Unit-e Alpha Testnet (v0.1) Technical Design

Unit-e is a scalability-focused proof-of-stake cryptocurrency. The goal of Unit-e is to achieve throughput and latency close to the physical limits imposed by the underlying network, using a combination of techniques outlined in the Unit-e Research Manifesto. Critically, this must be achieved in a decentralized fashion. We are launching a Unit-e alpha testnet (v0.1) in April 2019. The Unit-e reference implementation is forked from Bitcoin Core. Like Bitcoin, Unit-e is UTXO-based; unlike Bitcoin, Unit-e is currently fully proof-of-stake. Our v0.1 testnet includes many of the features that will ultimately end up in the Unit-e mainnet, as well as some intermediate features that are necessary for Unit-e's long-term scaling plan.

The purpose of this design document is to outline the design choices associated with our testnet release and to highlight how these design choices relate to the final Unit-e design. This document contains high-level descriptions of our design choices; it is intended for a moderately technical audience with some knowledge of blockchain terminology and concepts. All of our design decisions have been documented in greater detail in our GitHub repository. In addition to an overview of the different components, this document also includes links to the relevant design documentation.

Our design document begins with a summary of the Esperanza consensus protocol in Section 1. Esperanza is an intermediate consensus protocol that is implemented in v0.1 as a stepping stone to the eventual implementation of Unit-e's main consensus protocol, Prism. Since Prism is designed to operate at the physical limits of the underlying network, having a fast network layer is critical. Section 2, therefore, contains a summary of Unit-e's P2P network-level features. We discuss wallet support in Section 3, including hardware wallets and remote staking, and a roadmap for integrating the components of Unit-e's design in Section 4. Finally, in Section 5, we give operational details about how the testnet will be run and how to participate.

1. Esperanza Consensus Protocol

The consensus protocol in Unit-e v0.1 is called Esperanza. Esperanza is a stepping stone to Unit-e's eventual consensus protocol, Prism [1]. Esperanza is a chain-based consensus protocol inspired by PoSv3 [2], with two key differences. First, its proposal mechanism uses a randomness selection mechanism that provides stronger security guarantees than PoSv3. Second, Esperanza includes a finalization procedure. We describe both parts here.

Block proposal

At a high level, block proposal follows a procedure similar to a popular chain-based proof-of-stake consensus mechanism called proof-of-stake version 3 (PoSv3) [2]. PoSv3 is designed to emulate Nakamoto consensus in a PoS setting. It does this by splitting time into slots. Unit-e v0.1 currently uses a time slot of 4 seconds. In each time slot, each UTXO has the opportunity to propose a block, with a random chance of success. A UTXO is allowed to propose a block if

```
hash(previousStakeModifier << utxo.time << utxo.hash << utxo.n << blockTime) < threshold
```

where `blockTime` is the current time slot, `threshold` is the difficulty threshold, and `previousStakeModifier` is a reference to the parent block. The honest protocol is to take the `previousStakeModifier` from the tip of the longest visible chain in the system. Under this policy, the scheme proceeds similarly to Nakamoto consensus. Notice that if the difficulty is chosen to be small, it may be the case that nobody proposes a block in a particular time slot. Indeed, under our current parameter settings, Unit-e's expected time to propose a block is 8 seconds (i.e., two time slots), with a block size of 4 MB.

A key concern in PoS settings is so-called *nothing-at-stake attacks (NAS)*: because it is cheap to construct blocks, users can generate arbitrarily many of them by using arbitrarily old blocks for the previousStakeModifier at each time slot. These attacks are security-critical: by executing a nothing-at-stake attack, a malicious adversary can amplify its effective power by a factor of up to e (where e denotes the logarithmic constant, ~2.7). In other words, if the adversary starts with a fraction β of the stake, it can use a NAS attack to behave as if it owned up to fraction $e\beta$ of the stake [3].

A key innovation of Esperanza is to carefully choose the randomness used to select the next proposer. At a high level, the idea is to not choose the previousStakeModifier as the parent block, but as the last block modulo k in the longest chain, for some small constant k. This idea is illustrated in Figure 1 for k=2:

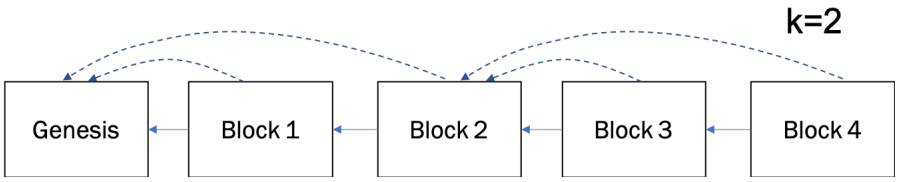


Figure 1. k-correlation controls the randomness of block selection to only use blocks of height 0 mod k for randomness generation. So if k=2, block 4's stake modifier would be taken from block 2.

Notice that each block still has a link to a single parent block, but the randomness used to decide whether a UTXO is allowed to propose a block is determined by the dashed arrows, which are not necessarily the block's immediate parent. This relatively small change gives substantial improvements in security. In [3], we show that as k grows, we can reduce the adversary's amplification factor from e down to 1 (with high probability). This neutralizes the effect of NAS attacks on chain-based PoS systems. In practice, we intend to use k=15, which gives a maximum amplification factor of about 1.04; however, at the moment,

we are still using k=1. The parameter k=15 was chosen to be as small as possible while achieving what we perceive to be an acceptably small amplification ratio (i.e., less than 5%). Keeping k small is important because the larger k is, the easier it is for an adversarial party to predict who will be a block proposer at a given point in time. This can lead to other adversarial behavior, such as *bribery attacks,* in which an adversary bribes a future proposer to build on an adversarial fork.

[1] Bagaria, V., Kannan, S., Tse, D., Fanti, G., & Viswanath, P. (2018). Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*.
[2] Earls, Jordan. The missing explanation of Proof-of-Stake Version 3. http://earlz.net/view/2017/07/27/1904/the-missing-explanation-of-proof-of-stake-version

Finalization

One of the key features of Esperanza is its finalization mechanism. Finalization is useful for protecting against various security concerns like long-range attacks and, to some extent, nothing-at-stake attacks.

Finalization Mechanism. The finalization mechanism in Esperanza is based on Casper the Friendly Finality Gadget [1]. Casper was initially proposed for Ethereum as a way to use BFT voting for finalization in a blockchain setting. In Casper FFG, a set of dedicated validators deposit stake in order to participate in the procedure. During this period, validators vote on checkpoints, which occur every 50 blocks.

Confirmation of a block has two steps: first, the block must be justified, then it must be confirmed. To finalize or justify a checkpoint, validators create so-called *vote transactions*, which link sequences of checkpoints on the same chain. These checkpoints need not be adjacent on the blocktree. Once a link receives a supermajority of votes (⅔ of validators' deposits), it can be used to justify or finalize a checkpoint. As stated in [1], a checkpoint block c is called "justified" if (1) a supermajority of validators votes on a link with c as the source, or (2) there exists a supermajority link c'→c where checkpoint c' is justified. A checkpoint c is called finalized if (1) it is the genesis block or (2) it is justified and there is a supermajority link c→c' where c' is a direct child of c. Equivalently, checkpoint c is finalized if and only if: checkpoint c is justified, there exists a supermajority link c→c', checkpoints c and c' are not conflicting, and h(c') = h(c) + 1, where h(c) denotes the *height* of a checkpoint or its distance from the genesis block.

Notice that once a checkpoint is finalized, it implies that at least ⅔ of the validators have approved it. Hence, if two conflicting checkpoints were to be finalized, it would imply that at least ⅓ of the validators have equivocated, or placed conflicting votes. If this happens, the offending validators' deposited stake is confiscated in a process called slashing. In particular, validators can be slashed for one of two reasons:
(1) The validator posts two vote transactions, (s1,t1) and (s2,t2), with endpoints at the same height, i.e., h(t1) = h(t2).
(2) The validator posts two vote transactions, (s1,t1) and (s2,t2), such that h(s1) < h(s2) < h(t1) < h(t2). In other words, the two transactions overlap one another.

The initial stake deposit is chosen to be large for validator nodes so that the penalty for equivocating is a substantial disincentive against misbehavior. In Unit-e v0.1, we choose the deposit amount to be 10,000 units. This quantity is likely to change, and a more detailed description of the incentives and economics in Unit-e can be found below. Slashing is initiated by other nodes submitting transactions with evidence of wrongdoing (i.e., conflicting votes signed by the same validator). Notice that these security guarantees are fundamentally different from the guarantees typically provided in chain-based cryptocurrencies: our finalization mechanism provides *economic* security guarantees, whereas the security provided by k-correlation is probabilistic, and does not model rational actors. Hence these two notions of security are complementary.

Security Properties. The main benefit of finalization is to provide stronger security against common threats in PoS systems. In particular, nothing-at-stake (NaS) attacks can be mitigated through finalization by limiting the horizon over which an adversary can append blocks to the blocktree. More precisely, once a checkpoint is validated, the adversary has no incentive to build on blocks at a level *lower* than the latest finalized checkpoint because the blockchain prior to a finalized checkpoint cannot be changed without validators losing a substantial amount of deposited stake. The main tradeoff associated with finalization is security vs. efficiency: the more often we finalize, the smaller the number of blockchain levels that can be built on in a NaS attack. At the same time, finalization consumes resources, so we would like to finalize as infrequently as possible. We have chosen a checkpoint interval of 50 blocks or approximately every 7 minutes. This decision will likely change as we stabilize our economic incentive models.

Another common threat addressed by finalization is *long-range attacks*, in which an adversary releases a private chain that (a) is longer than the current main chain, and (b) forks from the current main chain many blocks in the past. Long-range attacks are a greater threat in PoS systems than proof-of-work (PoW) for the same reason that NaS attacks are prevalent: it is much cheaper to create blocks in PoS, and there is no fundamental resource constraint preventing users from simultaneously building many chains and releasing the longest one. Although these two attacks are very similar, one soft difference between long-range attacks and NaS attacks is the end goal: NaS attacks are typically discussed in the context of an adversary indiscriminately trying to augment its profit by earning more block rewards, whereas long-range attacks are often discussed in the context of an adversary trying to double-spend a particular transaction far in the past. Because NaS and long-range attacks are so similar, finalization protects against both. Since finalized checkpoints cannot be overwritten without some validators losing substantial stake, the most significant long-range attacks that are possible in Unit-e originate from the last finalized checkpoint.

[1] Buterin, V., & Griffith, V. (2017). Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*.

2. P2P Network Improvements

Graphene

Graphene is a recently-proposed approach for reducing the bandwidth consumption associated with gossiping blocks in blockchain networks [1,2]. The key idea of graphene is to exploit redundancy in the network: some nodes may already possess a subset of the transactions contained in the blocks being gossiped. If so, there is no need to re-transmit those transactions. Hence a key question is how to tell which transactions are redundant, and which must be transmitted. To solve this problem, Graphene uses widely-studied ideas from coding theory. The core idea is to use both Bloom filter and Invertible Bloom Lookup Tables (IBLT) to minimize the amount of transferred data. IBLT is used to store short transaction hashes that comprise the block and Bloom filter allows to further minimize IBLT size. The key savings come from choosing the size of both of these data structures judiciously.

In experiments, Graphene has been shown to reduce transmission bandwidth by as much as an order of magnitude. Since Unit-e ultimately aims to operate at the limits of the network's capabilities, reducing the network's load is critical. Graphene is therefore implemented in the Unit-e testnet. One of the key required changes for Graphene was to implement the so-called canonical transaction ordering (CTO), which we discuss in the 'Wallet Support' section.

In our implementation Graphene co-exists with compact block. At creation, both graphene and compact block sizes are compared and only the smallest one is relayed.

Issue documentation:
https://github.com/dtr-org/uips/blob/ad576643f165418f582bbdab52be537590e34d12/UIP-0026.md

[1] https://people.cs.umass.edu/~pinar/ozisik.cbt.2017.pdf

[2] https://scalingbitcoin.org/stanford2017/Day1/graphene.BC-scaling.2017.key.pdf

Dandelion Lite (Alex)

Dandelion is a P2P-layer gossip algorithm that helps protect against network-level deanonymization. The core idea is that instead of passing transaction messages to all of a node's neighbors, the transaction first propagates over a randomly-selected line in the P2P topology (stem phase), and then gets flooded using the normal relaying mechanism (fluff phase). Dandelion provides statistical anonymity guarantees without incurring substantial delays in transaction propagation. The exact guarantees are outlined in [1,2]. One challenge with Dandelion is understanding when and how to change the randomized path that transactions

traverse. [2] suggests doing so at periodic intervals of ~10 minutes, but the transitions between stems can lead to implementation complexity and conflicts with features like RBF.

Dandelion Lite is a modified version of Dandelion that achieves weaker privacy guarantees than full Dandelion while exhibiting much lower complexity. The key idea is that the stem phase lasts only one hop. That is, instead of passing transactions along with a randomized path before the fluff phase, the creator of the transactions passes it to exactly one neighboring node, who broadcasts the transaction normally. This idea is described and analyzed in [3], which shows that when the adversary knows the underlying topology, Dandelion and Dandelion Lite have comparable anonymity guarantees.

We chose to implement Dandelion Lite in Unit-e. This decision was made primarily due to its reduced complexity and comparable privacy benefits to full Dandelion. The more nodes use Dandelion Lite - the more efficient it is. That is why it is enabled by default. The exact duration of embargo is to be researched on the testnet. Finalizer Commits are not affected by Dandelion Lite, because we want them to propagate as fast as possible(even at the cost of some security).

Issue description: https://github.com/dtr-org/unit-e/issues/210

[1] Bojja Venkatakrishnan, S., Fanti, G., & Viswanath, P. (2017). Dandelion: Redesigning the Bitcoin Network for Anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *1*(1), 22.
[2] Fanti, G., Venkatakrishnan, S. B., Bakshi, S., Denby, B., Bhargava, S., Miller, A., & Viswanath, P. (2019). Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. *ACM SIGMETRICS Performance Evaluation Review*, *46*(1), 5-7.
[3] Fanti, G. Dandelion Lite. http://www.andrew.cmu.edu/user/gfanti/dandelion-lite.pdf

Fast sync (snapshot)

The purpose of fast sync is to reduce the initial syncing time and the overall bandwidth usage. It can be achieved by downloading the snapshot instead of the full blockchain. Snapshot is the set of all UTXOs at the given block height plus extra metadata of that block.

After the fast syncing, nodes will be in the same state as after they did the full sync with the pruning. They will be able to do full validation of new blocks/transactions and create new ones.

Security is guaranteed by the network itself, as every node computes snapshot hash for every block and it is stored in the blockchain. When downloading the snapshot, its hash must be part of the finalized epoch (2/3 of finalizers voted for it and ensured the correctness of that chain).

More details in the UIP-11:
https://github.com/dtr-org/uips/blob/master/UIP-0011.md

Canonical Transaction Ordering

CTOR stands for “Canonical Transactions Ordering Rule”, sometimes also referred as LTOR (Lexicographical Transactions Ordering Rule). Which refers to a canonical order imposed over the transactions in a block (except for the coinbase transaction)

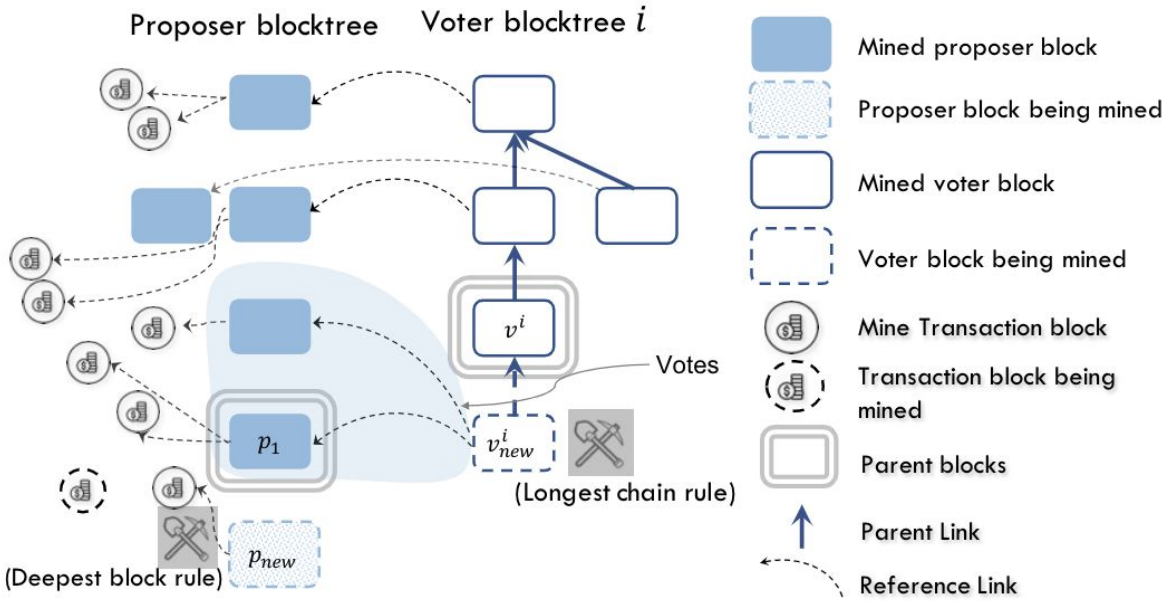
CTOR allow us to implement other interesting features, like Graphene to decrease block propagation times, or transactions processing parallelization to decrease processing times and optimize the usage of our multi-core CPUs.
From a more speculative perspective, it also allows us to implement Merklix trees (that gives us more powerful presence/absence proofs than “standard” Merkle trees) in a simpler way than with the “Topological Transactions Ordering Rule”.

For more details & references:
https://github.com/dtr-org/uips/blob/master/UIP-0024.md

3. Roadmap

Consensus - Prism (Layer 1)

Our long-term scaling plan relies on integrating a consensus protocol called Prism. Prism is a blockchain consensus protocol designed to achieve close to the physical limits on what a network can support in terms of transaction throughput and latency. At a high level, Prism achieves good throughput and latency by decoupling two functionalities of blocks: (1) committing transactions into blocks, and (2) confirming other blocks. The first function governs throughput, while the second governs latency. To this end, Prism generates three types of blocks: transaction blocks, proposer blocks, and voter blocks. Transaction blocks carry transactions, proposer blocks indicate which transaction blocks to include in the ledger, and voter blocks are used to confirm proposed blocks. As seen in the figure below, voter blocks are arranged in many separate blocktrees (e.g. 1,000), and proposer blocks are also arranged in a tree. However, Prism does not stipulate that the longest chain of proposer blocks dictates the ledger. Instead, the voter chains choose which proposer blocks are valid by voting on them. Notice that this results in a (highly structured) blockDAG.



We can speed up the generation of transaction blocks to improve throughput, and we can add more voter blocktrees to reduce confirmation latency. A key property of Prism is that an adversary cannot control which type of block they generate--this property is needed for the security of the scheme.

Current literature on Prism is designed for the proof-of-work setting (PoW) [1]. However, a proof-of-stake (PoS) version is forthcoming and will be implemented in Unit-e. A key observation is that Prism is chain-based: that is, confirmation is determined by the position of a block in the blockDAG. This is in contrast with methods that use round-based BFT protocols to achieve consensus (e.g., Algorand). Because of this, Prism can use a similar block proposal mechanism to Unit-e v0.1, and the basic data structures will remain the same. A key difference is that we will need to implement more copies of the blocktrees.

Another major difference between Prism and Unit-e v0.1 is the finalization procedure. Prism does not include a built-in finalization procedure. Incorporating finalization is challenging because it is difficult to establish which nodes possess what stake at a given point in time. In Bitcoin or PoSv3, this is easy-- we can simply use the ledger based on the longest chain. However, in Prism, the ledger does not depend on a single longest chain: it depends on the states of the transaction block pool, the proposer tree, and all of the voter chains. This makes it difficult to verify which nodes can participate in finalization without having access to the views of every node in the network. Hence, we currently plan to not include finalization in the first implementation of Prism-PoS.

[1] Bagaria, V., Kannan, S., Tse, D., Fanti, G., & Viswanath, P. (2018). Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*.

Spider (Layer 2)

Another important scalability innovation from the Unit-e Research Manifesto is Spider, a layer 2 scaling solution. Payment channel networks (PCNs) are viewed as an important scalability solution for cryptocurrencies. However, existing PCNs use naive routing schemes that lead to congestion and low throughput. As Unit-e develops, we intend to add support for PCNs that use Spider routing [1]. The key idea behind Spider is balance-aware routing: that is, we make routing decisions that actively prevent channels from becoming imbalanced. Sometimes this involves taking longer or less obvious routes; however, the end effect is to boost performance for everybody. In simulation, Spider multipath transport layer protocol can boost throughput by as much as 100% compared to state-of-the-art solutions.

Because of the significant throughput improvements associated with Spider, we intend to implement it in Unit-e. There are two main steps associated with implementing Spider. The first is to add support for PCNs. This will be a priority in 2019, and we intend to build on the infrastructure of Lightning Labs. The second is to implement Spider routing in PCN clients. Doing so will involve a number of major changes. First, the network should be packet-switched, meaning that transactions can be split into smaller components and routed over different routes at different times. This improvement is already being implemented in several existing PCNs (e.g., AMP). A second change is that transactions must be able to be queued at intermediate routers. This enables a packet in a congested channel to wait until the channel is clear to proceed. Queueing is central to Spider and makes a substantial difference in practice due to randomness in the arrivals of transactions. Third, individual routers should implement the Spider routing logic.

[1] Sivaraman, V., Venkatakrishnan, S. B., Alizadeh, M., Fanti, G., & Viswanath, P. (2018). Routing cryptocurrency with the spider network. *arXiv preprint arXiv:1809.05088*.

5. Alpha Testnet

The alpha testnet is the space for testing and gradually ramping up the protocol and the network. It will be used for testing, ironing out bugs, and to experiment with parameters, for example for the economics. With the launch of the alpha testnet, the first version of the Esperanza Consensus Protocol and the P2P Network Improvements is in place. More of the features and components envisioned for Unit-e will be added and released over time.

The testnet will be used for scalability tests and, at a later stage, for building up an ecosystem of validators. It's open for participation of any interested party. The goal is to build a community around Unit-e which will take part in the development. With the testnet, the project is becoming open for the blockchain and open source community and provides the means for further development and collaboration.