavior of its user. The consumer simulator purchases every content listed on the blockchain in random order and requests content immediately after purchase. The consumer simulator runs a peerclient to download files and a peerserver that serves downloaded files. The producer simulator generates random files and lists them for purchase on the blockchain. After the producer simulator has created the files it starts a peer server to serve its newly created files to the network. The random files are 5.5 gigabyte blank files with a small amount of random data appended at the end to prevent hash collisions. The files are transferred uncompressed, and the size of 5.5 gigabytes was chosen as a middle ground between small files and large files that have been compressed.

The index simulator manages indexserver data for each file on the blockchain, while

checking for new content on the blockchain periodically. The miner simulator will run the miner, so data can be added to the blockchain. The programs need Ether to append data onto the blockchain or purchase content. The bootnode simulator runs a bootnode and sends Ether to peers for testing. The bootnode was arbitrarily chosen to send Ether for testing.

Peers needing Ether send messages with their Ethereum addresses to the bootnode using Amazon's simple queue service (SQS). SQS stores a queue on an AWS operated server, so peers can push items onto or pull items from the queue over a network connection. The bootnode reads $n$ messages, with $n$ being the number of peers requiring Ether, this is required for testing and is not part of the system. The mining difficulty was minimized to reduce resources required for testing. This should not affect the results since the difficulty is automatically adjusted to meet the targeted block interval of 17 seconds, and blocks were mined in advance to create the Ether sent out to peers for testing.

# Chapter 6: Results

## 6.1 Data Gathering Method

All the timing data is gathered from the consumer simulators and focuses on the consumer experience. The data was gathered by pushing the current time onto an AWS SQS queue at the start and end of each task. After each test, the messages were gathered from the queue and the time to complete each task was calculated. The time reported is the average time to complete each task during a test. Multiple tests were run with an increasing number of consumers to test the scalability of the system. The test configurations are shown in Table 5. In every configuration a producer generates and serves 1 file. There is one Bootnode in each test to coordinate the Ethereum clients. There is one miner in each test to produce blocks. There is one index server in each test to coordinate P2P filesharing peers. The program was not ran at scales larger than those in Table 5 because of the costs to run tests.

Table 5: Test Configurations

| BootNodes | IndexServers | Producers | Consumers | Miners | Cost/Hour |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | $0.45 |
| 1 | 1 | 10 | 10 | 1 | $1.69 |
| 1 | 1 | 10 | 20 | 1 | $2.37 |
| 1 | 1 | 10 | 40 | 1 | $3.74 |
| 1 | 1 | 10 | 80 | 1 | $6.48 |
| 1 | 1 | 10 | 160 | 1 | $12.00 |
| 1 | 1 | 10 | 237 | 1 | $17.24 |

## 6.2 Results

The total time it takes to purchase and completely download a file was measured. To better characterize the performance of the prototype implementation, the components of total time are also measured these are purchase time, get index server time, get peers time, and download time. For analysis, the components are combined, as shown in Table 6, to produce the combined metrics of query time, blockchain time, pre download time, and total time.

Table 6: Combined Metrics

| | Purchase | Get IndexServer | Get Peers | Get File |
|---|---|---|---|---|
| Total Time | X | X | X | X |
| Query Time | | X | X | X |
| Blockchain Time | X | X | | |
| Pre-Download | X | X | X | |

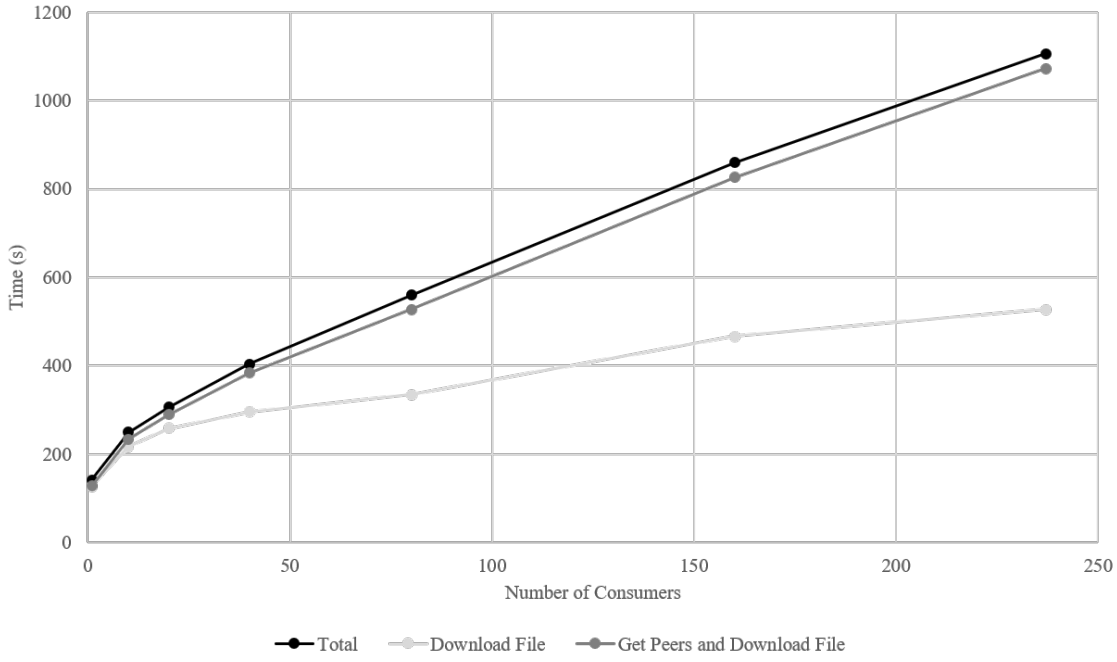


Figure 16: Total Time

Total time is the duration from when a consumer decides to get a file until they have the entire file. Figure 16 shows that total time grows linearly as the number of consumers is increased. In the small scale testing with 10 consumers the total time is just over 4 minutes (249 seconds) with 87% of the total time being download time and 6% of the total time being get peers time. At the largest scale tested total time is over 18 minutes (1105 seconds) with 48% of the total time being download time and 49% of the total time being get peers time. The total time is dominated by get peers time and download time with get peers becoming a larger portion of total time as scale increases.

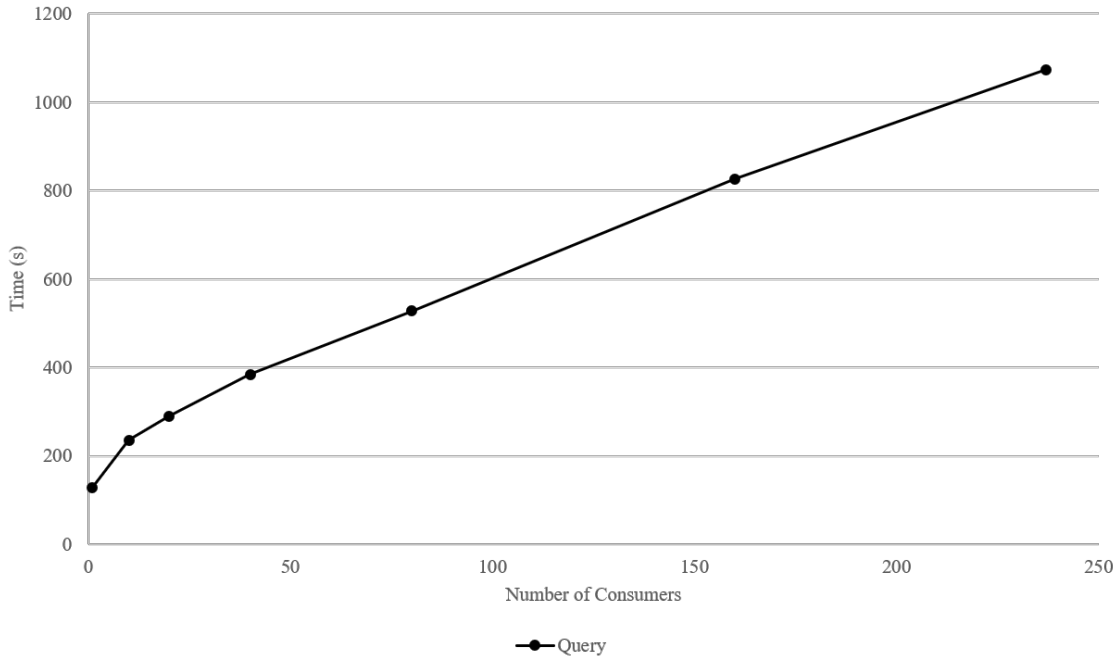


Figure 17: Query Time

Query time is how long it takes a consumer to retrieve a file they own. As shown in Figure 17 the query time increases linearly as the number of consumers is increased. Query time is like total time since they are made of the same components with total time also having purchase time. With 10 consumers query time is just below 4 minutes (234 seconds). At the largest scale tested the query time is just below 18 minutes (1073 seconds). Query time is dominated by get peer time and download time which make up over 99% of query time at each scale tested.

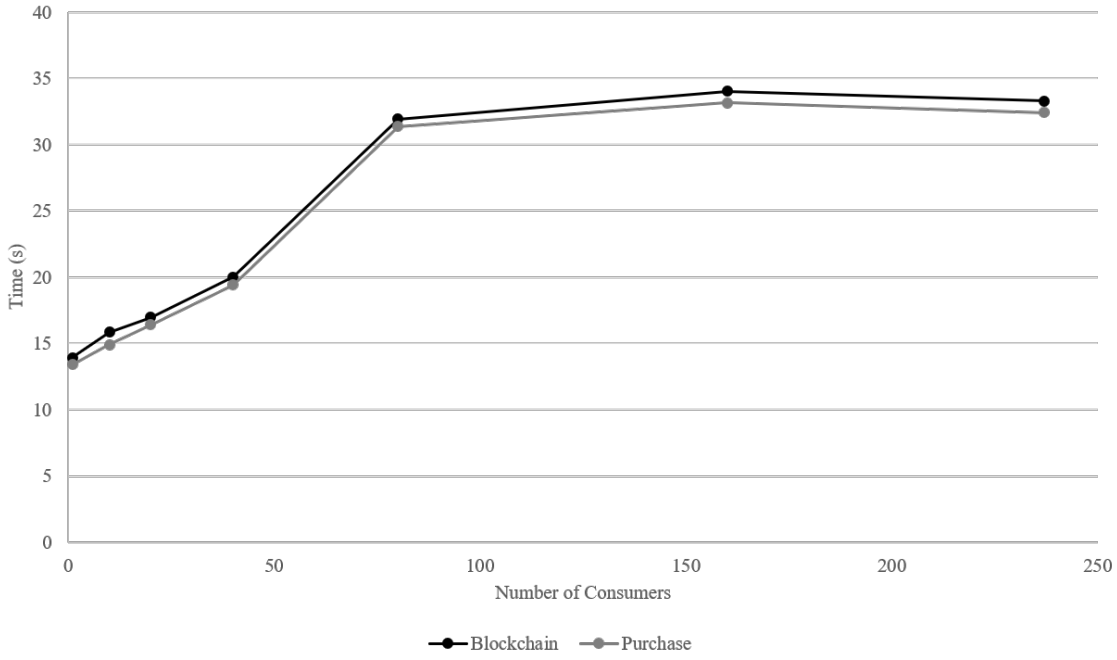


Figure 18: Blockchain Time

Blockchain time is how long it takes for the consumers blockchain read and write operations. Figure 18 shows that blockchain time increases linearly as the number of consumers increases, with the slope decreasing after the 80 consumer test. In the lower scale tests, the blockchain time is between 15 to 20 seconds. In the larger scale tests, the blockchain time is between 30 and 35 seconds. Blockchain time is dominated by the time to purchase the content, which is the only time the consumer writes to the blockchain. This confirms that writing to a blockchain takes longer than reading the blockchain.

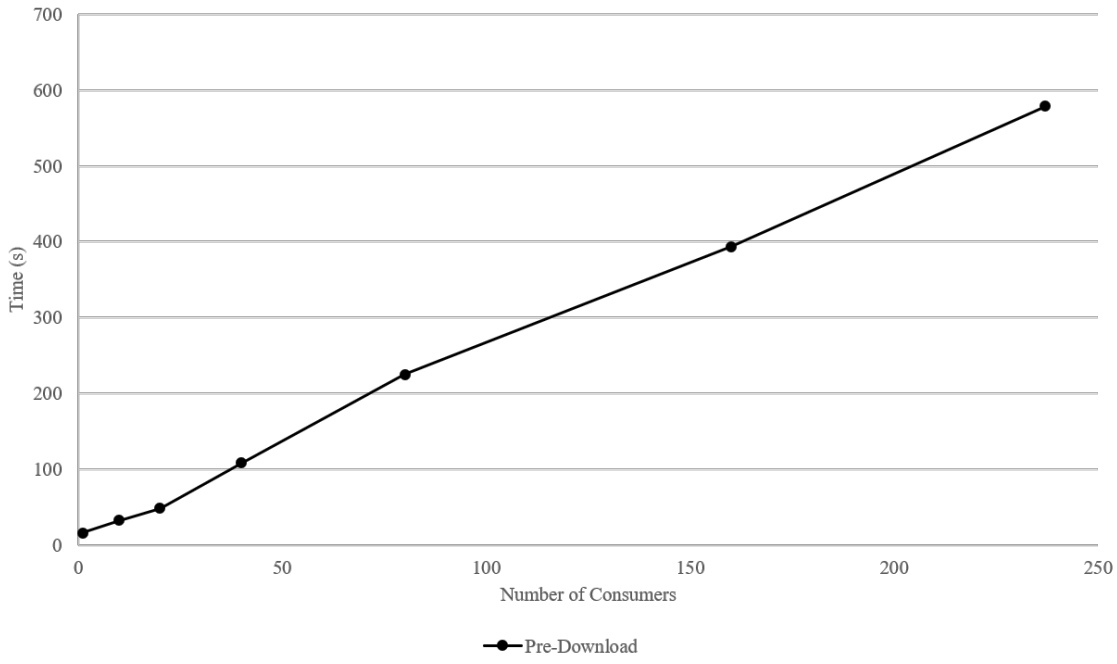


Figure 19: Pre-Download Time

Pre-download time is how long a consumer must wait before starting the file download. As shown in Figure 19 pre-download time increases linearly as the number of consumers increases. In the small scale test with 10 consumers pre-download time is 32 seconds, with 46% of the time being purchase time. In the largest scale test pre-download time is over 9 minutes (578 seconds), with 5% of the time being purchase time and 94% of the time being get peers time. As the scale is increased the get peers time becomes the dominant component of pre-download time.

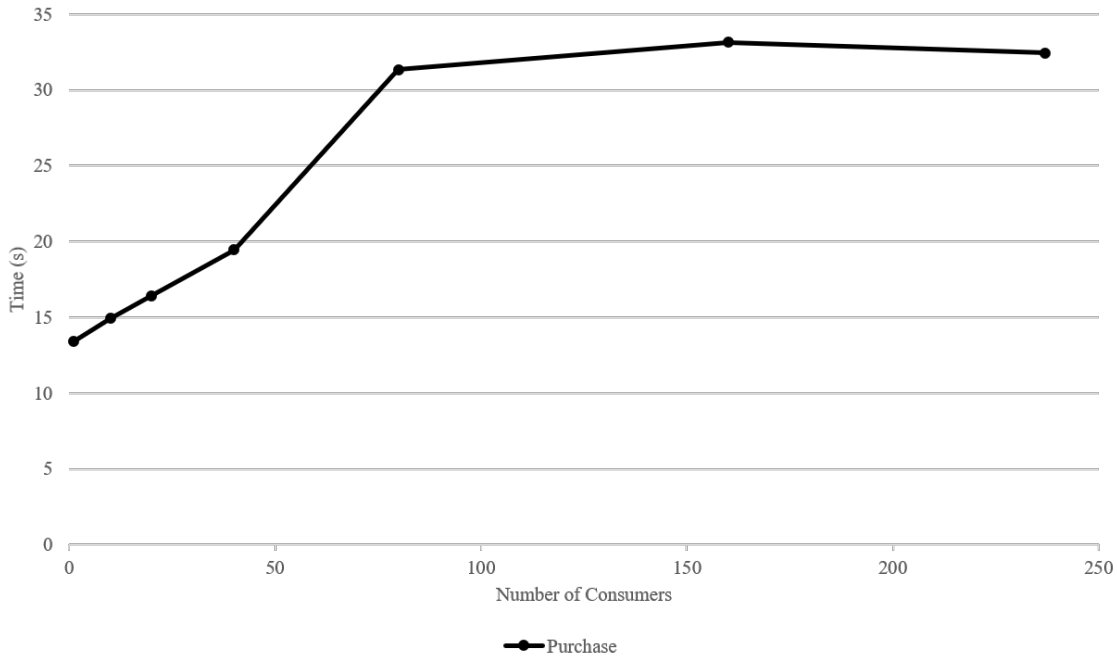


Figure 20: Purchase Time

Purchase time is how long it takes to acquire rights after a consumer decides to purchase a file. Purchase time involves transactions in Ethereum and requires the consumer making the purchase and the miner. As shown in Figure 20 the purchase time doubles as the number of consumers (and transactions) increases. Initially the purchases were included in the next block, but as the number of transactions increased it took two blocks for the purchase to be included. The decrease in slope after 80 consumers is caused by reduced parallel purchase transactions which resulted in relatively faster purchase times. Due to implementation issues, which are described later in this chapter, some consumers wait significantly longer than others for a file. This makes some consumers purchase their next file later than the others with the purchase transaction sent after other consumers transactions

have been mined.

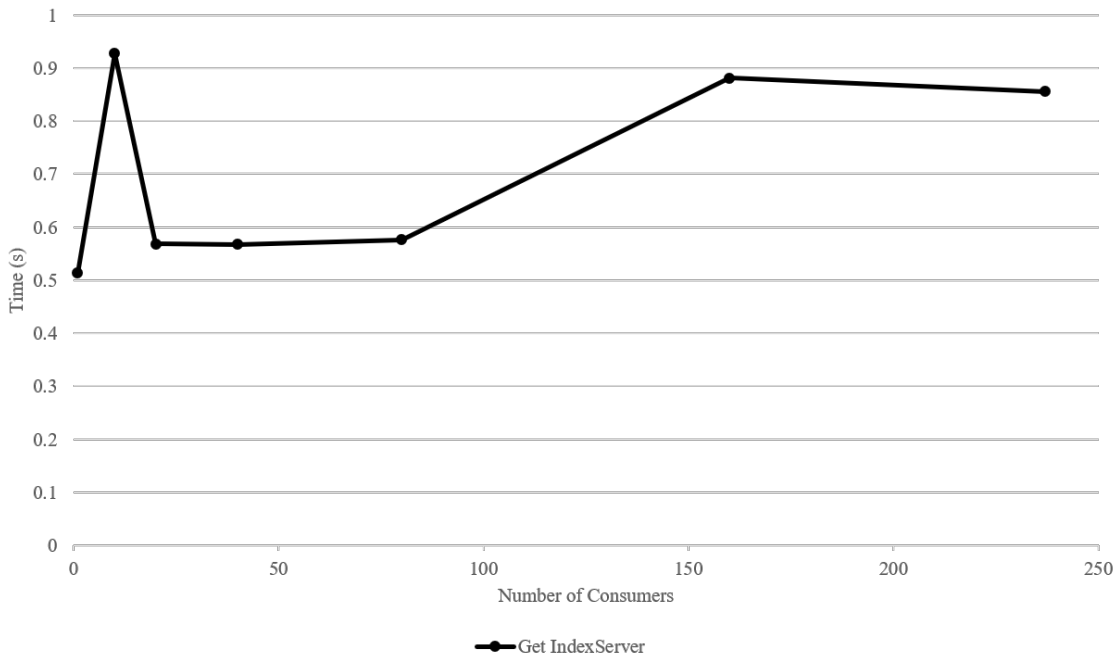


Figure 21: IndexServer Time

Get index server time is how long it takes for a consumer to get the list of index servers for a content. Get index server time is the consumer reading from their copy of the blockchain with the index server having previously sent a transaction to the miner to list themselves as index serving for the content. Figure 21 shows that the time to get the list of IndexServers is consistently under 1 second. The spike at 10 consumers is likely caused by inconsistency in the testbed not the system design. Three peers in this test had high get index times that were consistently between 2 to 3 seconds, while the other peers performed similar to peers in other tests. Get index time is not affected by test scale because it is a local disk read.

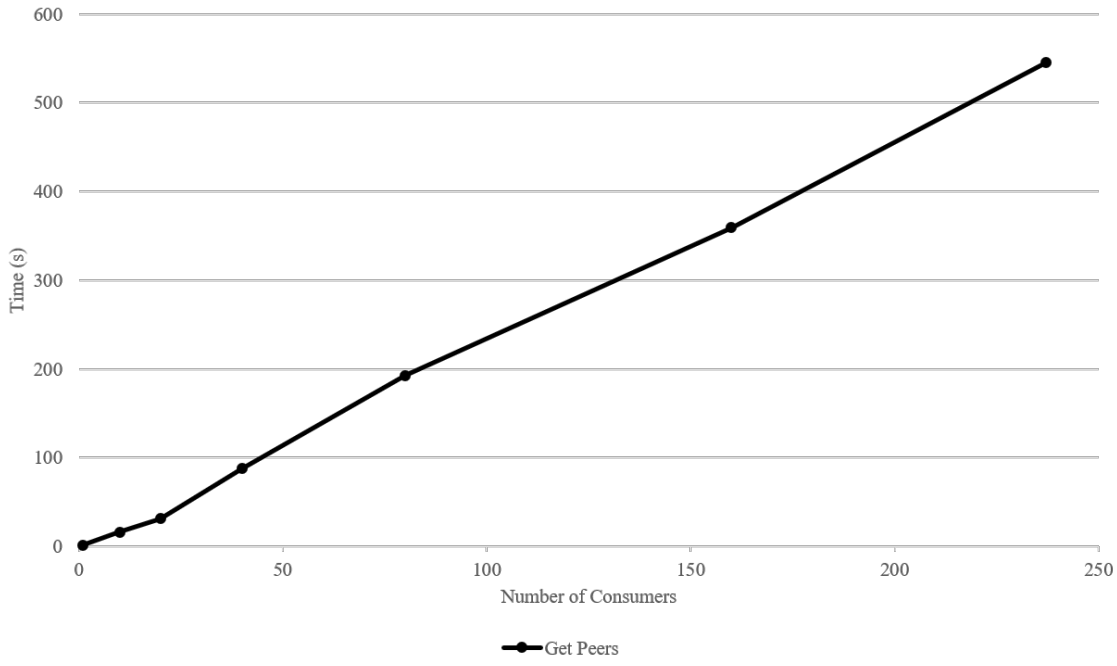


Figure 22: Peer Time

Get peers time is how long it takes for the consumer to get the list of serving peers from the index server. The peers involved in get peers time are the consumer and the index server and has no activity occurring on the blockchain. Figure 22 shows that the time to get the peer information from the index server increases linearly with the number of consumers. The poor performance of the index server is caused by the implementation and is not inherent in the concept of combining P2P filesharing and blockchain. The performance is caused by limiting the number of peers allowed to communicate simultaneously with the index server to prevent out of memory exceptions. This limit was needed since libraries used to run the index server consumed large amounts of memory to create threads when serving clients. Limiting the number of peers allowed to communicate simultane-

ously causes a line of peers waiting for routing information. This line of peers causes some peers to begin downloading much earlier than others causing them to finish downloading first. This reduces the number of consumers sending parallel purchase transactions.

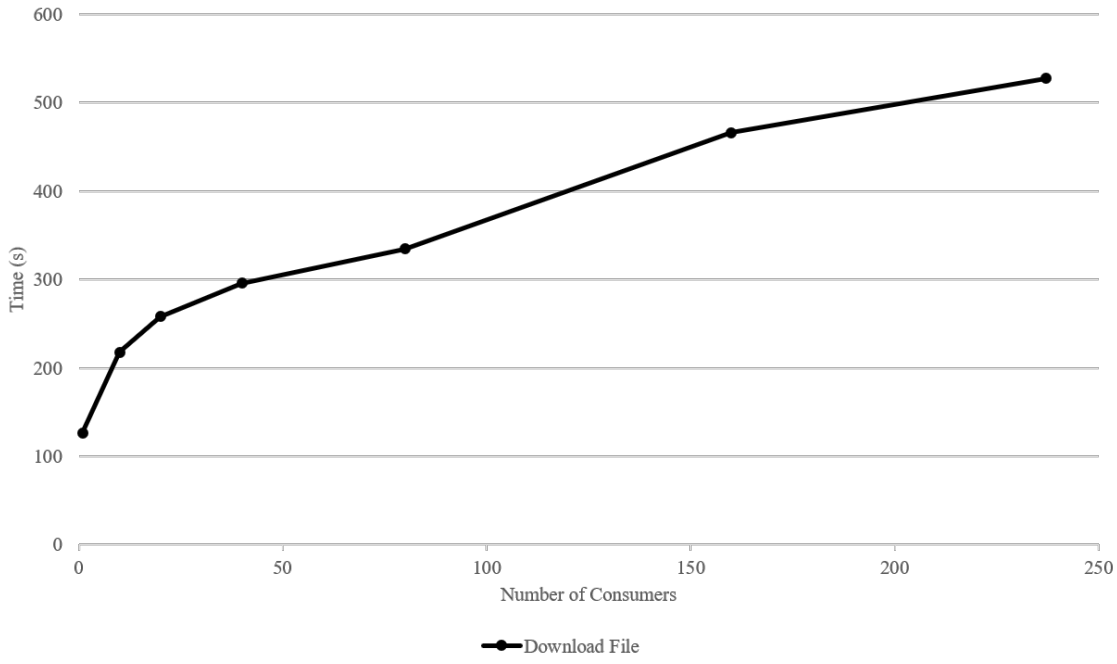


Figure 23: Download Time

Download time is how long it takes to download the file from a peer. The peers involved in download time are the consumer and a peer running a peer server, this could be the producer or a consumer who has the file. No activity occurs on the blockchain, but the peer server requires the consumer to have purchased the file. As shown in Figure 23 the download time increases linearly as the number of consumers increases. The increase in average download time is caused by increased load on the peer servers with more clients asking for files at once from each index server. The load is the worst during each con-

sumer's first download when only the producers are serving the files to all the consumers. The implementation of the P2P network does not have peers serve the chunks of file they have downloaded before the download has finished. Serving chunks would reduce the effects of increased load by having peers serve sooner.

# Chapter 7: Discussion

Given the performance of the P2P filesharing network implementation it is important to look at the theoretical performance of the blockchain. The transactions per second (tps) of the system will be used to measure the scalability. Purchase transactions are used to calculate the tps because they are the most common type of transaction and have the largest direct effect on the consumer experience. Equation 7.1 will be used to calculate tps.

$$t_s = t_b / s \tag{7.1}$$

with

$t_s$ = transactions per second

$t_b$ = transactions per block

$s$ = seconds per block

The transactions per block is unknown and the seconds per block is known with the time between blocks at 17 seconds. Equation 7.2 is used to calculate the number of transactions per block.

$$t_b = c_b / c_t \tag{7.2}$$

with

$t_b$ = transactions per block

$c_b$ = complexity per block

$c_t$ = complexity per transaction

Computational complexity in Ethereum is measured in a unit called gas. The compexity per block is the default targeted block gas limit of 4000000. The complexity per transaction is the average complexity of a purchase. Before gathering results, multiple purchases were performed and the average complexity of a purchase was 57400 gas. With the complexity per block and the complexity per transaction about 70 purchase transactions can fit in a block.

The test results confirmed the calculation when purchase time doubled between 40 consumers and 80 consumers, showing that a block was filled, and another was required to process the transactions. The purchase time data shown in Figure 20 appears to outperform the theoretical results in Figure 24 in the larger scale tests but the number of consumers is not equivalent to the number of parallel purchases. In smaller scale tests, the number of consumers was essentially equivalent to the number of parallel purchases, but as scale increased and performance issues occurred consumers were sending purchases in parallel less often weakening the relationship between the two. The stairstep pattern of the theoretical scalability is because multiple values for parallel purchases require the same number of blocks. When only part of a block is required the consumers still must wait the full 17 seconds for the block to be made.

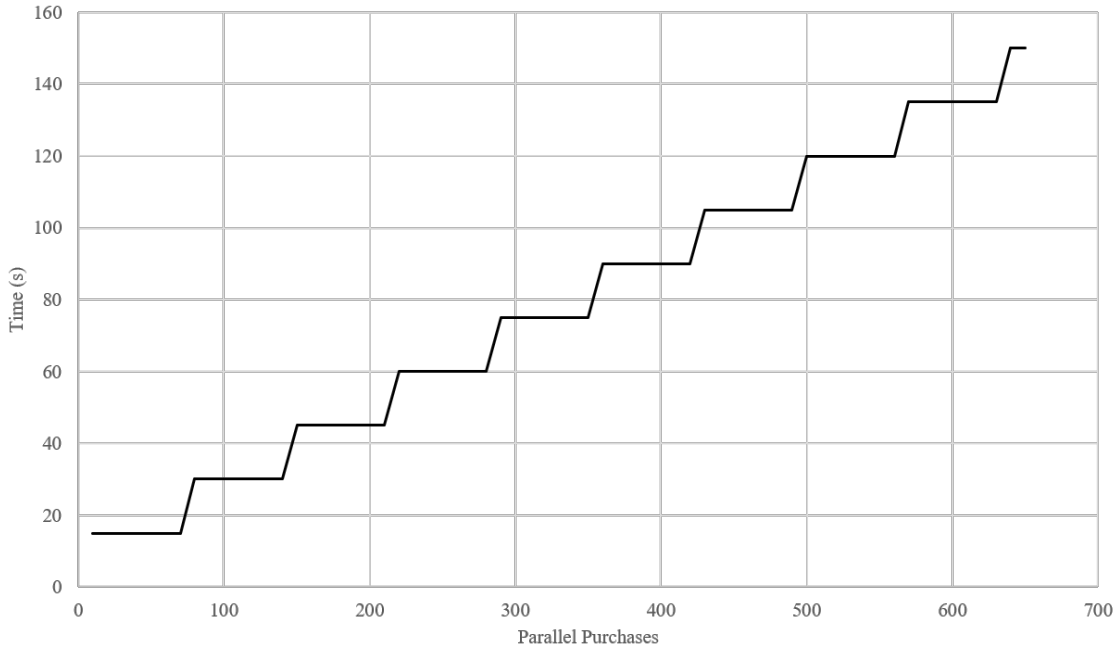


Figure 24: Theoretical Scalability

## 7.1 Scale Comparison

In 2012 Netflix users viewed 5.1 TV shows and 3.4 movies per week [30]. Netflix had 40.28 million viewers in the 3rd quarter of 2012 and 109.25 million viewers in the 3rd quarter of 2017 [31]. This averages 8.5 transactions per week. Using the 2012 viewership requires a CDN that supports 566.10 transactions per second (tps). Using the 2017 viewership, assuming content consumption is the same as 2012, requires a tps of 1535.42. The current upper bound for Bitcoin is about 7 tps [32]. The current upper bound for Ethereum is about 15 tps [32]. The implemented system is capable of 70 purchases per block, with 1 block produced every 17 seconds, and thus can support 4.1 purchases per second. When compared to enterprise-scale CDNs, the implemented system faces bottlenecks with the

current generation of blockchain technology.

## 7.2 Future Work

Future work should address the scalability issues to improve consumer experience. Methods to improve scalability that can be investigated to improve a blockchain digital rights management system are block limit increase, the lightning network, and sharding. Increasing the block size or complexity limits has been used by BitcoinCash [33] and Ethereum miners [34]. This method quickly increases the transaction rate, but it is not a long term solution because it requires additional resources proportional to the increase in transaction rate. The lightning network creates secure transactions off the blockchain [35]. The lightning network routes currency through cryptographically secured payment channels. If a peer does not correctly route currency, then proof can be published to the blockchain to resolve the conflict. Sharding divides the verification or storage of transactions among multiple subsets of peers [32]. Each subset handles its portion of transactions and requests information from peers in other subsets when required. Sharding is supposed to be secure if each subset of peers is sufficiently large.

Other challenges that can be addressed in future work are the free rider problem and multiple copy storage. In a P2P filesharing network the free rider problem is a peer that downloads content and does not serve peers, which results in worse service for peers. A potential fix to the free rider problem is to use cryptocurrency micropayments between peers to cover the costs of and incentivize file serving. Files with more copies stored are more likely to be available in the future. Cryptocurrency distribution for storing copies of files might incentivize storage of a minimum number of copies of a file.

# References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[2] H. V. Jagadish, B. C. Ooi, and Q. H. Vu, "Baton: A balanced tree structure for peer-to-peer networks," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 661–672.

[3] D. Tsolis, S. Sioutas, A. Panaretos, I. Karydis, and K. Oikonomou, "Decentralized digital content exchange and copyright protection via p2p networks," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*. IEEE, 2011, pp. 1056–1061.

[4] E. Diehl, *Securing Digital Video: Techniques for DRM and Content Protection*. Heidelberg: Springer, 2012.

[5] (2008) Yahoo music going dark. [Online]. Available: http://arstechnica.com/uncategorized/2008/07/drm-still-sucks-yahoo-music-going-dark-taking-keys-with-it/

[6] (2014) Diesel ebooks to close. [Online]. Available: http://www.publishersweekly.com/pw/by-topic/digital/retailing/article/61586-diesel-ebooks-to-close.html

[7] J. Leyden, "Ubisoft undone by anti-drm ddos storm," p. 1, 2010. [Online]. Available: https://www.theregister.co.uk/2010/03/08/ubisoft_anti_drm_hack_attack/

[8] (2008) Walmart shutting down drm server. [Online]. Available: http://boingboing.net/2008/09/26/walmart-shutting-dow.html

[9] (2008) Msn music store support notification. [Online]. Available: https://web.archive.org/web/20090124025750/http://community.winsupersite.com/blogs/paul/archive/2008/06/19/msn-music-store-support-notification.aspx

[10] Y. Tang, X. Li, Y. Liu, C. Liu, and Y. Xu, "Review of content distribution network architectures," in *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*. IEEE, 2013, pp. 777–782.

[11] J. Peltotalo, S. Peltotalo, A. Jantunen, L. Väätämöinen, J. Harju, R. Lehtonen, and R. Walsh, "A massively scalable persistent content distribution system." in *Communications, Internet, and Information Technology*. Citeseer, 2007, pp. 269–274.

[12] Y.-M. Chen and W.-C. Wu, "An anonymous drm scheme for sharing multimedia files in p2p networks," *Multimedia tools and applications*, vol. 69, no. 3, pp. 1041–1065, 2014.

[13] P. Kumar, G. Sridhar, V. Sridhar, and R. Gadh, "Dmw-a middleware for digital rights management in peer-to-peer networks," in *16th International Workshop on Database and Expert Systems Applications (DEXA'05)*. IEEE, 2005, pp. 246–250.

[14] H. Shi, Y. Zhang, J. Zhang, E. Beal, and N. Moustakas, "Design and implementation of digital right management in acollaborative peer-to-peer network," in *Com-*

*puter and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*. IEEE, 2007, pp. 198–203.

[15] R.-I. Chang, T.-T. Wei, and C.-H. Wang, "A cost-effective key distribution of p2p iptv drm over opportunistic multicast overlay for e-commerce systems," *Electronic Commerce Research*, vol. 15, no. 1, pp. 49–71, 2015.

[16] J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira, and A. Akutsu, "The blockchain-based digital content distribution system," in *Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on*. IEEE, 2015, pp. 187–190.

[17] J. Herbert and A. Litchfield, "A novel method for decentralised peer-to-peer software license validation using cryptocurrency blockchain technology," in *Proceedings of the 38th Australasian Computer Science Conference (ACSC 2015)*, vol. 27, 2015, p. 30.

[18] T. McConaghy and D. Holtzman, "Towards an ownership layer for the internet," 2015.

[19] Staff, "Our work," p. 1, 2017. [Online]. Available: https://ujomusic.com/

[20] J. Guagliardo, "Music and the blockchain," p. 1, 2016. [Online]. Available: http://www.pepperlaw.com/publications/music-and-the-blockchain-2016-09-07/

[21] (2018) Muse faq. [Online]. Available: https://museblockchain.com/faq

[22] Staff, "Blockchain content distribution," p. 1, 2017. [Online]. Available: https://decent.ch/

[23] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[24] (2016) Cryptocurrency. [Online]. Available: http://www.investopedia.com/terms/c/cryptocurrency.asp?lgl=no-infinite

[25] A. Miller, "Provable security for cryptocurrencies," Ph.D. dissertation, University of Maryland, College Park, 2016.

[26] (2016) Ethereum homestead documentation. [Online]. Available: http://www.ethdocs.org/en/latest/index.html

[27] G. Wood, "Ethereum yellow paper," 2014.

[28] V. Buterin, "Toward a 12-second block time," p. 1, 2014. [Online]. Available: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/

[29] Staff, "White-box cryptography," p. 1, 2017. [Online]. Available: https://www.arxan.com/technology/white-box-cryptography/

[30] ——, "Average netflix user watches 5 tv shows, 3 movies per week via the service," 2012. [Online]. Available: https://www.businesswire.com/news/home/20120906006400/en/Average-Netflix-User-Watches-5-TV-Shows

[31] ——, "Number of netflix streaming subscribers worldwide 2011-2017," 2018. [Online]. Available: https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

[32] V. Buterin and E. Staff, "On sharding blockchains," 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQ

[33] (2018) Bitcoincash peer-to-peer electronic cash. [Online]. Available: https://www.bitcoincash.org/

[34] (2018) Evolution of the average block gas limit. [Online]. Available: https://www.etherchain.org/charts/blockGasLimit

[35] (2018) Lightning network scalable, instant bitcoin/blockchain transactions. [Online]. Available: https://lightning.network/#intro

# APPENDIX A: Storage Calculations

Definitions

n = number of content

k = number of license standards

s = Total storage on single blockchain node

b = size of bytecode

d = size of state variables

$b_f$ = size of factory bytecode

$d_f$ = size of factory state variables

Assumptions

$b_0 = b_1 = b_n$

$d_0 = d_1 = d_n$

$b_{f0} = b_{f1} = b_{fn}$

$d_{f0} = d_{f1} = d_{fn}$

$k << n$

Factory Method Calculations

$s = k(b_f + d_f) + n(b + d)$

License Method Calculations

$s = nd + kb$

Difference Calculations

$$kb_f + kd_f + nd - (nd + kb)$$

$$kb_f + kd_f + nb - kb$$