N Chainz: A High Performance, Decentralized Cryptocurrency Exchange

Nick Egan
negan@mit.edu

Ryan Senanayake
rsen@mit.edu

Lizzie Wei
ewei@mit.edu

1 Introduction

Centralized exchanges rely on trusting that their owners will take the proper security precautions. This has led to many incidents of stolen cryptocurrency adding up to billions of dollars worth of losses, and is a stark contrast to the decentralization of the rest of the space. On the other hand, decentralized exchanges have the potential to be much more secure: theoretically, users are not vulnerable to server downtime and hacks, and can retain anonymity.

In this project, we present N Chainz, a decentralized cryptocurrency exchange. Specifically, N Chainzs features include block generation, limit orders, and the ability to trade a base token with another token. Our main goals are high performance and fault-tolerance. While we would ideally focus on high performance, the reality is that building high-performance exchanges is a complex problem even when built in a centralized fashion. We believe that we have been able to achieve a relatively simple design while not sacrificing performance. In this paper, we will discuss our major design choices and implementation.

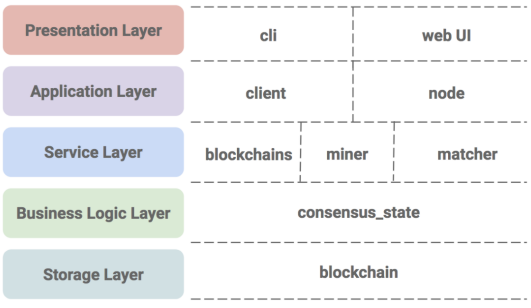

Figure 1: *System Overview. N Chainz's implementation is organized in several layers, which we discuss in detail in the next few sections.*

2 Multiple Blockchains

The most unique aspect of our design is the use of multiple blockchains to reach a unified consensus state. In our design, we maintain a separate **token chain** for each token added to the exchange as well as a **match chain**, which coordinates interactions between tokens. This has a number of advantages over a single chain design, which will be discussed in the following sections.

2.1 Comparison to Single Blockchain

The main improvement that a multiple blockchain solution offers is scalability. A single blockchain design results in a linear performance decrease proportional to the number of tokens as all nodes must agree on the exact same state. Our design allows for each chain to operate independently with a loose consensus enforced of trades by the match chain. We assume that inter-token commands such as orders do not dominate the usage of this system and that intra-token commands such as transfers are used frequently.

In our prototype, miners randomly choose chains to mine, but as the system scales, miners will be incentivized to select certain chains to follow (all miners must also follow the match chain). This shards the blockchain across overlapping sets of miners to save in the cost of storing this information and also allows for processing many transfers of a single token without affecting the performance of the rest of the system. Even though all miners must follow the match chain, this chain is also the most space-efficient as transactions on this chain consist of pointers to information on token chains (see section 2.3). This also means that less information must be globally agreed upon (typical match transactions in our system are 56 bytes whereas typical bitcoin transactions are 250 bytes), which theoretically improves our throughput.

The bottleneck of most blockchain systems is

network latency and bandwidth. If the latency of the peer-to-peer network was lower, the mining difficulty can be decreased to produce blocks more frequently, and if the bandwidth was higher, the block size could be increased. Producing a block in single-chain systems creates a flooding effect on the network that increases latency of block propagation and leads to inefficient bandwidth utilization. We believe that a multi-chain system can lead to improved network utilization as each chain reaches consensus at different random times. As forks between chains are less frequent than forks within the same chain (due to the loose consensus mechanism discussed in the following sections), this theoretically allows for us to choose more aggressive parameters (mining difficulty, block size, etc.) to achieve higher throughput than a single blockchain system.

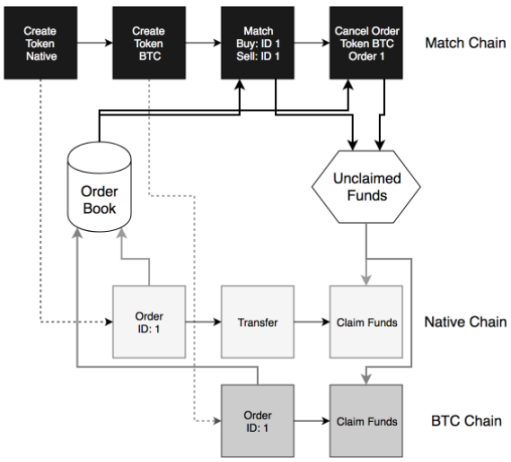

Figure 2: *Multiple Blockchains. N Chainz uses multiple blockchains (One Match and two Token chains are pictured) to store its state. Interaction between tokens is ordered by the Match chain and coordinated by Order Book and Unclaimed Funds. All transaction types are pictured as long as the direction they cause for funds to move.*

While this system does have some advantages, there are also some disadvantages to address. The first is that hashing power is split among multiple chains, making each chain more vulnerable to a 51% attack. However, this system also provides isolation between chains, which helps to limit the effects of an attack. Attacking the match chain can only allow for an attacker to freeze funds currently kept in orders. Attacking a token chain allows for the attacker to only steal funds from this chain. Future work should be done to further mitigate such an attack. One possibility is to adapt how miners choose which chain to mine such that a 51% attack is detected and countered by increased hashing power being devoted to the chain. Alternatively, different consensus mechanisms can be used such as proof-of-stake (see section 4.1). The other disadvantage is that it is less clear when it is okay to assume that a block will be included in the final state and isnt part of a shorter fork. However, we decided that this tradeoff was justified by the increased performance of such a network. A decentralized exchange provides a unique opportunity to attempt such a multi-chain design and believe that future work will resolve these problems just as over time they were resolved for single-chain systems.

2.2 Token Chain Behavior

By following a single token chain, the balances of all token holders can be recovered. As discussed in section 3.1, N Chainz chooses to keep a map of balances for all token holders instead of the Bitcoin model of tracking unspent transaction outputs. Three different transactions are supported on a token chain: Transfers, Orders, and Claim Funds. Transfers allow transferring funds between two addresses. Orders allow offering one token for another at a specified exchange rate (see section 2.4). Once an order has been completed or cancelled, resulting funds are kept in an Unclaimed Funds store. By issuing a Claim Funds transaction, users can make these funds available for spending on the token chain. Logically, token funds are loaned to the match chain via the order book and then reclaimed to their respective match chains via the Unclaimed Funds store.

The data stored in each type of token chain transaction is listed below:

- Order
  - ID: unique id for order
  - Buy Symbol: other symbol (not the chain that this transaction is published on) to exchange with
  - Amount to Sell: amount of the current chains token to exchange
  - Amount to Buy: amount of the other token to receive in exchange for Amount to Sell. Combined with Amount to Sell, this defines the price of the order.

  - Seller Address: address to withdraw funds from this token and deposit into the other token
  - Signature: signature by private key to prove that order was sent by Seller Address

- Claim Fund
  - ID: unique id for claim funds
  - Address: address to attempt to claim funds for
  - Amount: amount to claim from Unclaimed Funds Store

- Transfer
  - ID: unique id for transfer
  - Amount: amount to transfer between addresses
  - From Address: address to send from
  - To Address: address to send to
  - Signature: signature by private key to prove that order was sent by From Address

2.3 Match Chain Behavior

The match chain acts as a master log that provides a global ordering of events on token chains. Match chains contain three different transaction types: Create Tokens, Matches, and Cancel Orders. All tokens are initially created on the master chain via a Create Token. Matches are found by connecting orders in the order book (see section 2.4). Finally, a Cancel Order refunds the money from an order and prevents it from being matched.

The data stored in each match chain transaction is below:

- Match
  - Match ID: unique id for match
  - Sell Symbol: token on sell-side of match
  - Sell Order ID: id for matched sell-side order
  - Seller Gain: amount given to seller in buy tokens
  - Buy Symbol: token on buy-side of match
  - Buy Order ID: id for matched buy-side order
  - Buyer Loss: amount taken from buyer in buy tokens
  - Transfer Amount: amount taken by seller and amount given to buyer in sell tokens
  - Matcher address: Address of node that found match, which will be credited the surplus of the match

- Cancel Order
  - Order Symbol: symbol of order to cancel
  - Order ID: id of order to cancel
  - Signature: signature by private key to prove that Cancel Order was sent by creator of order

- Create Token
  - Symbol: symbol of new token. This acts as a unique identifier for the token
  - Total Supply: the total supply of the token. This amount will be credited to the creators address on creation of the token
  - Decimals: number of decimal places of a token. This is used only for displaying the currency to humans.
  - Creator Address: the address of the creator. This address must have sufficient funds to create a token on the native token chain (the first token chain created). This address will also be sent the Total Supply in the new token.
  - Signature: signature by private key to prove that order was sent by Creator Address

2.4 Multi-chain Consensus

Generally, consensus can be reached independently on each chain, however a few invariants must be maintained in order to ensure a valid state. In single-chains systems, it is always possible to rollback and to maintain a valid state. However, in our system two invariants must be checked when rolling back chains. First, when rolling back Orders, they cannot be referenced in Matches on the match chain (if they are rollback the match chain). Second, when rolling back Matches and Cancel Orders on the match chain, no values in the Unclaimed Funds store can become negative (if they do rollback the corresponding token chain).

When matching orders, it is important to only allow mined orders to occur in a match. Other-

wise, an invalid state can occur if a match references an order that is mined in the same block as a Claim Funds that is dependent on the current match. This block would be rejected by the network, but could be mined by a miner.

3 Consensus State

Each token chain results in an agreed upon consensus state. This state can be recovered by replaying all of the transactions in each blockchain. It is important to note that any valid replaying of the blockchain log results in the same consensus state. If this was not true, then the stored blockchains would not be a real log, which would make recovery and consensus impossible.

3.1 Token Chain Consensus State

The state associated with each token is the following:

- Balances: a map from addresses to balances. This is different than the bitcoin model of unspent transaction outputs. This allows for higher performance and simplicity at the expense of lightweight SPV nodes, which we decided was a worthwhile tradeoff.
- Unclaimed Funds: a map from addresses to the amount of this token in the Unclaimed Funds store. Funds are placed here after a completed match or a cancelled order and can be moved to balances by the user issuing a Claim Funds transaction.
- Used Transfer IDs: a set of used transfer ids. This ensures that a transaction only occurs once and cant be replayed by an attacker.
- Open Orders: a map from order ids to Order transactions. This allows a Cancel Order or Match transaction to reference the order id directly. Orders are removed when they are cancelled or depleted by Match transactions
- Order Update Counts: a map from order ids to how many mutations have occurred on the order. This tracks how many matches have been applied to an open order. This is necessary information to ensure that an order that has already been matched doesnt get rolled back (described in 2.4). Order IDs are removed from this map once the order is removed from Open Orders.
- Deleted Orders: a map from order ids to deleted Orders. This allows for deduplication of orders as well as the ability to rollback Match and Cancel Order transactions. We discuss in Section 7 how this does not need to be stored in memory and a future implementation should store this on disk with the relevant block.

3.2 Match Chain Consensus State

The match chain mostly modifies the state on token chains, but also maintains the following state:

- Created Tokens: a map from token symbols to the information about the token. This includes the total supply, creator address, and number of decimals.
- Used Match IDs: a set of used match ids. This is not necessary, but allows for us to deduplicate matches without processing the match, which provides a performance optimization.

4 Proof of Work

4.1 Comparison to Other Consensus Algorithms

We chose proof of work over proof of stake for the sake of simplicity. We are already attempting to build a complex multi-chain consensus system and decided that for a prototype, built from scratch, proof-of-work would be better suited.

In **proof-of-work**, miners solve computationally hard problems to verify the transactions on each block. Thus, a miners probability of success in creating a block is proportional to the fraction of computational power they have. Bitcoin and Ethereum are currently based on proof of work, which helps provide real-world evidence of the security of such a system.

In contrast, **proof-of-stake** depends on a network of trusted forgers. Due to their stake, they are motivated to validate. Thus, a forgers probability of creating a block is proportional to the fraction of the total number of coins that they own. Looking into chain-based and BFT-style proofs of stake, we decided that proof of work would be simpler to implement and therefore well-suited for an initial prototype. We looked into many proof-of-stake libraries that purported that it would be simple to build on top of. However, we found that many of these hid significant design flaws. For

example, Tendermint seemed to be a good library until we discovered that blocks are only proposed by a small set of proposer nodes. These nodes are chosen round-robin, which means that a denial-of-service attack can be mounted on the entire network by attacking one node at a time. We did not trust current implementations, but look forward to new proof-of-stake algorithms such as Ouroborus Genesis and Algorand, which do not suffer from the same simple DOS attack as these systems. We believe that our model of one blockchain per token is perhaps well-suited for a proof-of-stake or hybrid proof-of-work and proof-of-stake solution.

4.2 Miners

New transactions are added to an unverified transaction pool. Each miner chooses a random token chain to mine. Then, for that token, the miners try to mine transactions that are currently in the pool as well as new transactions as they come in.

We ensure that transactions are valid by making all transactions in the pool unique, and by validating each one through the consensus state before mining.

We currently have a set constant difficulty for mining. We looked into implementing adjusting the difficulty - for example, Bitcoin adjusts its difficulty such that blocks are mined roughly every 10 minutes. However, we decided to stay with a constant difficulty for our initial prototype.

Each chain reaches consensus separately. Although nodes keep track of all chains, they only mine one chain at a time. The longest valid chain is chosen to be correct. If a miner is on a shorter chain, they will get the more up to date version from a peer. As each chain reaches consensus separately, we can replace one chain at a time.

4.3 Matchers

Matchers are a subset of miners, which also submit match transactions based on orders on the order book. Any node can choose to find matches, but we expect that only a small subset will participate as they are only incentivized by collecting the surplus of matches. This is actually by design as the computation and memory needed to find matches is not necessary to be replicated over all nodes. By limiting this computation to a subset of high-powered nodes, we actually decrease wasted computation and increase throughput. This is another big advantage of having a separate match chain as many matches can be mined to each block of this chain. All nodes must still validate all matches created and therefore there is not a fear of 51

5 Order Matching

As an exchange, the primary function of N Chainz is to allow users to exchange one kind of token for another. This exchanging of tokens can go from any token to any other token, and all relevant data about token exchange is stored on the blockchains.

5.1 Orders

Like any other exchange, N Chainz allows users to exchange tokens through limit orders. On most exchanges, a limit order specifies that the user would like to buy or sell a certain amount of an asset (like Bitcoin) for at most or at least a certain price (in a base currency like USD). With N Chainz however, the notion of the asset versus the base currency is ambiguous, since every token can be exchanged for any other token. Therefore, an N Chainz order specifies that the user would like to buy a certain amount of one token (the buy amount) in exchange for at most a certain amount of another token (the sell amount). Thus, all orders are both buy orders of one token and sell orders of another token.

Limit orders in N Chainz are Good-Til-Canceled (GTC), which means that they never expire. If a user would like to cancel their order, they can do so with a cancel-order transaction. Both order transactions and cancel-order transactions are stored on the token chain for the token they are selling.

5.2 Orderbook

An orderbook is a data structure used by exchanges to match up buyers of an asset with sellers of an asset, where the buyers and sellers are using the same base currency. A typical exchange has a quote side, containing orders wishing to buy the asset, and a base side, containing orders wishing to sell the asset. In N Chainz, each currency pair has an orderbook, for which the quote side is buying the token that has an alphabetically higher symbol in exchange for the alphabetically lower symbol, and the base side is buying the token that has an alphabetically lower symbol in exchange for the alphabetically lower symbol.

It is important to note that the orderbook is a data structure kept in memory by the node in order to facilitate matches, and not on the blockchain explicitly. Within the context of this orderbook, or-

ders can be categorized as buy orders that buy the quote token in exchange for the base token, and sell orders that sell the quote token in exchange for the quote token, and orders can be given a price in the base token.

5.3 Matches

A match can be made between a buy order and a sell order on an orderbook if the buy price is at least the sell price. A match has an amount transferred, which is the amount of the quote currency transferred from the seller to the buyer, a seller gain, which is the amount of base currency transferred from the buyer to the seller, and a buyer loss, which is the amount of quote currency transferred from the seller to the buyer. Matches can either completely fill orders, if they deplete the orders buy amount completely, or partially fill orders otherwise. It is possible for orders to be completely filled and still have a non-zero sell amount left over.

If the buy price is exactly the sell amount, then the seller gain is equal to the buyer loss, but if not, then the seller gain is less than the buyer loss, a property that can be derived from the definition of a limit order. In this situation, the leftover amount of base currency is given to the miner as a reward.

5.4 Matching Engine

The matching engine is a component of our system that tracks an orderbook for every currency pair and finds matches. It stores each orderbook as 2 heaps, with a max heap tracking the quote side and a min heap tracking the base side. The matcher checks for matches whenever the system reaches a quiescent state (no blocks are currently being rolled back or applied). When this occurs, the matcher repeatedly extracts from both heaps until there is no more overlap. Every match is gossipped to other nodes and mined in blocks. The matcher only attempts to match mined orders as attempting to match unmined orders can lead to the invalid state discussed in Section 2.4.

Most exchanges that people are familiar with have an quote asset (such as Bitcoin) that is being exchanged for various prices in a base currency (such as USD), and thus a clear notion of buy orders used to buy the asset and sell orders used to sell the quote asset. On N Chainz however, we allow any token to be exchanged for any other token, which makes the notions of quote and base sides and buy and sell orders ambiguous.

6 Network

Like any other blockchain system, N Chainz is distributed across nodes in a peer to peer network. Every N Chainz node is responsible for storing a copy of the blockchain, maintaining a copy of the consensus state, and responding to network RPC calls. In addition, N Chainz nodes can optionally mine new blocks and match orders.

6.1 RPC Messages

N Chainz nodes exchange messages via RPC calls. The messages used are similar to those of Bitcoin, but modified to fit our network design.

6.1.1 VERSION

VERSION is used by network peers as a handshake when establishing a connection. When a node hears about a new peer, it send it a VERSION message that includes the version of the N Chainz protocol it is running. When a node receives a VERSION message, it makes sure that it is running the same version of N Chainz, and if so, responds with its own VERSION message. Every node rejects RPC calls from a peer if they havent successfully exchanged this VERSION handshake in order to ensure that all nodes in the network are following the same protocol.

6.1.2 ADDR

ADDR, meaning addresses, is used by nodes to share a list of known peers with other nodes. Whenever an N Chainz node connects to a new peer, it broadcasts ADDR to its existing peers to notify them of this peer. When a node receives an ADDR message, it attempts to connect to all of the new peers with a VERSION message. A consequence of this behavior is that our N Chainz network is fully connected, with every node communicating with every other node. While this design makes it harder for degenerate network topologies to emerge, it will put excessive load on the network as this system scales. To deal with this, we plan on introducing a system of nodes limiting the amount of other nodes they communicate with.

6.1.3 TX

TX, meaning transaction, is used to inform a node of a transaction on the network that should be eventually mined into a block. A TX message can be sent by either a client when it makes a new transaction, or by a peer node when it hears about a new transaction. When a user of N Chainz would

like to make a new transaction, it uses its client to send a TX message to a running node. When a node hears about a new transaction, it verifies it against its current state. This verification depends on the transaction type, which can include checks like making sure the user has a sufficient balance for a transfer transaction or that an order wasnt already filled for a match transaction. If the new transaction is valid, the node broadcasts the TX message to its peers. If the node is a miner node, it adds the transaction to its mempool to be mined.

6.1.4 INV

INV, meaning inventory, is used by a node to share blockhashes of its chains, and N Chainz nodes periodically broadcast it to all their known peers. In our current system, an INV call includes all blockhashes of all chains, but we plan on improving the protocol to include only the most recent blockhashes in unsolicited messages while providing older blockhashes on demand. When a node receives an INV call, it compares its blockhashes to its peers blockhashes, using the chain reconciliation mechanisms described below.

6.1.5 GETBLOCK

GETBLOCK is used by a node to get the blockdata for a block given the blocks hash and chain, and is used during the chain reconciliation process to bring a node up to date with a peers chain. When a node receives a GETBLOCK from a peer for a blockhash, it returns the block data if it has it in its chain, and returns an error if it doesnt.

6.1.6 Client RPC Messages

N Chainz nodes are also equipped to respond to a few message types used by N Chainz clients in order to get the current state of the node. Nodes currently support a GETBALANCE call which returns the balance of a user, a GETBOOK call to get the orderbook, and a DUMPCHAINS call that returns metadata about recent blocks on all the chains. These RPC calls are used to build our web interface, and we may find it useful to add additional message types to allow better client interaction with the network.

6.2 Chain Reconciliation

When a node receives an INV call from a peer, it will often find that one of the peers chains is different than its own corresponding chain. In order to reach consensus about the system state, it is important for a node to determine who has the more up to date chain, which is defined as the longest chain where every block on the chain is valid. If a node hears about a peer with a longer match or token chain, it will roll back its own chain until the blockhash at the tip of the chain matches the hash of the block on the peers chain with the same height. The node is guaranteed to find such a block because the genesis block for a chain is guaranteed to be the same across peers. It will then use GETBLOCK to request the missing blocks from its peer, and then apply each one in order. While applying, it will ensure that every transaction in every block is valid, and if it finds an invalid transaction, it aborts the chain reconciliation.

6.3 Bootstrapping

In order for a new N Chainz node to join the N Chainz network, it needs to know the network address of at least one existing node. Each node keeps a list of seed addresses to bootstrap its connection.

7 CLI and Web Interface

We built a command line interface for users to manage accounts, create transactions, and run nodes and miners. Specifically, in terms of managing accounts, users can create wallets, get the balance of a specific address in a specific token, and print all addresses. For creating transactions, users can put in limit orders between different tokens, transfers, cancels orders, claim funds, and create new tokens. Finally, for running nodes and miners, users can start up a node at a specific host and port, and print all blocks in a specific chain.

```
N Chainz: a high performance, decentralized cryptocurrency exchange.

Usage: nchainz COMMAND [ARGS]

The commands are:

Account management
        createwallet
                Create a wallet with a pair of keys
        getbalance ADDRESS SYMBOL
                Get the balance for an address
        printaddresses
                Print all adddreses in wallet file

Creating transactions
        order BUY_AMT BUY_SYMBOL SELL_AMT SELL_SYMBOL ADDRESS
                Create an ORDER transaction
        transfer AMT SYMBOL FROM TO
                Create a TRANSFER transaction
        cancel SYMBOL ORDER_ID
                Create a CANCEL_ORDER transaction
        claim AMT SYMBOL ADDRESS
                Create a CLAIM_FUNDS transaction
        create SYMBOL SUPPLY DECIMALS ADDRESS
                Create a CREATE_TOKEN transaction

Running a node or miner
        node HOSTNAME:PORT
                Start up a full node providing your hostname on the given port
        printchain DB SYMBOL
                Prints all the blocks in the blockchain
```

Figure 3: *A screenshot of our Command Line Interface help menu.*

We also have a web interface that visualizes order matching and a 3-minute-window price chart in real-time.

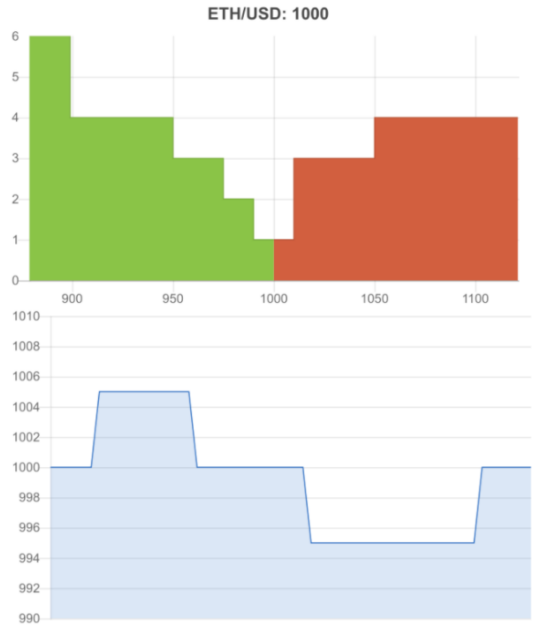

Figure 4: *A screenshot of our Web Interface. The first chart displays orders as they are matched. The second chart displays the price history over the last 3 minutes.*

8 Conclusion

In the future, we plan on adding more features to our initial prototype. Specifically, we will implement freezing and burning of tokens, transactions feeds in native coin, a target adjusting algorithm for our proof of work difficulty, and execution reports triggered by trades.

We also plan to investigate implementing proof of stake in order to improve performance.

Finally, we would like to make some improvements: for example, improving our peer to peer network, and how we store deleted orders. (Currently, we keep deleted orders in memory, instead of writing them to a disk associated with the block.)

Acknowledgments

We would like to thank Ivan Kuznetsov for his blockchain implementation tutorial [1] that helped us get started.

Code

Our repository can be found at the link https://github.com/rsenapps/nchainz

[1] https://github.com/Jeiwan/blockchain_go