MS Project Report

Danny Allen

Introduction

The Domain Name System (DNS) and Certificate Authorities (CAs) play vital roles in the secure operation of the Internet. The DNS maps human-memorable names to IP addresses, allowing users to find services of interest. Once a user connects to an address, CAs (or rather the certificates they issue) enable them to verify the identity of the entity on the other end of the connection. Unfortunately, the architectures of both the DNS and CAs rely on centralized elements making them potential targets for actors aiming to undermine Internet infrastructure. In the case of the DNS, this centralization takes the form of the root zone; CAs, while perhaps less so than the DNS, are centralized in that their respective chains of trust resolve to relatively few cryptographic keys when compared to the scale of the Internet at large.

Handshake, a blockchain-based protocol for updating and serving a decentralized domain name registry across a peer-to-peer network, was developed to mitigate the potential security pitfalls associated with the DNS and CAs. Historically, using these technologies has required that users place a certain measure of trust in the people and organizations which provide them. With Handshake, users no longer need to place trust in any single institution because the protocol is based on a decentralized consensus mechanism; as long as no one party can control a majority

of the network's computational power, it's effectively impossible for malicious actors to impact users on a large scale.

At its core, Handshake is a software package that seamlessly replaces an existing DNS resolver with one that connects to a network of nodes (i.e., servers) running the same protocol. Domain records created through Handshake are stored on its blockchain, and are protected by that technology's cryptographic characteristics. When a Handshake node responds to a client query for a domain record, it automatically verifies the records' associated cryptographic proof, ensuring that the record has not been tampered with. In the event that a record doesn't exist on the blockchain (which would be the case for any domain not deliberately created through Handshake), the node takes on the role of a typical DNS resolver and requests the record through the traditional DNS.

This description of the software's functionality highlights what is likely going to be the protocol's most serious challenge moving forward: adoption. Only domain records registered using the protocol are secured on the blockchain. Consequently, if domain owners never migrate away from existing DNS infrastructure, there may never be any impetus for everyday users to do so either.

This project is a measurement of Handshake's test network (testnet) blockchain, which was taken offline when the main network was launched in February of this year. While others will likely perform analyses of the main network (mainnet) blockchain in the future, I chose to focus on measuring the testnet blockchain specifically because those data may never be widely available again. Additionally, I hoped that my network-adoption measurements would reflect genuine user activity and not be impacted by squatting.

The next section details the work involved in writing the measurement program, and the section after that presents my findings.

Methodology

This section begins with a brief discussion on what motivated my choice of tools for accessing and working with the Handshake blockchain. The remainder of the section details the iterative development of the core measurement program.

Using Existing Resources

While my initial intent was to use the existing Handshake code to build out a program to extract the blockchain data and prepare them for measurement, this ultimately ended up being unrealistic for the scope of this project. At the outset, the most compelling reason for not utilizing the existing code was my complete lack of experience with Node.js. This, combined with the scale and complexity of the code base itself, meant that working with the code directly would be infeasible without considerable time investment. Under normal circumstances, this alone may not have been particularly prohibitive, but I also felt it was important to begin working with the data early so I could evaluate the protocol's potential as a research topic.

Rather than adapting existing code, I instead elected to primarily use the Handshake node's HTTP API. Access to the data was not as granular and making HTTP requests rather

than functions calls necessitated a few extra considerations, but the fact that I could gain access to the blockchain data immediately made it more suitable for this project.

First Method – HTTP Requests

When a new node starts up for the first time, it reads from a hard-coded list of "seed nodes" addresses to determine where it should start trying to make connections. A seed node is a public node that serves as an entry point to the network for new clients. Once a client connects to a seed node (or any other node for that matter), the seed node responds with a list of addresses for other active nodes that the client can try to connect with. In this way, clients expand their lists of available network entry points as they interact with the network.

This system is common among peer-to-peer networks, but it obviously relies on the seed nodes being regularly available. In the months leading up to the closure of the testnet, however, I was consistently unable to connect to a seed node. Fortunately, I eventually learned about a prominent website – *HNScan.com* – that operated and promoted a reliable public node, in addition to providing a web interface where users could query the node directly.

Because the node and the web server were effectively my only entry points to the network anyway, I essentially designed the first iteration of my measurement program as a Handshake-specific HTTP wrapper that would perform the following tasks: 1) query the node for a summary of all new data on the blockchain (based on a "checkpoint" file from a prior connection); 2) query the web site for every page serving new resource information for a block,

name, or transaction (rate limited to one every five seconds); and 3) aggregate the data and write them to a local file to serve as the checkpoint during the next connection.

Second Method – Deconstructing the Blockchain

After using the HTTP-based program for a few weeks, it became apparent that a considerable amount of measurable data wasn't being represented in the API query responses. The Handshake software constructs and updates its blockchain based on the transactions recorded in each new block. Each transaction output has a type and a value (or multiples of these), which indicate specific changes that need to be made to the overall state of the blockchain, similar to how deposits and withdrawals inform a bank of when money needs to be added to or taken from a customer's account. While this kind of cumulative account representation is sufficient to answer a bank customer's basic questions ("*what's my balance?*"), it fails to answer more interesting questions ("*what are my most common expenditures?*"). By the same token, the cumulative current state of the testnet blockchain only ever provides a one-dimensional perspective of user adoption of the protocol.

While *HNScan.com*'s web interface allowed me to search for previous transactions, it didn't provide me with a way to *recreate* the state of the blockchain at the time those transactions were submitted. The most thorough way to achieve this kind of functionality would have been to reimplement the protocol's state transition operations (i.e., the logic which determines how different transactions affect the chain's state), but the complexity of both the protocol and the code base seemed to put that approach out of my reach.

The solution I discovered for this problem presented itself in one of the node's API calls: *reset*. This call is primarily intended to allow users to purge prior transactions from a node's database in the event of a recording error, but the operation it performs also made it possible for me to retroactively evaluate past chain states. Normally, the call resets the chain to a height prescribed by the user and, upon completion, immediately begins requesting missing blocks from connected peer nodes. However, if the node is *unable* to connect to a peer node and restore the missing information, it disregards the discarded transactions and continues to operate from the past block state. By starting with an up-to-date copy of the blockchain, disabling the computer's networking (to prevent the node from connecting to peers), and iteratively resetting the chain to every possible block height, I was able to track and record every available value across the entire history of the blockchain.

Third Method – Progressive Name Tracking

Although iteratively deconstructing the blockchain one block at a time allowed me to identify the exact block height where a given value of interest had changed, it said nothing as to whether or not more changes existed at other points in the blockchain's history. As a result, I was forced to check for new values for each name, at each block height. By the time my reset program reached a height of 25,000 blocks, I was querying my node for the domain records of more than 8,000 names every time I reset another block, leading to processing times of roughly ten minutes per block. With another 15,000 blocks still left to examine in the testnet, my only option for finishing the measurement in a reasonable time frame was to reimplement some of the protocol's underlying state transition functions.

I had previously considered this approach impractical due to the protocol's complexity, but that had been prior to obtaining a sample data set. Once the time came to actually implement the functions, I had 25,000 blocks' worth of transaction and name history that I could use to validate my assumptions (in conjunction with an extensive review of the node's source code). While this method took drastically more time to implement than either of the other two, it also reduced the time and space complexity of my program by a few orders of magnitude.

This third method begins by querying the API for all of the blocks in the blockchain (which is notably different from resetting the blockchain to each height), and recording the transactions they contain. The program takes the full set of transactions and maps each domain name to the subset of transactions which correspond to (i.e., modify) it. Each domain name's set of transactions gets passed into the state transition function and converted into a list of block heights where changes to the name resource's values are known to occur. The program then uses these lists of derived block heights to perform a modified version of the iterative deconstruction, differing from the original in that this method only requests a name resource at block heights where it's known to change. Finally, as this produces a sparse data set when compared to the previous brute force method (which recorded a resource for every name at every block), the name resource values are duplicated to fill in the measurement model for each block height where the name resource was not requested.

Results

After having worked with the testnet data for quite some time, I've concluded that the most meaningful way to approach the problem is to divide the set of domain names into subsets defined by some common substrings and behavior (or by the lack thereof). Unfortunately, this is definitely not an approach that can be generalized to other platforms or data sets. As will become evident in the remainder of this section, the commonalities which define the domain subsets are so glaring that they require neither intuition nor expertise to identify. This is understandable for a test network, but it's unlikely that similar patterns would emerge in a live system.

Following the qualitative discussion on name groups and their observable trends, this section finishes with data visualization to help the reader understand how certain key metrics tracked over the blockchain's life span.

Namebase.io

The domain names which were the most indicative of what appeared to be genuine adoption behavior were those registered through *Namebase.io,* a website offering domain name- and wallet-management products to users. In total, 1,343 of the 8,281 (16.2%) names recorded on the testnet blockchain were determined to have been registered through that service.

While there were not necessarily any prevailing commonalities among the domain names themselves, the name resources created through *Namebase.io* almost always retained a set of default values given to them when they were created: the IP address *52.71.101.8* in the host

record, and the message "*Registered with namebase.io/*" in the text record. Accessing the IP address above, I was served a static web page linking to the Handshake protocol's documentation and to *Namebase.io* itself.

The following are some additional values taken from records created through *Namebase.io*, which still mapped to active services:

- 159.65.38.15: *Socii*, a social media management company.
- 1.1.1.1: *Cloudflare*'s well-known DNS service.
- 206.189.189.149: *binarynoggin*, a software development company.
- 165.22.151.242: *easyhandshake.com*, a Handshake service provider.

Note, that even though a resource on the handshake network is free to include any IP address the person in control of it wants, there's no way to know for certain that the person in control of that domain on the Handshake network is the same person that's in control of the domain on the traditional DNS (and in fact, in most cases, it won't be).

Of the remaining 6,938 (83.8%) registered names which were not created through *Namebase.io* (or at least which didn't have the typical name resource values which would indicate they were), only three included non-empty resources at some point during the testnet's life span. It's possible that this hints at there being a distinction between the behavior of *Namebase.io* users and that of all other users. However, a problem with the data set has to be resolved first, before I can make conclusions about such a possible distinction.

Test Groups

On multiple occasions, the network was flooded with nearly identical names, which seemed to indicate that a test was being performed. These test groups are denoted below according to common substrings and transaction windows:

- "*one-domain-and-one-bid-testing*": 98 names opened (0 on *Namebase.io*)
- "*one-domain-and-one-bid-test*": 100 names opened (1 on *Namebase.io*)
- "*one-domain-one-bid-test names*": 100 names opened (0 on *Namebase.io*)
- "*three-hundred-domains-and-one-bid-testing*": 600 names opened (19 *Namebase.io*)
- "*three-hundred-domains-ten-bids*": 5,382 names opened (11 *Namebase.io*)
- "*test*": 36 names opened (0 *Namebase.io*)
- "*synack"*: 134 names opened (2 *Namebase.io*)

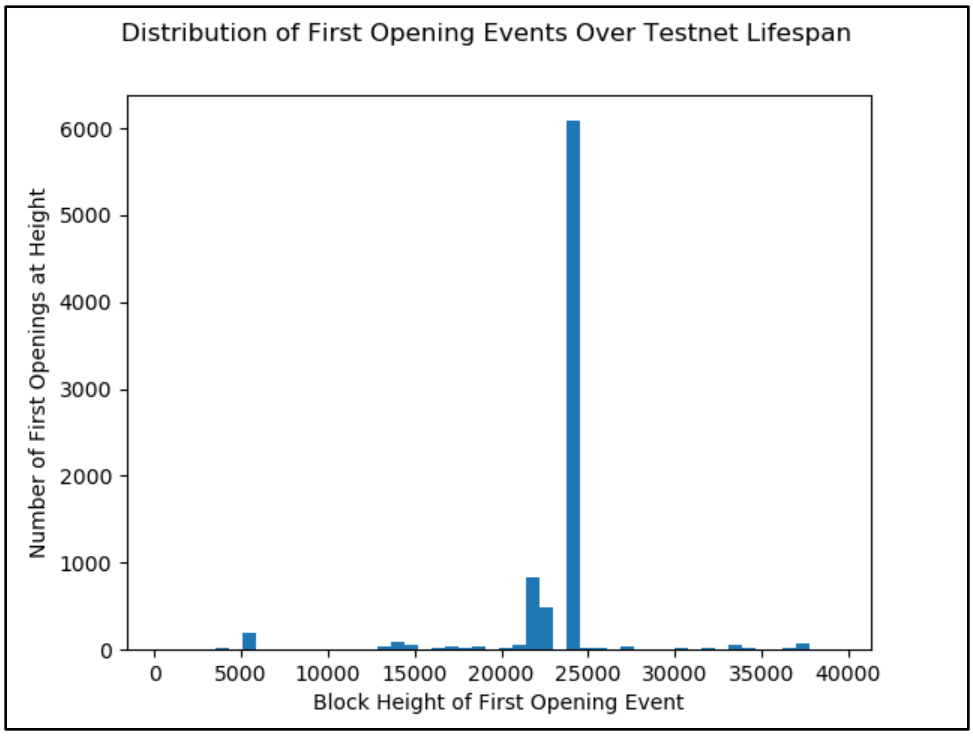
This leaves 1,831 unique names which are not obviously part of some test group. Across the 1,310 of these names which were registered through *Namebase.io,* there were twenty user-defined IP addresses and five user-defined canonical records. Across the 521 of these names which were not registered through *Namebase.io,* there were two user-defined IP addresses and one user-defined canonical record. Excluding test-group names, then, a resource created through *Namebase.io* is approximately four times as likely to be associated with a unique user-defined IP address as a non-*Namebase.io* resource, and approximately twice as likely to be associated with a unique user-defined canonical record.

While these results confirm that there *is* a substantial difference in how *Namebase.io* users and non-*Namesbase.io* users interacted with the testnet to record meaningful data, it's clear

that this difference is not nearly as drastic as initially believed. Further investigation would be needed to determine if these groups' usage patterns remained consistent after the launch of the mainnet.

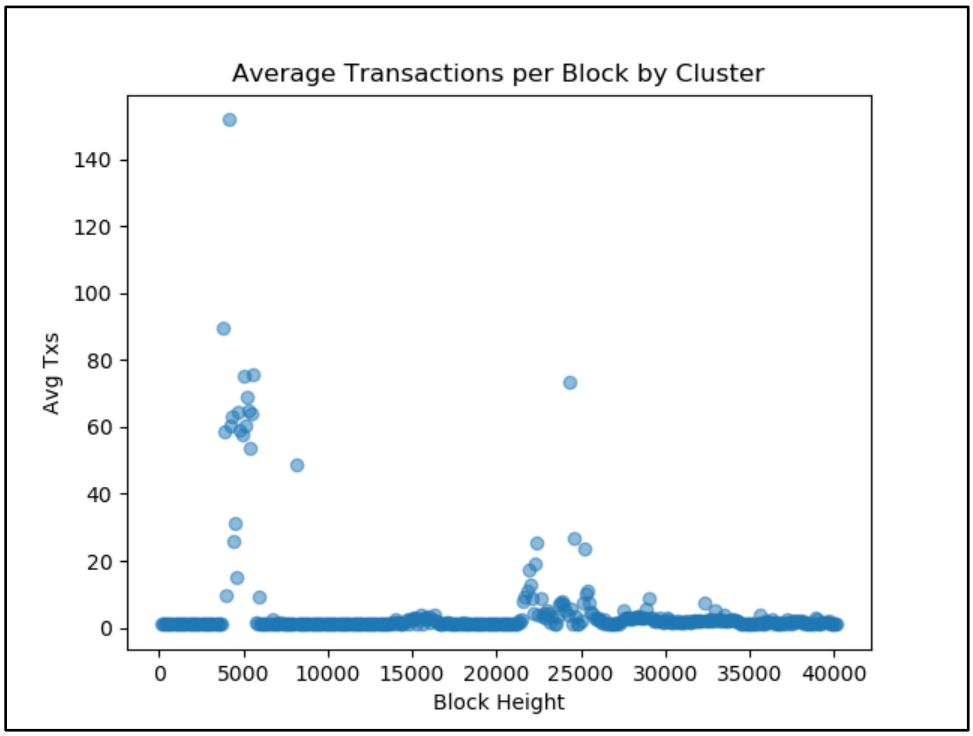
Visualizing the Data

One surprising observation from analyzing the testnet blockchain was the existence of the "*three-hundred-domains-ten-bids*" group which ended up constituting more than half of all the names that were ever recorded on the network. The image below shows how drastically the introduction of this group differed from the protocol's typical usage patterns:



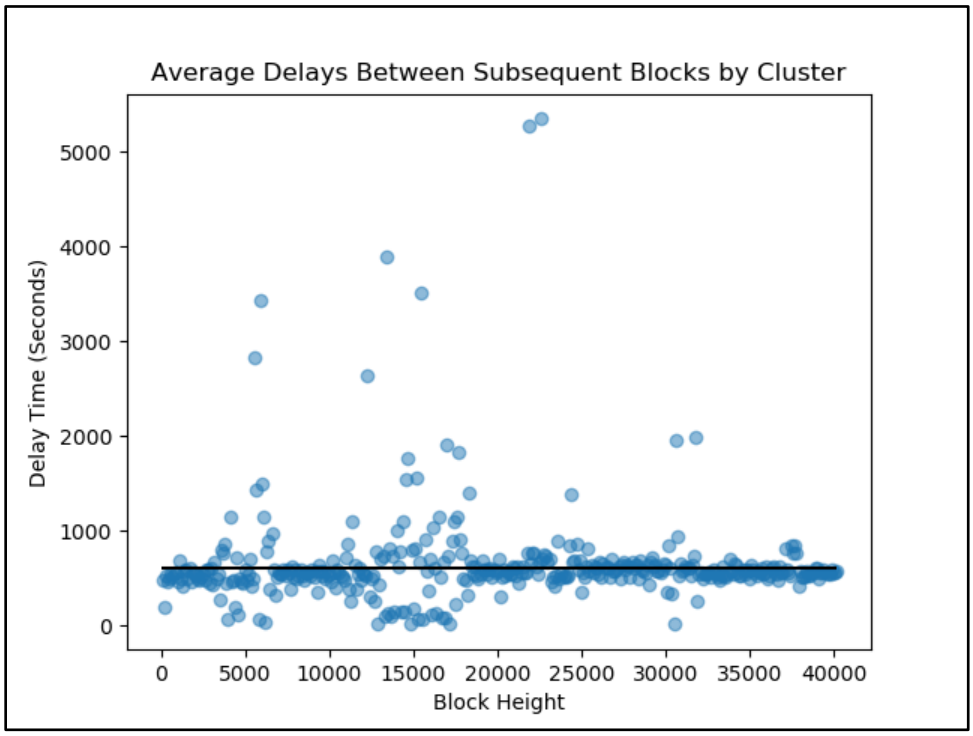
As its title states this graph shows the frequency of transactions intended to “open” a domain name for auction for the first time. The previously mentioned test group is obviously not the only set of names represented in the graph’s central spike, but it’s clear it is the predominant one.

When events like the one shown above occur, they typically have an impact on other network metrics. The image below shows how the average number of transactions per block was forced to spike to accommodate the influx of new opening events:



Interestingly, while the sudden influx of new transactions around block 25,000 is clearly visible, it’s much smaller than one that occurs at 5,000. How can the sharp increase in average transactions per block at 5,000 be explained without a corresponding increase in published transactions like the one found in the first image?

While it's hard to know for certain what caused the spike at 5,000, it should be remembered that high transaction volume isn't the only thing that can cause the ratio of transactions per block to increase. Rather than increasing the number of transactions per unit time (which typically corresponds to a fixed block publication rate), the average transaction metric can also be altered by *decreasing* the block publication rate and holding transactions constant. The next image shows how far each block deviated from its target block time (of 10 minutes):



Ultimately, unusual deviations from target metrics are likely caused by a multitude of factors, especially in a system as complex as a peer-to-peer network.

Limitations

As demonstrated in the qualitative portion of the last section, there are still some parts of the measurement process that rely on unproven assumptions about the either protocol or the transactions contained in the blockchain. Unfortunately, even with a perfect algorithm for reconstructing the blockchain on demand, there would still be fundamental elements of my approach that would preclude me from recapturing *all* of the data that were lost following the original publication of a new block.

This is especially noticeable when operating what is known as a mempool. When individuals want to publish a transaction to a blockchain network, they send the transaction to a node's mempool, where it stays until a node has seen the transaction and included it in a new block; they are like read buffers for the blockchain at large. From my perspective looking backward, I can only see transactions that were *successfully* added to the network. In other words, it's entirely possible that faulty nodes or malicious users might try to send incorrectly formatted data which, if not identified and rejected by the node operating the mempool, could lead to a corrupted blockchain. But as those kinds of activities are never intentionally recorded, the only way to measure and learn about them would be to operate the mempool in real time.

Another limitation is how I'm able to interact with archived copies of the blockchain's database. Whenever changes are rolled out to the code base, only different versions of the code that were released close to one another can maintain full interoperability. However, at some point, it's possible that different archived copies of the Handshake software lose that interoperability because of stagnant data. Since the launch of the mainnet in February, I've experienced instances when my blockchain has been rendered unusable, (presumably) due to

changes to the code base. In these cases, I've been forced to rollback to previous versions of the code in order to access my data again. Frankly, performing a retrospective analysis of a blockchain can involve inadvertently mixing conflicting versions in a way that leads to an inaccurate or unpredictable reconstruction process, and it can be difficult to diagnose such an issue.

Conclusion

At the onset of my work with the Handshake protocol, the fact that thousands of domain names were recorded on its blockchain seemed to indicate that users were receptive to the idea of moving away from a reliance of Internet systems with centralized points of failure. While the results of my measurement don't *disprove* that claim, they don't offer any compelling evidence in favor of it either. With that said, it's important to mention that my measurement did, in fact, identify approximately 30 unique host record values that were created by users who presumably found themselves motivated by what the protocol had to offer.

Currently, Handshake's mainnet sits at a block height in excess of 45,000, just a few thousand blocks more than where the testnet was terminated. And, even despite the lack of widespread adoption during the test phase, the mainnet has already attracted enough user interest that more than 360,000 names have been opened thus far. Moving forward, I believe that Handshake and other security-focused alternatives to legacy systems will continue to attract the attention of researchers and – hopefully – the attention of everyday users, in the process.