Maxima - A Fast Fearless Secure Sharded Block-Chain

Maximilian E. Chamberlin
MA (Oxford) M.Phil (Cambridge)
Maxima Organisation
mec@Maxima.org

Abstract

Recently, high demand and limited scalability have increased the average transaction times and fees in popular cryptocurrencies, yielding an unsatisfactory experience. Here we introduce Maxima a cryptocurrency with a sharded block-chain, one where the network is divided into partitions called shard which maintain their own blocks.

In Maxima, validator committees are sampled for each shard. Validators vote on the availability of blocks to the network, and are shuffled off shards very quickly. Once a committee has determined if a block is accessible to nodes on a shard, nodes on that shard execute the transactions and verify the blocks validity- producing a succint fraud proof if the block is invalid. Cross shard transactions are managed using tools from distributed systems, like locks and yanks.

Maxima is a globally distributed computer that is secure under an honest majority assumption, and puts in place mechanisms to prevent adaptive adversaries from corrupting the network. With 20 shards, Maxima is projected to process over 20,000 transactions per second.

Part I

Introduction

1 Motivation and Outline - Core Sharding

In Bitcoin, blocks are added to the block-chain approximately every 10 minutes, at a relatively fixed pace that results in a TPS of 5-7. Simple solutions to expanding the capacity of the block-chain have found natural limits to their efficacy: enlarging blocks runs up against user bandwidth limits; hastening the rate at which blocks are mined increases orphaning.

A more promising solution to the problem of scaling is to run separate chains that process transactions within partitions of the network, called shards. The key challenges are: (1) how to manage the communication between the different shards, a problem which is closely related to ensuring the atomicity of transactions in a distributed system and (2) how to run these many separate block-chains in a way that does not compromise on security, since each shard-chain will only have a fraction of the mining/validation power of the network.

In this paper, we propose mechanisms to ensure the atomicity of transactions and to ensure that the security of separate chains is not compromised. We adopt an honest majority assumption for the network, assuming fewer than a fraction f of 0.25 nodes are Byzantine. Note $f = 0.25$ is an

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

arbitrary constant bounded below 1/3 to ensure good constants. We also put in place safe-guards against adaptive adversaries who may bribe network participants.

With this in mind, our protocol can be described below. Validator committees are sampled for each shard. Sampling validators randomly means that with a sufficient number (400) we can ensure an honest majority of 2/3 of per shard almost surely.

Validators then vote on the availability of blocks to the network, and are shuffled off shards very quickly. This fast shuffling is to ensure that an adaptive adversary does not have the time to find and corrupt the validators of each shard. Because of this fast shuffling, the work that a validator can perform must be very minimal. Validators check that the data corresponding to the Merkle root of a block is accessible to a shard, so that the members of a shard can execute those transactions. If there is a dispute about transaction execution, validators also resolve disputes by considering succinct proofs of invalid execution. Cross shard transactions are managed using tools from distributed systems, like yanks, to ensure that transactions can be executed atomically.

1.1 Protocol Structure

The construction of our protocol depends on the following, relatively independent components, which can be subdivided into three themes: voting schemes; data creation and concurrency controls.

1. **Motivation and Outline - Sharding**
   (a) Protocol Structure
2. Consensus:
   (a) Voting Scheme
   (b) Validator registration and Shuffling
3. Concurrency Controls
   (a) Locking State for Cross Shard Transactions
   (b) Stateless Clients
4. Minimal Sharding - No Smart Contracts
   (a) Summary - a simple protocol to transfer currency.
5. **Motivation and Outline - Advanced Sharding**
6. Separating Validity Checking from Availability Checking
   (a) Erasure Coding
   (b) Fraud Proofs
   (c) Summary
7. Virtual Machine
   (a) EWASM VM
   (b) Transpilers and the future of block-chain
8. Solving the State Problem
   (a) **Introducing rental fees to the block-chain**
   (b) **Building on top of IPFS**

2 Achieving Fork-Free Consensus

2.1 Consensus Scheme - Modified Dfinity

One critique of proof of stake are that the economies of scale eventually lead to a few centralised stakers. This led to the adoption of delegated proof of stake. But with centralisation there is a good chance an actor can be bribed. More recently, consensus models have emerged that randomly sample participants from a global pool. Both Dfinity and Zilliqa have byzantine fault tolerant like algorithms that have O(n) communication complexity, enabling the scaling of ths algorithm to the order of 400 nodes.

Weighing up the benefits and disadvantages of our approach

There may be criticisms in that marginal incentives are not properly aligned in these schemes, compared with slashing proof of stake schemes like casper. However, as both rely on an honest majority assumption we deem them both as safe as each other. Slashing may also be introduced probabilistically to the Dfinity scheme as we will later show. Though we could also adopt the Zilliqa consensus algorithm, we adopt the Dfinity model for the simplicity of its algorithm which we outline here.

Algorithm in depth

In Dfinity, the network is grouped into threshold relay groups. These groups are created through random sampling and have a size of about 400. A random number is generated which selects the first relay group. These relay groups then sign on the random number to produce another random number and select the next relay group. The random number also selects a number of block proposers, and the blocks if valid may be signed by the current threshold relay group. This algorithm (with block proposers) and notaries who sign on blocks have similarities with PBFT schemes. Whereas PBFT commits only one block and forges consensus after a round fo prepare and commits, the Dfinity scheme may commit more than one block. Dfinity achieves its high speed and short block times exactly because notarization is not full consensus. However, notarization can be seen as optimistic consensus because it will frequently be the case that only one block gets notarized. Hence, whenever the broadcast network functions normally a transaction is final in the Dfinity consensus after two notarized confirmations plus a network traversal time. The notarization step makes it impossible for the adversary to build and sustain a chain of linked, notarized blocks in secret. For this reason, Dfinity does not suffer from the selfish mining attack [4] or the nothing-at-stake problem.

Advantages

Fork-free protocols typically use BFT style algorithms. These are systems where a vote for a block consists of a prepare followed by a commit. They are fork-free, which is essential for cross-shard communication. Traditionally, PBFT couldn't scale to the size of a network. So instead, we:

Subsample from the entire population so many validators are voting but not all (in Zilliqa, they use X). If say 400 participants are voting and we assume that f<25% are byzantine. We can show that the probability more than 2/3 of those voting are also byzantine is less than the number of atoms in the universe.

What are nodes voting on?

However, a key question here is: what precisely are the nodes voting on? A simple scheme would be for nodes to run the transactions in a block and verify that they are valid. If more than half the nodes agree that a block is valid it is finalised during notarisation. However, this presents problems in the context of sharding: the validators do not have the storage capacity or bandwidth to maintain the state of all shards, and yet they must be shuffled quickly between shards. One solution to this problem is to store the witness data with the transaction in a block, entailing that a block is "self-authenticating", and the witness data need not be downloaded.

Another solution is for validators to just check that data is available, rather than the correctness of a block. If a block is unavailable some honest actor in the network may respond with a fraud proof and take the faulty block creator's deposit. Ultimately, we offload the actual checking of whether a block is correct or not to the network. Just checking data availability has a second advantage: the committee can share the workload of the task. Below we give an outline of how this may be made possible:

Blocks are expanded with an erasure code to a size of 400, say. The erasure code is a redundant encoding scheme which ensures that if 1/6 of the code is available, the original block can be reconstructed by honest nodes within the network. Then each of the 400 committee members checks for 3 or 4 pieces of data. By the probabilistic sampling, we can ensure that 2/3 of the committee is honest. We also require a threshold signature of 1/2 for a block to be notarised for its validitiy.

If a block is available, then 2/3 of the committee will vote and so we have no issues.

If a block is unavailable, then fewer than 1/6 of honest members can have pieces of data. And so even with these votes, and 1/3 of the dishonest majority, they cannot reach the 50% threshold required. We devote a further section to data availability proofs.

Slashing Conditions

For all those who voted that a block is available, we require a random section of the group to reveal the data that they got from the network. This will disincentivise some from creating a block. Creating a slashing condition going the other way is more tricky, since we don't know if a block is unavailable or a a validator is just witholding data. However, liveness is a less serious issue, since the beacon group can always be changed.

Fall-back Mechanism

One pertinent question is what should happen if a Beacon group is corrupted? We use fast shuffling of validators to ensure that this is near impossible.

2.2 Consensus Scheme - Validator Registration and Shuffling

Here , we just say a little more about the actual mechanisms and concrete data structures underpinning the validator sampling. Validators have their own shard, which they register a deposit to. This deposit is a non-trivial sum. **TODO - copy from DFINITY**

3 Cross Shard Communication Through Message Passing

Message Passing is the facvoured approach for concurrency control in distributed systems.

We use a message passing model based on Erlang, which has a simple implementation of CSP (Communicating Sequential Processes) where soft-threads (threads) have each a single mailbox that can receive messages. The soft-thread can pop off and react to messages in its mailbox, also supporting some form of pattern matching for prioritization.

Contracts are like the threads of execution and we have a single mailbox per contract. Conceptually message passing is simple, it is no different from running the transaction, but with updated data, but with a special message type to distinguish it from ordinary data. Howeever, the only difference will be in the accumulation of messages. Validators bundle up the messages and feed them in as data to the contract, which can then pop messages off and react to them. Diagramatically, we could envidage something like this.

This introduces few overheads to the protocol, since popping off is just a data operation, which can be defined in the main-loop of any contract.

	S_1	S_2	S_3
t_0	M_1 to C_1 on S_3 on chain Sent to S_2 by val		
t_2		M_2 to C_1 on S_3 on chain Sent to S_2 by val	
t_3 t_4			Actor bundles M_1 and M_2 $[M_2, M_1]$ validated against Mroot C_1 reads $[M_2, M_1]$

We now talk about guarantees in our message passing model:

Availability:

A pertinent question arises: **what mechanism ensures that a messag**e on one shard makes its way over to a message on another shard? We charge the validators on Shard1 and Shard2, with disseminating the information to the third shard. Since there are approximately 400 validators per shard, this large number ensures that with very high likelihood $1 - (\frac{1}{3})^{400}$, that a message will get through. Certainly a message will get through eventually, if we take it that shards might be DOS'd. The message is sent with a transaction fee, and the transaction fee is split between block miners on both shards. In fact shards will periodically relay the message until it is included in a shard.

At Once:

Each message contains a signed nonce, which ensures that messages cannot be replayed.

Message Ordering:

Messages sent directly from one shard to another will not be received out-of-order.

Shard S1 sends messages M1, M2, M3 to S2

Actor S3 sends messages M4, M5, M6 to S2

However, we do not provide guarantees about the ordering of messages between shards.

Furthermore, specific concurrency control mechanisms may be built ontop of message passing systems. For instance locking schemes and other such things.

Other Concurrency Management

A question that emerges is what if the transaction fees are too low and it takes a long time for a message to be included on another shard? More broadly, should we be providing guarantees on message passing? We have provided a robust system, but we adopt the Erlang philosophy that we make the fallibility of communication explicit through message passing, and do not try to provide a leaky abstraction. Instead users can write implementations that provide guarantees on message delivery, and delegate these to higher level protocols. This is a model that has been used with great success in Erlang and requires the users to design their applications around it. You can read more about this approach in the **Erlang documentation (section 10.9 and 10.10)** and Akka.

Another angle on this issue is that by providing only basic guarantees those use cases which do not need stronger reliability do not pay the cost of their implementation; it is always possible to add stronger reliability on top of basic ones

On top of these other concurrency controls can be implemented in smart contracts, for instance various kinds of locking scheme.

The specific case of sending money?

	S_1	S_2	S_3
t_0	M_1 to C_1 on S_3 on chain Sent to S_2 by val; deduct		
t_3 t_4			Actor bundles M_1 and M_2 $[M_2, M_1]$ validated against Mroot C_1 reads $[M_2, M_1]$

4 Minimal Sharding - No Smart Contracts

For each shard, the threshold relay assigns a different committee group to the shard. Each committee votes of the current block,checking for the validitiy of the transactions. Initially, we will only support money being sent from one shard to another. However, we will later add virtual machine functionality. A question arises: what if a beacon group stalls because it is DOS'd. After a few minutes, if no relay group has been selected, two relay groups can vote to correct a shard.

5 Advanced Sharding - Motivation and Introduction

We have outlined a simple protocol to achieve sharded message calls. However, we want to position ourselves at the forefront of technology. Key to this is understanding the direction

Virtual machines and Interoperability.

A sharded chain allows one to run the consensus logic of many chains. For instance you could imagine one chain running bitcoin transactionson one chain and the transactions of another coin on another. Key to this would be cross compilation. So, if we had a WASM virtual machine for our transactions, all that would be needed would be to find a way to transpile code from one virtual machine to another. Then one could have all the code written on EThereum . Transpilation is a well studied topic, and has been used to convert C++ to Javascript etc.

The language of the WEB

Having a wasm virtual machine has another benefit in that it is becoming the de facto standard for the web. This means that any code that could be executed in the browser, something that is key to hosting websites could be run on our block-chain. This would need to be coupled with efficient storage mechanisms. Currently on Ethereum the main cost, is in storing data. This is because Ethereum is getting to be at capacity in terms of the blocks that each node must sync on when toring data. In sharding, this removes the storage problem by a huge constant factor.

Efficient Storage

Apart from sharding, one idea to improve storage is to move to a model where there is an efficient market for storing data. Nodes Store When you want to store data an auction is done.

6 Storage

Suppose that we use the state transition lingo, STF(S, B) -> S', where S and S' are states, B is a block (or it could be a transaction T), and STF is the state transition function. Then, we can transform: S -> the state root of S (ie. the 32 byte root hash of the Merkle Patricia tree containing S) B -> (B, W), where W is a "witness" - a set of Merkle branches proving the values of all data that the execution of B accesses STF -> STF', which takes as input a state root and a block-plus-witness, uses the witness as a "database" any time the execution of the block needs to read any accounts, storage keys or other state data [exiting with an error if the witness does not contain some piece of data that is being asked for], and outputs the new state root. That is, full nodes would only store state roots, and it would be miners' responsibility to package Merkle branches ("witnesses") along with the blocks, and full nodes would download and verify these expanded blocks. It's entirely possible for stateless full nodes and regular full nodes to exist alongside each other in a network; you could have translator nodes that take a block B, attach the required witness, and broadcast (B, W) on a different network protocol that stateless nodes live on; if a miner mines a block on this stateless network, then the witness can simply be stripped off, and the block rebroadcasted on the regular network.

The simplest way to conceive the witness in a real protocol is to view it as an RLP-encoded list of objects, which could then be parsed by the client into a {sha3(x): x} key-value map; this map can then simply be plugged into an existing ethereum implementation as a "database".

One limitation of the above idea being applied to Ethereum as it exists today is that it would still require miners to be state-storing full nodes. To solve this, we put the witness outside the signed data in the transaction, and allow the miner that includes the transaction to adjust the witness as needed before including the transaction. If miners maintain a policy of holding onto all new state tree nodes that were created in, say, the last 24 hours, then they will necessarily have all the needed info to update the Merkle branches for any transactions published in the last 24 hours.

Miners and full nodes in general no longer need to store any state. This makes "fast syncing" much much faster (potentially a few seconds). All of the thorny questions about state storage economics that lead to the need for designs like rent (eg. https://github.com/ethereum/EIPs/issues/35 18 http://github.com/ethereum/EIPs/issues/87 14 http://github.com/ethereum/EIPs/issues/88 11) and even the current complex SSTORE cost/refund scheme disappear, and blockchain economics can focus purely on pricing bandwidth and computation, a much simpler problem) Even for state-storing clients, the account lists allow clients to pre-fetch storage data from disk, possibly in parallel, greatly reducing their vulnerability to DoS attacks. In a sharded blockchain, security is increased by reshuffling clients between shards frequently; the more quickly clients are reshuffled, the more adaptive the adversaries that the scheme is secure against in a BFT model. In a stateless client, this cost drops to zero, allowing clients to be reshuffled between every single block that they create. One problem that this introduces is: who does store state?

Any new state trie object that gets created or touched gets by default stored by all full nodes for 3 months. This will likely be around 2.5 GB, and this is like "welfare storage" that is provided by the network on a voluntary basis. We know that this level of service definitely can be provided on a volunteer basis, as the current light client infrastructure already depends on altruism. After 3 months, clients can forget randomly, so that for example a state trie object that was last touched 12 months ago would still be stored by 25% of nodes, and an object last touched 60 months ago would still be stored by 5% of nodes. Clients can try to ask for these objects using the regular light client protocol.

Clients that wish to ensure availability of specific pieces of data much longer can do so with payments in state channels, similar to file coing. A client can set up channels with paid archival nodes, and make a conditional payment in the channel of the form “I give up $0.0001, and by default this payment is gone forever. However, if you later provide an object with hash H, and I sign off on it, then that $0.0001 instead goes to you”. This would signal a credible commitment to being possibly willing to unlock those funds for that object in the future, and archival nodes could enter many millions of such arrangements and wait for data requests to appear and become an income stream. We expect dapp developers to get their users to randomly store some portion of storage keys specifically related to their dapp in browser localstorage. This could even deliberately be made easy to do in the web3 API. In practice, we expect the number of “archival nodes” that simply store everything forever to continue to be high enough to serve the network until the total state size exceeds ~1-10 terabytes after the introduction of sharding, so the above may not even be needed.

Links discussing related ideas:

Witnesses: BLS accumulator =>

7 Separating Validity checking from State Execution

- Key to security is fast shuffling. This means that the excutors can’t do a lot.
- They attest to availability of blocks, fixed if others agree not so if others disagree.

8 Execution disputes and Validity

- Block creaters order and create transactions. The creators are sampled randomly from afixed set.
- They need a way of determining whether a transaction can pay them the gas to run the TX, and also if the ordering of transactions is correct.
- They need to do this without having access to all the state.
- Light clients can store all this information, and transactions need to include access lists of data that they need.
- Zero knowledge proofs that I have the gas needed.
- Q: how can a transaction determine if two things are blocking. Just create a merkle tree that tracks these things.
- We can use a multi-dimensional eraure code to determine quickly if some data is available.
- Key to security is fast shuffling. This means that the excutors can’t do a lot. They can’t maintain state on each shard.

Blocks are created within each shard, and the merkle root is signed on and shared with other shards. The question we now have is this?

(1) How can we ensure that all of the data contained within a shard is available to all members of the shard?

(2) Given that data isavailable, this means anyone can report flaws.

The question is how can we prove that a block is available?

One could require a majority of half the sampled validators perform availability checks and then sign off on the availability of a root.

Question is how can we feed in the right marginal incentives?

- Have rewards for attesting yes, but with penalties if they don’t have proof they checked for data availability. This means nodes are incentivised to say yes if they do have the data, but won’t declare yes if they don’t and know others won’t check them. small reward for saying no.
- randomly check availables for a reveal, with sufficinelty high penalty that makes lying unprofitable. Also allow challenges for those who know ing wrong. For those who don’t

require a random reveal with sufficiently high penalties. => say that a person has 10 blocks to do so before being slashed.

- unavailable|available are losing their rewards

If they are aware many other people do not have access to the data, they may vote yes anyway.

However, we don't want validators to have to actualy execute the state. This would take too long.

1. How can we resolve this issue? Ethereum uses Truebit, which takes the form of an interactive verification game. The main issue with truebit is that it requires multiple rounds of verification and thus can result in low latency in cases where there is a dispute.
2. The approach that we take is to erasure encode the trace of the execution. We use a 3d erasure encoding to ensure that the fraud proofs are of a size cube-root(n).
3. Again using sampling techniques, we can assign validators to each transaction.

Self-Authenticating Erasure Codes

We want validators to be able to guarantee that honest nodes can access data on the network, or stated more succinctly that **data is available**. A simple way for validators to check that data is available is to download a whole block. So, in the current scheme we would sample a set of validators - say 400, which is enough to guarantee an honest majority of participants. Then all of these shards would attempt to download the data from the network, and finaly take a vote between themselves on whether the data is available.

However, to increase efficiency, we would like to use a scheme where each validator need not download the entire block. If blocks were really large, say GB sized, which would be ideal for scalability, it may be impractical to have validators enter a shard and download multiple blocks to verify availability.

We can avoid this, if we take a sampling approach. However, validators will not sample directly from the block itself, but instead will sample from an erasure code.

Why do we want validators to check that data is available for a block? If the data for a block is available to the network, this means that honest executors on a shard can check to see if there are any faults with it and produce succint fraud proofs, O(1) sized proofs that a block contains invalid data.

What this ultimately means is that executors on a shord can check to see if

We don't actually want the validators to perform the executions themselves because doing so would take too long for large blocks of for transactions that take a long tiem to execute.

- N - the number of verifiers
- P - the number of checks made by each client
- Denot

Collectively N verifiers want to check a block of size M is available. They wish to do so without

9 Construction

The below is inefficient. If we have a row of root(m) values , we can recompute the entire row from that. To do a fraud proof, we still need to authenticate the column. What if we don't have a column value? So this still requires availability of some values to prove consistency, but we may not have those values.

9.1 Glossary

K - The expansion factor of each read solomon code

Let D_{ij}, F_{ij}, G_{ij} represent the data int he first, second and third squares respectively

x,y,z are respectively the bilinear accumulator constants for the committed values

First Square:

Let d_{ij} represent chunk i, j of the original data

- x_{ij} is the bilinear accumulator witness for the tuple (d_{ij}, i, j), that is $e[(d_{ij}, i, j), x_{ij}] = x$
- D_{ij} represent $(d_{ij}, i, j), x_{ij}$

Second Square:

- Let f_{ij} represent the j th evaluation point in the erasure code for row D_{i*}
- y_{ij} is the bilinear accumulator witness for the tuple (f_{ij}, i, j), that is $e[(f_{ij}, i, j), y_{ij}] = y$
- F_{ij} represent $(f_{ij}, i, j), y_{ij}$

Third Square:

- Let g_{ij} represent the i th evaluation point in the erasure code for column F_{*j}
- z_{ij} is the bilinear accumulator witness for the tuple (g_{ij}, i, j), that is $e[(g_{ij}, i, j), z_{ij}] = z$
- G_{ij} represent $(g_{ij}, i, j), z_{ij}$

An example is given below with K =2/3:

Definition: for an element $G_{ij} = (g_{ij}, i, j), z_{ij}$ to be **available**, an honest participant of the network must have access to it and it must be authenticated, so the following constraint holds: $e[(g_{ij}, i, j), z_{ij}] = z$.

$(d_{00}, 0, 0), x_{00}$	$(d_{01}, 0, 1), x_{01}$	$(f_{00}, 0, 0), y_{00}$	$(f_{01}, 0, 1), y_{01}$	$(f_{02}, 0, 2), y_{02}$
$(d_{10}, 1, 0), x_{10}$	$(d_{11}, 1, 1), x_{11}$	$(f_{10}, 1, 0), y_{10}$	$(f_{11}, 1, 1), y_{11}$	$(f_{12}, 1, 2), y_{12}$
		$(g_{00}, 0, 0), z_{00}$	$(g_{01}, 0, 1), z_{01}$	$(g_{02}, 0, 2), z_{02}$
		$(g_{10}, 1, 0), z_{10}$	$(g_{11}, 1, 1), z_{11}$	$(g_{12}, 1, 2), z_{12}$
		$(g_{20}, 2, 0), z_{20}$	$(g_{21}, 2, 1), z_{21}$	$(g_{22}, 2, 2), z_{22}$

9.2 Authentication/Availability Transitivity

If a set of values X in an erasure code are available, and can reproduce some data Y, then if y∈Y is not authenticated this can be proven in $O(\sqrt{m})$ steps. The authentication transitivity assumption is this : unless an $O(\sqrt{(m)})$ fraud proof is given by the network: then if X is available and produces Y, Y must be authenticated and thus available.

9.3 Proof of Uniqueness

Since all available elements are authenticated. If it is possible to construct more than one element in the $ijth$ index of any of D, F, G, then an O(1) proof is given by presenting something like:

$(d_{00}, 0, 0), x_{00}$, $(d^{'}_{00}, 0, 0), x^{'}_{00}$, where $d_{00} \neq d^{'}_{00}$

This means if two distinct items are unavailable a succinct fraud proof can be given.

9.4 Proof of Liveness

Given points 8.1, we can prove liveness.

Suppose no authentication fraud proof is given by the network, then:

If the data for a row D_{i*} is unavailable then: (by authentication transitivity)

#available elements in $F_{i*} < \sqrt{m}$ or equivalently:

#unavailable elements in $F_{i*} > (k-1)\sqrt{m}$. This implies:

#unavailable elements in $G > (k-1)^2 m$ or equivalently: (by authentication transitivity)

#available elements in $G < k^2m - (k-1)^2m = (2k-1)m$

Therefore a liveness check needs to ensure that more than $(2k-1)m$ elements of G are available with a high likelihood. If this be the case, we can reconstruct some dataset D.

Collaboratively Generated Erasure Codes

Collaborative generation, check two merkle roots.

$(d_{00}, 0, 0), x_{00}$ $(d_{10}, 1, 0), x_{10}$	$(d_{01}, 0, 1), x_{01}$ $(d_{11}, 1, 1), x_{11}$	$(f_{00}, 0, 0), y_{00}$ $(f_{10}, 1, 0), y_{10}$	$(f_{01}, 0, 1), y_{01}$ $(f_{11}, 1, 1), y_{11}$	$(f_{02}, 0, 2), y_{02}$ $(f_{12}, 1, 2), y_{12}$	
$x_{0*}, \bar{x}_{0*}$	$x_{1*}, \bar{x}_{1*}$	f	f	f	
		$(g_{00}, 0, 0), z_{00}$	$(g_{01}, 0, 1), z_{01}$	$(g_{02}, 0, 2), z_{02}$	$z_{0*}, \bar{z_{0*}}$
		$(g_{10}, 1, 0), z_{10}$	$(g_{11}, 1, 1), z_{11}$	$(g_{12}, 1, 2), z_{12}$	$z_{*1}, \bar{z_{*1}}$
		$(g_{20}, 2, 0), z_{20}$	$(g_{21}, 2, 1), z_{21}$	$(g_{22}, 2, 2), z_{22}$	$z_{*2}, \bar{z_{*2}}$

What are the implications to censoring of codes created by trust scores? Competition amongst many sellers with high trust rating, think amazon. So should not degenerate to a one proposer model. Anyone can build up trust. **So then partly depends on fee, could also require to rotate proposers.**

9.5 Another note, if the attackers bandwidth is absorbed he can't respond to all the requests, so would need just more than the attackers bandwidth, which will be much less than gigabytes..with gigabyte sized blocks... still a bit rickety.

9.6 Attacks against Dfinity

DKG for threshold groups registered on main chain.

New threshold group is selected every round using the randomness produced.

One idea would just be to select the next threshold group if after a period of a few blocks, the threshold group has not responded.

Practical: Not possible for an attacker to catch up with threshold relay... Therefore, can't get people in advance. What if he stalls a relay. Then we need a fall-back mechanism, then it's ok.

9.7 Other

The whole point of an erasure code is that:

(1) I can check all blocks for myself as long as the light client assumption holds

(2) Reduces the amount of checking for everyone and utilises a fraud proof mechanism

(3) without erasure codes we must rely on just consensus, which may not be enough for a sharded block-chain

(4)

Data Availability, State Execution and fast shuffling

Randomness:

Virtual Machine - Web Assembly?

(1) Transpilers ; (2)

The first question we should ask of the architecture is the nature of forks. Should a sharded-block-chain be fork-free?

Dfinity overview:

- Random number generation to select block maker and notarisation group
- Notarization proves that a block has been published at some time, based on a group vote.
- Notarisation is more secure than a single vote. => Finality after two rounds.
- **Kind of taking a hybrid approach pBft and block based consensus.**
- Good: **notarization in Dfinity is not primarily a validity guarantee but rather a timestamp plus a proof of publication'**
- **Good: almost fork-free; Good: still uses probabilistic consensus; Prevents long-range stake attacks**

Neo:

- Delegated Proof of Stake.
- Finality after a single round. Why based on voting.

Zilliqa:

- Sampling into shards.
- Honest majority assumotio + PBFT, much more robust.
- **Good: Uses random sampling so robust vopting groups;**
- **Bad: no slashable conditions.**

Ethereum:

- Splitting problems of availability from validity?
- Availability can be decided quickly.
- Validity decided by a kind of Trubit systle mechanism, which means validity can then be checked quickly.

Our Approach:

- Solve availability:
  - Block-chain based .. check the last 25 nodes for availability by downloading
  - dfinity based.. similar, perhaps more robust.
  - Proof based: Present a proof of data availability, i.e erasure coding a block and getting signatures from half of validators
  - Validators need only sign a hash, so the amount of work is constant* number of shards.
  - **Putting accountability into dfinity, all one needs is a numnber that gets revealed after the event.**
  - **Q: How do we put in place marginal rewards on voting**
  - **A1:** You sample a set of nodes to vote for availability, say 400; each round secret number per user, they need to compute a number based on this with their vote. ==> they have checked...
  - **a1**: Lying is just contradicti9ng the majority... study largest block reversions in bitcoin...
  - **want key shares to be revealed dfinity style.**
  - **A2: deposit into the chains to validate them, withdraw out when you can.**
- Solve validity
- truebit style execution
  - main disadvantage is the interactivity.
  - one can imagine erasure codign execution steps to find efficient fraud proofs
  - **Instead of a trubeit style verification game, one can just introduce hash checkpoints in compuitation. This reduces interactivity to 2 rounds, while keeping the amount of work done linear in the program size and**

- Ensure that consensus is fork-free, fast and robust

There is a new style file for papers submitted in 2016!

NIPS requires electronic submissions. The electronic submission site is

`https://cmt.research.microsoft.com/NIPS2016/`

Please read carefully the instructions below and follow them faithfully.

9.8 Style

Papers to be submitted to NIPS 2016 must be prepared according to the instructions presented here. Papers may only be up to eight pages long, including figures. Since 2009 an additional ninth page *containing only acknowledgments and/or cited references* is allowed. Papers that exceed nine pages will not be reviewed, or in any other way considered for presentation at the conference.

The margins in 2016 are the same as since 2007, which allow for $\sim$15% more words in the paper compared to earlier years.

Authors are required to use the NIPS LaTeX style files obtainable at the NIPS website as indicated below. Please make sure you use the current files and not previous versions. Tweaking the style files may be grounds for rejection.

9.9 Retrieval of style files

The style files for NIPS and other conference information are available on the World Wide Web at

`http://www.nips.cc/`

The file `nips_2016.pdf` contains these instructions and illustrates the various formatting requirements your NIPS paper must satisfy.

The only supported style file for NIPS 2016 is `nips_2016.sty`, rewritten for LaTeX 2_{ε}. **Previous style files for LaTeX 2.09, Microsoft Word, and RTF are no longer supported!**

The new LaTeX style file contains two optional arguments: `final`, which creates a camera-ready copy, and `nonatbib`, which will not load the `natbib` package for you in case of package clash.

At submission time, please omit the `final` option. This will anonymize your submission and add line numbers to aid review. Please do *not* refer to these line numbers in your paper as they will be removed during generation of camera-ready copies.

The file `nips_2016.tex` may be used as a “shell” for writing your paper. All you have to do is replace the author, title, abstract, and text of the paper with your own.

The formatting instructions contained in these style files are summarized in Sections 10, 11, and 12 below.

10 General formatting instructions

The text must be confined within a rectangle 5.5 inches (33 picas) wide and 9 inches (54 picas) long. The left margin is 1.5 inch (9 picas). Use 10 point type with a vertical spacing (leading) of 11 points. Times New Roman is the preferred typeface throughout, and will be selected for you by default. Paragraphs are separated by 1/2 line space (5.5 points), with no indentation.

The paper title should be 17 point, initial caps/lower case, bold, centered between two horizontal rules. The top rule should be 4 points thick and the bottom rule should be 1 point thick. Allow 1/4 inch space above and below the title to rules. All pages should start at 1 inch (6 picas) from the top of the page.

For the final version, authors’ names are set in boldface, and each name is centered above the corresponding address. The lead author’s name is to be listed first (left-most), and the co-authors’ names (if different address) are set to follow. If there is only one co-author, list both author and co-author side by side.

Please pay special attention to the instructions in Section 12 regarding figures, tables, acknowledgments, and references.

11 Headings: first level

All headings should be lower case (except for first word and proper nouns), flush left, and bold.

First-level headings should be in 12-point type.

11.1 Headings: second level

Second-level headings should be in 10-point type.

11.1.1 Headings: third level

Third-level headings should be in 10-point type.

Paragraphs There is also a `\paragraph` command available, which sets the heading in bold, flush left, and inline with the text, with the heading followed by 1 em of space.

12 Citations, figures, tables, references

These instructions apply to everyone.

12.1 Citations within the text

The `natbib` package will be loaded for you by default. Citations may be author/year or numeric, as long as you maintain internal consistency. As to the format of the references themselves, any style is acceptable as long as it is used consistently.

The documentation for `natbib` may be found at

```
http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf
```

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

> Hasselmo, et al. (1995) investigated…

If you wish to load the `natbib` package with options, you may add the following before loading the `nips_2016` package:

```
\PassOptionsToPackage{options}{natbib}
```

If `natbib` clashes with another package you load, you can add the optional argument `nonatbib` when loading the style file:

```
\usepackage[nonatbib]{nips_2016}
```

As submission is double blind, refer to your own published work in the third person. That is, use "In the previous work of Jones et al. [4]," not "In our previous work [4]." If you cite your other papers that are not widely available (e.g., a journal paper under review), use anonymous author names in the citation, e.g., an author of the form "A. Anonymous."

12.2 Footnotes

Footnotes should be used sparingly. If you do require a footnote, indicate footnotes with a number[1] in the text. Place the footnotes at the bottom of the page on which they appear. Precede the footnote with a horizontal rule of 2 inches (12 picas).

Note that footnotes are properly typeset *after* punctuation marks.[2]

12.3 Figures

All artwork must be neat, clean, and legible. Lines should be dark enough for purposes of reproduction. The figure number and caption always appear after the figure. Place one line space before the figure caption and one line space after the figure. The figure caption should be lower case (except for first word and proper nouns); figures are numbered consecutively.

You may use color figures. However, it is best for the figure captions and the paper body to be legible if the paper is printed in either black/white or in color.

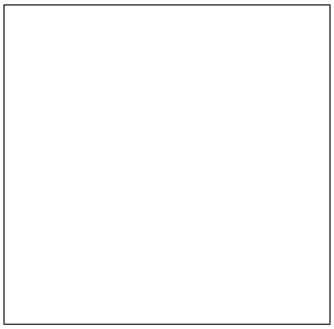

Figure 1: Sample figure caption.

12.4 Tables

All tables must be centered, neat, clean and legible. The table number and title always appear before the table. See Table 1.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

Note that publication-quality tables *do not contain vertical rules.* We strongly suggest the use of the `booktabs` package, which allows for typesetting high-quality, professional tables:

`https://www.ctan.org/pkg/booktabs`

This package was used to typeset Table 1.

13 Final instructions

Do not change any aspects of the formatting parameters in the style files. In particular, do not modify the width or length of the rectangle the text should fit into, and do not change font sizes (except perhaps in the **References** section; see below). Please note that pages should be numbered.

[1] Sample of the first footnote.

[2] As in this example.

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

14 Preparing PDF files

Please prepare submission files with paper size "US Letter," and not, for example, "A4."

Fonts were the main cause of problems in the past years. Your PDF file must only contain Type 1 or Embedded TrueType fonts. Here are a few instructions to achieve this.

- You should directly generate PDF files using `pdflatex`.
- You can check which fonts a PDF files uses. In Acrobat Reader, select the menu Files>Document Properties>Fonts and select Show All Fonts. You can also use the program `pdffonts` which comes with `xpdf` and is available out-of-the-box on most Linux machines.
- The IEEE has recommendations for generating PDF files whose fonts are also acceptable for NIPS. Please see `http://www.emfield.org/icuwb2010/downloads/IEEE-PDF-SpecV32.pdf`

  The `\bbold` package almost always uses bitmap fonts. You should use the equivalent AMS Fonts:

  ```
  \usepackage{amsfonts}
  ```

  followed by, e.g., `\mathbb{R}`, `\mathbb{N}`, or `\mathbb{C}` for $\mathbb{R}$, $\mathbb{N}$ or $\mathbb{C}$. You can also use the following workaround for reals, natural and complex:

  ```
  \newcommand{\RR}{I\!\!R} %real numbers
  \newcommand{\Nat}{I\!\!N} %natural numbers
  \newcommand{\CC}{I\!\!\!\!C} %complex numbers
  ```

  Note that `amsfonts` is automatically loaded by the `amssymb` package.

If your file contains type 3 fonts or non embedded TrueType fonts, we will ask you to fix it.

14.1 Margins in LaTeX

Most of the margin problems come from figures positioned by hand using `special` or other commands. We suggest using the command `includegraphics` from the `graphicx` package. Always specify the figure width as a multiple of the line width as in the example below:

```
\usepackage[pdftex]{graphicx} ...
\includegraphics[width=0.8\linewidth]{myfile.pdf}
```

See Section 4.4 in the graphics bundle documentation (`http://mirrors.ctan.org/macros/latex/required/graphics/grfguide.pdf`)

A number of width problems arise when LaTeX cannot properly hyphenate a line. Please give LaTeX hyphenation hints using the `\-` command when necessary.

Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper.

References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. **Remember that you can use a ninth page as long as it contains *only* cited references.**

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.