TRUST-TO-TRUST DESIGN OF A NEW INTERNET

MUNEEB ALI

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE
ADVISER: PROFESSOR ANDREA S. LAPAUGH

JUNE 2017

Abstract

The internet's original design, guided by the end-to-end design principle, pushed all application-specific logic and complexity to the edges of the network and kept the core of the network focused on the simple task of delivering data. The original end-to-end principle, however, did not explicitly account for trust and security. There are several central points of trust and failure on the traditional internet. These include root servers for the Domain Name System (DNS) and public-key infrastructure like Certificate Authorities (CAs) that publish security certificates. Further, the success of cloud hosted services in the last decade means that most user data is stored on remote servers and end-users need to trust these remote servers for correct execution of their applications.

In this thesis, we present a new internet architecture that explicitly follows the trust-to-trust design principle, i.e., end-users don't need to trust the core of the network for anything, and end-users can use applications and services in a fully decentralized way. We make the observation that cryptocurrency blockchains, like Bitcoin, can be used to bootstrap trust for new nodes joining a network. We identify the various limitations, like high latency and limited bandwidth, of contemporary blockchains and discuss how our architecture can scale by moving most operations outside of the blockchain layer.

We detail our experience of running a large production system on top of a cryptocurrency blockchain and how that experience guided our design. We present the implementation of a new decentralized internet, called Blockstack, that takes the trust-to-trust architecture from a theoretical concept to a production system. Deploying new systems by modifying production blockchains is hard because it requires coordination and agreement from several parties. We introduce *virtualchains*, a virtual blockchain constructed by processing data from underlying blockchains, to enable the seamless introduction of new functionality on top of blockchains without requiring any consensus-breaking changes. Blockstack is already powering several fully decentralized applications, like OpenBazaar; it's released as open-source software and, to date, more than 70,000 domains have been registered on it.

Acknowledgements

My dissertation took a non-traditional route because I took a leave from Princeton, in 2013, to start a tech startup. In the end, the different pieces came together in ways that I could never have imagined: I ended up not only doing research and publishing papers on our work but, more importantly, was able to take these ideas from theory to practice.

The work presented in this thesis is a collaboration with several people. Ryan Shea, my co-founder, and I were frustrated by a few large companies having monopoly control over the personal data and social graphs of internet users and started with an idea to re-decentralize the internet. Our work later evolved into the open-source project Blockstack. We sometimes joke that, given his contributions, Ryan probably deserves an honorary PhD. Jude Nelson joined our project in 2015 and took the lead on the software development of Blockstack Core, the reference implementation of Blockstack. Jude deserves credit for his work on virtualchains, the Atlas network, and for the several recent software releases amongst other things. I'd like to thank Guy Lepage for his contributions and specifically for designing early versions of the system architecture diagram. I'd like to thank open-source contributors Larry Salibra, Aaron Blankstein, Patrick Stanley, and the 2,500+ community members in our chat channel that make working on Blockstack fun every day.

I enjoyed my time at Princeton; the CS department feels like my home away from home. Over the years, I got to work with several Professors and explored a broad range of potential thesis topics. Heartfelt thanks go to my adviser, Andrea LaPaugh for giving me the freedom to explore different problem areas (datacenters, search systems, and blockchains) and for always pushing me to find a problem that I enjoy working on. I'd like to thank Mike Freedman for his time and support and for thinking through every detail of various distributed systems problems with me. I learned the undocumented art of being a systems researcher from watching him design and optimize systems. I'd like to thank JP Singh for being a mentor and sounding board for a broad range of ideas and topics. Blockstack is lucky to have both JP Singh and Mike Freedman as advisors to the project. I'd like to thank Arvind Narayanan, Jennifer Rexford, and Brian Kernighan for serving on my thesis committee and for giving valuable feedback on the work. Jen has been gracious with her time and support since my first day at Princeton. The intellectually stimulating and friendly culture of the department made it a pleasure to work there. Wyatt Lloyd, Jeff Terrace, David Shue, Michael Wawrzoniak, Anirudh Badam, Matvey Arye, Sid Sen, and Patrick Wendell raised the bar for excellence and were some of the best colleagues I got to work with.

I'd like to thank Larry Peterson and Phil Levis for hosting me at PlanetLab and Stanford for a summer respectively. Koen Langendoen, Thiemo Voigt, Umar Saif, and Zartash Uzmi guided my early research career before Princeton. I'm grateful to our investors, especially Albert Wenger and Naval Ravikant, for the financial resources and guidance needed to carry out our ambitious mission of building a new decentralized internet. I was fortunate enough to form some lasting friendships at Princeton; Markus, Nima, Milena, Fahad, Sophie, Bouchra, Eden, and Darshana made my time on the campus memorable. My grandmother, siblings, and mother-in-law always believed in me and my younger sister, Saira, celebrated every small milestone of this journey. Finally, I'd like to thank my parents, Akhtar and Naseem, and my wife, Leena, for their unconditional support and my baby girl, Mila, for brightening up my days.

Parts of this dissertation were published earlier in the following papers:

- Muneeb Ali, Jude Nelson, Ryan Shea and Michael J. Freedman, *"Blockstack: A Global Naming and Storage System Secured by Blockchains"*, 2016 USENIX Annual Technical Conference, Denver, CO, June 2016.

- Jude Nelson, Muneeb Ali, Ryan Shea and Michael J. Freedman, *"Extending Existing Blockchains with Virtualchain"*, Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, July 2016.

- Muneeb Ali, Jude Nelson, Ryan Shea and Michael J. Freedman, *"Bootstrapping Trust in Distributed Systems with Blockchains"*, USENIX ;login: Issue: Vol. 41, No. 3, Pages 52-58, Fall 2016.

I was supported by a Princeton Graduate Fellowship in the initial years of graduate school. The research in this dissertation was done while I was on leave from Princeton 2013-2017.

To my parents and wife.

Contents

List of Figures

Chapter 1

Introduction

> *"When Hiro first saw this place, ten years ago, the monorail hadn't been written yet; he and his buddies had to write car and motorcycle software in order to get around. They would take their software out and race it in the black desert of the electronic night."*
>
> – NEAL STEPHENSON (SNOW CRASH, 1992)

Our lives are increasingly dependent on the internet. Its hard to get any work done if the internet goes down. The internet was designed more than 40 years ago and is showing signs of age. Critical internet services can be taken offline by straight forward attacks like the recent DDoS attack on DNS servers [109]. Further, in the current internet architecture users implicitly trust certain hidden services and intermediaries like domain name servers and certificate authorities (CAs). These trust points can be exploited to trick users into connecting to malicious websites like the recent incident where a Turkish CA issued false security certificates for Google [124].

Over the last decade, we've seen a shift from desktop applications running locally to web services running on remote servers and storing user data remotely. These centralized services are a prime target for hackers and frequently get hacked. In 2016, Yahoo! admitted to losing information for 500 million people [117]. Security problems with core internet infrastructure and the data models of centralized web services built on top have exposed the ugly underbelly of the internet and exposed flaws in the internet's original design.

1.1 Internet Architecture

Internet's Original Design

The internet started as an inter-network of different packet-switched networks in the 1970s. Since the beginning, the internet was decentralized by design. Nodes on the internet were designed to talk to each other directly without relying on any central services that controlled internet traffic. The Domain Name System (DNS), a federated network, enables the use of human-readable names, like cnn.com instead of IP addresses, for connecting to other nodes. The internet gained wider adoption in the early 1990s with the introduction of the World Wide Web and internet browsers, but the underlying architecture remained unchanged.

The internet was originally designed to keep all application-specific logic at the edges and was designed as a "dumb network," i.e., it doesnt know what data it delivers and just transfers packets from point A to point B. This principle of not keeping complexity in the network and pushing all complexity and logic to the edges is called the *end-to-end design principle* [126] by David Clark et al.

Trust-to-Trust Design

The original end-to-end principle that guided internet design for the past decades did not explicitly account for trust and security. Over time, trusted resources like DNS root servers and Certificate Authorities (CAs) became a part of the core network. David Clark and Marjory Blumenthal **extended** the end-to-end principle [44, 51]. According to the updated principle, called the *trust-to-trust principle* [52], a new internet design should:

1. Give the end user explicit control over trust decisions, and

2. Move trust from the core of the network to the edges.

This movement of trust from the core of the network to the edges is also called *decentralization* and marks a shift in how new networks are being designed. A big part of that shift comes from the advent of new decentralized blockchain networks like Bitcoin [127].

1.2 Blockchains and Decentralized Services

Blockchain networks started as networks for digital currencies (also called *cryptocurrencies*). Digital currency networks need to solve a *double spend* problem where a digital token, unlike physical cash, can be spent twice [127]. Blockchains provide a solution to this problem without introducing any central services. Blockchain networks are open membership networks that store data logs, called *blockchains*. These data logs are organized into blocks that are cryptographically linked together in a *hash chain* [127]. Blockchain data is stored redundantly on all computers connected to the network. Only new data can be appended to blockchains, and historic logs cannot be modified or tampered with. Every connected computer can verify that new data being written to logs obeys the rules of the open blockchain network's consensus mechanism (like checking that a new transaction is not double spending a digital token). All nodes on the network have the same view of the blockchain data. The network doesn't have any central points of trust or failure.

Applications Beyond Cryptocurrencies

Blockchains were originally designed for new cryptocurrencies, like Bitcoin and Litecoin, but they are useful beyond exchanging digital tokens. The cryptographically auditable, append-only ledgers of blockchains are already being used to build new, *decentralized* versions of the Domain Name System (DNS) [104] and public-key infrastructure (PKI) [112], along with other applications like file storage [67] and document timestamping [48]. They enable a new class of decentralized applications and services that minimize the degree to which users need to put trust in a single party, like a root certificate authority.

Blockchains have attracted interest from enthusiasts, engineers, and investors with 1.5 billion USD invested in blockchain startups over the last several years [54]. With the rapid capital infusion, infrastructure for blockchains is getting quickly deployed [56]. However, blockchains are at an early stage, there is very little production data available to guide design trade-offs, and there are many scalability and performance challenges with blockchains (Chapter 2). We believe that in spite of the current limitations, blockchains provide important infrastructure for building secure, decentralized services.

1.3 A New Internet Architecture Secured by Blockchains

The decentralization benefits of blockchains motivated us to design a new internet architecture that explicitly follows the trust-to-trust design principle and removes all points of trust from the core of the network. We identify that the DNS, a federated network, and PKI services, usually operated by large companies like Verisign [21], are the two most important components of core internet infrastructure that need to be replaced with their decentralized versions. There is a school of thought that argues that human-readable names are not important and long cryptographic IDs combined with search engines can be a replacement for DNS [33, 80, 99]. In this thesis, we take the view that human-readable names are essential for providing a good user experience and, in practice, it'd be very hard to convince internet users to change their habits and stop using human-readable names online.

Decentralizing naming and PKI can enable end-users to connect to web services, like Facebook and LinkedIn, without trusting any (hidden) parties in the middle. This appears to satisfy the trust-to-trust principle in theory (with users and their web services in separate trust zones) but doesn't fully achieve the intended benefits of decentralization; web services like Facebook and LinkedIn still own and store user data and two users, say Alice and Bob, need to first connect to a centralized service before they can establish a connection between each other. We consider decentralization of user data to be an essential part of the trust-to-trust design. End users should not trust third-parties with their personal data and should not rely on any third-parties for connecting to other users.

In this thesis, we narrow down the design goals of a new trust-to-trust internet to the following three requirements:

1. Users should be able to register and cryptographically own human-readable names without trusting any service providers or intermediaries.
2. Users should have access to decentralized storage systems where they can store their data without revealing it to any party that they need to trust.
3. Users should get comparable performance to the traditional internet, and a trust-to-trust design based internet should be able to scale to billions of users.

In this thesis, we present how blockchains solve the bootstrapping of trust problem in distributed systems (Chapter 2). We argue that bootstrapping of trust is the only functionality that needs to be handled at the blockchain layer and all other operations can be moved off-chain for better scalability. We present our experiences from operating a production naming and PKI system on the Namecoin network, which is one of the largest services built on top of a blockchain to date (Chapter 3).

Our experience with Namecoin informed the design and implementation of a new blockchain-based system, called Blockstack, which provides services for naming and storage that can run on top of any blockchain (Chapter 4). Unlike previous blockchain-based systems, Blockstack *separates its control and data plane considerations*: it keeps only minimal metadata (names, data hashes and state transitions) in the blockchain and uses external datastores for actual bulk storage. Blockstack enables users to register human-readable names (Chapter 5). We have released Blockstack as open source software [42].

Modifying production decentralized systems like Bitcoin (and introducing new functionality for which it was not designed) is quite difficult, particularly that the system still needs to reach "consensus." We extended the single state machine model of blockchains, by introducing *virtualchains*, to allow for arbitrary state machines without requiring consensus breaking changes in the underlying blockchain. Virtualchains (Chapter 6) are to blockchains what virtual-machines are to physical computers; for applications running on a virtualchain it's a blockchain but is itself built on top of underlying blockchains and enables fault tolerance and isolation. This design was non-intuitive before our work; indeed, the standard approach for the past four years was to fork the main Bitcoin blockchain to add new and different functionality.

Further, we present the design of two new peer networks for decentralized discovery of content (Chapter 7) and outline the architecture of a decentralized storage system that can give comparable performance to traditional centralized cloud providers (Chapter 8). We discuss real-world use cases and present a case study of a decentralized marketplace that uses Blockstack (Chapter 9).

1.4 Contributions

This thesis makes the following contributions:

- We present the design of a new internet architecture that extends the end-to-end design of the traditional internet and explicitly follows the trust-to-trust design principle. Our design removes any trust points from the middle of the network. We make the observation that, in theory, blockchains are sufficient for enabling trust-to-trust networks, but the inherent bandwidth and latency limitations of contemporary blockchains mean that practical trust-to-trust networks need to move most communications off blockchains.

- We present the implementation and early performance numbers of Blockstack, a system that implements different components/layers of a new decentralized internet and takes the trust-to-trust architecture from a theoretical concept to a production system. We discuss lessons learned from running the Blockstack network in production and how these lessons helped evolve the system for improving reliability and achieving comparable performance to traditional internet services.

- We present the first security and network reliability analysis of a blockchain other than Bitcoin and report a critical security problem where a major alternate blockchain, Namecoin, had a single miner with well over 51% of the compute power.

- We present the design of *virtualchain*, a virtual blockchain constructed by processing data from underlying blockchains. With virtualchains, it's possible to introduce new functionality on top of production blockchains without requiring any consensus-breaking changes from the underlying blockchains.

- We present a novel Sybil-protection mechanism for peer-to-peer networks that uses blockchains to introduce Sybil-protection against (a) junk data writes and (b) node eclipse attacks. Unlike previous works, our Sybil-protection mechanism doesn't introduce any central gatekeepers.

Chapter 2

Bootstrapping Trust

"The city's central computer told you?

R2-D2, you know better than to trust a strange computer."

– C-3PO (THE EMPIRE STRIKES BACK, 1980)

In this chapter, we consider the problem of bootstrapping trust in distributed systems. Bootstrapping trust in a commodity computer is a more general problem [115]. Assuming that a commodity computer's local execution environment can be trusted, we define the bootstrapping trust in distributed systems problem as follows:

Problem 1 *Let n be a new node that (a) booted into a trusted local execution environment, (b) is joining a distributed system of N nodes, and (c) doesn't have any pre-established secure communication channels. If $M > 1$ network states are presented to node n, then how can node n independently decide which state $m \in M$ to trust?*

In this thesis, we make the observation that proof-of-work (PoW) blockchains, like Bitcoin [127], can be used to solve the bootstrapping of trust problem defined above under certain conditions. This chapter first reviews blockchains and lists their limitations.

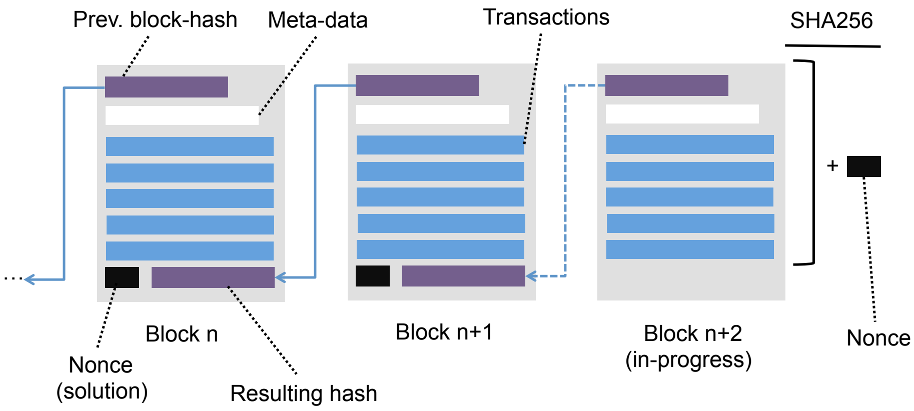


Figure 2.1: A blockchain with transactions organized in cryptographically linked blocks.

2.1 Background on Blockchains

Blockchains provide a global append-only log that is publicly writeable. Writes to the global log, called *transactions*, are organized as *blocks* and each block packages multiple transactions into a single atomic write. Writing to the global log requires a payment in the form of a *transaction fee*. Nodes participating in a blockchain network follow a leader election protocol for deciding which node gets to write the next block and collect the respective transaction fees. Not all nodes in the network participate in leader election. Nodes actively competing to become the leader of the next round are called *miners*. At the start of each round, all miners start working on a new computation problem, derived from the last block, and the miner that is the first to solve the problem gets to write the next block. In Bitcoin, the difficulty of these computation problems is automatically adjusted by the protocol so that 1 new block is produced roughly every 10 minutes.

Figure 2.1 shows a simplified version of the Bitcoin blockchain. Blocks n and $n+1$ are confirmed on the network and block $n+2$ is in progress. Blocks include (a) hash of the previous block (a reference to the previous block), (b) meta-data like software version, current timestamp, difficulty level, and (c) transactions. The mining process calculates the SHA256 hash of the Merkle root hash of all transactions proposed in a new block, meta-data, hash of the previous block, and a nonce. The nonce is repeatedly incremented until a resulting hash with a particular property is calculated, e.g., assume that we want a resulting hash that starts with 18 zeros like:

000000000000000000ec9897b4a82c526d8b7f93a0fa8f02fcc1dc5e90ae6c77

The mining process will continue to increment the nonce until a valid resulting hash is calculated. The miner that finds the hash will package the new block and announce it to the rest of the network. Miners have an economic incentive to work on the longest known blockchain [127]. Even if there are temporary blockchain forks because of different miners proposing different versions of the blockchain, the forks resolve within a few blocks as miners on the network discard shorter blockchain forks and pick the longest blockchain to work on. The longest blockchain has the most proof-of-work on it. This way the system keeps making forward progress and participating nodes have exactly 1 global view of data logs written to the longest blockchain. See [45, 106] for further details on how blockchains work and how they reach consensus.

2.2 Limitations of Blockchains

The decentralized nature of blockchains introduce meaningful security benefits (no central points of trust or failure), and various distributed systems and applications can be designed to use blockchains. However, certain aspects of contemporary blockchains present technical limitations and building systems with blockchains presents challenges:

• **Limits on Data Storage:** Individual blockchain records are typically on the order of kilobytes [127] and cannot hold much data. Moreover, the blockchain's log structure implies that *all* state changes are recorded in the blockchain. All nodes participating in the network need to maintain a full copy of the blockchain, limiting the total size of blockchains to what current commodity hardware can support. As of May 2017, Bitcoin nodes need 112GB total disk space, for blockchain data, to stay synchronized with the network [2].

- **Slow Writes:** The transaction processing rate is capped by the blockchain's write-propagation and leader-election protocol, and it is pegged to the rate at which new blocks are announced by leader nodes, called *miners* in many blockchain networks [45]. New transactions can take several minutes to a few hours to be accepted and the latency of a single block confirmation is typically on the order of 10-40 minutes [45].

- **Limited Bandwidth:** The total no. of transactions per block is limited by the *block size* of blockchains. To maintain fairness and to give all nodes a chance to become the leader in the next round, all nodes should receive a newly announced block at roughly the same time. Therefore, the *block size* is typically limited by the average uplink bandwidth of nodes [45]. For Bitcoin, the current bandwidth is 1MB ($\sim$1000 transactions) per block.

- **Endless Ledger:** The integrity of blockchains depends on the ability for anyone to audit them back to the first block. As the system makes forward progress and issues new blocks, the cost of an audit grows linearly with time, which makes booting up new nodes progressively more time-consuming. We call this the *endless ledger problem*. Bitcoin's blockchain currently has $\sim$463,000 blocks, and new nodes take 1-3 days to download the blockchain from Bitcoin peers, verify it, and boot up.

In spite of these limitations, blockchains like Bitcoin are gaining commercial adoption [54] and offer a secure base-layer on top of which other services and applications can be built [28]. The cost of tampering with blockchains grows with their adoption: today, it would require hundreds of millions of dollars to attack a large blockchain like Bitcoin [3].

2.3 Bootstrapping Trust in Distributed Systems

For proof-of-work (PoW) blockchains like Bitcoin, the PoW mining puzzle is hard to compute but easy to verify. This implies that any node on the network can independently calculate the PoW on a blockchain on commodity hardware. Given two blockchains b_n and b_m, a node can independently check if:

$$PoW(b_n) > PoW(b_m) \tag{2.1}$$

We can use this property to give a solution to the bootstrapping trust in distributed systems problem (Problem 1). We require that the M states presented to the new node n be written to M distinct PoW blockchain forks, i.e., we require that the only communication channel used to bootstrap a new node n is a read access to PoW blockchains. The node n can independently calculate $PoW(b_m)$, for all $m \in M$, pick the blockchain fork with the most proof-of-work, and trust the state m written to b_m.

Attack Vectors

An attacker that wants to give a new node n fabricated state will need to either:

1. Write malicious state to the longest blockchain and spend more energy than the rest of the honest network to make her blockchain longer than the rest of the network. Or

2. Try to stop node n from discovering the global longest blockchain and write malicious state to a local longest blockchain.

The first attack is economically infeasible for large production blockchain networks like Bitcoin and the second attack is not very practical because even a single honest peer node that is aware of blockchain forks outside of the local network partition can relay the longer blockchains to an otherwise disconnected network partition. The second attack translates to an attacker being able to disconnect a set of peer nodes into a network partition, meaning that the attacker can hide a subset of M states from the new node n. If an attacker can hide a subset of states from all nodes on a (disconnected) network, then no mechanism can enable a new node to discover the hidden states. In practice, as long as physical network links of participating nodes are not compromised, connecting to a large number of peers makes it hard for an attacker to compromise all peers and force them into a partition.

Non Proof-of-Work Blockchains

It's important to note that our solution to Problem 1 works for proof-of-work blockchains and not for other types of blockchains like proof-of-stake (PoS) [58]. In PoS cryptocurrencies the writer of the next block is chosen in a deterministic (pseudo-random) way, and the probability that a particular writer is chosen depends on its stake, i.e., the percent ownership of total tokens [87]. However, in a PoS blockchain if a bootstrapping node is presented with two conflicting transaction histories, then it is impossible for it to determine which one is the "true" chain without some external input [58]. Our definition of Problem 1 doesn't allow any pre-established secure communication channels and therefore PoS blockchains cannot be used. In general, as long as a node can independently differentiate between two conflicting blockchain forks, i.e., a node n can independently find a solution to a property similar to PoW comparisons (Equation 2.1), **without** any pre-established communication channels, then that blockchain can be used to distribute initial network state to new nodes and serve as a solution to the bootstrapping trust in distributed systems problem. In this thesis, when referring to blockchains, we assume PoW blockchains unless stated otherwise.

Bootstrapping Trust-to-Trust Networks

Blockchains, and their respective use for bootstrapping trust can be used to establish communication between two trust zones on the internet, e.g., Alice can discover Bob's domain name, public key, and IP address using a blockchain-based decentralized naming system (Chapter 5) and then connect directly with Bob. We argue that bootstrapping of trust is the only functionality that needs to be handled at the blockchain layer and all other operations in a trust-to-trust network can be moved off-chain for better scalability (Chapter 4).

Chapter 3

Lessons from Deployment

"In theory there is no difference between theory and practice. In practice there is."

– JAN VAN DE SNEPSCHEUT (1953–1994)

We discussed in Chapter 2 that proof-of-work (PoW) blockchains can be used to build trust-to-trust networks. There are many PoW blockchains in production [24] and there is little to no data available for their performance and reliability. There are many open questions. Is it better to build trust-to-trust networks by starting new blockchains or by using existing ones? What practical tradeoffs are there for the contemporary implementation options like sidechains [31], putting new logic on top of a blockchain [111], or on a main logic-heavy chain [46]? In this chapter, we describe our experience with running a year-long production system on a PoW blockchain, Namecoin, and the challenges we faced. We describe the lessons we learned and how they influenced our design decisions for using blockchains to build a new trust-to-trust internet (Chapter 4).

Namecoin is the oldest blockchain other than Bitcoin, that is still operational, with a cryptocurrency market capitalization of 12 million USD as of May 2017 [24] (the market capitalization of a cryptocurrency is the exchange-traded value of its coins multiplied by its number of coins in existence). Namecoin started as an alternate DNS-like system that replaces DNS root servers with a blockchain for mapping domain names to DNS records [104]. Given that blockchains don't have central points of trust, a blockchain-based DNS is much harder to censor and registered names cannot be seized from owners without getting access to their respective private keys [83].

The production system we operated on Namecoin was a type of a public-key infrastructure (PKI) system; users registered human-readable names like "naval" and then associated public keys and other data with their username [112]. We ran the system on Namecoin between March 2014 and September 2015 and had to overcome many challenges for registering and updating over 33,000 user entries and for sending over 200,000 transactions on the Namecoin network. Our production deployment led to many interesting experiences where we observed and analyzed network anomalies and security problems that were not discovered or documented before [27]. Our experience concludes that, for both security and reliability reasons, blockchain-based services should use the largest and most secure blockchain, which at the time of this writing is the Bitcoin blockchain.

3.1 Blockchain Security

The security of token or digital asset ownership is tied to the security of both the underlying blockchain and the software powering it. The most important factor in the security of a blockchain is the total cost of attacking the blockchain and tampering with recently written data. Miners often pool their resources to form a *mining pool*, which is essentially a super node on the network (a lot of computational power behind a single miner node). If the amount of computational power under the control of a single miner (or pool) is more than the rest of the network, called a *51% attack*, then that miner has the ability to attack the network and rewrite recent blockchain history, censor transactions (e.g., for name registrations), and steal cryptocurrency using *double spend* attacks [127]. This is because it will win the leader election for a majority of the time, and produce a blockchain history

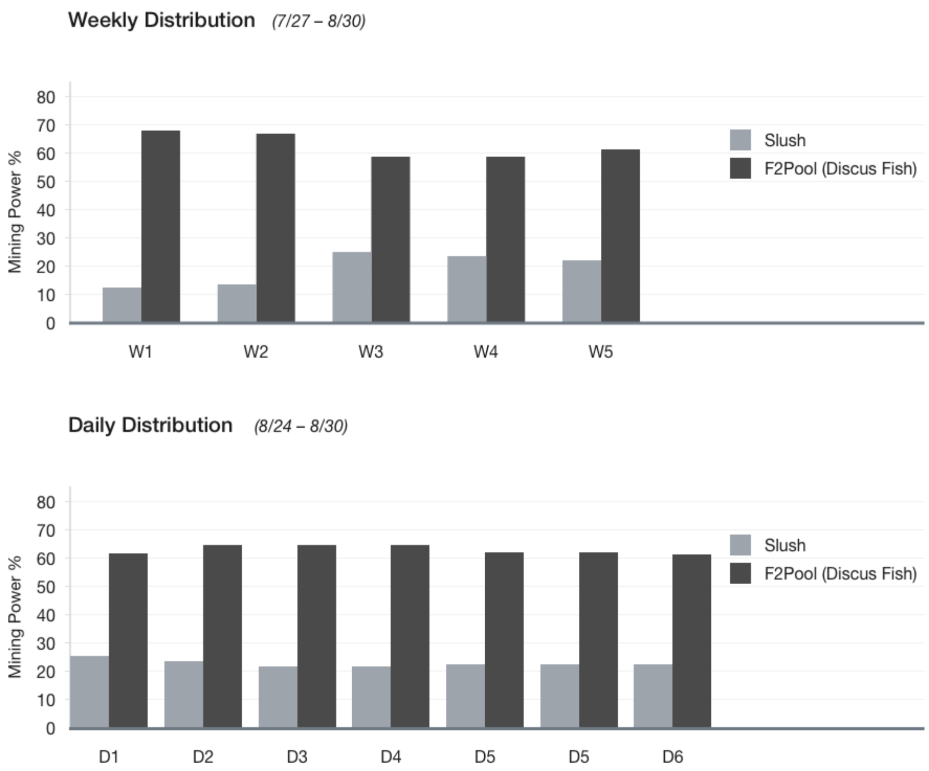


Figure 3.1: Weekly and daily mining distribution of the top two Namecoin miners in terms of hashing power (07/2015 – 08/2015)

with more proof-of-work than any disagreeing miner. The more expensive it is to control a majority of the compute power on a blockchain, the more secure the blockchain.

We noticed in late 2014 that a single mining pool consistently had more than 51% of the compute power on Namecoin. Later, the situation got even worse, with a single mining pool controlling over 60% of Namecoin's compute power in 2015. Figure 3.1 shows the weekly and daily distribution of mining power of the two top miners for the month of August 2015, right before we migrated our system away from Namecoin. In fact, we have observed F2Pool (also known as Discus Fish) control up to 75% of compute power in a particular week. At such concentration, Namecoin is effectively controlled by a single party; F2Pool writes most new blocks and can undermine the security of the blockchain.

Other than raw hashing power, software bugs can also introduce security problems, e.g., a Namecoin bug allowed people to steal names from anyone [74]. Denial-of-service attacks

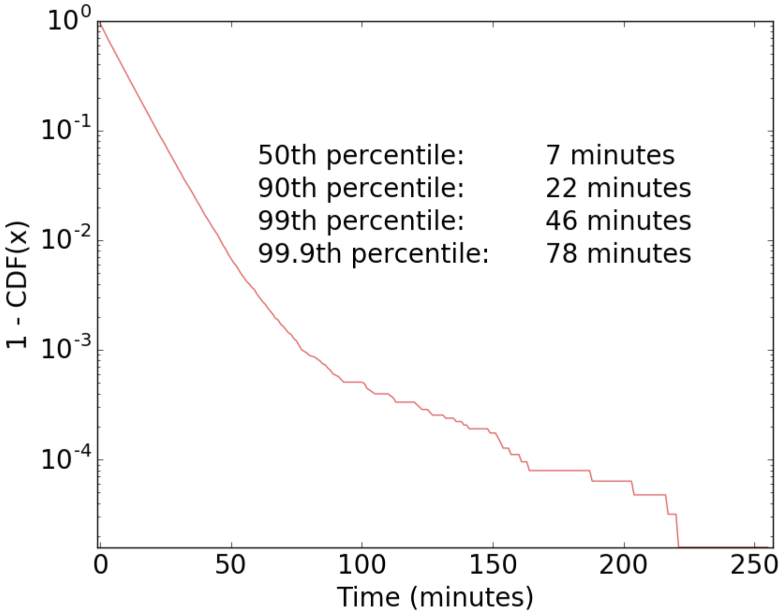


Figure 3.2: CCDF of Namecoin network latency (03/2014 – 04/2015)

are another attack vector; the more peers a cryptocurrency network has, the more resilient the network is to denial-of-service attacks.

Bitcoin currently has the largest amount of computational power securing the blockchain data. Bitcoin's codebase is more actively developed with more bug bounties and security reviews than other blockchains. Namecoin has many fewer peer nodes than Bitcoin (170 vs. 4,600 in Jan 2016 [11]), which makes it more vulnerable to DDoS attacks as well. The Bitcoin blockchain is currently by far the most secure blockchain. However, it's extremely hard to introduce new functionality to Bitcoin because that requires consensus-breaking changes (Section 3.4).

Lesson #1: There is a fundamental tradeoff between blockchain security and introducing new functionality to blockchains. Starting a new blockchain network is how developers typically introduce new functionality not provided by Bitcoin, e.g., a naming system that is of interest to many emerging applications. However, new blockchains are significantly less secure than Bitcoin. In Chapter 6, we overcome this tradeoff by creating *virtualchains* that introduce new functionality as a layer on top of Bitcoin.

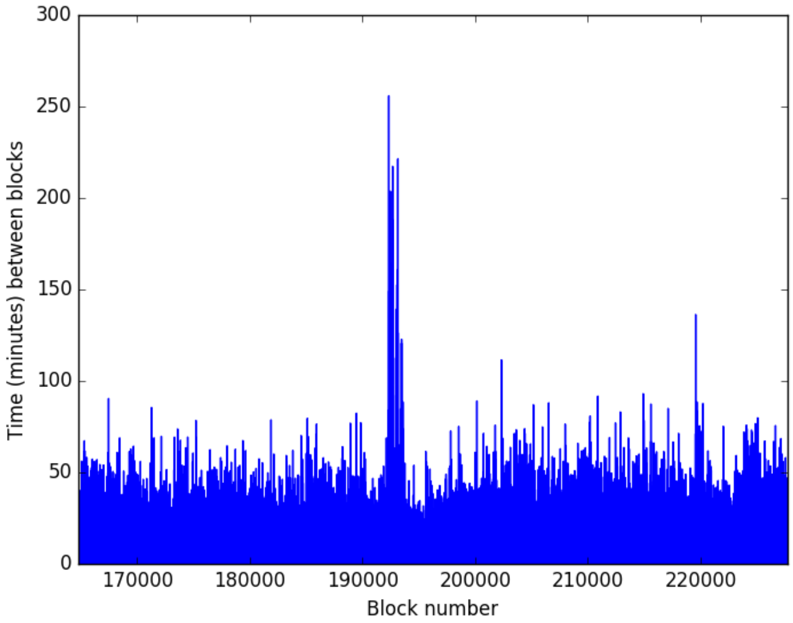


Figure 3.3: Namecoin network latency per new block (03/2014 – 04/2015)

3.2 Network Reliability and Throughput

The throughput of our production system (number of entries we can register/update) was directly dependent on the throughput of the underlying blockchain. The number of new register/update operations that can be performed per hour is limited by the number of transactions that can be sent (and confirmed) on the underlying blockchain per hour. Similarly, the reliability of our production system was impacted if the underlying blockchain network cannot perform operations reliably and consistently.

Network Latency Spike: As Namecoin is based on Bitcoin, it shares many protocol properties with Bitcoin, including a 10 minute average leader election time (the "latency target") and a 1MB bandwidth limit on block size (giving throughput of $\sim$1000 transactions per block). Figure 3.2 shows that after we launched our production system in March 2014, Namecoin on average performed well on the network latency target. As expected, most new blocks were written within 10 and 40 minutes (similar times have also been observed on Bitcoin [45]). Figure 3.3 shows an incident in late August 2014 (at block number 192000), where network latency skyrocketed for a couple of weeks ($\sim$1000 blocks are roughly a

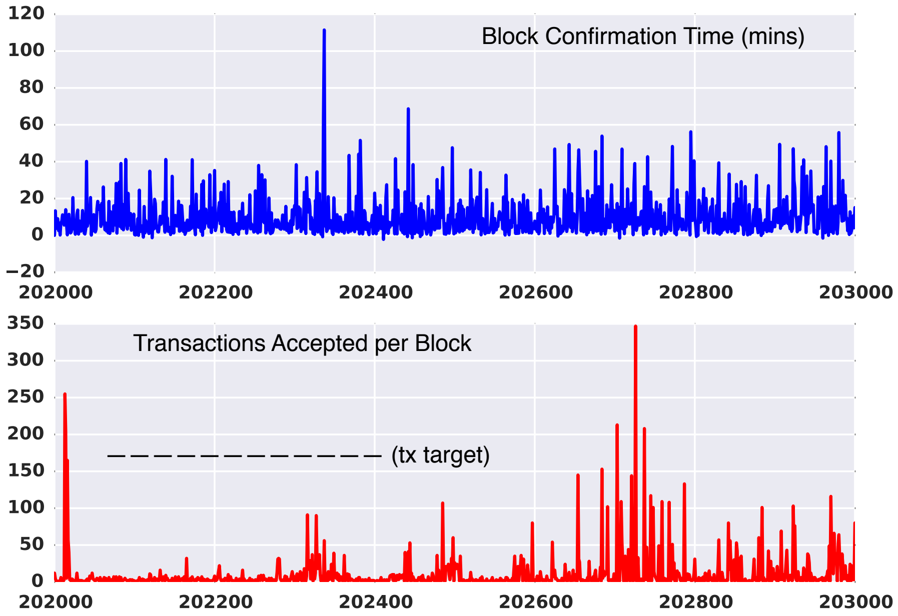


Figure 3.4: Throughput drop in the Namecoin network. (The number of transactions we were trying to send is shown as "tx target".)

week). After investigating the issue and having discussions with Namecoin developers, we discovered that the latency spike was caused by software issues in Namecoin. Someone on the network was sending transactions with a large number of data fields per transaction. This triggered a bug in the Namecoin software, causing reliability problems for the miners as their Namecoin daemons kept crashing. Without stable miner nodes, blocks were not getting appended in a timely fashion. This shows that unexpected protocol/software issues can trigger network latency problems. During this period, we noticed a slow down in rate of new registrations on our production system along with a spike in user complaints.

Network Throughput Drop: In early September 2014, right after the latency spike incident, we noticed that our transactions were not getting accepted for many consecutive blocks and, after a while, will get accepted in bulk in a single block that packaged a lot of transactions. We noticed that a lot of new blocks had no transactions in them. This issue persisted for over a week and Figure 3.4 plots the number of transactions that we were trying to send (shown as "tx target") vs. the number of transactions that were getting accepted by the network. Network latency was completely normal (shown at top of Figure 3.4), but network throughput went down because of no transactions in new blocks. We tried software upgrades and transaction rebroadcasts, but the issue persisted. After more analysis, We concluded that a large mining pool was either intentionally refusing to or was unable to package transactions in the new blocks it was writing. Our transactions would get packaged only when some other miner was elected to write the new block. We discuss the issue of blocks with 0 transactions and miner incentives in more detail in the next section.

Lesson #2: There is currently a significant difference between the network reliability of the largest public blockchain network (Bitcoin) and network reliability of the long tail of alternate blockchains. Problems with the Bitcoin network impact a lot more users and businesses than Namecoin and other smaller blockchains. This work is one of the first analysis of the network reliability of a blockchain other than Bitcoin.

3.3 Potential Selfish Mining

The signs that we noticed in the incident where miners were not accepting our transactions (Section 3.2) looked similar to a selfish mining attack [63]. In a selfish mining attack, (a) a miner needs to have a large amount of mining power (more than 33%), (b) people would notice a long delay in blocks followed by blocks in very quick succession, and (c) there will be a lot of rejected blocks. We noticed all these signs, and believe that the unusually high computing power of a single miner led to conditions similar to selfish mining; the miner was able to work on new blocks faster than the others and append them in rapid succession.

Lesson #3: Selfish-mining is not just a theoretical attack, but selfish-mining like behavior can already be observed in production blockchains. This is the first time that data collected from a production network shows signs of selfish-mining like behavior, regardless of if the miner was intentionally attacking the network or not.

3.4 Consensus-breaking Changes

For major updates, like protocol changes, Namecoin requires a “hard fork” in which everyone on the network must upgrade their software, and nodes on previous versions can no longer participate in the network. Anecdotal evidence suggests that it’s hard to get miners to upgrade their software because they don’t have enough incentive to spend engineering hours on maintaining a relatively small cryptocurrency like Namecoin, which is not their main reason for operating a mining pool. Our experience monitoring the Namecoin network showed that after Namecoin software updates there was a considerable fluctuation of computing power. In fact, we noticed that after the recent upgrade to Namecoin Core [105], a major upgrade to the Namecoin daemon, many miners dropped out and never came back.

Lesson #4: Other than the engineering problems, consensus-breaking changes are complicated because of fundamental incentive structures of the parties involved. System designers have never dealt with consensus-breaking changes before cryptocurrencies; it’s a novel challenge. For software upgrades to cryptocurrency networks, we should: (a) separate consensus-breaking upgrades from other upgrades as a software engineering rule (Bitcoin recently started doing this in their codebase [38]) and (b) try to align miner incentives given their cost (engineering time) of software upgrades. This resistance to upgrades is present in Bitcoin as well [118] and is not unique to smaller blockchains. In Chapter 4, we describe how we introduce new features without requiring miner upgrades.

3.5 Failure of Merged Mining

The security of a blockchain depends on the relative compute power of miners and the cost for a single party to have more computing power than the rest of the network. New, smaller blockchains have a *bootstrapping problem*: in the initial days of a new blockchain, it would be relatively easy for a single party to take it over, since the total compute power on the blockchain is not yet large enough to prevent this. To address this problem, Satoshi

Nakamoto (author of Bitcoin) introduced “merged mining” [102], where an alternate blockchain can re-use proof-of-work computations done for Bitcoin, allowing miners to participate in the new network without spending extra compute cycles. The miners can make extra profits on the new blockchain without adding computational overhead. For merge-mined cryptocurrencies the security of the blockchain is typically a subset of the “main blockchain”; in practice not all main blockchain miners set up merged mining.

Namecoin switched to merged mining with Bitcoin to increase the security of its blockchain [83]. Namecoin is the oldest and largest merged-mined cryptocurrency and inspired other cryptocurrencies to consider it as well. One of our key findings is that merged mining is currently failing in practice: the leading merged-mined blockchain, Namecoin, is vulnerable to the 51% attack (Section 3.1). Moreover, merged-mining provided a false sense of security. F2Pool controls 30-35% computing power of Bitcoin, but over 60% of Namecoin’s computing power through merged mining, leaving Namecoin vulnerable to a 51% attack. Unless the merged mined cryptocurrency can consistently attract a very high ratio of the main blockchain miners to support their software, merged mining will not keep it safe from 51% attacks.

Lesson #5: Merged-mining can give a false sense of security if you consider only the absolute hash rate. What’s more important is getting a very high ratio (90% or more) of main blockchain miner to participate on the smaller blockchain.

Summary

Namecoin deserves full credit for originally building decentralized naming on a blockchain. However, after our experience, we strongly believe that decentralized applications and services need to use the largest, most secure, and most actively maintained blockchain. Currently, no other blockchain even comes close to Bitcoin in terms of these security requirements. In the next chapter, we outline an architecture that decouples the choice of underlying blockchains from the rest of the system and is blockchain-agnostic.

Chapter 4

System Architecture

> *"Great scientists tolerate ambiguity very well. They believe the theory enough to go ahead; they doubt it enough to notice the errors and faults so they can step forward and create the new replacement theory. If you believe too much you'll never notice the flaws; if you doubt too much you won't get started. It requires a lovely balance."*
>
> RICHARD HAMMING (YOU AND YOUR RESEARCH, 1986)

In this chapter, we give an overview of a trust-to-trust principle based internet architecture and discuss design tradeoffs. We presented in Chapter 2 that blockchains can provide a solution to the bootstrapping of trust problem in distributed systems. Our internet architecture uses blockchains as the base-layer for bootstrapping trust and addresses the several scalability and performance limitations of using contemporary blockchains.

4.1 Architecture Overview

The primary goal of our internet architecture is to enable new nodes to boot up, connect to the network, and discover resources without trusting any remote party outside of their local execution environment. A secondary goal is to enable internet users to store their data reliably without revealing it to any remote party; user data cannot just stay on user devices for performance and redundancy reasons, and internet services should be able to read/write data directly from/to a user's personal storage buckets. These goals can be summarized as:

1. **Decentralized Naming & Discovery:** End-users should be able to (a) register and use human-readable names and (b) discover network resources mapped to human-readable names without trusting any remote parties.

2. **Decentralized Storage:** End-users should be able to use decentralized storage systems where they can store their data without revealing it to any remote parties.

3. **Comparable Performance:** The end-to-end performance of the new architecture (including name/resource lookups, storage access, etc.) should be comparable to the traditional internet with centralized services.

Until recently, decentralized naming systems with human-readable names were considered impossible to build (see Zooko's Triangle in Chapter 5) and decentralized storage systems like BitTorrent, etc. have never been able to offer performance comparable to centralized services [75]. Our architecture requires a solution to both these problems, is meant to scale to hundreds of millions of users and survive failures of individual components.

In this chapter, we first outline the high-level design choices we made for our architecture and then present the different components of a new decentralized internet, called Blockstack, that implements our trust-to-trust design. Blockstack is deployed in production and, to date, 72,000 new domains have been registered on it with several companies and open-source contributors actively developing new services on top [43].

Design Decision #1: Separation of the Control and Data Plane

Our architecture decouples the protocol for securely registering names and mapping them to network resources from the availability and performance of the data associated with name bindings. We do this by clearly separating the control and data planes. The control plane defines the protocol for a decentralized naming system and the associated bindings (Chapter 5) and defines the protocol used to bootstrap trust using blockchains (Chapter 6).

The data plane is responsible for data storage and availability. It consists of discovery services (pointers to user's data stores) and external storage systems for storing data (such as S3, IPFS [80], and Syndicate [81]). Data values are signed by the respective ownership keys, defined in the control plane. Any clients that read data values from the data plane can verify their authenticity by verifying the signatures.

We believe this separation is a significant improvement over systems like Namecoin or Ethereum, which implement both the control logic and the data storage plane at the blockchain level (although they leave open the possibility of using external data stores in the future). Our design not only significantly increases the data storage capacity of the system, but also allows each layer to evolve and improve independently of the other.

Design Decision #2: Agnostic of the Underlying Blockchain

Our architecture does not put any limitations on which blockchain can be used with it. Any blockchain can be used, as long as it provides total ordering of operations (which all blockchains do), but the security and reliability properties are directly dependent on the underlying blockchain. We believe that enabling the ability to *migrate* from one blockchain to another is an important design choice as it allows for the larger system to survive, even when the underlying blockchain is compromised. Section 6.2 gives more details on the migration process. Our architecture also allows for multiple underlying blockchains and treats blockchains as communication channels that deliver totally-ordered operations; any number of underlying communication channels can work as long as they can individually deliver totally-ordered operations.

Design Decision #3: Flexibility to Construct Arbitrary State Machines

We discussed in Section 3.4 that it's hard to introduce new features to blockchains after they've been deployed and gained real-world usage. In our architecture, we logically separate the blockchain layer from the rest of the architecture and give programmers the ability to construct arbitrary *state machines* after processing information from the underlying blockchain. More specifically, we introduce the concept of *virtualchains* (Chapter 6). A *virtualchain* treats transactions from the underlying blockchain as inputs to the state machine and valid inputs trigger state changes. At any given time, where time is defined by the block number of the underlying blockchain, the state machine can be in exactly one global state. Time moves forward as new blocks are written in the underlying blockchain, and the global state is updated. By using virtualchains in our architecture, we can introduce new types of state machines without requiring any changes from the underlying blockchain; virtually any new functionality can be added to the internet architecture over time without requiring modifications to blockchains. The abstraction of total ordering of operations, on top of the underlying blockchains. serves as the "narrow waist" of our architecture.

4.2 Overview of Blockstack

In this section, we present Blockstack, a system that implements different components of a new decentralized internet and takes our trust-to-trust architecture from a theoretical concept to a production system. Our architecture has four layers, with two layers (the blockchain layer and the virtualchain layer) in the control plane and two layers (discovery layer and data-storage layer) in the data plane. The control plane deals with smaller volumes of data and is mostly concerned with bootstrapping trust and defining the mapping between human-readable names and network resources. The data plane contains information on how to discover data (routes/pointers to data) and the actual storage backends. Data is replicated as widely as possible, and it doesn't matter from what source clients read data;

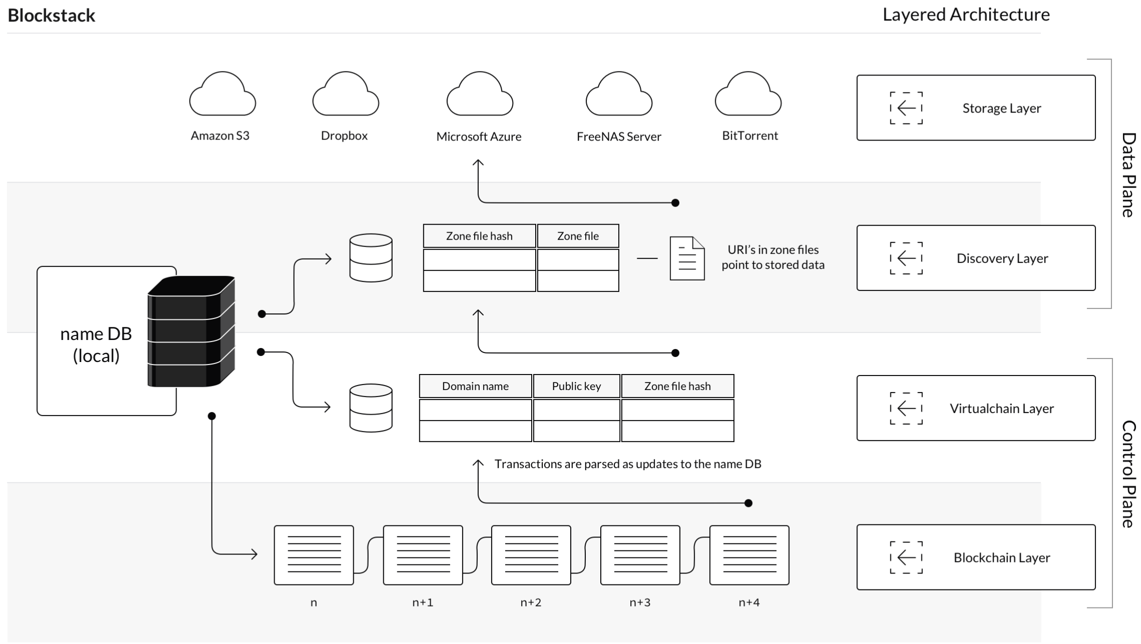


Figure 4.1: Overview of the Blockstack implementation (left) and the layered general architecture (right). Blockchain records give (name, hash) mappings. Hashes are looked up in the discovery layer to discover routes to data. Data, signed by name owner's public-key, is stored in cloud storage.

clients can independently verify from the control plane if they received the correct data or not. The 4-layers of our architecture are shown on the right side of Figure 4.1.

Layer 1: Blockchain Layer

In our architecture, the blockchain occupies the lowest layer (layer-1), and serves two purposes: it provides the storage medium for storing operations, and it provides consensus on the order in which the operations were written. Higher-layer operations are encoded in transactions on the underlying blockchain. The blockchain layer provides an abstraction of totally-ordered operations to the layer above and serves as the "narrow waist" of our architecture. A lot of complexity, like mining operations, consensus algorithms, cryptocurrency fluctuations, etc., are hidden underneath this abstraction. The higher layers only care about reading/writing totally ordered operations and can operate on top of any blockchain.

Layer 2: Virtualchain Layer

Above the blockchain, at layer-2, is a *virtualchain*, which defines new operations without requiring changes to the underlying blockchain. Nodes of the underlying blockchains are not aware of this layer. Virtualchains are like virtual machines, where a specific VM like Debian 8.7 can run on top of a specific physical machine. Different types of virtualchains (Chapter 6) can be defined and they run on top of the specific underlying blockchain. Virtualchain operations are encoded in valid blockchain transactions as additional metadata. Blockchain nodes do see the raw transactions, but the logic to process virtualchain operations only exists at the virtualchain level.

The rules for accepting or rejecting virtualchain operations are defined in the specific virtualchain e.g., a virtualchain that defines a single state machine implementing operations for a global naming system. Operations accepted by rules defined in the virtualchain are processed to construct a database that stores information on the global state of the naming system along with state changes at any given blockchain block.

Layer 3: Discovery Layer

The discovery layer (layer-3) is part of the data plane. Our architecture separates the task of *discovering* resources (i.e., routes to data) from the actual storage of data. This avoids the need for the system to adopt any particular storage service from the onset, and instead allows multiple storage providers to coexist, including both cloud storage and P2P systems.

The Blockstack implementation uses *zone files* for storing routing information, which are identical to DNS zone files in their format. The *zone files* are stored in the discovery layer, implemented as a peer-to-peer network by Blockstack (Chapter 7). **Users do not need to trust the discovery layer** because the integrity of any data record in the discovery layer can be verified by checking the respective hash of that data record in the control plane.

In Blockstack's current implementation, nodes form a peer network, called the Atlas network (Chapter 7), for storing *zone files*. The peer network only allows *zone files* to be written if $hash(zonefile)$ was previously announced in the blockchain. This effectively whitelists the data that can be stored in the peer network. Data records representing routes (irrespective of where they are fetched from) can be verified and therefore cannot be tampered with. In the current implementation of the Atlas network, peer nodes maintain a full copy of all *zone files* since the size of *zone files* is relatively small (4KB per file). Keeping a full copy of all routing data introduces only a marginal storage cost on top of storing the blockchain data (which is in the order of several GB s).

Layer 4: Storage Layer

The top-most layer (layer-4) is the storage layer, which hosts the actual data values and is part of the data plane. All stored data values are signed by an owner key defined in the control plane. By storing data values outside of the blockchain, our architecture allows values of arbitrary size and allows for a variety of storage backends. **Users do not need to trust the storage layer** and can verify their integrity in the control plane. Our design benefits from the performance and reliability of the backend cloud storage systems used and offers comparable performance to traditional internet services (Chapter 8).

Blockstack Components

Blockstack implements a decentralized naming system (Chapter 5), called the *Blockchain Name System* (BNS) by defining operations in a new virtualchain (Chapter 6) and storing discovery data in a peer network called the *Atlas Network* (Chapter 7). Our virtualchain uses the underlying blockchain to achieve consensus on the state of BNS and binds names to data records. Relying on the consensus protocol of the underlying blockchain, our virtualchain can provide a total ordering for all operations supported by BNS, like name registrations, name updates, and name transfers. Our virtualchain represents the global state of BNS, including who owns a particular name and what data is associated with a name. We present more details on these components in the next chapters.

Chapter 5

Secure Naming

"The comsats rented processor time almost as cheaply as ground stations, and an automatic payment transaction (through several dummy accounts set up over the last several years) gave him sole control of a large data space .. Mr. Slippery (the other name was avoided now, even in his thoughts) had achieved the fringes of the Other Plane .."

– VERNOR VINGE (TRUE NAMES, 1981)

The traditional internet uses the Domain Name System (DNS) for mapping human-readable names to IP addresses (which give the location of nodes and content). When internet users type in cnn.com in their browser, DNS servers return the mapping of the human-readable name to an IP address. ICANN, a non-profit organization, manages DNS and the root servers. These servers are a central point of trust and failure; they can be taken offline by DDoS attacks and mappings for domains can be changed by either forcing changes to the DNS servers or by spoofing replies from them.

Historically, in the early days of the internet, there was no concept of names for nodes (clients or servers) connected to the network. Every node had an IP address, and you could connect to that node by using the IP address as the identifier on the network. By convention, network admins maintained hosts.txt files where they'd map human- readable names to IP addresses. Over time, the manual system of syncing host files was replaced by DNS.

In our internet architecture, we need a decentralized replacement for DNS, i.e., a system that binds human-readable names to discovery data but doesn't have any central points of failure or control. There is a school of thought that argues that human-readable names are not important and long cryptographic IDs combined with search engines can be a replacement for DNS [33, 80, 99]. In this thesis, we take the view that human-readable names are essential for providing a good user experience and, in practice, it'd be very hard to convince internet users to change their habits and stop using human-readable names online. In this chapter, we outline the challenges of building decentralized naming systems, give a background on the current blockchain-based naming systems, and present a new decentralized naming system called the Blockchain Name System (BNS).

5.1 Background on Decentralized Naming

In this section, we describe the motivation for building naming systems that have no central point of trust and provide the relevant background on blockchain-based naming systems.

Zooko's Triangle

There is a fundamental computer science challenge with building naming systems. There are three properties we might want a name to have: the name is (1) unique (meaning there is no situation where two people can independently create and use a unique name like cnn.com), (2) human-readable (a name should look like Paul not 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa), and (3) decentralized (names should be chosen by users at the edges of the network and not on behalf of users by a central authority at the center). The computer science challenge is that, before blockchains, naming systems only allowed for any two of those three properties [83], never all three at the same time. This limitation is called *Zooko's Triangle* [84]. For example, public keys are unique and decentralized as users can generate them on their computers without talking to any central service but are not human-readable. Twitter handles are human-readable and unique, but not decentralized (Twitter, the company, controls the namespace). Nicknames are human-readable and decentralized (users can choose any nickname for anyone) but are not unique. Blockchains square Zooko's Triangle, and for the first time it's possible to have human-readable names that are unique without using any centralized service [84].

In this chapter, we use the term *naming system* to mean (a) names are **human-readable** and can be picked by humans, (b) name and any associated data have **a strong sense of ownership**—that is, they can be owned by cryptographic keypairs, and c) there is **no central trusted party** or point of failure. We use the term **name-value pair** to refer to a key-value pair where the key is a human-readable name registered in a naming system.

Blockchain-based Naming Systems

It's hard to tamper with data stored in a blockchain because it requires a prohibitively high computing resources; re-writing blockchain data requires proof-of-work [25]. There are several naming systems built using blockchains, like Namecoin [104], Ethereum Name System [62] and BitShares [23], that store name ownership data in blockchains. These naming systems make name registration decentralized, and no third-party can take away ownership of domains from users. Users trust their personal computers, or servers that they run, instead of relying on remote DNS servers, significantly reducing the attack vector of DDoS (an attacker will need to DDoS individual users instead of centralized servers). Further, blockchain-based naming systems make it prohibitively hard to tamper with name bindings. However, with a few exceptions like Namecoin and our work, blockchain-based naming projects are in an early stage and haven't been widely deployed [97]. In this chapter, we only focus on Namecoin and our work and discuss other related works in Chapter 10.

Namecoin

Namecoin [104] was the first system to build a decentralized naming system using a blockchain. Namecoin's blockchain gives consensus on the global state of the naming system and provides an append-only global log for state changes. Writes to name-value pairs can only be announced in new blocks, as appends to the global log. The global log is logically centralized (all nodes on the network see the same state), but is organizationally decentralized (no central party controls the log).

Before our work, it was common practice to start new blockchains (by forking them from Bitcoin) to introduce new functionality and make modifications required by the respective service/application, which is the precise approach taken by Namecoin [104]. In Namecoin, just like DNS, there is a cost associated with registering a new name. The name registration fee discourages people from registering a lot of names that they don't intend to use; the recipient of registration fees is a "black hole" cryptographic address from which money cannot be retrieved [83]. Namecoin defines a pricing function for how the cost of name registrations changes over time. Namecoin supports multiple namespaces, like top-level-domains (TLDs) in DNS, and the same rules for pricing and name expiration apply to all namespaces. By convention, the $d/$ namespace is used for domain names. For example, to register the domain $yahoo$ on Namecoin, one must register the name $d/yahoo$ and then put the IP address of the Yahoo! website in the name/value pair.

In Namecoin, name registration uses a two-phase commit method where a user first **pre-orders** a name hash in a new transaction that includes $hash(name)$ in the transaction. This does not reveal what $name$ she is trying to register. After the pre-order transaction has been confirmed by the network—, i.e., enough blocks (typically 10) are later added to the blockchain to make it computationally infeasible for any miner to re-write recent blockchain history and reverse the transaction—the user can reveal the $name$ she was trying to register. This is done by sending a second transaction on the network that completes the **register** step. The user also includes the $value$ of the name/value pair in the second transaction. The cryptocurrency address that signed the two transactions becomes the owner of the newly registered name/value pair. Name registrations expire after a fixed amount of time, measured in new blocks written (currently 36,000 blocks, which translates to roughly 8 months). Namecoin also supports updating the value associated with a name, as well as ownership transfers.

5.2 BNS: Blockchain Name System

In this section, we present the design of a new blockchain-based naming system, called the Blockchain Name System (BNS) and discuss an implementation of BNS that has been running in production for more than 2 years. In our trust-to-trust internet architecture, BNS is a replacement for DNS and is meant to provide similar functionality but without any central root servers. In BNS, names are owned by cryptographic addresses of the underlying blockchain and their associated private keys (e.g. ECDSA-based private keys used in Bitcoin [45]). As with Namecoin, a user *preorders* and then *registers* a name in a two-phase commit process. This is done to avoid front-running where an attacker can race the user in registering the name because an attacker will be able to see the unconfirmed transaction on the network. The first user to successfully write both a preorder and a register transaction is granted ownership of the name. Further, any previous preorders become invalid when a name is registered. Once a name is registered, a user can *update* the name-value pair by sending an update transaction and uploading the name-value binding. Name *transfer* operations simply change the address that is allowed to sign subsequent transactions, while *revoke* operations disable any further operations for names.

In BNS, names are organized into $namespaces$, which are the functional equivalent of top-level domains in DNS—they define the costs and renewal rates of names. Like names, namespaces must be preordered and then registered. As shown in Figure 5.1, in BNS the information for top-level domains (namespaces) is registered on a *root blockchain*. Entries

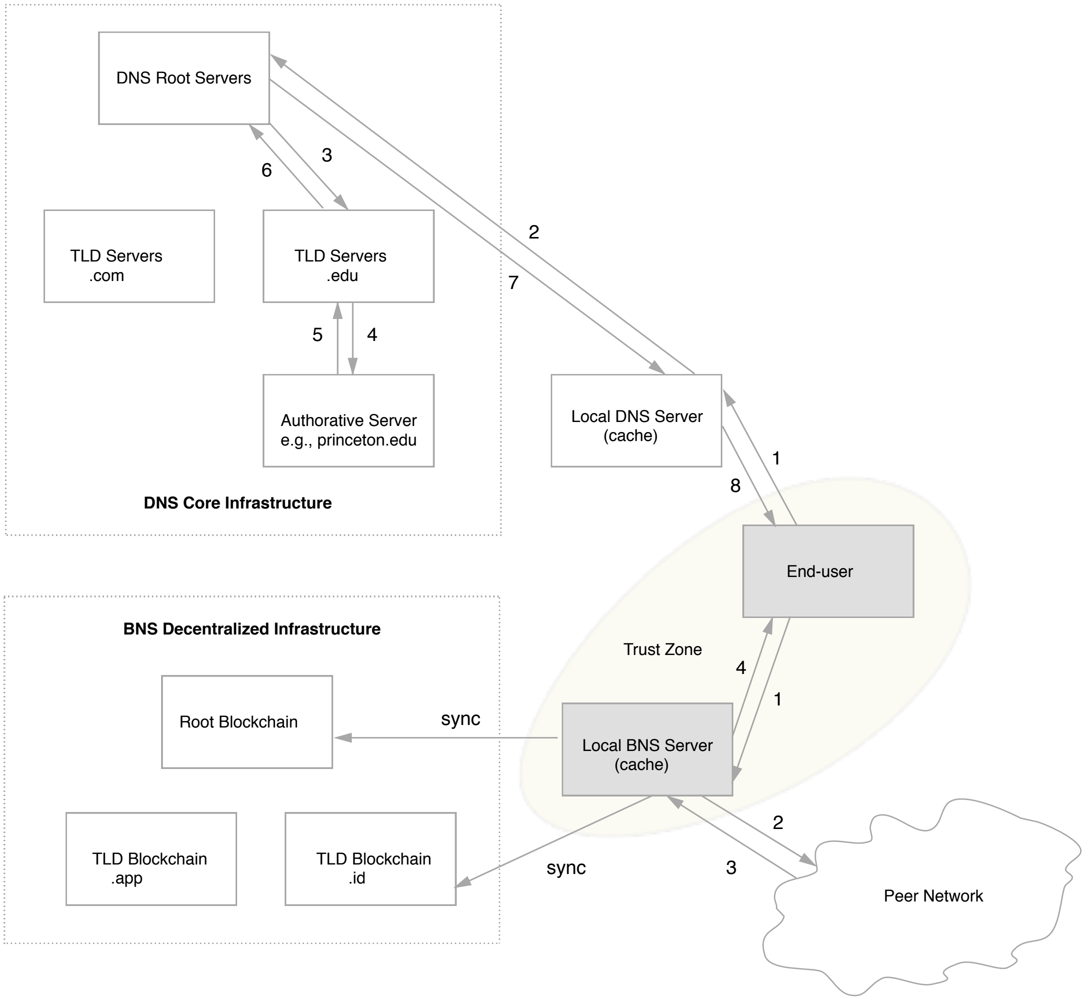
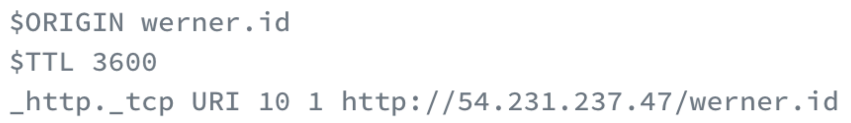


Figure 5.1: A recursive DNS query (top) for princeton.edu and an iterative BNS query for werner.id. End-user and the local BNS server are in the same trust zone.

for TLDs can point to other blockchains that store data for domains registered on that TLD. The root blockchain can also be used for defining TLDs, in which case the TLD entry points to the same blockchain. In DNS, the DNS root servers, TLD servers, and authoritative servers are outside the *trust zone* of the end-user, where the trust zone is defined as either a local machine or local network and can include a node controlled (and trusted) by the end-user in the wide-area. Figure 5.1 (top) shows a recursive DNS query for princeton.edu. The query is resolved outside the user's trust zone. In BNS, the local BNS server fetches blockchain data from the respective (decentralized) blockchain networks and keeps a local copy that is continuously synced with the blockchain networks. Individual blockchain



```
$ORIGIN werner.id
$TTL 3600
_http._tcp URI 10 1 http://54.231.237.47/werner.id
```

Figure 5.2: An example zone file for BNS.

records are small and contain pointers to data outside the blockchain, for example in a peer network. To resolve a name, say werner.id, the end-user makes a query to the local BNS server running inside her trust zone. The local BNS server looks at the respective blockchain record and fetches the respective zone file from an external source, like a peer network. The external source for zone files is untrusted. The hash of the zone file is present in the blockchain record, and any tampering attempt can be easily detected by checking the zone file against the hash. Figure 5.2 shows an example BNS zone file for a single domain.

Pricing Functions

Anyone can create a namespace or register names in a namespace, as there is no central party to stop someone from doing so. *Pricing functions* define how expensive it is to create a namespace or to register names in a namespace. Defining intelligent pricing functions is a way to prevent "land grabs" i.e., stop people from registering a lot of namespaces/names that they don't intend to actually use. BNS has support for sophisticated pricing functions. For example, we created a $.id$ namespace in our implementation of BNS with a pricing function where (a) the price of a name drops with an increase in name length and (b) introducing non-alphabetic characters in names also drops the price. With this pricing function, the price of *john.id* > *johnadam.id* > *john0001.id*. The function is generally inspired by the observation that short names with alphabets only are considered more desirable on namespaces like the one for Twitter usernames. It's possible to create namespaces where name registrations are free as well. Further, we expect that in the future there will be a reseller market for names, just as there is for DNS. A detailed discussion of pricing functions is out of the scope of this thesis, and the reader is encouraged to see [83] for more details on pricing functions and name squatting problems in decentralized naming systems.

BNS Implementation

Our implementation of a new decentralized internet, Blockstack, uses BNS as the default naming system. BNS is implemented by defining a state machine and rules for state transitions in a new virtualchain. We store zone files in a new peer network, called the *Atlas Network*. We present the details for our BNS implementation with virtualchains and Atlas in Chapter 6 and Chapter 7 respectively. Like names, namespaces also have a *pricing function* [42]. **To start the first namespace on Blockstack, the $.id$ namespace, we paid 40 bitcoins ($10,000 at the time) to the network. This shows that even the developers of this decentralized system have to follow the rules and pay appropriate fees.**

Simple Name Verification

In BNS, the local BNS server should be in the user's trust zone. The local BNS server needs to keep a consistent copy of all blockchains used by BNS, and the storage requirements can be in the order of hundreds of gigabytes (given the current size of contemporary blockchains like Bitcoin [2]). BNS has support for "thin clients," which can query the system without running BNS servers locally or having access to the full blockchain history. Support for thin clients is important for users on mobile devices.

Virtualchain nodes can independently calculate a *consensus hash* at any blockchain block. Consensus hashes help virtualchain nodes figure out if they have the same view of the global state at any given block. Each consensus hash $CH(n)$ is constructed from block n's sequence of virtualchain operations V_n, and (a Merkle root of) geometric series of prior consensus hashes P_n defined by:

$$CH(n) = Hash(V_n + P_n) \tag{5.1}$$

where

$$P_n = \{CH(p) | p = n - 2^i, i \in \mathbb{N}, n - 2^i >= n_0\} \tag{5.2}$$

and n_0 is the first block. Other than detecting that two virtualchain nodes have the same global view, consensus hashes also address the *endless ledger problem* (defined in Section 2.2). As the underlying blockchain grows in size, new nodes need to process more and more blocks before they boot up.

A new node can bootstrap by using an untrusted database of state information at a given block number, combined with a trusted consensus hash $CH(n)$ of the same block number. The block number is also termed the *block height* in the literature, and it increases with each new block. A new node can reconstruct the virtualchain from the untrusted database and reprocess virtualchain operations at each blockchain block, recalculating each $CH(n)$ along the way. If the final consensus hash matches the trusted consensus hash at block n, then the database associated with n is trustworthy, and the node can start processing blocks after n. This is much faster than the traditional approach of starting from the first block n_0 and fetching all transactions, even though most of them will be discarded.

The process of verifying the authenticity of a prior name operation with a later trusted consensus hash is called *Simplified Name Verification (SNV)*. As such, if a user trusts that $CH(n)$ is authentic, then she can query and verify the *virtualchain* operations V_n and previous consensus hash P_n for block n. The construction of $CH(n)$ allows a user to verify the authenticity of any *virtualchain* operation from a block with height $n_{prior} < n$, using only a logarithmic number of queries. More details on *consensus hashes* are in Section 6.1.

Figure 5.3 shows an example SNV query. Each row represents the blockchain, in increasing block height order from left to right ($n > n_0$). Here, the user can verify the authenticity of a name operation in a target block (marked with T_q). In each step, the user recursively trusts the consensus hash for the white outlined blocks. To do so, the user iteratively queries V_n and P_n for a given n verifies that they hash to $CH(n)$, and then se-

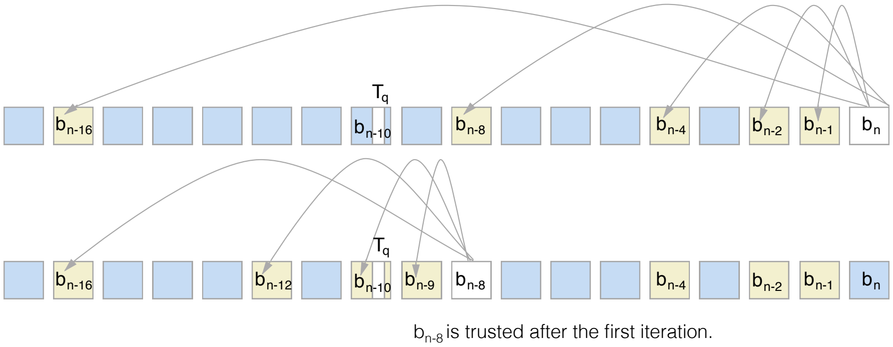


Figure 5.3: Overview of SNV. Example SNV query of a record (T_q) in block b_{n-10}.

lect n's predecessor n' to be the smallest height greater than n_{prior} for which $CH(n')$ was previously queried and verified.

5.3 Blockchain-based Public Key Infrastructure

On the traditional internet, DNS domains can be used with digital certificates to enhance security. Digital certificates for websites are the foundational building block of internet security. When users see the "green lock sign," they feel that they're on a secure connection. In the background, their browser checks the digital certificate of the website. The "green lock sign" represents that some Certificate Authority (CA), like Verisign, issued a digital certificate to a website and the website has ownership of that certificate. The Certificate Authority can issue "malicious" certificates that impersonate businesses and websites without their permission and users would end up trusting the malicious certificates – a real problem that has happened several times in recent history, e.g., Turktrust, a Turkish CA, issued malicious certificates for Google.com [124].

A blockchain can be used as a global distribution mechanism for public keys and digital certificates. Since blockchains give a global view and are extremely hard to tamper with, it'd be impractical for an attacker to alter a certificate after its issued or present incorrect information to only a subset of users. Also, in a blockchain-based public-key infrastructure there are no central CAs that can be compromised to attack the system.

Current blockchain-based naming systems like BNS or Namecoin, already provide public key associations with domain names and all domains, by default, get certificates. While efforts like Lets Encrypt [16] are reducing the cost of obtaining digital certificates and encouraging more websites to enable secure connections, a vast majority of the internet still runs on insecure connections. If the naming system is built using a blockchain, then all websites have security certificates; security is on by default. In BNS, domain names can serve as memorable identifiers for public keys. Names make no implication about identity and are used as memorable identifiers only. Third-party attestations can be attached to the memorable name later on. Further, all BNS nodes can see the global state, so any key revocations or state changes to public key mappings cannot be hidden from any user.

A Production Deployment on Namecoin

Before our implementation of BNS on Blockstack, we deployed a simple public-key infrastructure (PKI) service on Namecoin [112]. For our PKI service, we started a new namespace $u/$ on Namecoin. We defined the format for publishing public keys, like PGP [136], along with other profile data in the blockchain [6]. Namecoin already had support for human-readable names and registering name-value pairs. We launched a web service [112] in March 2014 that enabled people to easily register names. All registered names have an ECDSA public key [79] binding by default and a subset of users have added their PGP keys as well. According to a study by Harry et al. [83], our system was the second largest namespace on Namecoin by volume and the largest by the number of active users. Lessons from this deployment (Chapter 3) guided the design of BNS and how we bind BNS domains to public keys. Our design of storing most of the data outside of blockchains scales much better [27]. Data read/writes are not limited by the blockchain bandwidth at all.

Chapter 6

Virtualchain

"Time, he said, is what keeps everything from happening at once."

– RAY CUMMINGS (THE GIRL IN THE GOLDEN ATOM, 1922)

Public blockchains are becoming a universal network service. However, it's hard to make consensus-breaking changes to production blockchain networks. Introducing new features directly in a blockchain requires everyone on the network, including miners, to upgrade. These upgrades potentially break consensus and cause forks [45]. Our experience with the Namecoin blockchain shows that starting new, smaller blockchains leads to security problems, like reduced computational power needed to attack the network, and should be avoided when possible (Chapter 3). To overcome this, we created *virtualchains*, a virtual blockchain for creating arbitrary state machines on top of already-running blockchains. Virtualchains, like virtual machines, enable the ability to migrate (from one blockchain to another) and improve fault tolerance. We present lessons from a successful migration of a production network from Namecoin to Bitcoin, using a virtualchain. To the best of our knowledge, this was the first cross-chain migration of a production system running on blockchains. The migration showed that virtualchains could be used to cope with failures at the blockchain layer (by migrating systems away from the underlying failing blockchain).

Challenges with Building Applications with Blockchains

Blockchains provide a totally-ordered, tamper-resistant log of state transitions. New applications can store a log of all state changes in a public blockchain, such as Bitcoin [127], Litecoin [94], or Ethereum [46]. By using the blockchain as a shared communication channel, these applications can then **bootstrap global state** in a secure, decentralized manner, since every node on the network can independently construct the same state.

However, there are two key challenges to using blockchains as a building block for decentralized applications and services:

1. First, a blockchain can fail, i.e., it can go offline, or its consensus mechanism can become "centralized" by falling under the *de facto* control of a single entity. We presented in Chapter 3 how Namecoin, the oldest cryptocurrency other than Bitcoin, had a single miner with more than 51% mining power [133]. To tolerate such failures, it should be possible to migrate application state across blockchains efficiently.

2. The second challenge is that the application's log can be forked and corrupted if the underlying blockchain forks. Under a blockchain fork, nodes on different forks will write and read different events. The blockchain may drop and re-order transactions when the forks resolve, causing bootstrapping nodes to construct different state than already-running nodes. Applications must be able to recover from blockchain forks.

6.1 Design of Virtualchains

Virtualchain is a virtual blockchain (a logical layer) for building multiple state machines and their respective transition logs (also called journals) on top of a blockchain. Virtualchains process transactions in the underlying blockchains to construct state machines on top of blockchains. Virtualchains provide a fork*-consistency model [92]. Application nodes replay their logs to achieve *application-level* consensus at each block b, such that

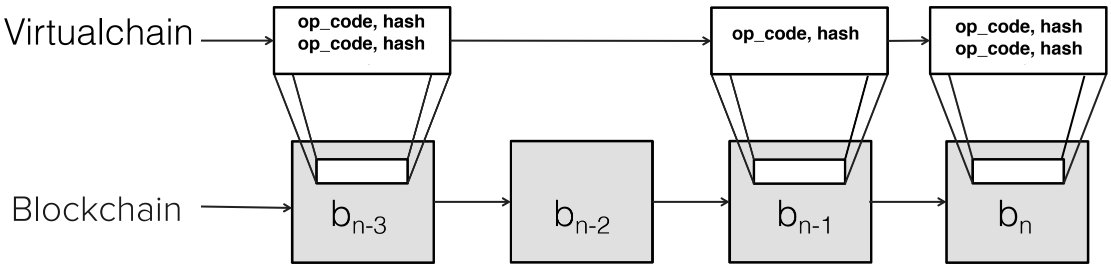


Figure 6.1: Virtualchain operations on top of an underlying blockchain.

two nodes will agree on a block if and only if the application transactions in that block leave the nodes in an identical state. If their resulting state after executing the operations in block b are identical, then their generated *consensus hash* (Section 5.2) for that block will be the same. Consensus hashes enable nodes to independently audit and efficiently query their log history, as well as detect forks and then migrate state between blockchains.

Figure 6.1 shows how virtualchains process only relevant transactions (transactions with valid virtualchain opcodes) from the underlying blockchain and ignore other transactions. The transactions accepted by virtualchain are organized in virtual blocks that are linked together by a consensus hash per block; the consensus hash at a block reflects all previous virtualchain history. We have used virtualchains to implement a BNS (Section 5.2) state machine in a new virtualchain. Our virtualchain currently uses Bitcoin as the underlying blockchain. The new opcodes are announced in Bitcoin transactions in a field designated for additional data, called *OP_RETURN*. This is one of the largest use cases of *OP_RETURN* transactions on the Bitcoin blockchain today [114].

Consistency Model

At a high level, blockchains are append-only, totally-ordered logs of transactions that are replicated across all nodes of the network [45]. A transaction t is a signed statement that moves tokens owned by a cryptographic keypair. Transactions are causally linked: t_1 happens before t_2 if unspent tokens in t_1 are used (spent) in t_2. Blockchain peers append transactions by announcing new (unwritten) transactions and executing a leader election protocol to determine the writer of the next block (containing new transactions) in the global blockchain. Most public blockchains use a variant of *Nakamoto consensus* [45], which allows concurrent leaders. Appending conflicting blocks creates *blockchain forks*, which peers resolve using a proof-of-work [78] metric. Transactions in the fork with the most proof-of-work are considered authoritative; conflicting transactions are silently discarded, while non-conflicting transactions are incorporated into subsequent blocks.

Nakamoto consensus gives blockchains the property that longer forks are exponentially rarer if there are no long-lasting network partitions and if most of the compute power is controlled by honest peers [45]. This means that most of the time, transactions are very likely to be durable and linearizable after a constant number of blocks (*confirmations*) have been appended on top of them. We use these properties to implement fork*-consistent replicated state machines (RSMs) on top of public blockchains. Application nodes read the blockchain to construct state machine replicas and submit new transactions to the blockchain to execute state transitions.

Our work differs from prior fork*-consistent systems [64, 65, 92], as we enable open membership: the sets of both application users and application nodes are dynamic and may be empty since we use an external blockchain to establish the ground truth of the state and to propagate state transitions. In this section, we describe how *consensus hashes* are calculated, how we enable efficient queries on prior state transitions *without* requiring a full blockchain or state machine replica, and how we detect and recover from forks caused by long-lived forks in the underlying blockchain.

Consensus Hashes

To make forward progress, nodes read new blockchain transactions and determine whether or not each transaction of the underlying blockchains represents a valid state transition in virtualchain. Since anyone can write transactions and they can get arbitrarily delayed, nodes must be able to filter transactions (and associated state transitions) and ignore trans-

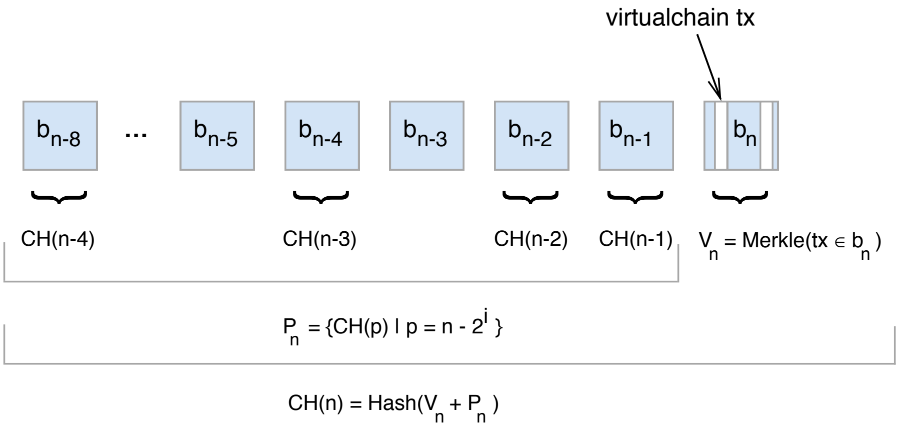


Figure 6.2: Consensus Hash, $CH(n)$, construction from virtualchain transactions.

actions that relate to a fork that they're not interested in. We achieve this by requiring that the current *consensus hash* is announced in new transactions.

A consensus hash is a cryptographic hash that each node calculates at each block. It is derived from the accepted state transitions in the last-processed block, and a geometric series of prior-calculated consensus hashes. Figure 6.2 shows this process. Let $tx \in b_n$ be the sequence of transaction logs found in block b_n, let $Merkle(tx \in b_n)$ be a function that calculates the Merkle tree root over these transactions, and let $Hash(x)$ be a cryptographic hash function. Then, we define $CH(n)$ to be the consensus hash at block n, where

$$V_n = Merkle(tx \in b_n) \tag{6.1}$$

$$CH(n) = Hash(V_n + P_n) \tag{6.2}$$

Block b_0 contains the first log entry, while P_n is the geometric series of prior consensus hashes starting from b, i.e., the consensus hash for the previous block, two blocks ago, four blocks ago, etc. Consensus hashes were discussed earlier in Section 5.2.

Users include their latest known $CH(n)$ in each transaction they submit through their clients, and applications ignore state transitions with "stale" (too old) or unknown consensus hashes. This way, applications ignore forks of their own log/journal, and application users (or the clients they're using) can tell when to retry lost transactions (announcing state transitions). In doing so, **consensus hashes preserve the join-at-most-once property of fork*-consistency**: an application will accept a state transition with $CH(n)$ only if it has accepted all the prior state-transitions that derived $CH(n)$.

Fast Queries.

Not all users will have a copy of the full blockchain on their machine. We use a protocol for fast queries that is useful for creating "lightweight nodes" that do not need blockchain or state replicas. Instead, they can query highly-available but untrusted "full nodes" (which have a full copy of the blockchain) as needed. For example, Blockstack's virtualchain uses this feature to implement a Simple Name Verification (SNV) protocol (Section 5.2).

For fast queries, application users obtain $CH(n)$ from a trusted node, such as one running on the same host. A user can then use this trusted $CH(n)$ to query previous state transitions from untrusted nodes in a logarithmic amount of time and space. To do so, it iteratively queries and verifies P_n and $Merkle(tx \in b_n)$ using $CH(n)$ until it finds $CH(n')$ and $Merkle(tx \in b'_n)$, where b' is the block that contains the state transition to query. Once it has $Merkle(tx \in b'_n)$, it can ask for and verify the previous state transitions ($tx \in b'_n$).

Blockchain Fork Detection and Recovery.

If the transaction logs never retroactively fork, the application logic and consensus hashes can preserve the legitimate-request property of fork*-consistency. Retroactive forks in proof-of-work blockchains are highly unlikely, but they *can* occur since an entity can (theoretically) come up with a longer blockchain with a different transaction history of old blocks (called a "deep chain reorg"). Short-lived forks, on the other hand, are fairly common and are not an issue for applications/services built with virtualchains. Nodes avoid **short-lived forks** by only accepting sufficiently-confirmed transactions. Applications may

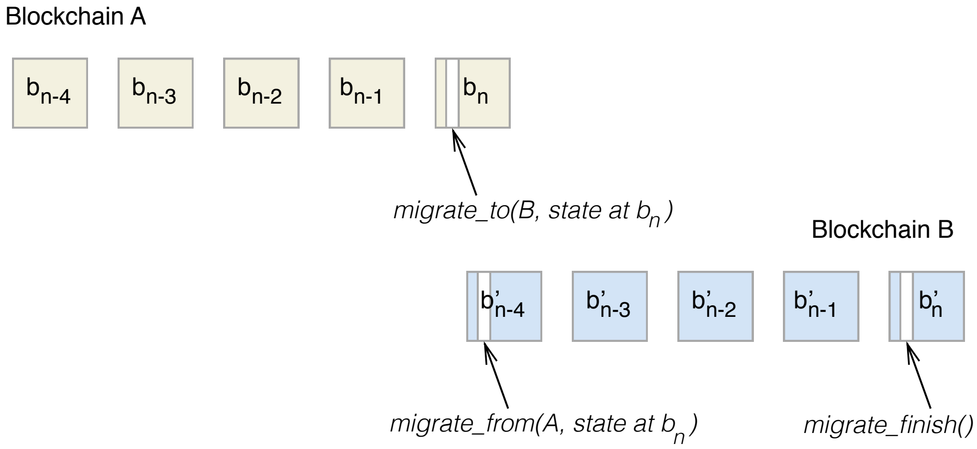


Figure 6.3: A framework for migrating from blockchain A to blockchain B.

increase the number of required confirmations to decrease the likelihood of loss or reordering, e.g., Blockstack requires 10 confirmations (in the Bitcoin blockchain).

To detect **deep chain reorgs**, a node runs multiple processes that subscribe to a geometric series of prior block heights. If a process at a lower height derives a different consensus hash than one from a higher height, then a blockchain fork might have occurred, and all processes at higher heights have potentially-divergent state. This means all running nodes may be in a separate fork set from bootstrapping nodes.

We can automate deep reorg discovery, but reconciling the fork sets requires human intervention, since irreversible actions taken by the application may be based on now-lost state transitions. Fortunately, long-lived forks are rare and severe enough to be widely noticed [36] [37] [39]. This means that when they happen, end-users or app developers can determine which transactions were affected, and re-send state-transitions.

6.2 Cross-chain Migration

Virtualchains can survive the failure of an underlying blockchain by migrating state to another blockchain. Doing so requires announcing a future block until which the current blockchain will be valid (no new transactions will be accepted on the current blockchain

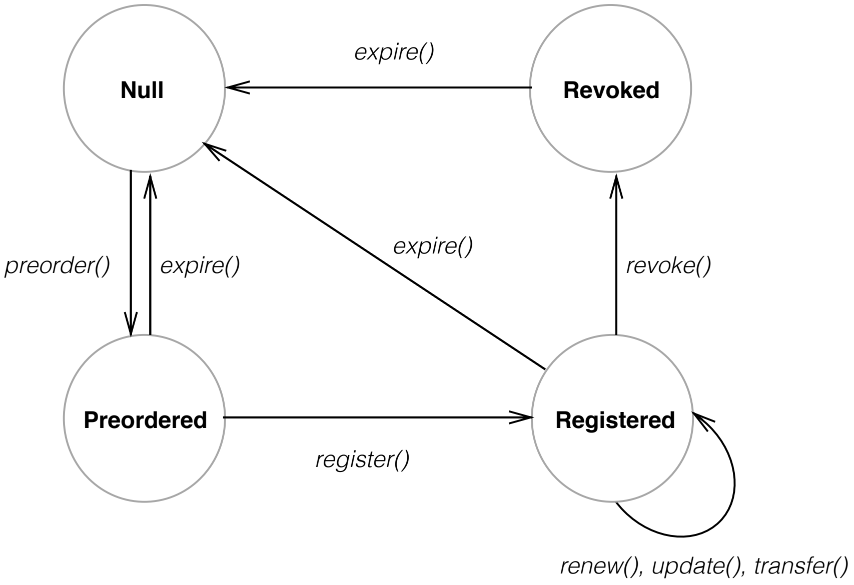


Figure 6.4: The state machine for BNS, showing states and transitions for a name.

after that block), and then executing a **two-step commit** to bind the existing state to the new blockchain. Figure 6.3 shows the framework where migrating from blockchain A to B. To begin, the app/service administrator(s) announces a future block after which the current blockchain will no longer be used for the app/service and sends special "migrate" transactions to both the current and the new blockchain (to announce the migration process). The administrator(s) (a) acquires a lock on the new blockchain, (b) writes the current application state (excluding historic state transitions) to the new blockchain, and (c) releases the lock on the new blockchain and opens up the new blockchain to new transactions. Virtualchain verifies that the migrate transactions are signed by the same principal and verifies that the last-known state on the old blockchain is consistent with the consensus hash announced on the new blockchain. This enables seamless cross-chain migration.

Virtualchain enables applications to use any blockchain for consensus and migrate state between them. Virtualchain stores each application's state-transition journal on the underlying blockchain, and handles application consensus at a logical-layer on top of the blockchain. Virtualchain is already used in a production system on the Blockstack network. Figure 6.4 shows the Blockstack Name System (BNS) state machine implemented

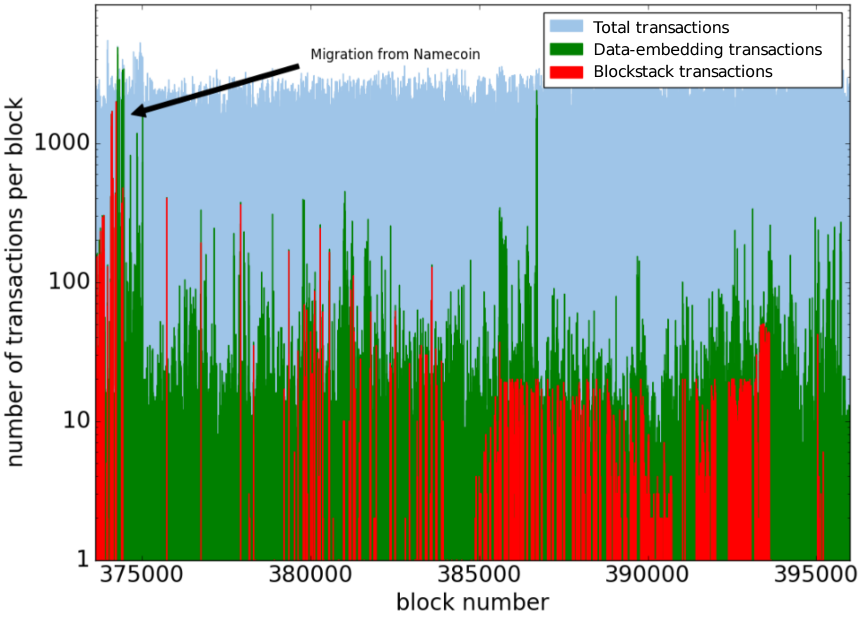


Figure 6.5: All data-embedding transactions on Bitcoin; non-financial applications and services on top of Bitcoin are already becoming a frequent use case.

by Blockstack, the different states a $name$ can be in and how state transitions work. This virtualchain implementation has been used to register 70,000 domains [27]. Virtualchain is released as open source [132] and developers can build different state machines using it.

Lessons from a Migration to Bitcoin

Migration in virtualchains, from one underlying blockchain to another, works by first acquires a lock on the new blockchain, transferring the current state of the state machine, and then explicitly release the lock when the migration completes. In September 2015, we completed migration of 33,000 users of our production system [112], from Namecoin to Bitcoin, using a virtualchain. These users were migrated from the $u/$ namepsace on Namecoin to the $.id$ namespace on BNS, the default naming system of the Blockstack network.

Our virtualchain for this particular application embeds additional data in Bitcoin transactions using special fields dedicated for including arbitrary data [40]. Embedding additional data in Bitcoin transactions is already a popular way of defining higher-level protocols on top of Bitcoin, like Counterparty [57], Open Assets [113], etc. Figure 6.5 shows recent bandwidth usage of data-embedding protocols on the Bitcoin blockchain. The spike of 10,000+ transactions, near block 375000, was during our migration to Bitcoin. The Blockstack network, running on Bitcoin, currently accounts for 18.6% of all data-embedding transactions ever made on Bitcoin [114]. Below are some observations we made while working with the Bitcoin network:

Network Throughput: Bitcoin currently supports between 3 and 7 transactions per second with a 1MB block size. Even after two years of heated debate [135] amongst the Bitcoin developers and the broader community, the block size has not been increased. We noticed these limitations first hand when we throttled our transactions so that our transactions wouldn't exceed 20-30% of Bitcoin blocks, which in turn significantly increased the amount of time it took for completing registrations. When scaling to millions of users, as opposed to thousands, even 8MB or larger blocks will not suffice, and we need to look into performing registrations across multiple chains [31] and explore novel methods for packing multiple name operations in a single transaction [48].

Network Attacks: During our migration to Bitcoin, a UK-based company called CoinWallet was performing a stress test on the Bitcoin network [55]. The stress test included a high volume of small transactions which had transaction amounts that were too low for miners to package in a block (due to protocol rules designed to prevent spam). This resulted in an extremely high number of unconfirmed transactions on the network and we ended up paying 2-3 times higher transaction fees to get our transactions packaged by miners. This experience shows how a single actor can force high mining fees on the rest of the network (although in this case there was a cost factor attached to the attack which limits the duration of the attack). We believe that networking attacks on blockchains, like the one we experienced or other DDoS attacks [100], are likely to become more frequent. Protections against such attacks is an important area of future research.

Chapter 7

Content Discovery

"Information is the resolution of uncertainty."

– Claude E. Shannon (1916-2001)

Blockchains have limited bandwidth and cannot store much data. Every node on the network has a copy of the data stored on blockchains, and they typically grow linearly with time, e.g., the Bitcoin blockchain grew from 14GB to 120GB between 2014 and 2017 [2]. In our architecture, only pointers to data values are kept in the blockchain; peer-networks are used as additional storage. Using peer networks significantly increases the storage capacity but comes with other challenges: traditional peer-networks are susceptible to Sybil attacks [59] and are not a reliable source of data, especially under high churn. In this chapter, we present two new peer-networks that we designed to improve the reliability of peer networks and protect them against Sybil attacks.

These peer-networks logically exist at the discovery layer of our architecture and, in addition to extending the storage capacity of blockchains, provide a discovery service for other network resources like backend storage systems. In the Blockstack implementation, the peer-networks store zone files for BNS (these zone files are identical to DNS zone files).

7.1 Peer Networks for Content Discovery

In peer-to-peer networks (also called *P2P networks* or simply *peer networks*) participating nodes are equally privileged and collaborate to perform a function or provide a service. Peer networks were popularized by file sharing networks like Napster in 1999 [47]. Nodes in a peer network maintain a connection to a subset of other peers on the network and these connections can be structured or unstructured (random connections to peers). In our architecture, we use peer networks for content discovery. Pointers to large data files are stored in peer networks, while the actual data resides on storage backends (Chapter 8).

The reliability of the applications and services running on our internet architecture depends on the reliability of the blockchain layer and discovery/storage peers. Out of the different layers, the peer networks used for discovery are the most vulnerable to reliability issues (cloud storage providers have 99.9$\%$ uptime SLAs [15] and blockchains are fully-replicated across peers). Theoretically, any person or company can decide to run a (centralized) index of discovery data for their particular app/service. Apps can also choose to index/mirror only a particular namespace (TLD), and they don't have to index the pointers to all data. This helps with scalability. Let's say there are m namespaces with n name-value pairs in each namespace. Instead of indexing $O(m \times n)$ records, you can index $O(n)$ records and n for your namespace could be significantly small. However, realistically, the global Blockstack network should have at least one, if not more, default discovery service for all data in addition to any specialized app-specific discovery services. Further, the global discovery service cannot violate the trust-to-trust principle and cannot be centralized. This implies the need to use decentralized peer-to-peer networks for content discovery.

Challenges with Peer Networks

Peer networks are well studied in distributed systems [30, 75] and researchers have identified several challenges with peer networks.

- **Scalability:** In unstructured peer networks, as the number of participant peers increases, the number of messages exchanged for a lookup grows [30]. Practical unstructured peer networks, either use "super peers" (KaZaA [93]) or use centralized trackers (eDonkey 2000 [76]). Structured peer networks, mostly based on Distributed Hash Tables (DHTs) reduce no. of messages needed for lookups, to typically $O(log_N)$, but suffer from other problems like Sybil-attacks and node churn [95].

- **Performance:** Reads and writes on peer networks have very variable latency depending on the underlying design, but in most cases, the worst-case performance of lookups is unacceptable; a request can needlessly bounce through several high-latency network links before being handled. Some work on DHTs (like DSHTs [70], and Beehive [120]) try to fix this for frequently-requested data, but it doesn't help the long-tail performance and works for only certain type of workloads.

- **Reliability:** Public peer networks allow anyone to write to them. To deal with so much data, they simply delete stale data. Applications need to re-announce data every so often to keep it available. Data sources can go off-line before republishing [69]. Structured peer networks can split into one or more disjoint networks due to partitions, and re-join later on. This can lead to inconsistent state; some clients can see one value for the key, and other clients can see a different value.

- **Junk Data Writes:** Without some rate-limiting or access-control mechanism, peer networks have no way to limit the amount of data inserted. An adversary can flood the peer network with lots of garbage data and knock nodes off-line.

- **Node Eclipse Attack**. In structured peer networks, an attacker can take over the neighbors of all nodes storing a particular key/value pair and effectively censor nodes/keys from the network. Such Sybil-attacks are a general problem for structured peer networks with no good solutions available without requiring centralized gatekeepers or human input on peer connections [91].

7.2 Kadamlia-TX: A Sybil-resistant DHT Network

Like other DHT-based peer networks, the Kademlia DHT [98] is susceptible to Sybil attacks. Sybil attacks can be on (a) the DHT data storage or (b) the routing tables of the DHT. Malicious nodes can write "junk data" to the DHT and overload the storage capacity of individual nodes or can insert "junk entries" in the routing tables affecting the routing of queries. We modified Kademlia to design a new DHT-based protocol, called *Kademlia-TX*. Our work makes a contribution towards Sybil-resistant DHTs by enabling a cost to launch an attack without introducing any central trusted party, i.e., the DHT system stays decentralized while adding a significant cost of attack.

Protection Against Junk-Data Writes

We address this problem by creating a decentralized "white-list" of keys that can be written to the DHT. Our mechanism requires that a *(key, value)* pair can only be written to the DHT if $Hash(key)$ was announced earlier in a proof-of-work blockchain transaction:

$$if\ \ key == Hash(value)\ \ and\ \ Hash(value) \in T_n\ \ then\ \ insert(key, value) \tag{7.1}$$

where T_n is the set of all transactions up to block n of a blockchain. The writer needs to present a blockchain transaction ID ($txid$) that contains the $Hash(key)$ of the $(key, value)$ pair being written. The DHT node verifies that (a) the blockchain transaction has enough confirmations (i.e., cannot be dropped from the blockchain) and (b) contains the correct hash. This forces writers to pay for data writes to the DHT and associates a cost to launching SPAM attacks. The cost of a write can be a minimum blockchain transaction fee (e.g., 0.0001 or approx 5-10 cents). This also makes the DHT storage content-addressable.

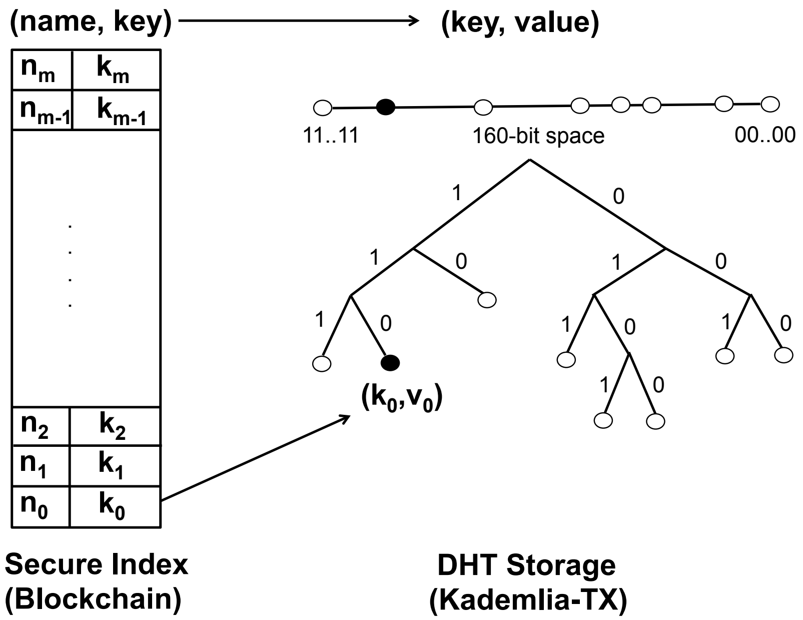


Figure 7.1: Relationship between the secure index (stored on the blockchain) and the peer network (Kademlia-TX). The peer network accepts writes only for keys in the secure index.

Protection Against Node Eclipse Attacks

For Sybil attack on routing tables, we modified Kademlia's default way of assigning random 160-bit node identifiers to include the requirement the node identifiers should be derived from valid blockchain transactions, e.g., from Bitcoin transactions that have > 6 confirmations and > 0.0001 BTC transaction fee. **In Kademlia-TX, a new node can only be included in the routing tables if the new node can sign a challenge/message with the private key that corresponds to the blockchain transaction from which the 160-bit node identifier is derived.** If the new node is unable to respond to the challenge, it is not included in the routing table. This ensures that there is a cost to launching node eclipse attacks and the cost can be made arbitrarily large simply by increasing the TX fee requirement. Figure 7.1 shows how Kademlia-TX uses a secure index, announce through the blockchain channel, to rate-limit data writes to the DHT. We can additionally require:

- Proof of ownership of a certain amount of cryptocurrency; putting a floor on the money required to be part of the network, and

- proof of ownership of certain age of cryptocurrency [10]; limiting an attacker from moving crypocurrency between different addresses to generate node IDs.

Kademlia-TX gives a solution to the Sybil-attack problem without introducing any centralized gatekeepers or human input and provides tunable parameters for decreasing/increasing the cost of attack. We deployed a variant of Kademlia-TX as a production peer network between Fall 2015 and Fall 2016.

7.3 Atlas Network

For BNS, the size of individual zone files is fairly small (<4KB) and the total space needed to store them increases linearly with the no. of domain registrations. Currently, it takes only 300MB to store all zone files of the 70,000 domains registered on BNS and 100GB space can store zone files for all 250 million ICANN domains (which is smaller than the size of the current Bitcoin blockchain). Inspired by the need to store the (small-sized) zone files of BNS, we designed a new peer network called the *Atlas Network*. The Atlas network solves a particular case of decentralized storage using peer networks–the case where:

1. The data set is small in size and
2. There is a full index of data available to the network.

All Atlas nodes maintain a 100% state replica, and they organize into an unstructured overlay network. The unstructured approach is easier to implement, has no overhead for maintaining routing structure and is resilient against targeted node attacks. When a new Atlas node boots up, it first gets the index of all data $keys$ and hashes of $values$ stored in the blockchain. After getting the index, Atlas nodes talk to their peers to fetch key/value pairs they dont have. The Atlas network implements a K-regular random graph. Each node selects K other nodes at random to be its neighbors using the Metropolis-Hastings Random Walk algorithm with delayed acceptance (MHRW-DA [90]), and regularly asks them for the set of key/value pairs they have. Peers pull missing key/value pairs in rarest-first order to maximize availability, i.e., new key/value pairs written to the network are given preference for propagation through the network. In addition to storing key/value pairs locally, peers can also write them to remote backup locations (e.g., a service like Dropbox or S3) for additional protection against data loss. When a peer receives a missing key/value pair, it pushes it to its immediate neighbors that dont have it yet.

Like Kademlia-TX, Atlas nodes already know the hashes of the zone files so that no one can upload invalid data. But unlike Kademlia-TX, data is replicated on $O(N)$ nodes instead of only on a small subset of nodes. The Atlas network makes censoring attacks expensive. Censoring the entire network requires attacking $O(N)$ nodes. By contrast, only $O(log_N)$ DHT nodes need to be taken over to censor a key/value pair for everyone. Even then, the victim node will detect the censorship unless the attacker also eclipses the victims Bitcoin node (which requires building a fraudulent blockchain fork with sufficient proof-of-work). We believe that the Atlas network is a significant step forward towards having a reliable, hard-to-censor, and decentralized peer network.

7.4 Analysis of a Production Deployment

The Blockstack production network used a variation of Kademlia-TX for hosting BNS zone files between Fall 2015 and Fall 2016. Our implementation ensured that (1) each key was the hash of its value, and (2) each key corresponded to an accepted registration transaction in our virtualchain. This prevented most classical DHT attacks–no one could overwrite a user's key/value pairs, and no one could flood the DHT with lots of junk data writes. However, we didn't implement the node eclipse attack protection of Kademlia-TX, and our production DHT deployment was still (theoretically) vulnerable to routing attacks.

In our production deployment, we didn't notice any explicit node eclipse attacks, but we did encounter partitions of the DHT overlay where some nodes hosted in Hong Kong and

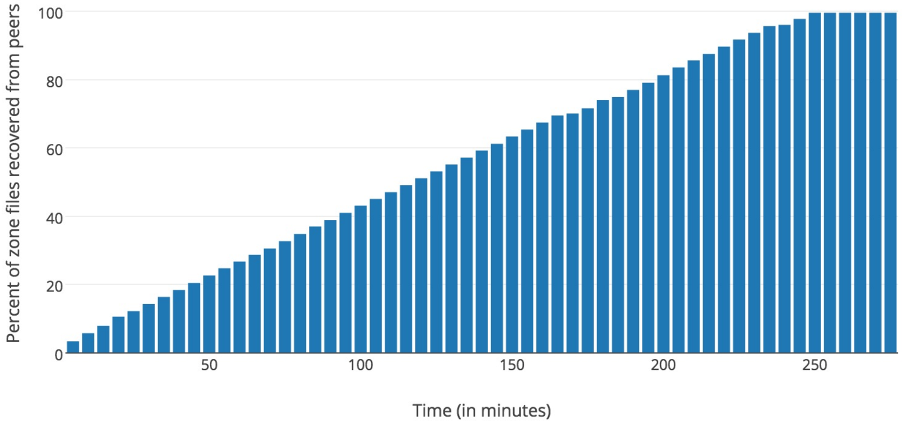


Figure 7.2: Avg. time for Atlas nodes to recover from a 100% data loss (80320 items).

Europe would end up on a different partition. Churn is a general problem with structured DHTs. Our DHT nodes were programmed not to accept data writes unless a hash of the data is present in the blockchain, i.e., someone has paid a fee to gain access to write data. The DHT-based discovery network served as an acceptable initial design, but with a growing network, the daily and hourly churn became a bigger issue. The Blockstack implementation switched to the Atlas network from the DHT-based discovery network in Fall 2016, and since November 2016, we have been distributing BNS zone files using the Atlas network.

Network Partitions

The Atlas network is more reliable than the previous Kademlia-TX-based DHT network. For our Kademlia-TX deployment, we frequently ran into network partition issues where some nodes, e.g., in Hong Kong would get disconnected from the “mainline” DHT. Between September 2015 and September 2016, there were at least **7 major incidents** where we had to work with our community to restore network partitions in our Kademlia-TX DHT deployment. Since moving entirely to the Atlas network, between November 2016 and the time of this writing (May 2017), we’ve had **0 incidents of network partitions** or any other network outage. In fact, there is no concept of a network partition on the network since Atlas is unstructured and all nodes have a full replica.

Node Recovery

Atlas nodes can recover from failures on their own. If the local index of the Atlas data becomes corrupt, the nodes can reconstruct it from the blockchain data. We ran an experiment where we would intentionally destroy the Atlas index (containing information about names and hash pairs) repeatedly and let the nodes repair the index. On current commodity hardware, it took nodes an **average of 5 hours and 45 minutes to locally regenerate the full data index**. In a second experiment, we destroyed all locally stored zone files on Atlas nodes and measured how long it took for Atlas nodes to self-heal and fetch the zone files from their peers. Figure 7.2 plots the average time it took for our nodes to re-fetch the zone files. We repeated the experiment for different peer topologies. We see that nodes can re-fetch data at the rate of roughly 1200 zone files every 5 minutes. Our network currently indexes 80320 zone files in total (57199 zone files have unique hashes), and 248 of them are not present on any Atlas node, i.e., the client application that registered the name was unable to announce the zone file to any Atlas node and the information didn't propagate through the network. Our nodes recovered 99.57% of the data in 250 minutes (Figure 7.2) and then kept looking for the remaining 248 files. **The important metric is that 100% of our nodes were able to recover from a complete data loss within hours fully**. The Atlas network is self-healing in that aspect and can recover from failures even if very few copies of data remain on the peer network.

Chapter 8

Decentralized Storage

"The greatest performance improvement of all is when a system goes from not-working to working."

– JOHN OUSTERHOUT (1954–PRESENT)

On the traditional internet, once end-users establish a secure connection to a website like Facebook.com they then log in to the service and keep all their data with the remote service. This model, along with advances in cloud computing, pushes all complexity and user data to the remote cloud and user devices exist as "dumb screens." This is a full departure from the spirit of the end-to-end design where end-user devices were meant to handle complexity and logic (Chapter 1).

Currently, with frequent use of an online service user data gets locked into "data silos", e.g. data that is understood and stored by Facebook, Yahoo!, Google and others respectively but cannot be migrated across services. This leads to a centralized data model; the data silos inevitably get hacked eventually, e.g., the recent hack of 500 million Yahoo! users [117].

Our trust-to-trust internet architecture has the potential to release users from these data silos by giving users access to decentralized storage systems that provide comparable performance to centralized cloud providers. Users can log in to apps and services by using blockchain-based decentralized identity systems, like the identity system used by our

Blockstack implementation [41] and uPort [96], and save data generated by apps/services on storage backends owned by the user (instead of the service provider). Decentralized identity systems enable users to control a unique identity recorded on the blockchain that can be recognized by any site (rather than a username and password combo that can only be recognized by the site that had you create an account). Users can log in to websites by proving ownership of their identity, i.e., by cryptographically signing a challenge from the website. A detailed discussion of blockchain-based identity systems is out of the scope of our work and readers should see [41, 96] for further details.

Encrypted & Signed Data on Storage Backends

In our architecture, our design philosophy is to reuse existing cloud providers and infrastructure in a way that end-users don't need to trust the underlying cloud providers. We treat cloud storage providers (like Dropbox, Amazon S3, and Google Drive) as "dumb drives" and store encrypted and/or signed data on them. The cloud providers, like Dropbox, have no visibility into user's data; they only see encrypted data blobs. Further, since the associated public keys or data hashes are discoverable through the blockchain channel, cloud providers cannot tamper with user data. There are two modes of using the storage layer, and they differ in how the integrity of data values is verified:

(a) Mutable Storage is the default mode of operation for the storage layer. The user's zone file contains a URI record that points to the data, and the data is constructed to include a signature from the user's private key. Writing the data involves signing and replicating the data (but not the zone file), and reading the data involves fetching the zone file and data, verifying that $hash(zonefile)$ matches the hash in the blockchain, and verifying the data's signature with the user's public key. This allows for writes to be as fast as the signature algorithm and underlying storage system allow, since updating the data does not alter the zone file and thus does not require any blockchain transactions. However, readers and writers must employ a data versioning scheme to avoid consuming stale data.

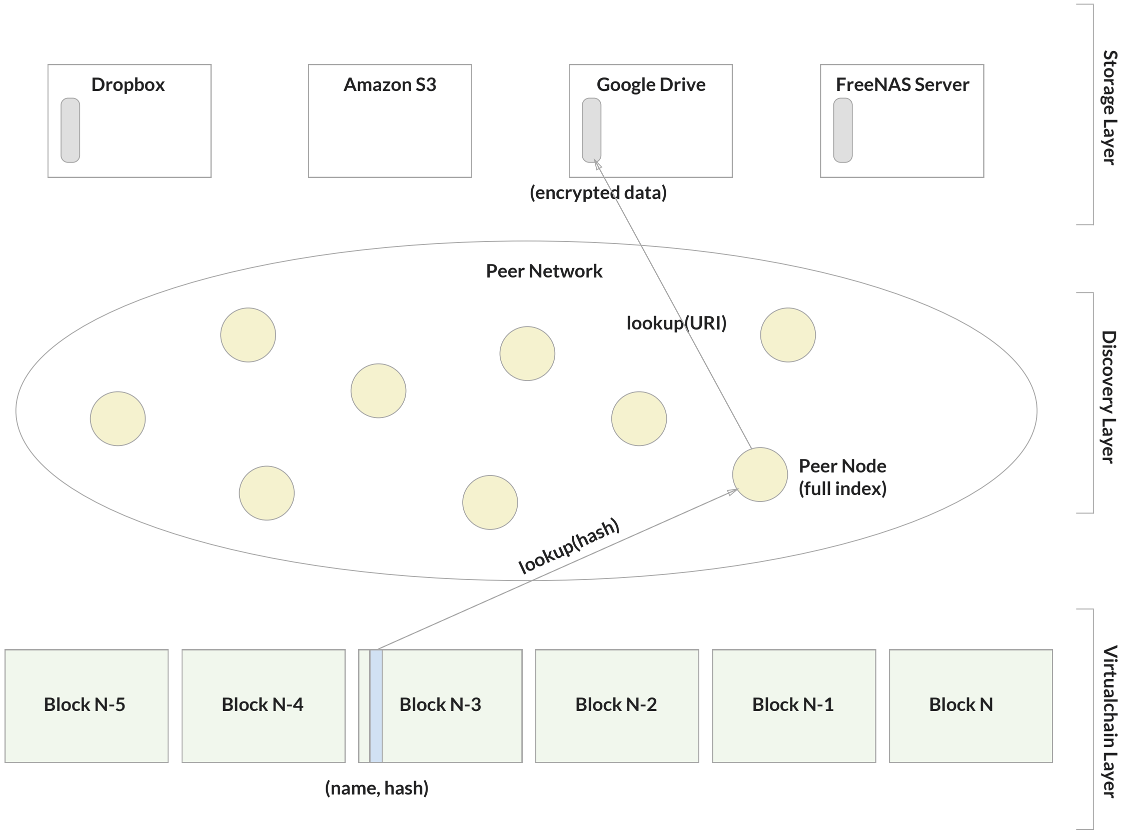


Figure 8.1: Overview of our storage system and steps for looking up data.

(b) Immutable Storage is similar to mutable storage, but additionally puts a TXT record in the zone file that contains $hash(data)$. Readers verify data integrity by fetching the data and checking that $hash(data)$ is in the zone file, in addition to verifying the data's signature and the zone file's authenticity. This mode is suitable for data values that don't change often and where it's important to verify that readers see the latest version of the data value. For immutable storage, updates to data values require a new transaction on the underlying blockchain (since the zone file must be modified to include the new hash), making data updates much slower than mutable storage.

Figure 8.1 shows an overview of our storage system. We show an example encrypted data blob with three replicated copies at Dropbox, Google Drive, and a FreeNAS Server (and not on Amazon S3). In our Blockstack implementation, we have drivers for individual cloud providers like Dropbox and S3, and integrate them as a storage backends. This hides the individual APIs for storage backends and exposes a simple PUT/GET interface to Blockstack users. Looking up data for a name, like $muneeb.id$, works as follows:

1. Lookup the $name$ in the virtualchain to get the $(name, hash)$ pair.
2. Lookup the $hash(name)$ in the peer network (Atlas in the Blockstack implementation) to get the respective zone file (all peers in the Atlas network have the full replica of all zonefiles).
3. Get the storage backend URI from the zonefile and lookup the URI to connect to the storage backend.
4. Read the data (decrypt it if needed and if you have the access rights) and verify the respective signature or hash.

8.1 Performance of Reads and Writes

The goal of our architecture is to give comparable performance to traditional cloud providers. We introduce meaningful security and fault-tolerance benefits by removing central points of control and failure and paying a small overhead on read/write performance is totally worth it as long as the overhead is not significant and not noticeable to the average users. We implemented Blockstack in 40,344 lines of Python code [42]. The current implementation uses Bitcoin as the underlying blockchain. Currently, Blockstack core developers decide which underlying blockchain(s) to support in which version of the software. Individual applications can decide to run the software version of their choice and keep their namespace on a particular blockchain if they prefer not to migrate.

We evaluated the performance of reads and writes through Blockstack to demonstrate that it reads and writes files at competitive rates with the underlying storage. Blockstack adds a negligible constant storage space overhead per file (roughly 5% larger files with compression). There is CPU overhead for encryption and compression, but since the file

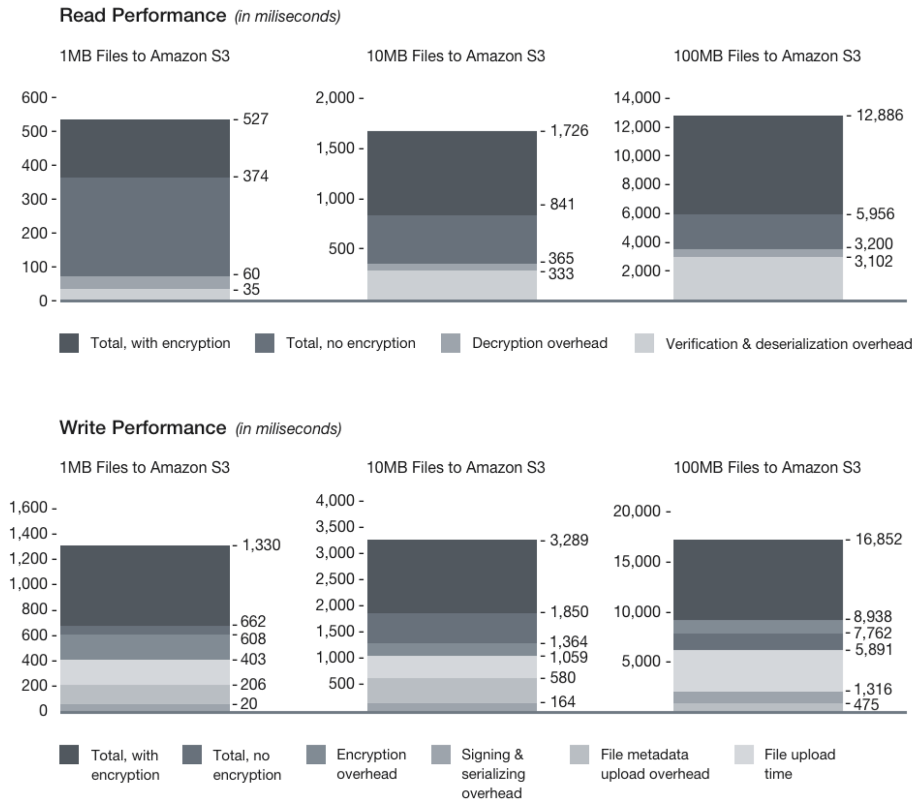


Figure 8.2: Performance overhead of Blockstack.

size difference is very small, the network performance for reads and writes is similar to directly accessing the underlying storage service.

The write performance and overheads associated with uploading 1, 10, and 100 megabyte files to Amazon S3 is shown in Figure 8.2 (each trial was performed 25 times). We see that the CPU-bound overhead is in the order of 2 seconds for large (100MB) files. Many low-hanging performance optimizations still remain in our implementation. Similarly, reading encrypted files from Blockstack with S3 as storage backend is competitive with a direct read from S3 (Figure 8.2). We omitted the file download time to emphasize the overhead in the graph. The sources of overhead, verifying the signature and decrypting the data, are CPU-bound while in practice performance will largely be network-bound for wide-area usage. On current commodity hardware, booting new Blockstack nodes can take 1-2 hours with SNV (Section 5.2), compared to 2-4 days without SNV. Further engineering improvements in our Python implementation are currently possible.

8.2 System Scalability

The storage layer of our architecture is not a scalability bottleneck. Contemporary cloud storage systems are highly scalable [15]. The Atlas network also scales well because it does not index individual user files or file-chunks but indexes pointers to user's storage backends. The storage backends deal with the bulk of data read/writes, and the Atlas network is involved only when (a) a user is changing or updating her storage backends or public key mappings, or (b) new users are registered on the system. When registering new domains/users, zone file hashes must be announced on the underlying blockchain. The underlying blockchains typically have low-bandwidth and are the bottleneck on scalability (relative to the Atlas network). We're exploring the option to pack multiple virtualchain transactions into a single blockchain transaction [48] for addressing blockchain scalability. This can enable us to register several hundreds of millions of end-users. Scaling Blockstack to billions of users in practice will likely uncover scalability issues that are not obvious right now and addressing these challenges is an area of future work. As the design of Blockstack evolves, the latest design will be available at [8].

Chapter 9

Applications

"People tend to think of the Web as a way to get information or perhaps as a place to carry out ecommerce. But really, the Web is about accessing applications."

– Marc Andreessen (Wired Magazine, 2012)

Our implementation of Blockstack, a new decentralized internet, is already attracting interest from application developers. As of May 2017, 5,571 meetup group members have organized developer events in 7 different countries [7]. Two large cloud providers, Ama-

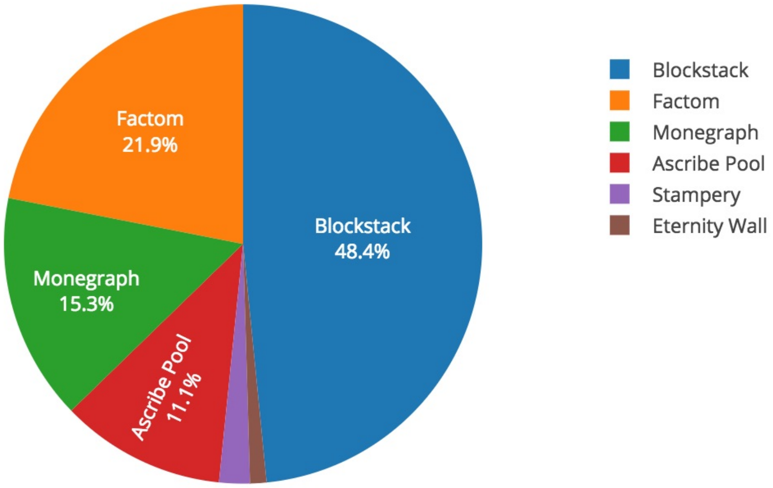


Figure 9.1: Total data-embedding transactions for non-financial use cases on the Bitcoin network (all time transactions as of 05/2017).

zon Web Services and Microsoft Azure, have integrated support for easily deploying new Blockstack nodes on their respective cloud platforms [4, 5]. In this chapter, we first give an overview of a wide range of applications/services that are using Blockstack (or are testing an integration) and then discuss one case study (OpenBazaar) in more detail.

Overview of Applications

A broad range of applications and services are already using Blockstack. Figure 9.1 shows the distribution of data embedding transactions, called *OP_RETURN* transactions, on the Bitcoin network. As of May 2017, Blockstack is the largest non-financial use case on the Bitcoin blockchain with 200,184 total transactions which account for 48.4% of all non-financial use cases and 18.6% of all *OP_RETURN* transactions (including financial use cases). This data is publicly available in the Bitcoin blockchain [20, 114]. Below is a list of some applications/services that are either using Blockstack or are testing an integration:

- **MediaChain:** Mediachain is a blockchain-based media platform. It allows parties to make statements about creative works; metadata statements are cryptographically signed by the contributor and timestamped in the Bitcoin blockchain. Medichain uses Blockstack for providing human-readable names in a decentralized way [107].

- **Syndicate:** is a wide-area file system that combines existing cloud storage providers into a single (virtual) cloud storage [81]. Syndicate is using Blockstack for distributing public keys to virtual machines without introducing a centralized trusted party.

- **Rushwallet:** is a client-side JavaScript Bitcoin wallet that uses Blockstack to convert human-readable names to Bitcoin addresses [18]. Rushwallet enables users to enter memorable names for making payments (instead of copy/pasting Bitcoin addresses).

- **Tor Project:** is considering using Blockstack for replacing onion addresses with domains while preserving user privacy [131]. There is already an implementation available that is testing the use case [9].

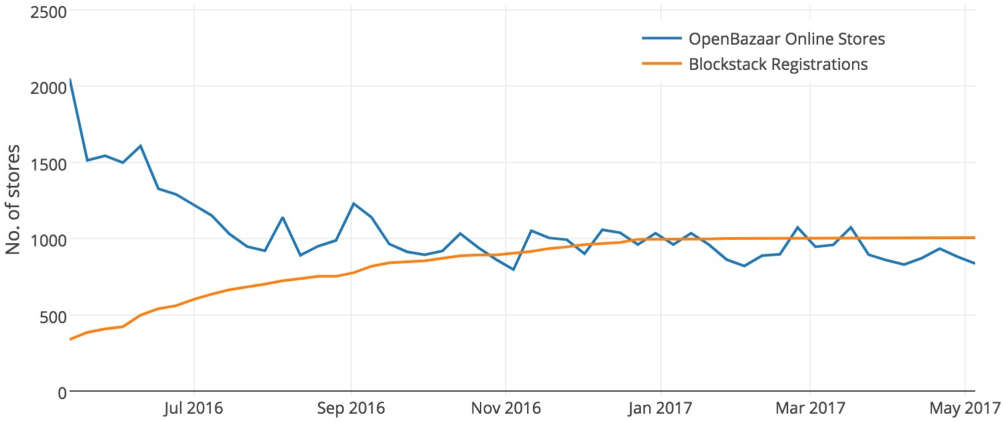


Figure 9.2: OpenBazaar network analysis (05/2016 – 05/2017).

- **Souq:** is a Blockstack-powered decentralized asset tagging and funding application. Souq uses Blockstack to create an index of projects, e.g., road repair needs of a local community and then allows people to fund initiatives [19]. It is one of the several apps that came out of a decentralized app building challenge [12].

There are several other application uses cases, e.g., identity systems built by the Decentralized Identity Foundation [13] (an initiative by Microsoft and several other companies) and IDEO [61]. A detailed discussion of all current use cases of Blockstack is out of the scope of this chapter. We will focus on one particular case study and present an analysis of how OpenBazaar, a decentralized peer-to-peer marketplace uses Blockstack.

9.1 Case Study: OpenBazaar

OpenBazaar is a decentralized peer-to-peer marketplace that uses the Bitcoin network for making payments [22]. It started in April 2014 as a hackathon project, and the production release came out in April 2016. OpenBazaar is fully decentralized; users run their own nodes and connect with each other directly over a peer-to-peer network. OpenBazaar required a decentralized naming system and couldn't introduce a centralized naming system as that would go against their primary design goal. OpenBazaar uses Bockstack as the naming system since the initial releases. Figure 9.2 plots the number of active OpenBazaar stores and the number of store name registrations on Blockstack.

We collected OpenBazaar data from an OpenBazaar monitoring service that takes a snapshot of all stores every 15 minutes [1] and collected the respective Blockstack data from the Bitcoin blockchain. Below are some observations:

- We see that there was an initial spike in usage when more than 2000 active stores were online in mid-May 2016. This dropped off in the next two months. The more dedicated users who were actively selling goods on OpenBazaar not only kept their nodes online but also registered human-readable domain names for their stores on Blockstack. Our software was not fully polished, and these users still went through all the manual steps to register names and debugged issues with our assistance.

- It's interesting to see the correlation between name registrations on Blockstack and active OpenBazaar nodes between mid-September 2016 to May 2017. Our analysis shows that the Blockstack functionality was very popular amongst OpenBazaar users. We crawled the OpenBazaar network (using data from [14]) in May 2017 to find that 81% of all online stores were reachable via their Blockstack domains and had listings.

- The total number of registered stores surpassed the total no. of active nodes in 2017. This is likely due to old registrations of stores that are no longer actively maintained by their owners, but the domains registered on Blockstack are still valid.

- We noticed a slowdown in new store registrations in 2017; this is likely because OpenBazaar is planning a major upgrade to their software and move to a new network, called OpenBaaar 2.0 [17], and store owners are testing the new network instead of the current production network.

Chapter 10

Related Work

"It will seem at first that you're working on the proverbial needle, a tiny fragment of the world, a minute crystal, beautiful but in the scheme of things, microscopic. Work with it. And the more you work with it, ... you will come to see that your work, your subject, encompasses the world. In time, you will come to see the world in your grain of sand."

– MANUEL BLUM (WHAT IS RESEARCH, 2001)

A few clean-slate internet designs have been proposed over the years that note several issues with the current internet architecture [34, 53]. Notable proposals include, Data-Oriented Network Architecture [88], Software-Defined Internet Architecture [119], Serval [110], i3 [129], Content-centric Networking [116], and others [66]. Our work differs from these architectures in two ways (a) we explicitly focus on removing trust points from the middle of the network (instead of other problems addressed by earlier works), and (b) we have a production deployment for more than 3 years used by over 70,000 users and our design incorporates the real-world lessons we learned. The trust-to-trust design principle was originally proposed by David Clark and Marjory Blumenthal [44, 51] and we extend their work by presenting an architecture and implementation that follows that principle.

We presented the Blockchain Name System (BNS) and use it in our Blockstack implementation. Binding names to values in naming systems is a well-explored problem space. UIA [68] gives a great overview of global naming systems and their importance. We encourage the reader to UIA [68] for a detailed background on naming systems. Unlike Namecoin [104] or Blockstack, UIA doesn't try to provide globally unique names. In authentication systems like OpenID [122], InCommon [77], and certificate authorities (CAs), a federation of authorities attests to bindings. BNS, however, does not require a federation.

Naming systems have been designed since the early days of the internet and distributed systems [89]. The most comprehensive naming system other than DNS that has seen actual deployment and use is Digital Object Architecture (DOA) [82] that is an object archival system where the name/handle doesn't change ever. We share that design philosophy where names can be permanent (in a namespace where names don't expire) as the underlying data is stored in blockchains forever.

Other than Namecoin, blockchains like Ethereum [46] and BitShares [23] also have support for human-readable names. Further, sidechains [31] enable implementation of naming systems as an alternate blockchain that is linked to the main Bitcoin blockchain. All these designs involve smaller, alternate blockchains and Blockstack directly uses the most secure blockchain (Bitcoin). Virtualchains [108] builds new state machine logically on top of underlying blockchains and allow the system to migrate to any underlying blockchain, e.g., a migration from Bitcoin to Zcash [35] or Litecoin [94] requires minimal engineering effort given their similarity. Non-blockchain based PKI systems, like Keybase [86] and CONIKS [101], achieve some of the same goals as Blockstack's name ownership bindings with public keys. Keybase and CONIKS focus on keeping remote servers accountable, i.e., a remote server cannot lie without getting caught. Blockstack, however, focuses on removing trusted remote services all together while giving the global state.

In networked systems, it's hard to get global state without involving central trusted parties [85], Blockstack is able to provide global state (and not just approximate global state). Our system is open ("permissionless"), whereas existing wide-area systems like OceanStore [60] and Bonafide [50] have a closed ("permissioned") set of peers that use BFT agreement to make progress for the whole system. Blockstack differs from decentralized storage systems which allow open membership but offer stronger-than-eventual data consistency (like Shark [29], Pond [123], and Scatter [73]) by focusing on decentralization while supporting a wide variety of external datastores that give strong consistency.

Storage-oriented cryptocurrency blockchains like Filecoin [67], Permacoin [103], and Storj [128] seek to replace cloud storage by distributing files as sets of transactions within a blockchain, and rewarding miners for proof-of-storage (instead of proof-of-work). Blockstack differs from these systems by decoupling hosting data from operations of the underlying blockchain, allowing developers to use storage systems appropriate for their problem domains. Blockstack currently has drivers for Dropbox, Amazon S3, and Google Drive. Our design allows other storage systems, like Syndicate [81], IPFS [80] or Tahoe-LAFS [134], to be "plugged in" as storage backends and provides a simple meta-level filesystem interface to storage systems.

In INS [26] names are based on intention e.g., "nearest printer" instead of location. In Content Centric Naming [32, 49, 72], DNS-like binding to human-readable names is dropped for flat namespaces. In self-certifying naming [33] names are not human-readable and are self-certifying, meaning that it is possible to verify the association between the name and the object it refers to with signed metadata [71]. Flat name structures have to be resolved to their appropriate bindings by some infrastructure. This usually implies the use of peer networks. We use peer networks for storing discovery data, but names in our system are unique and human-readable.

DHTs such as Chord [130], CAN [121], Pastry [125] and Kademlia [98] provides structured peer networks with theoretical bounds on lookup times. DHTs are susceptible to Sybil attacks and are not reliable under high-churn. We propose two peer networks (a) a modification to Kademlia's DHT that adds blockchain-based Sybil protection and (b) an unstructured peer network, Atlas, with full-replication of indexed data. Other improvements like caching optimizations like Beehive [120] are also possible.

Chapter 11

Conclusion

"We reject: kings, presidents and voting.

We believe in: rough consensus and running code."

– David D. Clark (IETF talk, 1992)

A lot of issues with the traditional internet, like attacks on the integrity or availability of the domain name system and public-key infrastructure, along with exploits concerning end-user data stored on centralized web services are a direct result of trusting intermediaries and remote services. We propose a new internet architecture where users don't need to trust anything on the network other than their machines. In addition to pushing all complexity and application-specific logic to the edge of the network (end-to-end principle), our architecture also explicitly removes any trust points from the middle of the network as well (trust-to-trust principle). Blockchains enable this trust-to-trust design by providing a mechanism for new nodes to join the network without trusting any remote party. Our architecture enables a much-needed security and reliability upgrade to the traditional internet.

Our experience with running a production network on a cryptocurrency blockchain Namecoin, one of the oldest blockchain other than Bitcoin, shows how a single miner consistently had more than 51% hashing power and how we ran into network reliability problems. Our data shows that out of the hundreds of blockchains currently in production, even the more popular blockchains like Namecoin cannot currently be used for large-scale production systems. Currently, the security of Bitcoin far outweighs other blockchains.

We present Blockstack, a blockchain-based system that implements several components of a trust-to-trust internet architecture, including services for naming, discovery, and storage. Blockstack introduces separate control and data planes, and by doing so, it enables the introduction of new functionality without modifying the underlying blockchain. The design of Blockstack was informed by a year of production experience from one of the largest blockchain-based production systems to date. We have made several novel improvements (like introducing the ability to do cross-chain migrations, faster bootstrapping of new nodes, and keeping data updates off the slow blockchain network) that make it easier to build decentralized services using publicly-available infrastructure. Our performance results show that Blockstack can give comparable performance to the underlying storage service and only introduces a small CPU overhead.

We present two peer-networks, Kademlia-TX and Atlas, that can be used to augment the limited storage capacity of contemporary blockchains. In our design, the global state of the index is provided by blockchains, and the blockchain entry contains a pointer to data stored outside. The peer-networks use this global state to (a) reject any junk/invalid data write attempts and (b) fetch any missing data entries from their peers. Further, we introduce mechanisms for Sybil-protection against node eclipse attacks without introducing any central gatekeepers. Kademlia-TX is a kademlia-based structured peer network that adds mechanism for Sybil protection. The Atlas network uses an unstructured approach and keeps a full replica of all data at every node to improve data reliability. Our production Blockstack network switched from Kademlia-TX to Atlas because the unstructured approach is better suited for networks with high churn.

Blockstack takes our trust-to-trust internet architecture from a theoretical concept to a production system and is already being used by decentralized applications like OpenBazaar, a decentralized marketplace. We've released Blockstack as open-source [42].

Bibliography

[1] BazaarBay: OpenBazaar Search and Analytics. http://bazaarbay.org/.

[2] Bitcoin blockchain size. https://blockchain.info/charts/blocks-size.

[3] Bitcoin hashrate. https://blockchain.info/charts/hash-rate.

[4] Blockstack Core v14 on the Amazon Web Services Marketplace. https://aws.amazon.com/marketplace/pp/B06XGBTRD7.

[5] Blockstack Core v14 on the Microsoft Azure Marketplace. https://azuremarketplace.microsoft.com/en-us/marketplace/apps/blockstack.blockstack-core-v14.

[6] Blockstack ID format, version 2. https://blockstack.org/docs/blockstack-profiles.

[7] Blockstack meetup groups. Retrieved from https://www.meetup.com/topics/blockstack/ in May 2017.

[8] Blockstack whitepaper. http://blockstack.org/papers.

[9] Blockstack/tor integration. https://github.com/jcnelson/blockstack-tor.

[10] Coin age and days destroyed. https://en.bitcoin.it/wiki/Bitcoin_Days_Destroyed.

[11] Crypto-currencies stats – active nodes. https://bitinfocharts.com.

[12] Decentralize the web challenge. https://forum.blockstack.org/t/recap-of-the-decentralize-the-web-challenge/746.

[13] Decentralized identity foundation. https://github.com/decentralized-identity.

[14] DUO Search: Discover OpenBazaar. http://duosear.ch/.

[15] Google Cloud Storage SLA. Retrieved from https://cloud.google.com/storage/sla in May 2017.

[16] Let's encrypt. https://letsencrypt.org.

[17] Milestone 1 developer release for openbazaar 2.0. https://blog.openbazaar.org/milestone-1-developer-release-for-openbazaar-2-0.

[18] Rush wallet. https://rushwallet.com/.

[19] Souq. http://cryptocracy.io/.

[20] Usage of blockchain protocols. https://blockchainprotocols.org/bitcoin.

[21] Verisign. https://www.verisign.com.

[22] What is openbazaar. https://blog.openbazaar.org/what-is-openbazaar.

[23] Bitshares namespaces, 2016. http://docs.bitshares.eu/namespaces/index.html.

[24] Cryptocurrency market cap, May 2017. http://www.coincap.io.

[25] Adam Back. Hashcash - A Denial of Service Counter-Measure. Tech report, 2002. http://www.hashcash.org/papers/hashcash.pdf.

[26] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. *SIGOPS Oper. Syst. Rev.*, 34(2):22–, April 2000.

[27] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael Freedman. Blockstack: A global naming and storage system secured by blockchains. In *Proc. USENIX Annual Technical Conference (ATC '16)*, June 2016.

[28] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Bootstrapping trust in distributed systems with blockchains. *USENIX ;login:*, 41(3):52–58, 2016.

[29] Siddhartha Annapureddy, Michael J. Freedman, and Daved Mazieres. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd NSDI*, Boston, MA, 2005.

[30] B Pourebrahimi B, K Bertels, and S Vassiliadis. A survey of peer-to-peer networks. In *16th workshop on circuits, systems, and signal processing (proRISC)*, 2005.

[31] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timon, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. White paper, Blockstream, 2014. https://blockstream.com/sidechains.pdf.

[32] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the internet. In *Proc. of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 343–352, 2004.

[33] Hari Balakrishnan, Scott Shenker, and Michael Walfish. Semantic-Free Referencing in Linked Distributed Systems. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 197–206. Springer Berlin / Heidelberg, 2003.

[34] Steven M. Bellovin, David D. Clark, Adrian Perrig, and Dawn Song. A Clean-Slate Design for the Next-Generation Secure Internet. Technical report, July 2005.

[35] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.

[36] Bitcoin Improvement Proposal 50. `https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki`.

[37] Bitcoin Improvement Proposal 66. `https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki`.

[38] Bitcoin Core Developers. Bitcoin core version 0.10.0 release notes: Consensus library, February 2015. `https://bitcoin.org/en/release/v0.10.0`.

[39] List of Bitcoin CVEs. `https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures`.

[40] Bitcoin.org: Bitcoin developer guide, 2015. `http://bitcoin.org/en/developer-guide`.

[41] What is a blockstack id? `https://blockstack.org/docs/blockchain-identity`.

[42] Blockstack source code release v0.14, 2017. `http://github.com/blockstack/blockstack-core`.

[43] Blockstack website, 2017. `http://blockstack.org`.

[44] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[45] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 104–121, 2015.

[46] Vitalik Buterin. A next-generation smart contract and decentralized application platform. Technical report, 2017. https://github.com/ethereum/wiki/wiki/White-Paper.

[47] Bengt Carlsson and Rune Gustavsson. *The Rise and Fall of Napster - An Evolutionary Approach*, pages 347–354. Springer Berlin Heidelberg, 2001.

[48] Chainpoint white paper. https://tierion.com/chainpoint.

[49] David R. Cheriton and Mark Gritter. Triad: A new next-generation internet architecture, 2000. http://gregorio.stanford.edu/triad.

[50] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Tiered fault tolerance for long-term integrity. In *FAST*, pages 267–282, 2009.

[51] David D. Clark and Marjory S. Blumenthal. The end-to-end argument and application design: the role of trust. In *Proceedings of the Conference on Communication, Information and Internet Policy (TPRC)*, September 2007.

[52] David D. Clark and Marjory S. Blumenthal. The end-to-end argument and application design: The role of trust. *Federal Comm. Law Journal*, 63(2), 2011.

[53] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: Defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475, June 2005.

[54] Coindesk. State of blockchain Q1 2016: Blockchain funding overtakes bitcoin. http://www.coindesk.com/state-of-blockchain-q1-2016/.

[55] Coindesk. Bitcoin network stress test could occur next week, Sep 2015. http://coinde.sk/1Ku5oWc.

[56] Coindesk. State of bitcoin 2015: Ecosystem grows despite price decline, 2015. http://coinde.sk/1tJDDvv.

[57] Counterparty protocol specifications. http://counterparty.io/docs/protocol_specification/.

[58] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake, 2016. https://eprint.iacr.org/2016/919.pdf.

[59] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.

[60] Patrick Eaton, Hakim Weatherspoon, and John Kubiatowicz. Efficiently binding data to owners in distributed content-addressable storage systems. In *IEEE 3rd Security in Storage Workshop (SISW'05)*, pages 40–51. IEEE, 2005.

[61] Dan Elitzer. Digital identity today is brokenŁŁbut we can fix it. https://medium.com/ideo-colab/digital-identity-is-broken-but-we-can-fix-it-b15398180bdf.

[62] Ethereum name service. https://ens.domains.

[63] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[64] Ariel J Feldman, Aaron Blankstein, Michael J Freedman, and Edward W Felten. Social networking with Frientegrity: Privacy and integrity with an untrusted provider. In *Proc. 21st USENIX Security Symposium*, August 2012.

[65] Ariel J Feldman, William P Zeller, Michael J Freedman, and Edward W Felten. SPORC: Group collaboration using untrusted cloud resources. In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, pages 337–350, October 2010.

[66] Anja Feldmann. Internet clean-slate design: What and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.

[67] Filecoin: A Cryptocurrency Operated File Network. Tech report, 2014. http://filecoin.io/filecoin.pdf.

[68] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. Persistent personal names for globally connected mobile devices. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*, Seattle, Washington, November 2006.

[69] Michael J. Freedman. Experiences with coralcdn: A five-year operational view. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, 2010.

[70] Michael J. Freedman, E. Freudenthal, and David Mazieres. Democratizing content publication with coral. In *Proc. 1st NSDI*, San Francisco, CA, 2004.

[71] Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pages 1–6, 2011.

[72] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[73] Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas Anderson. Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 15–28. ACM, 2011.

[74] Michael Gronager. Namecoin was stillborn, i had to switch off life-support, Oct 2013. https://bitcointalk.org/index.php?topic=310954.

[75] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume 02*, ITCC '05, pages 205–213, 2005.

[76] Oliver Heckmann, Axel Bock, Andreas Mauthe, and Ralf Steinmetz. The eDonkey file-sharing network. In Peter Dadam and Manfred Reichert, editors, *GI Jahrestagung (2)*, volume 51 of *LNI*, pages 224–228. GI.

[77] Incommon federation. https://www.incommon.org/federation/.

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

[79] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *Int. J. Inf. Sec.*, 1(1):36–63, 2001.

[80] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. Draft, ipfs.io, 2015. https://github.com/ipfs/papers.

[81] Jude Nelson and Larry Peterson. Syndicate: Virtual cloud storage through provider composition. In *ACM BigSystem*, 2014.

[82] Robert Kahn and Robert Wilensky. A framework for distributed digital object services. *Int. J. Digit. Libr.*, 6(2):115–123, April 2006.

[83] Harry Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of Namecoin and lessons for decentralized namespace design. *WEIS '15: Proceedings of the 14th Workshop on the Economics of Information Security*, June 2015.

[84] Dan Kaminsky. Spelunking the triangle: Exploring aaron swartzs take on zookos triangle, Jan 2011. http://dankaminsky.com/2011/01/13/spelunk-tri/.

[85] S. Keshav. Efficient and decentralized computation of approximate global state. *SIGCOMM Comput. Commun. Rev.*, 36(1):69–74, January 2006.

[86] Keybase. https://keybase.io.

[87] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. http://www.peercoin.net.

[88] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 Conference on Applications,*

Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, pages 181–192, 2007.

[89] Butler W Lampson. Designing a global name service. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 1–10, 1986.

[90] Chul-Ho Lee, Xin Xu, and Do Young Eun. Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 319–330, 2012.

[91] Chris Lesniewski-Laas and M. Frans Kaashoek. Whānau: A Sybil-proof distributed hash table. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, San Jose, CA, April 2010.

[92] Jinyuan Li and David Maziéres. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *Proc. 4th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '07)*, February, 2007.

[93] J. Liang, R. Kumar, and K. Ross. The KaZaA Overlay: A Measurement Study. September 2004.

[94] Litecoin. `https://litecoin.org`.

[95] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Commun. Surveys Tuts.*, 7(2):72–93, April 2005.

[96] Christian Lundkvist, Rouven Heck, Joel Torstensson, Zac Mitton, and Michael Sena. Uport: A platform for self-sovereign identity. `https://whitepaper.uport.me`.

[97] Jim Manning. Ethereum name service launch delayed by bugs, March 2017. `https://www.ethnews.com/ethereum-name-service-launch-delayed-by-bugs`.

[98] 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.

[99] David Mazieres, Michael Kaminsky, M. Frans Kasshoek, and Emmitt Witchel. Separating key management from file system security. In *Proc. 17th SOSP*, Kiawah Island Resort, SC, 1999.

[100] Julia McGovern. Official statement on the last weeks ddos-attack against ghash.io mining pool, 2015. `http://bit.ly/1nu49vR`.

[101] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015.

[102] Merge mining specification.
`https://en.bitcoin.it/wiki/Merged_mining_specification`.

[103] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 475–490. IEEE, 2014.

[104] Namecoin. `https://namecoin.info`.

[105] Namecoin core, required software for miners, 2015.
`https://github.com/namecoin/namecoin-core`.

[106] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, Princeton, NJ, USA, 2016.

[107] Denis Nazarov. Blockstack + mediachain. `https://blog.mediachain.io/blockstack-mediachain-6a505e2c4ef1`.

[108] Jude Nelson, Muneeb Ali, Ryan Shea, and Michael J Freedman. Extending existing blockchains with virtualchain. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, Chicago, IL, June 2016.

[109] Lily Newman. What we know about fridays massive east coast internet outage, October 2016. `https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/`.

[110] Erik Nordström, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. Serval: An end-host stack for service-centric networking. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 85–98, San Jose, CA, 2012.

[111] Omni protocol specification (formerly mastercoin).
`https://github.com/OmniLayer/spec`.

[112] Onename. `https://onename.com`.

[113] Open assets protocol. `https://github.com/OpenAssets`.

[114] Statistics of usage for bitcoin OP_RETURN. Retrieved from `http://opreturn.org` in May 2017.

[115] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. Bootstrapping trust in commodity computers. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 414–429, 2010.

[116] Diego Perino and Matteo Varvello. A reality check for content centric networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ICN '11, pages 44–49, 2011.

[117] Nicole Perlroth. Yahoo says hackers stole data on 500 million users in 2014, September 2016. `http://nyti.ms/2oAqn0G`.

[118] Nathaniel Popper. A bitcoin believer's crisis of faith, January 2016. `http://nyti.ms/2pXr3OR`.

[119] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, 2012.

[120] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 8–8, 2004.

[121] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172, 2001.

[122] David Recordon and Drummond Reed. Openid 2.0: A platform for user-centric identity management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, pages 11–16, 2006.

[123] Sean C Rhea, Patrick R Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y Zhao, and John Kubiatowicz. Pond: The oceanstore prototype. In *FAST*, volume 3, pages 1–14, 2003.

[124] Seth Rosenblatt. Fake turkish site certs create threat of bogus google sites, January 2013. `http://cnet.co/2oArU6O`.

[125] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, 2001.

[126] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984.

[127] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Tech report, 2009. `https://bitcoin.org/bitcoin.pdf`.

[128] Shawn Wilkinson and Tome Boshevski and Josh Brandof and Vitalik Buterin. Storj: A peer-to-peer cloud storage network. Tech report, storj.io, 2014. `http://storj.io/storj.pdf`.

[129] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 73–86, 2002.

[130] 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, 2001.

[131] Kyle Torpey. Tor may use bitcoin to enable user-friendly onion addresses, April 2017. `https://bravenewcoin.com/news/tor-may-use-bitcoin-to-enable-user-friendly-onion-addresses/`.

[132] Virtualchain source code release v0.14.1, May 2017. `http://github.com/blockstack/virtualchain`.

[133] Why Blockstack is migrating to the Bitcoin blockchain. `https://blockstack.org/blog/why-blockstack-is-migrating-to-the-bitcoin-blockchain`.

[134] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: The least-authority filesystem. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 21–26, 2008.

[135] Aaron Van Wirdum. A closer look at bip100: The block size proposal bitcoin miners are rallying behind, August 2015. `http://bit.ly/1VbyoXP`.

[136] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.