EE 374: Fundamentals of Blockchain Infrastructure **Stanford, Spring 2025**

Lecture 9: Tendermint and Finality

April 28, 2025

Lecturer: Prof. David Tse Scribe: Justin Wu

1 Agenda

In this lecture, we review and extend the ideas from last lecture regarding the Tendermint protocol. We discuss partially synchronous networks, the ideas of Finality and Availability, and finally, we discuss Ebb and Flow Protocols, and their relation with Ethereum.

2 Tendermint Recap

2.1 System Model and Assumptions

1. **Validators:** A fixed set of n validators participate in consensus. In the usual configuration, one sets $n = 3f + 1$ so that the protocol can tolerate up to f Byzantine failures (i.e., arbitrary or malicious behavior) while still achieving both safety and liveness under synchrony.

2. **Faults:** Validators may fail in a Byzantine manner: they can equivocate, send conflicting messages, or remain silent. We assume at most f validators are Byzantine.

3. **Networks:** The network is *eventually synchronous*: it may experience unbounded delays initially (asynchronous period), but after some unknown Global Stabilization Time (GST), all messages are delivered within a known maximum delay Δ. For liveness guarantees, we require synchrony after GST; safety holds at all times even during asynchrony. This GST idea will be built further in this lecture.

2.2 Protocol State

Each validator maintains the following local state:

1. **Height (H).** The current block height in the chain.

2. **Round (r).** Consensus is organized into sequential rounds $r = 0, 1, 2, \ldots$ at each height. If consensus does not complete in a given round, the protocol advances to $r + 1$.

3. **Step.** Within each round, validators progress through four steps: {Propose, Pre-vote, Pre-commit, Commit}.

4. **Locked Value and Round.** A validator may be *locked* on a particular block B at round r_{lock} if it has sent a pre-commit for B in that round.

5. The validator also keeps track of the highest round r such that it has observed $2f + 1$ round-r pre-votes for a block B_r from round r.

2.3 Protocol Phases

At height H, the protocol proceeds in rounds $r = 0, 1, 2, \ldots$ until a block is committed.

1. **Proposal Step:** At the beginning of the proposal step (time $t = 3\Delta r$), the round-r proposer proposes a round-r block B by sending a proposal message $\langle \mathsf{Proposal}, r, vr, B \rangle$. We will later see what vr refers to – it can be -1 or a positive integer denoting some round.

2. **Pre-vote Step:** Let $\langle \mathsf{Proposal}, r, vr, B \rangle$ be the first round-r proposal message observed by a validator. At the beginning of the pre-vote step (time $t = 3\Delta r + \Delta$), the validator sends a round-r pre-vote, denoted by $\langle \mathsf{Prevote}, r, vr, h(B) \rangle$, for block B (vr is copied from the proposal message) if the following conditions are satisfied:

   - When $vr = -1$, the validator can directly send the pre-vote.
   - When $vr > 0$, the validator will send the pre-vote only if it has also observed $2f + 1$ round-vr pre-votes for B, i.e., $2f + 1$ messages of the sort $\langle \mathsf{Prevote}, vr, vr', h(B) \rangle$ (here, vr' does not matter).

   If a validator does not observe a round-r block it can vote for, it sends a round-r pre-vote, denoted as $\langle \mathsf{Prevote}, r, vr = -1, \perp \rangle$, for a special *nil* (empty) value.

3. **Pre-commit Step:** Let B be the round-r block (i.e., a block that came in a round-r proposal), for which a validator has first observed $2f + 1$ round-r pre-votes of the form $\langle \mathsf{Prevote}, r, vr, h(B) \rangle$ by distinct validators. At the beginning of the pre-commit step (time $t = 3\Delta r + 2\Delta$), the validator sends a round-r pre-commit, denoted by $\langle \mathsf{Precommit}, r, h(B) \rangle$, for the round-$r$ block B,

   If a validator does not observe such $2f + 1$ pre-votes for any round-r block, it sends a round-r pre-commit, denoted as $\langle \mathsf{Precommit}, r, \perp \rangle$, for a special *nil* value.

Confirmation Rule: At the end of the round ($t = 3\Delta r + 3\Delta$) or later, if a validator observes $2f + 1$ round-r pre-commits for B, it confirms B for its height (h), and terminates the protocol for height h. Otherwise, it goes into the next round $r + 1$.

A client c confirms a round-r block B, if it observes (at any time) $2f + 1$ round-r pre-commits for B by distinct validators. In that case, we say that the round-r block B became confirmed by the round-r pre-commits.

Proposal Rule: Let r' be the largest round such that the validator has observed $2f + 1$ round-r' pre-votes for a round-r' block B'. Then, if that validator is elected as a leader in a future round r'', it proposes this block B' for round r'' by sending the message $\langle \mathsf{Proposal}, r'', vr = r', B' \rangle$. If there is no such round r', i.e., if the validator has not observed $2f + 1$ (same-round) pre-votes for any block B', it can propose any block B'' by sending the message $\langle \mathsf{Proposal}, r'', vr = -1, B'' \rangle$.

Locking Rule: A validator v locks on a round-r block B at round r upon sending a round-r pre-commit for B. In a future round r'', the validator v does not send pre-votes for other blocks $B'' \neq B$ **unless the block** B'' **comes as part of a proposal** $\langle \mathsf{Proposal}, r'', vr = r', B'' \rangle$ **and** v **observes** $2f + 1$ **round** $vr = r'$ **pre-votes for** B''. *Note that an honest validator never releases a lock permanently: it can acquire a higher lock at a higher round in the future, or it might temporarily drop the lock to vote for block $B'' \neq B$ in the example above.*

Why two voting stages? A single vote per round allows an adversary to equivocate by withholding votes: honest validators could end up committing two conflicting blocks if they switch votes between rounds. With a two-stage protocol and locking, once more than 2/3 have prevoted a block, they lock and cannot compromise safety by switching to another block without first seeing a supermajority prevote for another block, which cannot happen if faulty power is limited to $f < n/3$.

3 Tendermint: security properties

Tendermint has three main security properties.

- **Safety.** No two honest validators ever commit conflicting blocks, regardless of network delays, if at most $f < n/3$ validators are Byzantine. Safety holds even during periods of asynchrony.
- **Liveness.** Transactions are eventually committed, provided the network eventually becomes synchronous, and at most $f < n/3$ validators are Byzantine.
- **Finality.** Tendermint is safe even without any assumptions on network delay. Even when the network is asynchronous, it is still safe (Safety under asynchrony is called finality).
- Tendermint is safe and live, i.e., secure, under the partially synchronous network model.

Even if the network never regains synchrony, Tendermint will never violate safety—it simply stops making progress until messages are delivered in a timely fashion. In other words, Tendermint sacrifices liveness under prolonged asynchrony but always preserves the guarantee that committed blocks cannot be undone. This "safety-first" approach is deliberate: temporary pauses in block production are far less harmful than a safety failure, which could irreversibly corrupt the ledger or cause financial loss. By contrast, networks like Solana occasionally suffer brief liveness outages with minimal impact, but any breach of safety would be catastrophic. In blockchain design, ensuring that once a transaction is final it remains final is more important than ensuring the chain never pauses.

4 History of Models/consensus

In the early development of distributed consensus, Lamport and colleagues initially formulated protocols under the synchronous model, leveraging known timing bounds to detect failures. However, real-world networks generally don't conform to these precise timing assumptions, which motivated a shift towards asynchronous designs. The issue with these designs was that they were too complex. For a partial compromise, Dwork, Lynch, and Stockmeyer introduced the partially synchronous model in 1988, with

1. Global Stabilization Time: There exists some known time GST after which the network behaves well and every message is delivered within a known bound Δ.
2. Safety Always: Even before GST, when delays may be arbitrarily long, no two non-faulty processes can decide conflicting values.
3. Liveness after GST: Once the network's delay falls below Δ post-GST, the protocol's timeout mechanisms and leader-rotation ensure that consensus is eventually reached.

Crucially, Tendermint achieves security under the partial synchrony model.

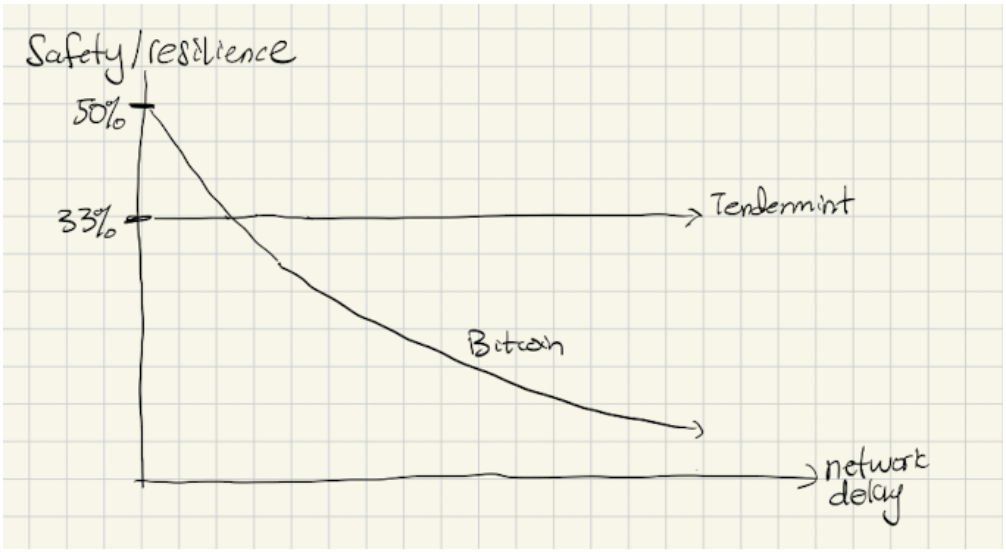

Figure 1: Safety vs Network Delay

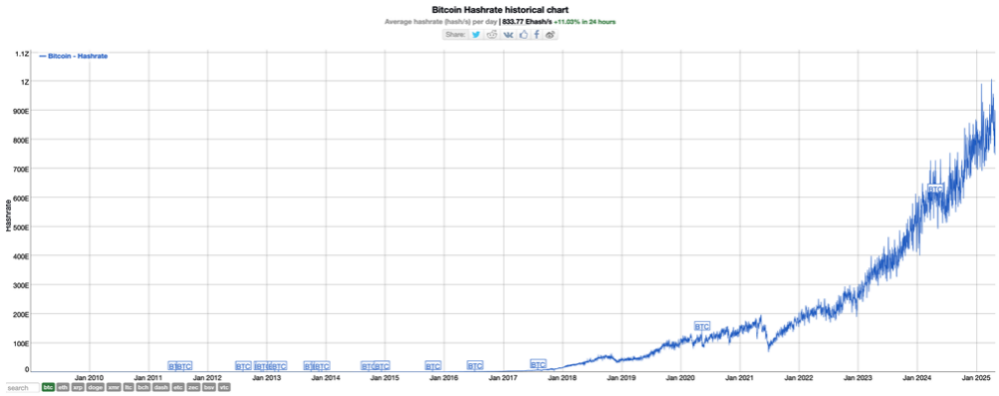

Figure 2: Bitcoin exhibits Dynamic Availability

5 Tendermint vs Bitcoin

In contrast to Tendermint's BFT design, where liveness stalls if more than one-third of validators go offline, Bitcoin's proof of work chain simply slows down and keeps growing whenever miners drop out. For example, when China's 2022 crackdown forced a large fraction of the global hash rate offline (see figure), Bitcoin didn't halt; blocks continued to be found at a reduced pace until difficulty readjusted downward. Because Bitcoin assumes that offline miners aren't actively malicious, any decline in hashing power only impacts block production rate, not safety or eventual confirmation. This resilience, where the ledger remains live despite shifts in the active miner set, is **availability under dynamic participation**.

This doesn't happen in Tendermint. Tendermint assumes that any validator whose votes don't arrive in time may be Byzantine and treats silence as a potential attack. Validators will refuse to

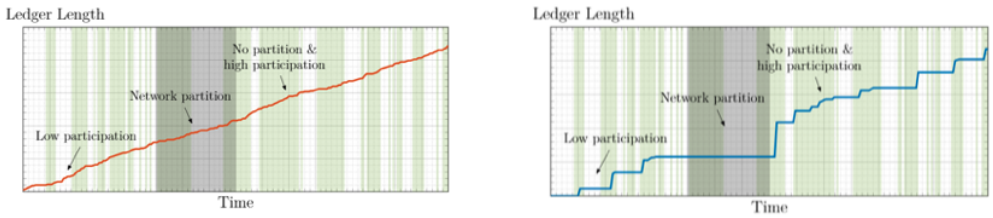

Figure 3: Availability (left), Finality (right)

recommit any block unless they say $\geq 2/3$ prevotes for it, and once locked on a block, they won't switch to another without the proper unlock conditions. This prioritizes safety over liveness and avoids ever operating under fully asynchronous assumptions.

Thus, Bitcoin and Tendermint are designed under essentially opposite paradigms.

6 Availability-Finality Dilemma

Bitcoin is dynamically available but is not final. Tendermint has finality, but is not dynamically available. Can we get the best of both worlds?

It appears the answer is no. If one pursues availability, we keep extending the chain regardless of offline peers. But if those absent validators later rejoin and build on a conflicting fork, we have no way to declare which history is final. If we are to pursue finality, we refuse to commit any block unless already sure no one can later override it. But if validators vanish or the network partitions, we can't reach that guarantee, so the ledger simply stops; in particular, availability is missing. These two goals rest on mutually incompatible assumptions about network participation.

This is precisely the availability-finality Dilemma.

7 Finality Gadgets

Ethereum's shift to PoS adds finality by layering a "finality gadget" (called **Casper FFG**) on top of GHOST. In the **Gasper** design, we first run *LMD GHOST* to extend the available chain, thus preserving dynamic availability. Then, every 32 slots, validators vote in Casper FFG on the newest checkpoint. Once $\geq 2/3$ of active stake attests to a checkpoint, it becomes final and cannot be overridden. By decoupling, we get

- Availability (LMD GHOST): The chain never stops growing.
- Finality (Casper FFG): A stable prefix of the chain is irrevocably locked in once enough votes arrive.

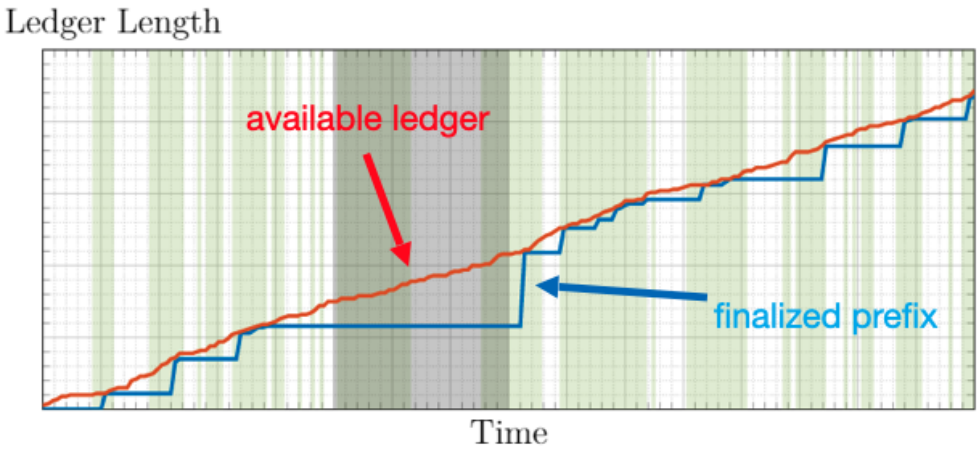

Figure 4: ETH PoS Visual

8 Ebb and Flow Protocols

Doesn't the above contradict the availability-finality dilemma?

Gasper resolves the dilemma with two synchronized views of the ledger.

1. Available Chain (Flow): GHOST's head rule lets blocks keep flowing—even during network partitions or low participation, the chain extends continuously.
2. Finalized Prefix (Ebb): Casper FFG checkpoints only when $\geq 2/3$ votes arrive. During partitions or when stake drops, finality ebbs (halts), but as soon as the network heals or participation rises it flows again.

Early security gaps in PoS checkpointing were partially addressed in the Gasper paper; however, the original Gasper paper did not come with a complete security proof, and formal guarantees remain an area of ongoing research. The protocol went live with **The Merge** on September 15, 2022. Today, over 34 M ETH is staked under Ethereum PoS. In practice, it is much easier to launch PoS chains, since PoW chains require miners.

References

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

[2] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[3] J. Kwon. Tendermint : Consensus without mining. 2014.

[4] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[5] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Electronic paper, 2008.

[6] J. Neu, E. N. Tas, and D. Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.