Blockstack: A New Decentralized Internet

`http://blockstack.org`

May 12, 2017

Abstract

The traditional internet has many central points of failure and trust, like (a) the Domain Name System (DNS) servers, (b) public-key infrastructure, and (c) end-user data stored on centralized data stores. We present the design and implementation of a new internet, called Blockstack, where users don't need to trust remote servers. We remove any trust points from the middle of the network and use blockchains to secure critical data bindings. Blockstack implements services for identity, discovery, and storage and can survive failures of underlying blockchains. The design of Blockstack is informed by three years of experience from a large blockchain-based production system. Blockstack gives comparable performance to traditional internet services and enables a much-needed security and reliability upgrade to the traditional internet.

1 Introduction

The internet was designed more than 40 years ago and is showing signs of age. Critical internet services can be taken offline by attacks like the DDoS attack on DNS servers [1]. 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 [2].

Over the last decade, we've seen a shift from desktop apps (that run locally) to cloud-based apps that store user data on remote servers. These centralized services are a prime target for hackers and frequently get hacked. In 2016, Yahoo! admitted to losing information for 500 million people [3]. Security problems with the core internet infrastructure and the centralized data models of web services built on top have exposed flaws in the internet's original design.

There are many fundamental technical challenges with creating a fully decentralized replacement for core internet components like DNS, public-key infrastructure, and storage backends. New users/nodes need to establish trust on the network and discover the relevant data without relying on any remote servers. The decentralized solutions need to give comparable performance to the traditional internet and scale accordingly as well. Our implementation of a new internet, called Blockstack, has three components.

1. A blockchain, implemented using *virtualchains* [4], is used to bind digital property, like domain names, to public keys. Blockstack's blockchain solves the problem of bootstrapping trust in a decentralized way i.e., a new node on the network can independently verify all data bindings.
2. A peer network, called *Atlas*, gives a global index for discovery information and
3. A decentralized storage system, called *Gaia*, provides high-performance storage backends without introducing central trusted parties.

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 using Blockstack [5]. We've released Blockstack as open-source [6].

2 System Architecture

Blockstack has the following design goals:

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 Section 3) and decentralized storage systems like BitTorrent, etc. don't offer performance/bandwidth comparable to centralized services [7]. Blockstack presents a solution to these problems.

Design Decision #1: Survive Failures of Underlying Blockchains
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 important as it allows for the larger system to survive, even when the underlying blockchain is compromised. 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 #2: Keep Complexity and Logic Outside of Blockchains
Many blockchains, like Namecoin [8] or Ethereum [9], 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). We believe that not using blockchains for data storage is necessary for scalability and keeping complex logic outside of blockchains is important for both security and scalability. Nodes on the network should not be required to compute complex untrusted programs just to stay synced with the network. Further, it's hard to introduce new features to blockchains after they've been deployed and gained real-world usage. We introduce the concept of *virtualchains* (Section 4) that can build arbitrary state machines on top of blockchains without requiring any modifications to the underlying blockchains. The abstraction of total ordering of operations, on top of the underlying blockchains. serves as the "narrow waist" of our architecture and keeps complexity outside of blockchains.

Design Decision #3: Scalable Index for Global Data
Any decentralized network requires an index to the data stored by it. Going back to the early days of peer networks, Napster introduced a centralized index with decentralized file transfer in 1999. BitTorrent started with centralized trackers (indexes) as well and later introduced DHT-based decentralized indexes. DHT-based peer networks are susceptible to Sybil-attacks and have historically been unreliable and hard to scale, especially under a lot of churn. We experienced these problems first-hand as our initial peer network for Blockstack was based on the Kademlia DHT. We introduced a new unstructured peer network, called the Atlas network, that solves a particular case of decentralized storage using peer networks–the case where (a) the data set is small in size and, (b) there is a global list of all indexed items available to the network. In Atlas, nodes maintain a 100% state replica. The unstructured approach is easier to implement, has no overhead for maintaining routing structure and is resilient against targeted node attacks (every node has a full copy of data).

2.1 Blockstack Layers

Blockstack's architecture has three layers as shown in Figure 1), with one layer (the blockchain layer) in the control plane and two layers (the peer network and data-storage) 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; clients can independently verify from the control plane if they received the correct data or not.

Layer 1: Blockchain
In our architecture the blockchain occupies the lowest layer, and serves two purposes: it provides the storage medium for operations and it provides consensus on the order in which the operations were written. Virtualchain encodes operations in transactions on

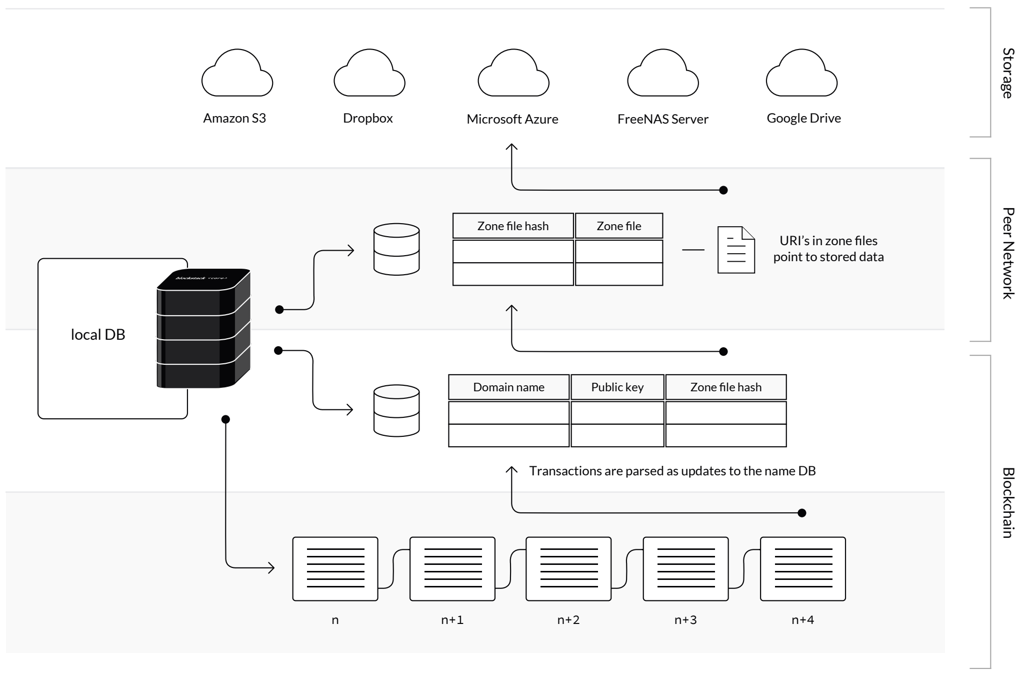

Figure 1: *Overview of the Blockstack architecture.*

the underlying blockchain. The blockchain provides an abstraction of totally-ordered operations to virtualchain 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.

The blockchain layer also includes 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 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 which defines a single state machine implementing operations for a global naming system. Operations accepted by rules defined in our 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: Peer Network

Blockstack uses a peer network for discovery. The peer network 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 network by Blockstack. **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 (Section 5), 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 GBs).

Layer 3: Storage
The top-most layer (layer-3) 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, Blockstack 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.

Blockstack implements a decentralized naming system, called the *Blockchain Name System* (BNS) by defining operations in a new virtualchain and storing discovery data in a peer network called the *Atlas Network* (Section 5). 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 sections.

3 BNS: Blockchain Name System

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.

In Blockstack, 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 replace DNS [10, 11, 12]. 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.

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 [13], never all three at the same time. This limitation is called **Zooko's Triangle** [14]. 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 [14].

Namecoin [8] was the first system to build a decentralized naming system using a blockchain. Our experience of running a production network on Namecoin revealed certain security and reliability issues that highlight the need for using the largest most-secure blockchain network [15].

3.1 BNS operations

We present the design of a new blockchain-based naming system, called the Blockchain Name System (BNS). Our implementation of BNS has been running in production for more than 3 years. In our new internet, 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 [16]). 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

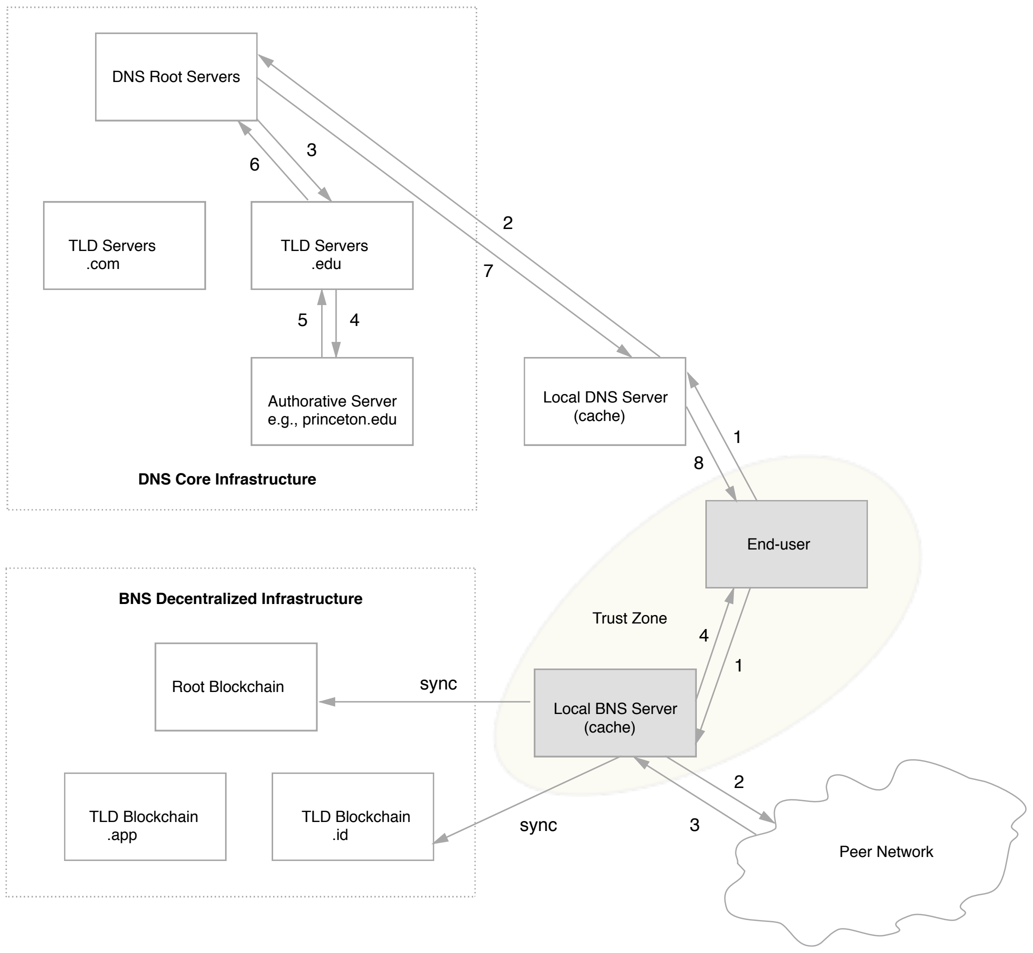

Figure 2: *A recursive DNS query (top) and an iterative BNS query.*

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 2, in BNS the information for top-level domains (namespaces) is registered on a *root blockchain*. Entries 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

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

Figure 3: *An example zone file for BNS.*

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 2 (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 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 hash). Figure 3 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 alphabetics 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 whitepaper, and the reader is encouraged to see [13] for more details on pricing functions and name squatting problems in decentralized naming systems.

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 Section 4 and Section 5 respectively. Like names, namespaces also have a *pricing function* [6]. 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.

3.2 Public Keys in BNS

On the traditional internet DNS domains can 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 [2].

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, itd be impractical for an attacker to alter a certificate after its issued or present incorrect information to only a subset of users. Also, in blockchain-based PKI there are no central CAs that can be compromised to attack the system.

BNS already provides public key associations with domain names and all domains, by default, get certificates. While efforts like Lets Encrypt [17] 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 binds public keys by default, then all websites have security certificates and security is on by default. In BNS, domain names can serve as memorable identifiers for public keys. The names themselves 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 have access to a single global state, so any key revocations or state changes to public key mappings cannot be hidden from any user.

4 Virtualchain

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 [16]. 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 [15]. 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 used virtualchains to migrate a production network from Namecoin to Bitcoin earlier [18]. The migration showed that virtualchains could be used to cope with failures with the underlying blockchain.

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 [19], Litecoin [20], or Ethereum [9]. 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. 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.

4.1 Design of Virtualchains

Virtualchain is a virtual blockchain (a logical layer) for building multiple state machines 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 [21]. Application nodes replay their logs to achieve *application-level* consensus at each block b, such that 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* for that block will be the same. Consensus hashes enable nodes to independently audit and efficiently query their histories, as well as detect forks and then migrate state between blockchains.

Figure 4 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. Blockstack's virtualchain implements a BNS state machine. Our virtualchain currently uses Bitcoin as the underlying blockchain. The new opcodes are announced in Bitcoin transactions in a field designated for additional data,

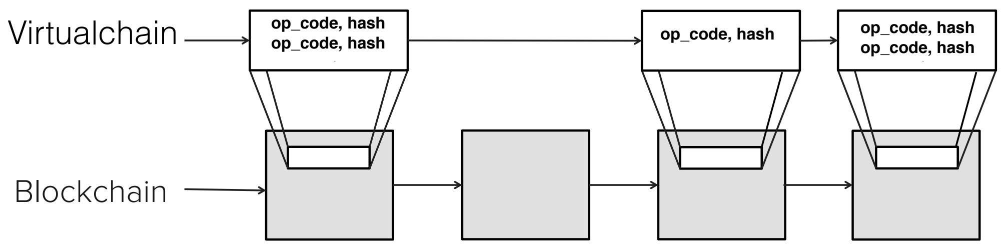

Figure 4: Virtualchain operations on top of an underlying blockchain.

called *OP_RETURN*. This is one of the largest use cases of *OP_RETURN* transactions on the Bitcoin blockchain today [22].

Consistency Model: Most public blockchains use a variant of *Nakamoto consensus* [16], which allows concurrent leaders. Appending conflicting blocks creates *blockchain forks*, which peers resolve using a proof-of-work [23] 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 [16]. 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.

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 transactions 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 5 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

$$V_n = Merkle(tx \in b_n) \tag{1}$$

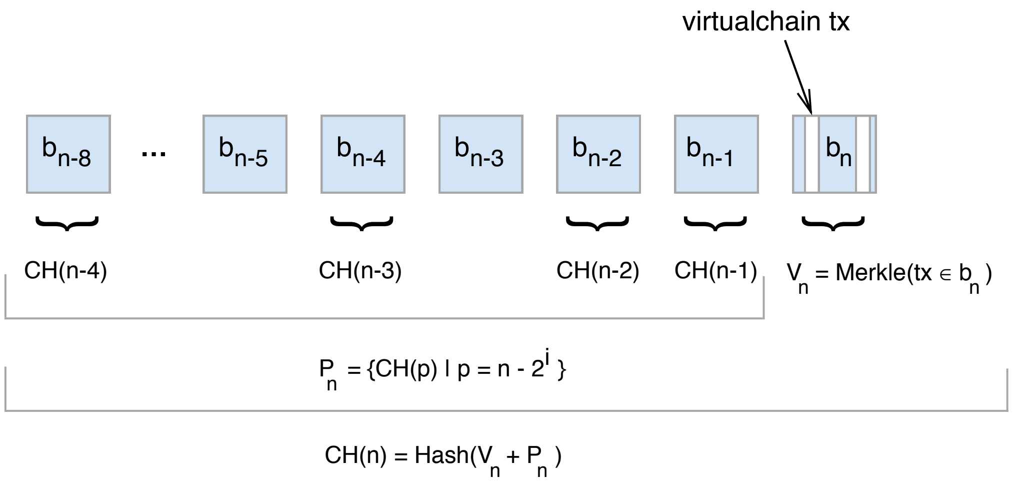

Figure 5: Consensus Hash, $CH(n)$, construction from virtualchain transactions.

$$CH(n) = Hash(V_n + P_n) \tag{2}$$

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.

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, 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 [24].

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-

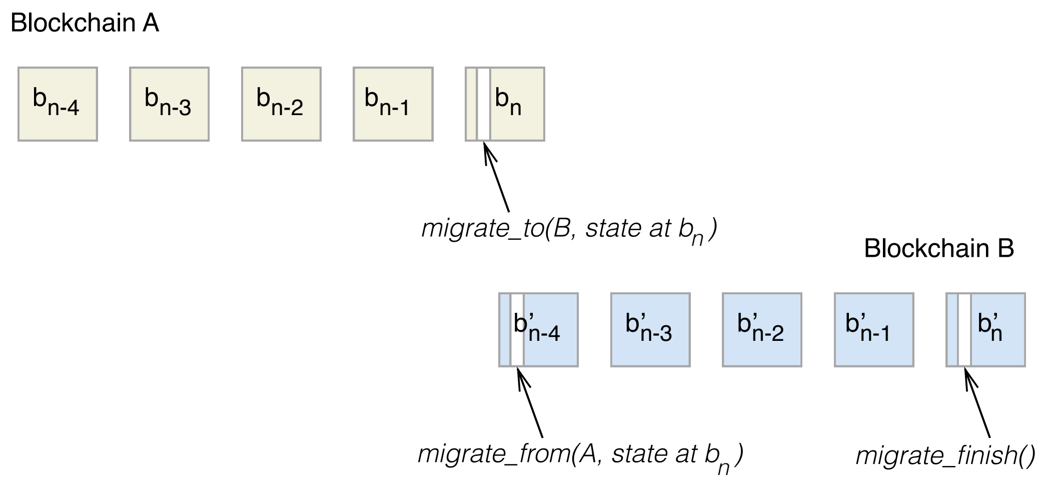

Figure 6: A framework for migrating from blockchain A to blockchain B.

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 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 [25] [26] [27]. This means that when they happen, end-users or app developers can determine which transactions were affected, and re-send state-transitions.

4.2 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 after that block), and then executing a **two-step commit** to bind the existing state to the new blockchain. Figure 6 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 is released as open source [28] and developers can build other types of state machines using it.

5 Atlas Network

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 [29]. In our architecture, only pointers to data values are kept in the blockchain; peer-networks are used as additional storage. In the Blockstack implementation, the peer-networks store zone files for BNS (these zone files are identical to DNS zone files). Using peer networks significantly increases the storage capacity but comes with other challenges: traditional peer-networks are susceptible to Sybil attacks [30] and are not a reliable source of data, especially under high churn.

In 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 [31]. 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 (Section 6).

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 [32] 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 [33, 7] 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 [33]. Practical unstructured peer networks, either use "super peers" (KaZaA [34]) or use centralized trackers (eDonkey 2000 [35]). 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 problems like Sybil-attacks and node churn [36].
- **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 [37], and Beehive [38]) 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 [39]. 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 [40].

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 [41]), 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.

Atlas nodes already know the hashes of the zone files so that no one can upload invalid data. Data is replicated on $O(N)$ nodes instead of only on a small subset of nodes (typical of DHT-based networks). 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.

In our production deployment, during September 2015 and Nov 2016, we used a Kademlia-based DHT network. 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 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 DHT network. For our DHT 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 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. Nodes can also re-fetch all zone files in case of a data loss. We ran an experiment where we intentionally destroyed zone files on Atlas nodes and **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.

6 Gaia: Decentralized Storage

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 a decentralized internet where end-user devices were meant to handle complexity and logic.

Blockstack has the potential to release users from these data silos by giving users access to a decentralized storage system, called *Gaia*, that provides comparable performance to centralized cloud providers. Users can log in to apps and services by using blockchain-based decentralized identity [42] and save data generated by apps/services on storage backends owned by the user (instead of the service provider). Gaia's 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.

In Gaia, 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 7 shows an overview of Gaia. 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

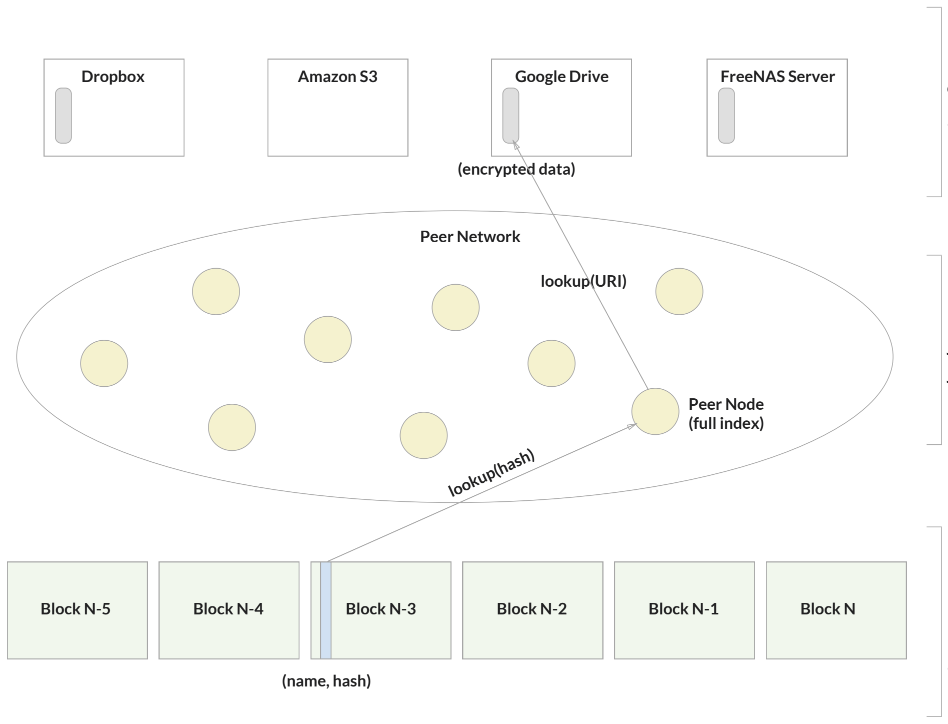

Figure 7: Overview of Gaia and steps for looking up data.

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 Atlas network 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.

Performance: 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 evaluated the performance of reads and writes of Gaia to demonstrate that it reads and writes files at competitive rates with the underlying storage. Gaia 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 size difference is very small, the network performance for reads and writes is similar to directly accessing the underlying storage service.

We measured the write performance and overheads associated with uploading 1, 10, and 100 megabyte files to Amazon S3. 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. 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.

System Scalability: The storage layer of our architecture is not a scalability bottleneck. Contemporary cloud storage systems are highly scalable [32]. 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 [43] 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.

7 Conclusion

We present Blockstack, a new decentralized internet secured by blockchains. Blockstack provides a full stack to developers for building decentralized applications including services for identity, discovery, and storage. Blockstack can introduce new functionality without modifying the underlying blockchains and can survive the failure of underlying blockchains. The design of Blockstack is informed by 3 years of production experience from one of the largest blockchain-based production systems to date. Our performance results show that Blockstack can give comparable performance to cloud services on the traditional internet and only introduces a small CPU overhead. We've released Blockstack as open-source [6].

References

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

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

[3] N. Perlroth, "Yahoo says hackers stole data on 500 million users in 2014," Sept. 2016. `http://nyti.ms/2oAqn0G`.

[4] J. Nelson, M. Ali, R. Shea, and M. J. Freedman, "Extending existing blockchains with virtualchain," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, (Chicago, IL), June 2016.

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

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

[7] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. 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, pp. 205–213, 2005.

[8] "Namecoin." `https://namecoin.info`.

[9] V. Buterin, "A next-generation smart contract and decentralized application platform," tech. rep., 2017. `https://github.com/ethereum/wiki/wiki/White-Paper`.

[10] H. Balakrishnan, S. Shenker, and M. Walfish, "Semantic-Free Referencing in Linked Distributed Systems," in *Peer-to-Peer Systems II*, vol. 2735 of *Lecture Notes in Computer Science*, pp. 197–206, Springer Berlin / Heidelberg, 2003.

[11] Juan Benet, "IPFS - Content Addressed, Versioned, P2P File System," draft, ipfs.io, 2015. `https://github.com/ipfs/papers`.

[12] D. Mazieres, M. Kaminsky, M. F. Kasshoek, and E. Witchel, "Separating key management from file system security," in *Proc. 17th SOSP*, (Kiawah Island Resort, SC), 1999.

[13] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. 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.

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

[15] M. Ali, J. Nelson, R. Shea, and M. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proc. USENIX Annual Technical Conference (ATC '16)*, June 2016.

[16] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. 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*, pp. 104–121, 2015.

[17] "Let's encrypt." `https://letsencrypt.org`.

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

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

[20] "Litecoin." `https://litecoin.org`.

[21] J. Li and D. 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.

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

[23] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*, pp. 258–272, Springer, 1999.

[24] "Simplified name verification protocol." `http://blockstack.org/docs/light-clients`.

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

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

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

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

[29] "Bitcoin blockchain size." `https://blockchain.info/charts/blocks-size`.

[30] J. R. Douceur, "The sybil attack," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, (London, UK), pp. 251–260, Springer-Verlag, 2002.

[31] B. Carlsson and R. Gustavsson, *The Rise and Fall of Napster - An Evolutionary Approach*, pp. 347–354. Springer Berlin Heidelberg, 2001.

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

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

[34] J. Liang, R. Kumar, and K. Ross, "The KaZaA Overlay: A Measurement Study," Sept. 2004.

[35] O. Heckmann, A. Bock, A. Mauthe, and R. Steinmetz, "The eDonkey file-sharing network.," in *GI Jahrestagung (2)* (P. Dadam and M. Reichert, eds.), vol. 51 of *LNI*, pp. 224–228, GI.

[36] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Commun. Surveys Tuts.*, vol. 7, pp. 72–93, Apr. 2005.

[37] M. J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing content publication with coral," in *Proc. 1st NSDI*, (San Francisco, CA), 2004.

[38] V. Ramasubramanian and E. G. 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, pp. 8–8, 2004.

[39] M. 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.

[40] C. Lesniewski-Laas and M. F. 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), Apr. 2010.

[41] C.-H. Lee, X. Xu, and D. Y. 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, pp. 319–330, 2012.

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

[43] "Chainpoint white paper." `https://tierion.com/chainpoint`.