

The Thunder Protocol

Rafael Pass and Elaine Shi

Thunder Research

1 Introduction

In this white paper, we introduce the Thunder protocol, a new and *provably secure* consensus protocol, which overcomes the two main bottlenecks of classic blockchains:

Throughput: Existing blockchains as employed by major crypto-currencies (e.g., Bitcoin or Ethereum) require flooding the whole network with all the transactions. As a consequence, these blockchains do not scale well to handle a large transaction volume. For instance, Bitcoin and Ethereum can handle less than 10 transactions/second. This small throughput severely hinders a wide-spread adoption of such crypto-currencies. As a comparison, VISA handles 2000 transactions/sec.

Confirmation times: The security of Nakamoto’s blockchain protocol requires the block-time to be significantly larger than the maximum network latency [20]; this is why in Bitcoin, new blocks of transactions get mined every 10 minutes, and a transaction is only considered confirmed once it has been incorporated in a block and there are 5 subsequent blocks following it; thus it takes on average one hour for a transaction to be confirmed (with low confidence). While Ethereum uses a smaller block-time, the average confirmation time still remains relatively high—approximately 13 minutes. These long confirmation times hinder many important applications (especially smart contract applications).

Our Thunder protocol overcomes both of these issues, achieving both high throughput and fast confirmation time (under typical circumstances within 2 network roundtrips), while being robust up to a 50% attack.

While we are aware of several other recently suggested approaches for overcoming the above two bottlenecks, these approaches are either restricted to only payments (and do not handle smart contracts), or provide security only against a weaker 1/3-attack; additionally, many approaches are based on heuristic arguments, whereas our protocol is accompanied by a rigorous proof of security; see Section 4 for more details.

The design of Thunder is inspired by a new theoretical paradigm for consensus proposed in our earlier work, Thunderella [21], and provides the first practical instantiation of this general paradigm. This paradigm can be instantiated using either Proof-of-Work or Proof-of-Stake.

2 The Thunder Protocol

Following our Thunderella paradigm [21], the key idea behind Thunder is to combine a “standard” blockchain—which we will refer to as the *slow chain*—with an *optimistic fast-path*. The fast path is coordinated by a new entity referred to as the **Accelerator**, and requires the use of a *committee of “stake-holders”*. As long as a) the **Accelerator** is acting honestly, b) network conditions are good, and c) $> 3/4$ of the committee are honest, we barely need to use the slow-chain at all; instead, transactions can be confirmed on the fast-path by the committee using an extremely simple and lean fast-path protocol. This protocol offers both high throughput and instant confirmation of transactions. But the fast-path only confirms new transactions assuming the above mentioned good conditions are met. When they are not (e.g., the **Accelerator** misbehaves or is under attack), we leverage the “slow chain” to recover in a provably sound manner.

We proceed to provide a more detailed description of our new consensus protocol.

2.1 Consensus and Blockchains

Roughly speaking, a blockchain is simply a permissionless implementation of a primitive known as *state machine replication*, or *consensus* in the distributed systems literature.

Consensus: A consensus protocol enables maintaining a “linearly ordered log” of transactions in a distributed fashion such that the following two properties hold:

- **Consistency:** at any point in the execution, all honest participants have consistent logs of transactions—that is, either their logs are identical, or one participant’s log is a prefix of the other’s.
- **Liveness:** any honest protocol participant can propose to add a transaction; this transaction is then guaranteed to be incorporated into their logs within some fixed, small amount of time; additionally, whenever a participant sees some transaction in their log, the same transaction will appear in every other participant’s log within some fixed, small amount of time.

In essence, these properties mean that from the view of the participants, they are communicating with a trusted third party that maintains a *global, ordered, append-only* log of transactions that anyone can add to—in essence a *public ledger*. Our goal here is to implement a consensus protocol in the permissionless setting with high throughput and fast confirmation of transactions.

Blockchains: A formal definition of a blockchain can be found in [20]. Strictly speaking, a blockchain does not directly give the above-described consistency property; rather, the consistency property defined for a blockchain is that players have a consistent view of the blockchain *when dropping the last κ blocks*, where κ is a *security parameter*: in other words, the last κ blocks may be “unstable”, but everything before that has been stabilized (except with exponentially vanishing probability as a function of the security parameter κ).

Other than the above-mentioned main properties of consistency and liveness, [20] also defines a notion of *bounded chain-growth*: the length of the blockchain should grow at a well-define pace, not too fast and not too slow. We will require the use of a slow-chain that satisfies such bounded chain-growth.

Pass et al. [20] show that Nakamoto’s protocol satisfies all the above properties once the mining hardness is appropriately set (as a function of the network delay and total computing power)

assuming that a majority of the mining is done by honest users (a.k.a. “computational honest majority”). More precisely, the Nakamoto protocol satisfies consistency and liveness *if and only if* the mining-hardness is set as a function of the maximum network latency so that the block-time (which in the Bitcoin system is set to 10 minutes) significantly exceeds the maximum network latency. Ethereum’s protocol is related to Nakamoto’s and ought to satisfy them as well.

Security Contexts: We will distinguish between two different conditions on the network participants.

- **Worst-case conditions \mathbf{W}** under which the protocol is guaranteed to provide consistency and liveness (i.e., \mathbf{W} are conditions under which the protocol is guaranteed to be *secure*). In our case, \mathbf{W} will be the conditions that:
 1. the underlying slow-chain is secure (which in turn follows from computational honest majority in the case of Nakamoto’s blockchain); and,
 2. a majority of the committee is honest.

Under such worst-case conditions, confirmation times may be slow, and the transaction load may be limited (just as is the case for Bitcoin and Ethereum).

- **Optimistic conditions \mathbf{O}** under which the transaction volume can be large, and transactions are “instantly” confirmed. In our case, \mathbf{O} will be the conditions:
 1. the slow-chain is secure;
 2. the Accelerator as is online and honest;
 3. $> \frac{3}{4}$ of the committee is online and honest.

Our new Thunder protocol will make use of a simple fast-path protocol in conjunction with an underlying slow-chain to ensure worst-case security under \mathbf{W} , and instant-confirmation and high-throughput under \mathbf{O} . As mentioned above, the key idea is to make use of the “fast-path” protocol when things work well, and only use the slow-chain to recover from faults.

2.2 A Simple Fast-Path Protocol

Consider first the following simple fast-path protocol:

- We have a designated entity: the Accelerator.
- All new transactions are sent to the Accelerator; the Accelerator bundles up transactions into micro-blocks, signs each micro-block *with increasing sequence numbers seq*, and sends out the signed micro-block to a “committee” of players.
- The committee members “ack” all Accelerator-signed micro-blocks by signing them, *but* only ack at most one micro-block per sequence number.
- When a micro-block has received acks (i.e., signatures) from more than $3/4$ of the committee members, we refer to the micro-block as being **notarized** (and the notarization of the micro-block is this collection of these signatures.¹ Participants can *directly output* their longest

¹In an actual implementation, one would make use of an *aggregate signature scheme* to describe the notarization in a compact form.

sequence of consecutive (in terms of their sequence numbers) notarized micro-blocks—all transactions contained in them are considered **confirmed**. In this way, the fast-path instantly finalizes transactions.

It is easy to prove (and well-known) that this protocol is consistent under the condition $\mathbf{W}^{\text{fast}} = > 1/2$ of the committee members are honest); additionally, it satisfies liveness with instant confirmation under the conditions $\mathbf{O}^{\text{fast}} =$ “1) the Accelerator is online and honest, 2) $> 3/4$ of the committee members are online and honest”. In fact, under these optimistic conditions, we only need 2 communication rounds to confirm a transaction! This approach is extremely practical and indeed similar protocols are often used in practice—for instance `chain.com` uses a related protocol for their permissioned consensus protocol.

The problem with this approach, however, is that the protocol does not satisfy liveness (even “slow” liveness) under condition \mathbf{W}^{fast} . If the Accelerator is cheating (or is simply taken down from the network), the protocol halts. (Indeed, in this case `chain.com` would have to manually fix the issue, which is impossible in a decentralized setting.)

2.3 How to Recover from Faults

To overcome this problem, we leverage the underlying “slow-chain” which we assume satisfies both consistency and liveness. Roughly speaking, we deal with this as follows.

Heartbeats: For every length ℓ of the slow-chain, the Accelerator takes the hash h of the current log (of the fast path) and sends out the tuple (ℓ, h, seq) (where `seq` is the current sequence number) to get notarized by the committee members (who all check that h is correct based on the history they have seen); we refer to a notarized version of such a tuple (ℓ, h, seq) as a **heartbeat**. The Accelerator is subsequently required to post the **heartbeat** to the slow-chain.

Cool-down: Next, if some user notices that its transaction is not getting confirmed by the accelerator/committee, some “evidence” of this will become apparent on the underlying slow-chain—as we shall see shortly, the evidence is simply the fact that no **heartbeat** has been observed on the slow-chain for a sufficiently long period of time. (Note that by liveness, and the bounded chain growth properties of the slow-chain, if Accelerator and more than $3/4$ of the committee are online and honest, we are guaranteed that a heartbeat for length ℓ gets incorporated close to length ℓ). Whenever such evidence of cheating (i.e., no recent heartbeat are contained on the slow-chain) has been found, we enter a “cool-down” period, where committee members stop signing messages from the Accelerator, yet we allow anyone to post any “new” notarized transactions (which were not included in the most recent heartbeat) to the slow-chain. The length of the cool-down period is counted in blocks of the slow-chain (say κ blocks where κ is a security parameter).

Slow mode: Finally, after the cool-down period ends, we can safely enter a “slow period” where transactions *only* get confirmed in the slow-chain blockchain (and the fast-path is no longer active). We stay in the slow-chain for an appropriate amount of time counted in blocks on the slow-chain (e.g., enough blocks to restore or replace a faulty Accelerator) and can next reboot with a new epoch of the fast-path protocol; more details on the reboot step can be found in Section 2.4.

Let us point out the reason for having a cool-down period: Without it, players may disagree on the set of fast-path transactions that have been confirmed before entering the “slow mode”, and thus may end up with inconsistent views. The cool-down period enables honest players to post all notarized transactions they have seen to the slow-chain, and thus (slowly) reach consensus on this set of transactions; once we have reached this consistent view (at the end of the cool-down), we can finally fully switch over to confirming new transactions on the slow-chain.

Collecting evidence of cheating: “yell” transactions It remains to explain how to collect evidence that the Accelerator (and/or committee) is cheating or is simply “offline”. If a player notices that his transaction is not getting confirmed by the Accelerator or committee, he can send a special `yell` transaction (which contains the transaction it wants to see confirmed) to the underlying slow-chain. The Accelerator is additionally instructed to confirm all such `yell` transactions it sees on the slow-chain.

Now, if a committee member sees some transactions on the slow-chain that have not gotten notarized within a sufficiently long amount of time—counted in blocks in the slow-chain (say within κ blocks)—they know that the Accelerator (or a large fraction of the committee members) must be cheating or offline. At this point, the committee member will stop signing `heartbeat` messages.

So, as long as just 1/4 of the committee are honest, a cheating Accelerator will be “choked”—no new `heartbeats` by Accelerator will be notarized. Consequently, everyone that observes the slow-chain will enter the “cool-down” phase and subsequently the slow mode.

2.4 How to Reboot the Fast Path

Once we have entered the slow mode where all transactions are being posted on the slow-chain, we need to have a way to return to the fast path. The simplest method is for the current Accelerator (once it has recovered from the fault), to post a “summon” message to the slow-chain to summon committee members to retry at a certain point of time (expressed in terms of some future height of the slow-chain).

More generally, Thunder nodes can use the slow-chain to discuss and vote on how and when to bootstrap, possibly renegotiate the committee and leader election strategy in the meanwhile. In the general form, nodes can inspect a stabilized prefix of the slow-chain, and imagine that some predicate can be applied to the prefix to decide when nodes should make an attempt to reboot the fast path (See Section 3.4 for more discussions).

2.5 How to Select the Committee

So far we have deferred the discussion of how the committee is selected. Also here, we may consider multiple approaches.

Rely on recent miners: Perhaps the simplest approach is to select the committee as the miners of, say, the 500 most recent blocks on the slow-chain (as was done in [23]). We note, however, that to rely on this approach, we need to ensure that the underlying slow-chain is “fair” [22] in the sense that the fraction of honestly mined blocks is close to the fraction of honest players. This is not the case for Nakamoto’s original blockchain nor Ethereum (see e.g., [14]), but as shown in [22], any blockchain can be turned into a fair one. If we use this approach, the resulting protocol will now be consistent and live under simply the

condition \mathbf{W} = “computational honest majority” (as this condition implies that a majority of the committee will be honest), and high-throughput and instant confirmation holds under the optimistic conditions \mathbf{O} = “1) The Accelerator is honest, 2) $> 3/4$ fraction of the mining power is honest.”

Subsample from all stake-holders: If the blockchain is used for a cryptocurrency (as we intend here), we may instead select the committee by subsampling from the recent stake-holders. This can be done using the method from our earlier work Snow White [11]:

The committee that is active when the slow chain has a length ℓ is selected based on the stake-holders when then the slow chain had length $\ell - 2\kappa$ by using a selection function that takes as input the hash of the chain at length $\ell - \kappa$. As explained in more detail in our earlier work [11], the reasons for the 2κ “look-back” is to prevent against attacks where an attacker adaptively selects its keys to have a higher probability of getting elected to the committee. This approach ensures consistency and liveness under the conditions \mathbf{W} = “1) the slow-chain is secure, 2) $> 1/2$ of the stake holders are honest”; and achieves fast confirmation under the conditions \mathbf{O} = “1) The Accelerator is online and honest, 2) $> 3/4$ of the stake holders are online and honest”.

The drawback of this approach is that it may be too optimistic to assume that a majority of stake-holders are online (and thus the conditions for the fast path may rarely hold). The next approach deals with that issue.

Select from stake-holders putting down an escrow: One potential drawback of the above approach is that not stake-holders may want to participate in transaction validation. Rather than subsampling from the full set of stake holders, we may instead ask stake-holders that intend to participate the to put down an *escrow* (as a message posted on the slow-chain). We can then select from these stake-holders who have put down escrow. One idea is to select the members putting down the most amount of escrow, since we would like to encourage more stake to join the committee and protect the fast path. The actual algorithm we implemented is motivated by this idea but a carefully crafted variant such that we can prove that the equilibrium behavior satisfies certain desirable properties.

More generally, committee selection can be performed by applying a general predicate to a stabilized part of the slow-chain.

3 The Full Thunder Protocol Specification

We here provide the full protocol specification. We let κ be a security parameter: as mentioned before, we will require the underlying slow-chain to guarantee consistency and liveness except with negligible probability in κ . (Concrete parameters will be presented in a forthcoming implementation paper.)

For simplicity of exposition, we first describe a protocol that starts off in the fast mode and simply enters the slow mode when the optimistic conditions stop holding; later, in Section 3.4, we explain how to extend the protocol to enable a “reboot” of the fast mode.

For simplicity, we also assume that each microblock contains just a single transaction tx —in practice, a microblock will contain a batch of transactions (and our protocol can be extended to deal with any batching method).

3.1 Notations and Useful Definitions

- *The underlying slow-chain.* Nodes (a.k.a. “miners”) run an underlying blockchain protocol referred to as the “slow chain”; we use the notation `chain` to denote a node’s the view of the slow chain. We use the notation `chain[: ℓ]` to denote the prefix of `chain` upto the ℓ -th block; we use `chain[: -ℓ]` to denote `chain` where the last ℓ blocks have been removed. (Recall that the consistency property of a blockchain does not guarantee that everyone has a consistent view of the whole chains, but they do once the last κ , where κ is a security parameter, blocks—that is the “unstable blocks” have been removed.) We use `chain[ℓ : ℓ’]` to denote the part of `chain` from the ℓ -th block to the ℓ' -th block.

We assume there there is some commonly known *starting length* ℓ^{start} for the slow-chain at which we begin the execution of the Thunder protocol.²

- *Committee selection.* We assume that there is a *publicly-known* function denoted `comm` which takes as input the stabilized part of the blockchain `chain[: -κ]` and outputs a set of committee members denoted `comm(chain[: -κ])`. (We discussed a few concrete committee selection methods in Section 2.5.)
- *Micro-blocks:* Each *micro-block* in the fast-path log has the form

$$(\text{mblock}, \text{seq}, \ell, \text{tx})$$

where `seq` $\in \mathbb{N}$ is a sequence number, ℓ is the *slow-chain length* associated with the micro-block (indicating the slow-chain length when this transaction was proposed), and `tx` is a transaction.

- *Heartbeats:* A `heartbeat` has the same form as a micro-block except that the transaction `tx` is replaced simply by the hash h of the log: more precisely, a `heartbeat` has the form

$$(\text{heartbeat}, \text{seq}, \ell, h)$$

We say that such a heartbeat is *associated with slow-chain length* ℓ .

- *Notarization.* Let m be of the form `(mblock, seq, ℓ, -)` or `(heartbeat, seq, ℓ, -)`. We say that σ is the *notarization* of m w.r.t. `chain` if a) $\ell \leq |\text{chain}|$ and b) σ is a valid aggregate signature from more than $\frac{3}{4}$ fraction of the members of `comm(chain[: ℓ - κ])` as well as from the Accelerator. Whenever `chain` is clear from context, we simply say that σ is a notarization of m . A node i considers σ to be a notarization of m if σ is a notarization of m w.r.t. the node i ’s view of the slow-chain.

Finally, a notarized microblock (or heartbeat) is simply a pair consisting of a microblock (or heartbeat) and a notarization of it.

- *Skipped heartbeats.* We say that the slow-chain `chain` has a *skipped heartbeat* if there exists some ℓ such that a) $\ell^{start} \leq \ell \leq |\text{chain}| - 2\kappa$, and b) `chain[ℓ - κ : ℓ + κ]` does not contain a notarized heartbeat associated with length ℓ .³
- *Yell transactions.* A yell transaction is a transaction on the slow-chain with a special type-modifier `yell`.

²Later when we may have multiple epochs of the fast-mode, we can use the slow-chain to reach common knowledge on the starting length ℓ^{start} .

³That is, there exists a 2κ windows in the *stabilized* part of `chain` which does not contain its appropriate `heartbeat`.

3.2 Fast-Mode Protocol

Everyone:

- **Storing notarized messages:** Upon receiving a message m and a notarization σ of m (w.r.t. chain), store m and mark m as being notarized.
- **Extracting the log:**
 - a) A lucky sequence is defined to be a sequence of notarized tuples with consecutive sequence numbers

$$(T_1, 1, \ell_1, -), (T_2, 2, \ell_2, -), \dots, (T_k, k, \ell_k, -),$$
 such that
 - $\ell_1 = \ell^{\text{start}}$;
 - for every $i \in [k]$, $T_i \in \{\text{mblock}, \text{heartbeat}\}$;
 - for every $i \in [k]$, $\ell_{i+1} = \ell_i + 1$ if $T_i = \text{heartbeat}$ and otherwise $\ell_{i+1} = \ell_i$.
 - b) Let ℓ be the first skipped heartbeat in $\text{chain}[: -\kappa]$. Find a lucky sequence denoted $\text{LOG}_{\text{stable}}$ that has been observed so far and is consistent with all heartbeats at lengths $\ell^{\text{start}}, \ell^{\text{start}} + 1, \dots, \ell - 1$ in $\text{chain}[: -\kappa]$.
 - c) Let LOG be a *maximal* lucky sequence observed so far — if there are ties, break ties arbitrarily. Output $\text{LOG}_{\text{stable}}$ concatenated with all transactions in LOG whose sequence number has not appeared in $\text{LOG}_{\text{stable}}$.
- **Transitioning to “slow-mode”:** If noticing that chain contains a *skipped heartbeat*, transition to executing the slow-mode protocol (described below in Section 3.3).

The Accelerator:

- **Init:** set $\text{seq} = 0$;
- **Microblock notarization requests:** Upon observing any new transaction tx (either transmitted over the network or part of a yell transaction in chain):
 - let $\text{seq} := \text{seq} + 1$;
 - sign the tuple $m = (\text{mblock}, \text{seq}, |\text{chain}|, \text{tx})$ producing the signature σ^{acc} ;
 - send the notarization request (m, σ^{acc}) to the members of $\text{comm}(\text{chain}[: -\kappa])$.
- **Heartbeat notarization requests:** Whenever the Accelerator notices that the length of the slow-chain chain has grown by 1:⁴
 - let LOG be the Accelerator’s current fast-path log;
 - let $\text{seq} := \text{seq} + 1$;
 - sign the heartbeat $m = (\text{heartbeat}, \text{seq}, \ell, H(\text{LOG}))$ where $H(\text{LOG})$ is a cryptographic digest of the fast-path log and ℓ is the length of the slow-chain before the increment—let σ^{acc} denote the signature.
 - send the notarization request (m, σ^{acc}) to the members of $\text{comm}(\text{chain}[: -\kappa])$.

⁴We may without loss of generality assume that the slow chain always grows by at most one step—we can always emulate a multi-step hop by several one-step hops.

- **Notarization:** Upon receiving signatures from $> \frac{3}{4}$ fraction of $\text{comm}(\text{chain}[: \ell - \kappa])$ on some message m of the form $(\text{mblock}, -, \ell, -)$ or $(\text{heartbeat}, -, \ell, -)$ for which a notarization request was sent out, compute an aggregate signature σ from the collected signatures as well as the Accelerator’s own signature on m and broadcast m as well as its notarization σ .

Committee members: Upon receiving a tuple m of the form $(\text{mblock}, \text{seq}, \ell, \text{tx})$ or $(\text{heartbeat}, \text{seq}, \ell, h)$ with a valid signature from the Accelerator:

- assert that the node is a member of $\text{comm}(\text{chain}[: \ell - \kappa])$
- assert that ℓ is within $\pm 0.5\kappa$ of $|\text{chain}|$;
- assert that no other tuple with the sequence number seq has been signed;
- If m is of type mblock , sign m and return the the tuple and signature to the Accelerator.
- If m is of the type heartbeat ,
 - assert that $h = H(\text{LOG}')$ where LOG' is the node’s fast-path log up to sequence number $\text{seq} - 1^5$
 - let TXs^* be all yell transactions in $\text{chain}[: -\kappa]$; assert that LOG contains all of them.
 - sign m and return the the tuple and signature to the Accelerator.

Clients: Upon receiving a transaction tx :

- Send tx to the accelerator.
- If the transaction has not gotten included in the node’s fast-path log before the slow-chain has grown by κ steps, broadcast a yell transaction for tx (to the underlying slow-chain).

Miners: Participate in the underlying slow-chain’s mining. The input to the slow-chain protocol includes:

- every yell transaction the node is aware of that has not yet been included in the slow-chain;
- every notarized heartbeat that has not been included in the slow-chain so far.

3.3 Slow-Mode Protocol

In the slow mode, committee members ignore all messages from the accelerator. Clients propose transactions only to the slow-chain by broadcasting any new transaction it receives as input.

Miners:

- Let ℓ^* be the smallest skipped heartbeat that triggered the slow mode.
- Let h^* be the hash digest inside the heartbeat associated with length $\ell^* - 1$ in chain , and let LOG^* be the prefix of the fast-path log that matches the hash h^* .
- Participate in the slow-chain mining; in every round, input to the slow-chain protocol: 1) every notarized transaction that is not contained in LOG^* and not in chain ; and 2) every other transaction it has observed (including those inside yell transactions) but has not been incorporated in LOG^* or chain yet.

Everyone:

⁵If this condition does not hold, store the transaction in an incoming message buffer until the sequence number in LOG exceeds seq .

- Broadcast the fast-path log including a notarization on every micro-block.
- Let ℓ be the earliest block such that $\text{chain}[: \ell]$ contains a skipped heartbeat⁶.
- Let LOG be the maximal lucky sequence (see Section 3.2) of notarized tuples $(T_1, 1, \ell_1, \text{tx}_1), (T_2, 2, \ell_2, \text{tx}_2), \dots, (T_L, L, \ell_L, \text{tx}_L)$, where each notarized tuple has been observed up until the end of the cool-down period, i.e., in the prefix $\text{chain}[: \ell + 2\kappa]$ ⁷. If there are multiple such lucky sequences, let LOG be an arbitrary one.
- Suppose that $\text{chain}[: -\kappa]$ contains heartbeats at consecutive slow-chain lengths $\ell_1, \ell_2, \dots, \ell_k$, where $\ell_i := \ell_{\text{start}} + i - 1$. Let $\text{LOG}_{\text{stable}}$ be a lucky sequence of notarized tuples consistent with the hashes of these heartbeats⁸.
- Output the concatenation of the following in every round:
 - $\text{LOG}_{\text{stable}}$;
 - every tuple in LOG whose sequence number is not included in $\text{LOG}_{\text{stable}}$;
 - all other transactions contained in $\text{chain}[: -\kappa]$ that have not yet been output (in the same order as they appear in chain).

3.4 Rebooting the Fast Path

When nodes are in the slow mode, they can reboot a fast path execution again as follows; let $\text{start}(\cdot)$ be some general predicate which determines when to leave the slow-mode (examples of such predicates can be found in Section 2.4).

Let ℓ be the earliest block such that $\text{chain}[: \ell]$ contains a skipped heartbeat. Let chain^* be the shortest prefix of chain such that if $\ell \leq |\text{chain}^*| - 3\kappa$ and $\text{start}(\text{chain}^*[: -\kappa]) = 1$ where start is a policy-predicate, a node would now enter the fast mode, and reset $\ell^{\text{start}} := |\text{chain}^*|$.

When there are multiple transitions between slow and fast modes, we need to assign a unique epoch number every time a new fast mode is initiated. All `mblock` and `heartbeat` messages should be additionally *tagged with the epoch number* (for simplicity, our earlier description omitted this epoch number). Finally, when there are multiple epochs (where each epoch contains a fast mode followed by a slow mode), nodes should sequentially output the logs of all epochs one by one (our above description explained how to output the log from one such epoch).

4 Comparison with Related Initiatives

We provide some comparisons with related initiatives.

- Offchain payment channels (e.g., Lightning Networks [1] and Raiden networks [2]) and state channels [13, 18] support fast offchain payments, but existing systems offer little to no smart contract programming support; moreover, parties need to engage in a prior setup.
- Several initiatives rely on “classical-style” consensus methods to build a decentralized cryptocurrency. For example, Zilliqa [3] and Tendermint [17] rely on PBFT [9] or its variants to reach consensus. In comparison with Thunder, which combines an asynchronous fast path and a *synchronous* “fallback”, PBFT instead relies on an asynchronous recovery path. This means that

⁶As an optimization in practice, it suffices to check the part of the slow-chain $\text{chain}[\ell^{\text{start}} - \kappa : \ell]$.

⁷if $\ell + 2\kappa$ exceeds $|\text{chain}|$, we round it down to $|\text{chain}|$.

⁸If such a lucky sequence has not been observed, query nodes on the network for such a lucky sequence; and before a reply is obtained, just output the same log as the previous round.

1) PBFT can tolerate at most $< \frac{1}{3}$ fraction corruption due to a well-known lower bound [12] (in contrast, our approach tolerates up to a 50% attack); 2) PBFT requires more rounds of voting in the normal path (which is necessary for an asynchronous recovery) and thus is at least twice as slow as our protocol; and 3) in comparison, our approach is conceptually much simpler than PBFT and thus friendlier to implementation and large-scale deployment [15, 19].

- The beautiful work of Algorand [10] designs a new synchronous state machine replication protocol and builds a cryptocurrency on top. Just like ours, their protocol is provably secure. However, it only tolerates up to $\frac{1}{3}$ corruptions; moreover, their approach requires complete redesign of the consensus layer, whereas our approach allows us to accelerate any existing blockchain using a simple fast-path protocol.
- Several works rely on a DAG-based approach [4, 25] (rather than chain-based) to prevent double-spending. These works do not realize our notion of a linearly ordered log; they also do not support general smart contracts/DApps.
- Dfinity’s approach [5] relies on offchain voting on blocks to reach finality in “2 blocks of time”. Their approach does not improve throughput w.r.t. to the underlying slowchain.
- Ethereum’s Casper [7, 8] also relies on offchain voting on blocks, but they do so to checkpoint blocks and not to improve speed and throughput. The elegant Plasma framework by Poon and Buterin [24] and several sharding initiatives [3, 6, 16] are conceptually related to ours. These approaches are orthogonal and complementary to our effort.

5 Acknowledgments

We are very grateful to the Thunder team and to Vitalik Buterin for his insightful feedback and comments.

References

- [1] <https://lightning.network/>.
- [2] <https://raiden.network/>.
- [3] <https://www.zilliqa.com/>.
- [4] <http://iota.org/>.
- [5] <https://dfinity.org/>.
- [6] On Sharding Blockchains. <https://github.com/ethereum/wiki/wiki/Sharding-FAQ>.
- [7] Vitalik Buterin. <https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c>, 2017.
- [8] Vitalik Buterin and Vlad Zamfir. Casper. <https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/>, 2015.

- [9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.
- [10] Jing Chen and Silvio Micali. Algorand: The efficient and democratic ledger. <https://arxiv.org/abs/1607.01341>, 2016.
- [11] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016.
- [12] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.
- [13] Stefan Dziembowski, Lisa Ekey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. *IACR Cryptology ePrint Archive*, 2017:635, 2017.
- [14] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, 2014.
- [15] Pedro Fonseca, Kaiyuan Zhang, Xi Wang, and Arvind Krishnamurthy. An empirical study on the correctness of formally verified distributed systems. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 328–343, New York, NY, USA, 2017. ACM.
- [16] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 19–34, 2018.
- [17] Jae Kwon. Tendermint: Consensus without mining. <http://tendermint.com/docs/tendermint.pdf>, 2014.
- [18] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017.
- [19] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA, 2016. ACM.
- [20] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.
- [21] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. <https://eprint.iacr.org/2017/913.pdf>.
- [22] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.
- [23] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.
- [24] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. <https://plasma.io/>.

- [25] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.