Dexm: A Simple and Usable blockchain for everyone

Antonio Groza, Sebastien Biollo, Ruggero Valli
Email: {*antonio,seba,ruggero*}@*dexm.space*
https://dexm.space

Abstract. In this paper we describe the model of a cryptocurrency that can be used as a platform to build dApps easily, in which smart contracts are used as a replacement for a traditional backend. The protocol was designed in such a way to allow compatibility with web browsers and IoT devices. This all results in a blockchain that can potentially handle infinite events per second if scaled properly and that could allow for blockchain to reach mass adoption.

1 Introduction

While developing Dexm we tried to adhere to the following set of principles:

- **Simplicity:** While deciding on the internals of our blockchain we tried to use as many existing solutions as possible and modifying them at times to fit our needs. This was crucial in developing our product as quickly as we did because we didn't have to reinvent everything. An example of this rule in our work is our network stack.
- **Risk:** If we have to make a decision we try to take as little risks as possible, but if the benefit for the network is major we will take that risk. An example of this principle in our work is Sharding.
- **Generality:** Instead of moving everything down to the protocol level we try to abstract as much of it as possible to smart contracts. However we don't do this for network functions, like consensus and sharding. We believe this more complex tasks should be separated from the chain.
- **Compatibility:** This is an extension of the Simplicity point. Because our protocol is so simple it allowed us to easily port it to other devices.

2 Networking

In this section we describe how the networking of our blockchain is built. This is crucial to **retro compatibility** with browsers, and very different from existing solutions because of our very different goals. The whole stack is built with **simplicity** and **compatibility** in mind. We think this is a good set of principles because we can implement light clients in browsers, **IoT devices** and make many more use cases possible.

2.1 Message marshaling

A key part of any blockchain's networking stack is how messages are encoded and sent from node to node. Here instead of coming up with our own encoding system like Ethereum's RLP or just using C structs like Bitcoin we decided to use **Google's Protobuf**.
Protobuf works by defining the structure of a message, then compiling that structure to some code that checks the structure of the message and encodes it to a standard format. This allows us to encode and decode messages in **many of different programming languages**, while keeping complexity down.

2.2 Network Segmentation

Because of how some of our scaling mechanisms work we had the necessity to deliver some messages only **to a subset of the network**, while others to everyone. To do this we added a network primitive called **interests**. In our network a client can signal a string that he's interested in, and he will receive every message that has that string as a **tag**. This is currently used only for sharding, but because of its intentionally generic design it could be used as a communication system for dApps.

2.3 WebSockets

WebSockets are the transport mechanism used for communication between nodes. WebSocket is a transport protocol compatible with most browsers and it is **TCP** based. It **supports SSL and proxies** out of the box. This makes our network stack is **compatible with 93% of web browsers** and most embedded devices that can make HTTP requests. We think this is crucial for mass adoption as many more people will be able to access the platform.

3 Blockchain

3.1 Introduction

Traditional ways of processing transactions on the internet rely on centralized entities to save the balance of each person using the service. In this section we describe a method to process **peer-to-peer transactions** without relying on a centralized entity. Traditionally the solution relied on Proof of Work as described by Nakamoto[1], but this method has proven to be very hard on the environment and has a limited speed because of it's reliance on hard computing problems.
Instead of using Proof of Work, which requires a lot of processing power to be used, we use **Proof of Stake (PoS)** that is a type of algorithm by which a blockchain network aims to achieve distributed consensus, an example of this is Casper's FFG[2]. Unfortunately their approach didn't fit our blockchain,so we had to come up with our own consensus algorithm that we called **Khoros**.

3.2 Assumptions

The only assumption we had while developing Dexm is that **2/3 of stake in the network is held by honest nodes** and that the remaining dishonest third is organized and tries to maximize their profit.

3.3 Khoros Consensus

In Khoros, a user deposits a set amount of money (defined as **stake**) to become a **validator**. New validators must be able to join, and existing validators must be able to leave. To accomplish this, we define the dynasty of a validator. The **dynasty** of a validator is a set of consecutive blocks. The start of the dynasty is simply the block in which a user becomes a validator + 200. This extra time is given to validators to download the chain they will take part in.
To leave the network a validator must send a **withdraw** message. The block number + 200 in which the withdraw message is included is defined as end dynasty. At the start of the end dynasty, the validators deposit is locked for a long period of time (6 month); if during the withdrawal delay the validator violates any rules, the deposit is slashed. This is to avoid long range revision attacks, as explained in [2].

3.3.1 Canonical Chain Choice

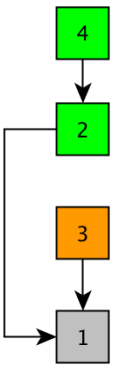

Fig. 1. *In this figure we show the behaviour of an honest validator. The number inside the block represents the order in which the honest validators received them. The green blocks followed both rules correctly*

In traditional Proof of Work blockchains choosing a chain that everyone agrees on is a solved problem. Unfortunately most of existing solutions for Proof of Stake chains rely on an elected leader to pick the canonical chain. We didn't like this approach as it didn't comply with our objective of being **totally decentralized**, so we came up with a **set of rules** honest nodes follow to address this problem:

- Always select the chain with the most blocks
- If multiple chains are the same length, build on top of the one you received first

If a block isn't included in the canonical chain all the transactions contained within it will be ignored and the validator will receive no reward. This gives an incentive for block generators to send their blocks as soon as possible. With this simplified model the only time a honest validator will not receive a reward is if two dishonest validators get picked in a row. The formula to calculate what percentage of blocks will be affected by this issue is the following:

$$z = \frac{1}{3}^2 \approx 11\% \tag{1}$$

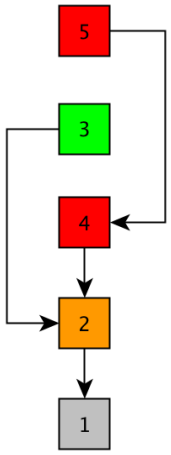

Fig. 2. *In this figure we show the behaviour of two dishonest colluding to take away a honest validator's reward*

To address the issue we pick n validators alongside the block proposers. The role of these validators is publishing a vote on what block they think the chain should be built upon and broadcast that to the network. If a validator makes more than one vote per block proposal his stake will be slashed. We picked n to be 11, this way the chance of what described in 2 to about 0.137%. We think this number is low enough to be negligible, but in case we are wrong we can keep increasing n till we are satisfied.

$$z = \frac{1}{3}^{\frac{n-1}{2}} \approx 0.137\% \tag{2}$$

Once the chosen validator gets the half plus one votes(the votes are stake based, it means that more stake you have, more high is you vote) he needs to append them in the block header for his block to be valid. Once this has done the validators that voted and the block proposer will get a reward. This gives an incentive for Validators to send their votes quickly, because if the block proposer has already reached a majority then the other votes will recive no reward. In case a majority cannot be reached then the block cannot be generated. Once every 100 blocks all the validators vote on which fork they think is right, and once 2/3 of the voters agree all the other blocks are purged from storage.

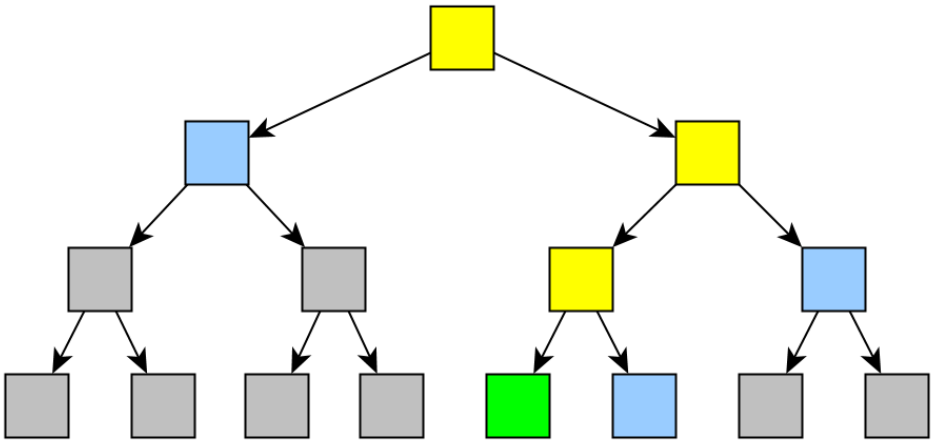

Fig. 3. *Merkle Proof explained: The leaves (bottom nodes) contain receipts of the transactions. Every other node contains the hash of the two nodes below. In order to prove that the green node is inside the tree, it is sufficient to send the blue nodes, along with the root.*

3.4 Scalability

Current blockchains are often bound by the amount of events/s they can handle, as every single peer in the network has to receive and process everything that happens. To address this and achieve over 4080x improvement in transactions per second we implemented sharding. **Sharding** is a mechanism in which **the network is split** into many separate blockchains (called from this point on **shards**) that can communicate with each other. Hereby is described the mechanism by which this is achieved.

3.4.1 Merkle Trees

A Merkle tree is a type of **binary tree**, composed of a set of nodes with a large number of leaf nodes at the bottom of the tree containing the underlying data, a set of intermediate nodes where each node is the hash of its two children, and finally a single root node, also formed from the hash of its two children, representing the ”top” of the tree, called Merkle Root.
Every block is a Merkle Tree, where every leaf contains a **receipt**, which is the ”summary” of a transaction. Merkle Trees can be used to prove a transaction is inside a block, without having to download the whole block, but only a subset of the nodes of the Merkle Tree, as shown in figure (3).

This is essential if you don’t want to store every shard, but only yours.
In the network there are two different types of transactions:

- **Intra-Shard Transactions**: In which the recipient of the transaction is inside your same shard
- **Inter-Shard Transactions**: In which the recipient of the transaction is in a different shard.

In the case of an Inter-Shard Transaction a Merkle proof needs to be generated and sent to the shard of the recipient. This will allow the other shard to verify:

- the inclusion of the transaction inside a block
- that the recipient is inside their shard
- that a proof for that transaction has not already been received

After that is done all the nodes in the shard save the hash of the transaction as ”burned” and credit the recipient with the value of the transaction.

3.4.2 Beacon Chain

The key function of the Beacon Chain is to manage the proof-of-stake protocol for itself and all of the shard chains. There are a number of aspects to this: a major part of the work of the Beacon Chain is maintaining the set of validators for every shard; nominating the chosen block proposer for each shard at each step; organising validators into committees to vote on the proposed blocks; applying the consensus rules; applying rewards and penalties to validators.

- **Block proposers:** When someone becomes a validator he can participate in the network, this means that he can generate blocks. This happen every 5 second (called round), in which a validator is chosen via **random selection, proportional to the validator's stake**
- **Providing Randomness:** Good quality randomness is really hard to generate in blockchain systems, but a key requirement of the proof-of-stake protocol is a source of randomness that is distributed, verifiable, unpredictable, and (reasonably) unbiasable. The current approach to random number generation is to use the previous 100 hash blocks, to **prevent someone from calculating when he will be chosen** and in which shard he will be placed. The shard chosen for the validator will be valid for 100000 rounds (around 1 day).
- **Rewards and penalties:** Validators receive rewards for behaving well and playing their part: this is the incentive for participating. But if validators break the rules, they are penalised by having some of their stake deposit reduced (slashed), and being ejected from the system. There is also a small penalty for being absent (not showing up to vote on blocks). The reasons for this are subtle, and are about allowing the system to keep processing even if huge numbers of validators go offline, such as in the event of a catastrophe. If a validators deposit falls below half of his stake, the Beacon Chain will remove it from the validator set.

4 Smart Contracts

One of the most promising uses for blockchains currently is smart contracts. A smart contract is a tiny program stored on the blockchain which has the ability to interact with digital assets according to an arbitrary set of pre-specified rules.

4.1 Design Choices

Some existing blockchains tried to use their own virtual machines to execute contracts, like Ethereum's EVM. This has the major disadvantage of coming up with new programming languages and compilers. While ETH has been trying to address the issue with eWASM[3], a modified version of the WebAssembly[4] standard, we are following a similar route although with more APIs and features to address security issues and common programmer issues. To move assets or interact with other contracts the programmer just needs to call a set of predefined APIs.

4.2 Peculiar Features

We have implemented a number of features that might help address some common use cases and problems in smart contract development.

- **Rate limiting**: To limit the potential damage done by a hack to the smart contract we define a set of APIs to limit the amount of assets that can be moved out of the contract in a set amount of time.
- **Contract Patches**: A common issue while developing smart contracts is modifying the code once it has been deployed to the network. Currently this is done on ETH using proxy contracts, small smart contracts which simply call another contract. To address the complexity of this process we added an API in which a contract can specify the hash of a binary patch that will be applied to the specified contract.
- **Contract Locking**: If a security issue takes place this API allows the contract to be called only by a select list of wallets. This could be used in conjunction with Contract Patches to provide a centralized way of fixing security issues.
- **K/V Database**: Very often there is the need for smart contracts to save data, and with these APIs we provide a very simple interface for users to store data and make it accessible to other contracts and our JS library.

5 Testnet

To test the performance and security of our design we implemented what was described in this paper and tested it's performance using 1000 virtual machines running on a Docker Swarm with a simulated latency of 30ms. We think that this is a realistic scenario. Our tests was run with only 10 shards enabled and we managed to achieve an average of 5500 transactions per second. Although this number is high compared to other blockchains our design supports scaling to 255 shards, which in turn would allow us to support around 140000 transactions per second assuming perfect conditions. This resulted in around 0.6MB/S being written to each node's SSD each second. If this were to be our constant rate of growth it would be unsustainable, but if we modify our code slightly we could support up to 4096 shards, reducing the used space by 16x, back to an usable rate.

6 Usability

A core goal when designing Dexm was to make it more accessible to more people. The result of our effort is a network that is fully compatible with current browsers. Users can

interact with smart contracts without installing plugins or apps, they just have to go to an url. Unfortunately because using non-ssl WebSockets on SSL pages is not supported we plan to develop a process where a node, after enough up time on the network, will be given an unique subdomain on our domain, along with an SSL certificate. This relies on a centralized entity for SSL support but this is already the case for all SSL requests as you trust all the Certificate Authorities present within your computer.

7 Conclusion

In this paper we have proposed a cryptocurrency that is designed both for users and developers, guarantees usability, and makes possible for all developers to make dApps. Also thanks to Khoros and PoS we have manged to get a strong decentralization and a high level of performance in the network.

Future Work. We think that some improvement needed in the sharding mechanism to support private enterprise chains. We are also missing the possibility to pay for transactions by using tokens, this mechanism would make transactions in which a third party pays the fees possible.

References

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. `https://bitcoin.org/bitcoin.pdf`

[2] Vitalik Buterin and Virgil Griffith. *Casper the Friendly Finality Gadget*. `https://arxiv.org/pdf/1710.09437.pdf`.

[3] Ethereum flavored WebAssembly *ewasm Design Overview and Specification*, `https://github.com/ewasm/design`