SirajCoin: The Cryptocurrency for Programming Wizards (V1)

Abstract

A peer to peer currency could enable a community in way no other tool could. SirajCoin acts as the fuel for a global community of developers to engage with Siraj directly by spending it on his attention via hourly meetings and video collaborations. It will also unlock access to exclusive content, merchandise, and events. Importantly, the coin enables a novel governance structure in the community that gives members voting powers on how to best spend community funds. These funds can be used to fund AI research initiatives, elect community representatives, and empower other creators that spread AI knowledge. This cryptocurrency is the first step in establishing The School of AI, a worldwide network of programming wizards dedicated to solving AI through research and education.

Introduction

In 2 years, Siraj Raval's Youtube channel has grown to 256,000 developers across the world with over 150 countries represented. The community has grown together, using platforms like Slack, Reddit, Twitter, Facebook, and the Youtube comment section to grow, engage, and learn from each other. Siraj alone can't maintain this community, and this currency is a gateway to empower the community through a novel governance structure and exclusive access to Siraj's attention. The advent of Bitcoin 7 years ago proved that we could exchange value over the internet in a way that didn't need to involve a third party, namely a bank. What made this possible was it's use of a data structure that distributes trust across many nodes instead of a few called the blockchain. Every node downloads a copy of it and stores every single transaction that's occurred in the network. It can't be hacked or tampered with because it's secured by the proof of work algorithm. Since then several cryptocurrencies have popped up adding their own features to the bitcoin blockchain to extend its functionality. Ethereum, for example, is a digital currency just like Bitcoin, but it's also got its own Turing complete programming language that lets you build what are called smart contracts. In the Bitcoin network, any coins that you buy are sent to a unique address. No coins are actually held at this address, the address just acts as a unique identifier, like a bank account number that allows the total of the transactions to and from this address to be calculated. We can think of Bitcoin as a ledger that record the deposits and withdrawals from accounts. When we send a transaction via the bitcoin network, the old balance is reduced by some amount and another accounts balance is increased by this amount but no physical value is ever moved. Ethereum works slightly differently. It stores the number of coins you own at an address called a smart contract. This is is a piece of code which is stored on the blockchain network i.e on each participant database. It defines the conditions to which all parties using a contract agrees. So if required conditions are met certain actions are executed. As the smart contract is stored on every computer in the network, they all must execute it and get to the same result. This way users can be sure, that outcome is correct. The ability to store information at these addresses is called the Ethereum State. Whereas Bitcoin only stores transactions that have been sent in the Bitcoin blockchain, any type of data can be stored in the Ethereum state and thereby the Ethereum blockchain. The information that is stored could range from the amount of Ether you have to the type of life insurance you own. Inspired by the smart contract functionality of Ethereum, SirajCoin utilizes its own blockchain that has its own turing complete scripting language that can be written in Javascript. All of this is powered by the Tendermint blockchain engine which we'll go over in the next section.

The TenderMint Blockchain Engine

Tendermint is software for securely and consistently replicating an application on many machines. By securely, we mean that Tendermint works even if up to 1/3 of machines fail in arbitrary ways. By consistently, we mean that every non-faulty machine sees the same transaction log and computes the same state. Secure and consistent replication is a fundamental problem in distributed systems; it plays a critical role in the fault tolerance of a broad range of applications, from currencies, to elections, to infrastructure orchestration, and beyond.

The ability to tolerate machines failing in arbitrary ways, including becoming malicious, is known as Byzantine fault tolerance (BFT). The theory of BFT is decades old, but software implementations have only became popular recently, due largely to the success of "blockchain technology" like Bitcoin and Ethereum. Blockchain technology is just a reformalization of BFT in a more modern setting, with emphasis on peer-to-peer networking and cryptographic authentication. The name derives from the way transactions are batched in blocks, where each block contains a cryptographic hash of the previous one, forming a chain. In practice, the blockchain data structure actually optimizes BFT design.

Tendermint consists of two chief technical components: a blockchain consensus engine and a generic application interface. The consensus engine, called Tendermint Core, ensures that the same transactions are recorded on every machine in the same order. The application interface, called the Application BlockChain Interface (ABCI), enables the transactions to be processed in any programming language. Unlike other blockchain and consensus solutions, which come pre-packaged with built in state machines (like a fancy key-value store, or a quirky scripting language), developers can use Tendermint for BFT state machine replication of applications written in whatever programming language and development environment is right for them.

Tendermint is designed to be easy-to-use, simple-to-understand, highly performant, and useful for a wide variety of distributed applications

Application BlockChain Interface

Tendermint Core (the "consensus engine") communicates with the application via a socket protocol that satisfies the ABCI.

To draw an analogy, lets talk about a well-known cryptocurrency, Bitcoin. Bitcoin is a cryptocurrency blockchain where each node maintains a fully audited Unspent Transaction Output (UTXO) database. If one wanted to create a Bitcoin-like system on top of ABCI, Tendermint Core would be responsible for

- Sharing blocks and transactions between nodes
- Establishing a canonical/immutable order of transactions (the blockchain)

The application will be responsible for

- Maintaining the UTXO database
- Validating cryptographic signatures of transactions
- Preventing transactions from spending non-existent transactions
- Allowing clients to query the UTXO database.

Tendermint is able to decompose the blockchain design by offering a very simple API (ie. the ABCI) between the application process and consensus process.

The ABCI consists of 3 primary message types that get delivered from the core to the application. The application replies with corresponding response messages.

The messages are specified here: ABCI Message Types.

The **DeliverTx** message is the work horse of the application. Each transaction in the blockchain is delivered with this message. The application needs to validate each transaction received with the **DeliverTx** message against the

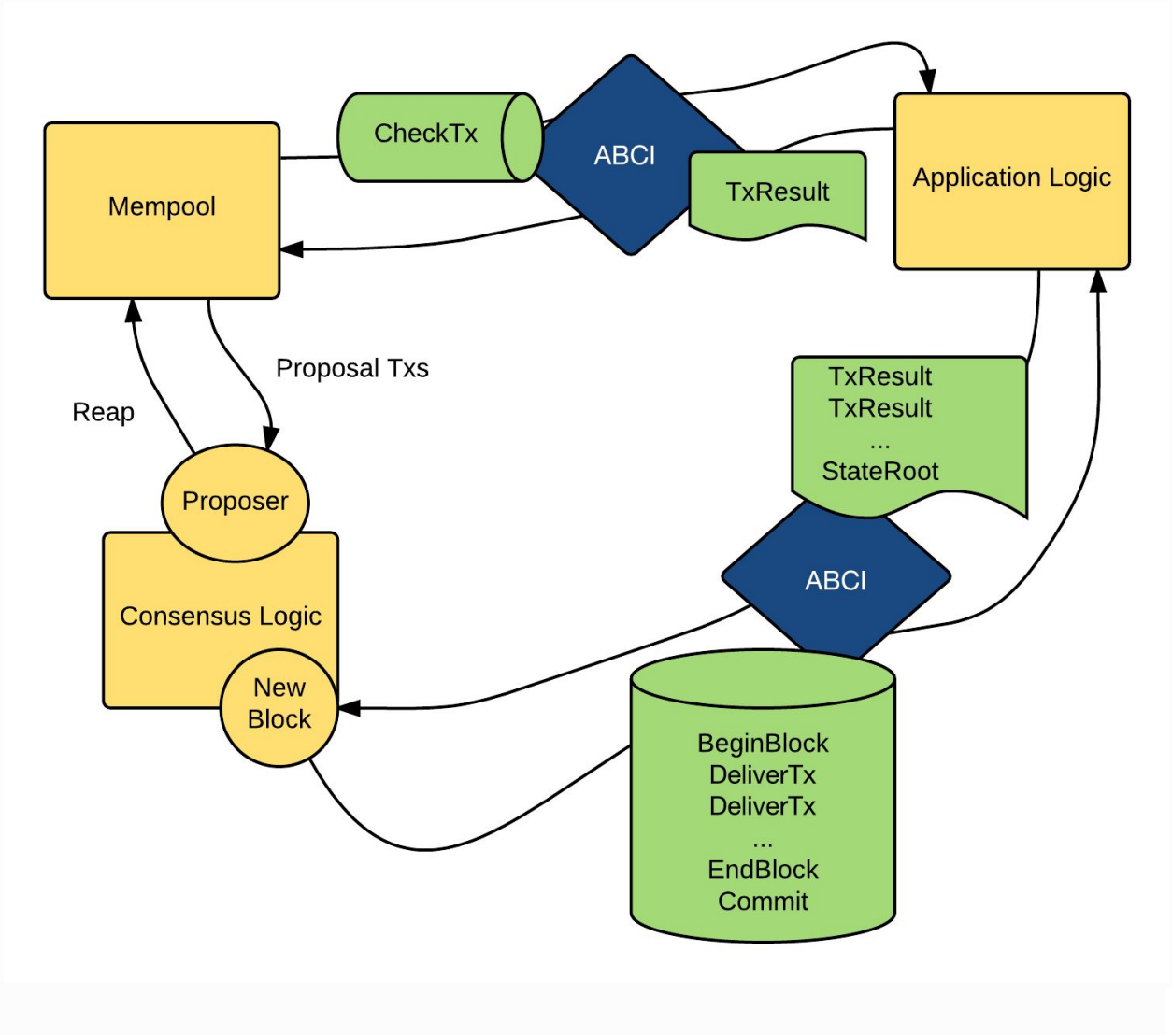
current state, application protocol, and the cryptographic credentials of the transaction. A validated transaction then needs to update the application state — by binding a value into a key values store, or by updating the UTXO database, for instance.

The **CheckTx** message is similar to **DeliverTx**, but it's only for validating transactions. Tendermint Core's mempool first checks the validity of a transaction with **CheckTx**, and only relays valid transactions to its peers. For instance, an application may check an incrementing sequence number in the transaction and return an error upon **CheckTx** if the sequence number is old. Alternatively, they might use a capabilities based system that requires capabilities to be renewed with every transaction.

The **Commit** message is used to compute a cryptographic commitment to the current application state, to be placed into the next block header. This has some handy properties. Inconsistencies in updating that state will now appear as blockchain forks which catches a whole class of programming errors. This also simplifies the development of secure lightweight clients, as Merkle-hash proofs can be verified by checking against the block hash, and that the block hash is signed by a quorum.

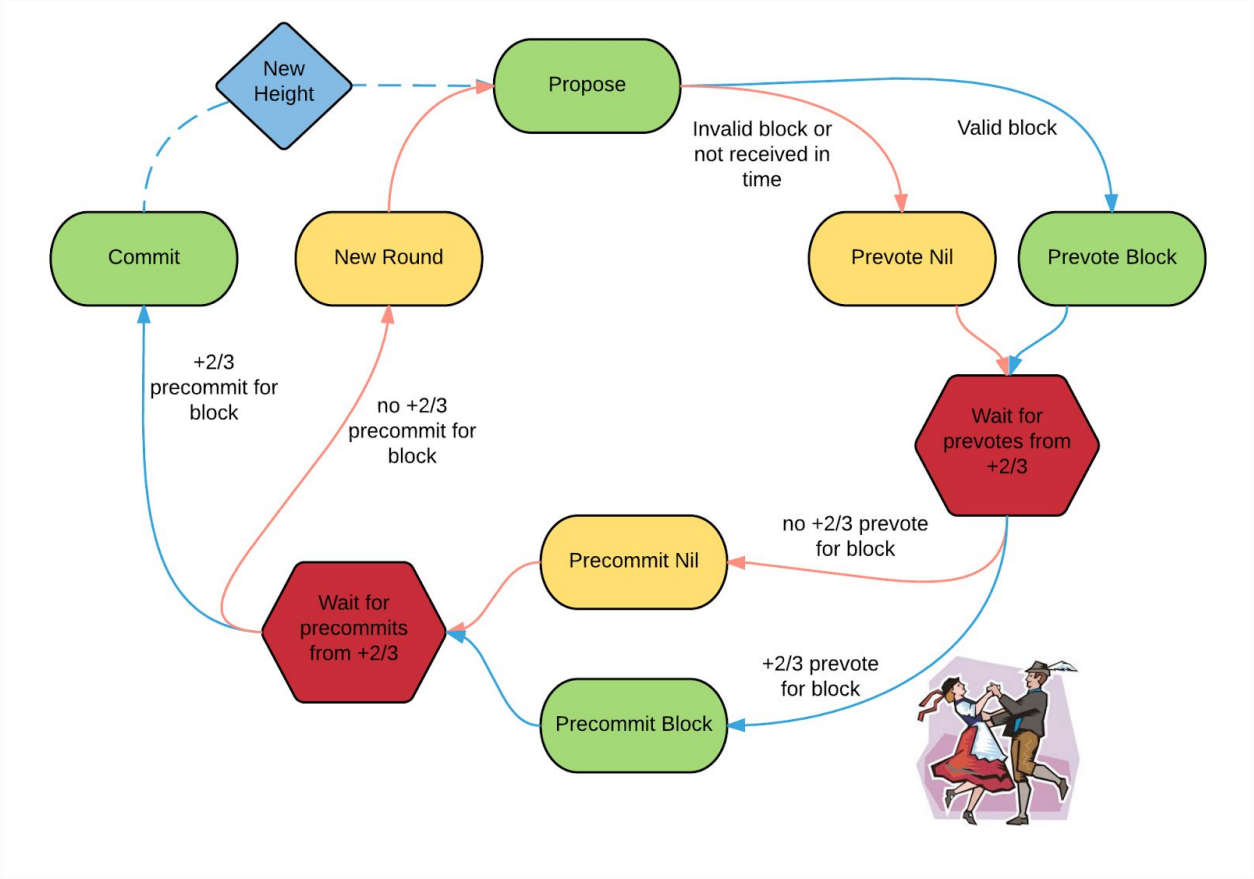
There can be multiple ABCI socket connections to an application. Tendermint Core creates three ABCI connections to the application; one for the validation of transactions when broadcasting in the mempool, one for the consensus engine to run block proposals, and one more for querying the application state.

It's probably evident that applications designers need to very carefully design their message handlers to create a blockchain that does anything useful but this architecture provides a place to start. The diagram below illustrates the flow of messages via ABCI.



Validators

Tendermint is an easy-to-understand, mostly asynchronous, BFT consensus protocol. The protocol follows a simple state machine that looks like this:



Participants in the protocol are called **validators**; they take turns proposing blocks of transactions and voting on them. Blocks are committed in a chain, with one block at each **height**. A block may fail to be committed, in which case the protocol moves to the next **round**, and a new validator gets to propose a block for that height. Two stages of voting are required to successfully commit a block; we call them **pre-vote** and **pre-commit**. A block is committed when more than 2/3 of validators pre-commit for the same block in the same round.

There is a picture of a couple doing the polka because validators are doing something like a polka dance. When more than two-thirds of the validators pre-vote for the same block, we call that a **polka**. Every pre-commit must be justified by a polka in the same round.

Validators may fail to commit a block for a number of reasons; the current proposer may be offline, or the network may be slow. Tendermint allows them to establish that a validator should be skipped. Validators wait a small amount of time to receive a complete proposal block from the proposer before voting to move to the next round. This reliance on a timeout is what makes Tendermint a weakly synchronous protocol, rather than an asynchronous one. However, the rest of the protocol is asynchronous, and validators only make progress after hearing from more than two-thirds of the validator set. A simplifying element of Tendermint is that it uses the same mechanism to commit a block as it does to skip to the next round.

Assuming less than one-third of the validators are Byzantine, Tendermint guarantees that safety will never be violated - that is, validators will never commit conflicting blocks at the same height. To do this it introduces a few **locking** rules which modulate which paths can be followed in the flow diagram. Once a validator pre commits a block, it is locked on that block. Then,

1. it must prevote for the block it is locked on
2. it can only unlock, and pre commit for a new block, if there is a polka for that block in a later round

In many systems, not all validators will have the same “weight” in the consensus protocol. Thus, we are not so much interested in one-third or two-thirds of the validators, but in those proportions of the total voting power, which may not be uniformly distributed across individual validators.

Since Tendermint can replicate arbitrary applications, it is possible to define a currency, and denominate the voting power in that currency. When voting power is denominated in a native currency, the system is often referred to as Proof-of-Stake. Validators can be forced, by logic in the application, to “bond” their currency holdings in a security deposit that can be destroyed if they’re found to misbehave in the consensus protocol. This adds an economic element to the security of the protocol, allowing one to quantify the cost of violating the assumption that less than one-third of voting power is Byzantine.

Cooperation

Since validators divide the transaction fees of block H amongst themselves, a greedy validator might be tempted to exclude some signatures when proposing the next block H+1. This is an inferior strategy when considering that other validators are game optimal participants. Given that the total amount of fees to be divided in a block is f1, and that the sum of the voting powers vi of all validators that have signed and successfully broadcasted their signatures is 1, consider proposer P with voting power vp < 1 who is considering whether to include validator Alice’s signature with voting power va < 1. At stake is Alice’s fair share of the fees which is f1 · va. Of this, P’s incremental benefit of excluding Alice’s signature is: f1 · va · vp/(1 − va) Then, Alice could react tit-for-tat by excluding P’s signature when it becomes Alice’s turn to propose the next block, where the sum of the fees in that 8 block is f2. In that case, P’s detriment is: f2 · vp P only gains a monetary advantage if the benefit outweighs the costs where: f1 · va · vp/(1 − va) > f2 · vp f1 > f2/(va/(1 − va)) Thus if P and Alice’s interactions were limited such that they only get to propose one block each, it’s clear that P doesn’t benefit overall unless the proposed block contains a much larger sum of fees f1 in reward than what Alice’s later block will contain, f2, assuming va 1. Even if Alice’s voting power is large, she could divide her stake amongst multiple smaller accounts. In the case where P and Alice aren’t limited to propose one block each, P and Alice might exclude each other’s signatures indefinitely. In this case, P’s expected benefit on each block is: E[fees] · va · vp · vp whereas P’s expected detriment on each block is: E[fees] · vp · va No matter the amount of voting power, no two validators benefit by excluding each other’s signatures indefinitely. Intuitively, this is because the other validators gain more when two validators exclude each other

There are 2 types of transactions, taxed and untaxed. The taxed transaction is the subscribe transactions when a referral is made. 10 percent is sent the founders reward vested over 2 years (Siraj and the original development team). 15 percent is sent to the treasury that the community can decide how the funds are allocated. Untaxed transaction is any send/receive between members to encourage exchange.

Oracles

An oracle, in the context of blockchains and smart contracts, is an agent that finds and verifies real-world occurrences and submits this information to a blockchain to be used by smart contracts.

Smart contracts contain value and only unlock that value if certain predefined conditions are met. When a particular value is reached, the smart contract changes its state and executes the programmatically predefined algorithms, automatically triggering an event on the blockchain. The primary task of oracles is to provide these values to the smart contract in a secure and trusted manner.

Blockchains cannot access data outside their network. An oracle is a data feed – provided by third party service – designed for use in smart contracts on the blockchain. Oracles provide external data and trigger smart contract

executions when predefined conditions meet. Such condition could be any data like weather temperature, successful payment, price fluctuations, etc.
Oracles are part of multi-signature contracts where for example the original trustees sign a contract for future release of funds only if certain conditions are met. Before any funds get released an oracle has to sign the smart contract as well.

Our oracle acts as a notary; it notarizes data in the external world and puts it into blockchain readable format. In our case, we're using an oracle that is a simple web app run and maintained by Siraj since he has no incentive to game the oracle. One way to earn Sirajcoin is to refer people to the network. The flow is simple.

1. Community Member gives their referral link to a friend i.e sirajcoin.com/public_key
2. Friend clicks the link and subscribes to Siraj's Youtube channel
3. Community Member gains SirajCoin since the friend subscribed

The web app that acts as the intermediary between calling the Youtube API for a new subscription and the SirajCoin blockchain is the oracle. The community has the public key of the oracle and can govern what the oracle is allowed to do i.e grant coins. It is publicly verifiable.

Conclusion

We proposed peer to peer currency could enable a community as fuel to engage with Siraj directly by spending it on his attention via hourly meetings and video collaborations. It will also unlock access to exclusive content, merchandise, and events. Importantly, the coin enables a novel governance structure in the community that gives members voting powers on how to best spend community funds. These funds can be used to fund AI research initiatives, elect community representatives, and empower other creators that spread AI knowledge. This cryptocurrency is the first step in establishing The School of AI, a worldwide network of programming wizards dedicated to solving AI through research and education.