Tendermint Protocol

Instructor: Nitin Awathare

September 17, 2024

In the previous lecture we have moved to the partially synchronous network model, which is more realistic match to the real network, now a days. There we have discussed about a impossibility proof. In this lecture we will discuss about one possibility result in terms of the Tendermint protocol.

While Tendermint is a modern protocol designed for blockchains, it closely resembles iterated versions of classical Byzantine agreement protocols from the 1980s and 1990s. We will first outlines three key ideas of the protocol, detailed of which would be presented in next section. While the concepts may seem intuitive, turning them into a precise protocol is tricky, as most approaches would fail consistency or liveness. In distributed protocols, intuition can be misleading—rigorous, precisely defined protocols and formal proofs are essential.

- **Iterated Single-Shot Consensus:** The first idea is to reduce the multi-shot SMR problem to single-shot consensus. Tendermint repeatedly invokes a Byzantine agreement-type protocol for each block (that consists of transactions). Nodes first agree on block 1, then block 2, and so on, focusing on one block at a time. Each message is tagged with the current block number, and nodes ignore messages related to other blocks. A challenge arises in the asynchronous phase when some nodes work on different blocks due to message delays.
- **Aggressive Restarts:** The second idea focuses on the leader proposing a block, say block 9. Two issues could prevent honest nodes from agreeing: the leader might be Byzantine, sending inconsistent information, or messages could be heavily delayed before the global stabilization time (GST). To address this, nodes restart with the next leader if progress is slow. Once GST passes and the leader is honest, agreement on the block is expected. A key challenge is ensuring consistency when some honest nodes see progress while others don't, due to Byzantine behavior or delays, which leads to the next idea.
- **Two Stages of Voting:** The third key idea in the Tendermint protocol is the use of two voting stages for each block proposal. Why two stages? With a single vote, different honest nodes might see conflicting outcomes due to Byzantine nodes sending inconsistent messages or network delays. This could lead to some nodes committing to block 9 while others restart with a new leader, violating consistency.

  Two voting stages introduce a third possible outcome: first-stage success but second-stage failure. This "intermediate" result allows nodes to lock in on a potential block without fully committing, remaining flexible if new information suggests a better block to finalize. This approach mitigates the risk of nodes diverging in their local histories.

Given this high level idea and goals let's now jump into the details of Tendermint Protocol.

1 The Tendermint Protocol

There are various versions of the Tendermint protocol, but this lecture focuses on the simplest implementation that stays true to Tendermint's three core ideas (discussed above) while ensuring correctness. This simplified approach trades some efficiency for clarity—for instance, we won't optimize who sends messages to whom, assuming all messages are signed and broadcast to everyone, nor will we fine-tune the protocol's block production rate. While these factors are crucial for a highly efficient version of Tendermint, this choice allows for more straightforward proofs of consistency and liveness, which will discuss soon once we are done with the protocol description.

Tendermint relies on the PKI assumption, starting with a known list of nodes and their public keys, with each node knowing its private key. While the PKI assumption isn't required for optimal fault tolerance in the partially synchronous model (as shown by Dwork, Lynch, and Stockmeyer), it is convenient and natural in a blockchain context. Furthermore, in the partially synchronous model, all nodes always know the current time step and round number, even during the asynchronous phase. There is a known upper bound, Δ, on message

delays after GST. Tendermint rounds last 4Δ time steps, with each round being one attempt to agree on a block. The protocol relies on the known value of Δ but not on the unknown global stabilization time.

Nodes will be voting on blocks. Such a vote has five attributes:

- The identity of the voter (as proved via a cryptographic signature that could only have be created by that node);
- The block (i.e., ordered sequence of not-yet-executed transactions) that the vote is for;
- The block number (e.g., block 9);
- The round number (e.g., round 117);
- The voting stage (first or second). Note that Tendermint works in two stages.

These attributes ensure votes are linked to specific nodes and blocks, annotated by block number, round, and stage, to avoid interference and ensure proper handling of multiple consensus attempts. The output of each voting stage, s, is a Quorum Certificate (QC), stated in Definition 1.1. Given this, A quorum certificate (QC) can be described as a triplet (h, r, s), where h is the block being certified, r is the round number, and s is the voting stage within that round. In other words, A QC signifies supermajority support for a specific block in a given referendum, effectively representing a "successful vote" for that block.

Definition 1.1. (Quorum Certificate (QC)) A quorum certificate (QC) is a set of votes from at least $2n/3$ distinct voters, all supporting the same block (h) in the same round (r).

Each QC has some properties associated with it, which we will discuss next:

Every pair of QCs overlaps in at least n/3 nodes. - Let Q_1 and Q_2 be two quorum certificates. Since at most $n/3$ nodes are not in Q_1 and similarly for Q_2, at least $n-(n/3)-(n/3) = n/3$ nodes are in both Q_1 and Q_2. So, if $f < n/3$, then every pair of QCs overlaps in at least one honest node. Honest nodes in Tendermint vote at most once per referendum. Byzantine nodes may double-vote, but with $f < n/3$, they cannot create quorum certificates (QCs) for two different blocks in the same referendum. This ultimately lead to the conclusion that, if Q_1 and Q_2 are QCs for the same referendum, Q_1 and Q_2 support the same block.

In the Tendermint protocol, each honest node maintains two local variables: a block B_i and a quorum certificate Q_i supporting B_i. Initially, when a node begins working on a new block number, it sets Q_i to null and B_i to the unexecuted transactions it knows about, ordered arbitrarily. B_i represents the node's current belief about the next block.

Nodes may replace an old QC with a newer one for the same block number. Specifically, a QC Q_1 from referendum (h, r_1, s_1) is more recent than a QC Q_2 from referendum (h, r_2, s_2) if: (i) $r_1 > r_2$(later round) or (ii) $r_1 = r_2$ and $s_1 > s_2$ (later stage). Any non-null QC is considered more recent than a null QC.

1.1 Protocol Pseudocode

As discussed earlier, each Tendermint round lasts 4Δ time steps, where Δ is the max message delay after GST. Honest nodes act only at time steps that are multiples of Δ, though messages may arrive between these steps. Each round has 4 phases, explained below in pseudocode and later in detail. The pseudocode applies to a node i working on block number h_i, ignoring messages about different block numbers (with one small exception). Every message must be signed, and any implausible message—e.g., unsigned messages, proposals from non-leaders, or multiple proposals from the same leader—is discarded by honest nodes.

Let's break down the pseudocode, in Figure 1 governing the behavior of honest nodes. Consider an honest node $i$working to determine block h_i, having already finalized blocks 1 through $h_i - 1$. At the start of round r (i.e., time step $4\Delta r$, assuming rounds are numbered from 0), node i knows both the current round r and the leader l for that round, as we're in a permissioned, partially synchronous setting with a shared global clock and a predefined leader rotation (e.g., round-robin based on node IDs).

First Phase The first phase applies only if node i is the round-r leader l. In this case, i proposes a block to the other nodes (similar to the Byzantine broadcast sender). It first checks for any more recent height-h_i QCs from other nodes, updating its local QC Q_i and block B_i if needed. Then, node i broadcasts a proposal for block B_i, including the supporting QC Q_i, to all nodes (including itself). The message is signed and includes the block number h_i and the round number r.

Second Phase In the second phase, non-leaders process block proposals from the leader. Each node i listens during time steps $4\Delta r$ to $4\Delta r + \Delta$ for a block-QC pair from the round-r leader l, ignoring multiple proposals or those from non-leaders. If no valid pair is received by $4\Delta r + \Delta$, node i does nothing. If a pair (B_l, Q_l) is

(A Version of) the Tendermint Protocol

Assumptions: local node i is working on block number h_i, with local variables B_i and Q_i (initialized as in Section 4.2). All messages for other block numbers are ignored (with one exception, see "in the background" below). Current round is r. Leader of round r is ℓ.

// First phase (executed at time $t = 4\Delta r$)

1 **if** $i = \ell$ **then** // local node is the current leader

2 **if** ℓ has received a height-h_i QC more recent than (B_ℓ, Q_ℓ) **then**

3 $B_\ell := B_j$, $Q_\ell := Q_j$ // update accordingly, to the most recent one (B_j, Q_j)

4 broadcast (B_ℓ, Q_ℓ) to all nodes // annotated with h_i, r, signature

---

// Second phase (executed at time $t = 4\Delta r + \Delta$)

5 **if** i has received (B_ℓ, Q_ℓ) from ℓ **then** // must be signed by ℓ

6 **if** Q_ℓ at least as recent as Q_i **then** // out-of-date, need to update

7 $B_i := B_\ell$, $Q_i := Q_\ell$

8 broadcast (B_i, Q_i) // keep all nodes up-to-date

9 broadcast first-stage vote for B_i // annotated with h_i, r, signature

// (no vote cast if (B_ℓ, Q_ℓ) not received on time)

// Third phase (executed at time $t = 4\Delta r + 2\Delta$)

10 **if** i has received at least $\frac{2}{3}n$ round-r (first-stage) votes for a block B **then**

11 $B_i := B$ // might or might not change the value of B_i

12 $Q_i :=$ the votes above // constitute a round-r stage-1 QC

13 broadcast (B_i, Q_i) // keep all nodes up-to-date

14 broadcast second-stage vote for B_i // annotated with h_i, r, signature

// (no vote cast if no round-r stage-1 QC is received on time)

// Fourth phase (executed at time $t = 4\Delta r + 3\Delta$)

15 **if** i has received at least $\frac{2}{3}n$ round-r second-stage votes for a block B **then**

16 $B_i := B$ // might or might not change the value of B_i

17 $Q_i :=$ the votes above // constitute a round-r stage-2 QC

18 broadcast (B_i, Q_i) // keep all nodes up-to-date

19 commit B_i to local history as block number h_i // worry: consistency?

20 increment h_i // next round, will start working on the next block

21 reset B_i to the known as-yet-unexecuted transactions (ordered arbitrarily)

22 reset Q_i to null

// (no block committed if no round-r stage-2 QC received on time)

// Addendum (at time $t = 4\Delta r + 4\Delta$, just before first phase of round $r + 1$)

// (Catch up with all future blocks that have already been decided upon)

23 **while** i is in possession of a height-h_i stage-2 QC Q for a block B **do**

24 carry out lines 18–22 from the fourth phase (with (B, Q) playing the role of (B_i, Q_i))

// In the background (at all times)

25 store all QCs received for blocks $h_i + 1, h_i + 2, \ldots$ // for use in lines 2 and 23--24

Figure 1: Tendermint Psuedocode

received and Q_l is at least as recent as Q_i, node i updates its B_i and Q_i to match (B_l, Q_l) and broadcasts these values with a first-stage vote. If Q_i is more recent, i ignores the proposal and skips the vote.

Third Phase In the third phase, nodes determine the outcome of the second-phase referendum. If a node receives a supermajority (at least two-thirds of votes) for block B by time step $4\Delta r + 2\Delta$, it considers B the winner. The node updates its local block B_i and forms a QC for B, adopting it as the new Q_i, and broadcasts (B_i, Q_i) along with a second-stage vote. If no supermajority is reached by $4\Delta r + 2\Delta$, the node does not cast a second-stage vote.

Fourth Phase In the fourth phase, nodes check for a supermajority of stage-2 votes for the same block B by time step $4\Delta r + 3\Delta$. If found, the node adopts B, assembles the votes into a stage-2 QC, updates B_i and Q_i, and broadcasts them. Since B passed two stages, the node commits to B as block h_i, updates its history, and begins working on block $h_i + 1$, resetting B_i and Q_i appropriately. If no supermajority is received, the node does nothing in this phase.

An honest node i working on block h_i ignores messages about earlier or future blocks, except when it receives a QC for a future block $h > h_i$. It saves these QCs for later use, potentially when i becomes the leader. At the end of each round, node i processes saved stage-2 QCs for future blocks, committing to them only after committing to all previous blocks. If a block is already agreed upon, node i skips its consensus process and directly adopts the block.

It's not immediately clear that Tendermint ensures consistency or eventual liveness with fewer than one-third Byzantine nodes, so careful proofs are essential. This highlights how protocol analysis informs design—intuition alone isn't enough for distributed protocols, as key details require precise mathematical reasoning. Will discuss the mathematical reasoning soon.

References:

1. Foundations of Blockchains `https://timroughgarden.github.io/fob21/`.
2. Blockchain gets better: moving beyond Bitcoin
`https://www.comp.nus.edu.sg/features/2018-blockchain-gets-better/`

Disclaimer :

Some of the content and/or images in these notes have been directly sourced from the books and/or links cited in the references. These notes are exclusively utilized for educational purposes and do not involve any commercial benefits.