Foundations of Blockchains
Lectures #7: The Tendermint Protocol (ROUGH DRAFT)[*]

Tim Roughgarden[†]

1 The Upshot

1. Lecture 6 showed that consistency and eventual (post-GST) liveness cannot be achieved by any SMR protocol in the partially synchronous model if $f \geq n/3$.
2. The Tendermint protocol is a widely used consensus protocol that achieves consistency and eventual liveness provided $f < n/3$.
3. The first high-level idea in Tendermint is to carry out iterated single-shot consensus (with one consensus instance per block).
4. The second high-level idea is to restart after a timeout if there is insufficient progress due to a Byzantine block proposer or delayed messages.
5. The third high-level idea is to use two stages of voting to enable an intermediate outcome, between success and failure.
6. The Tendermint protocol uses rotating leaders, with one leader per round.
7. Each round has four phases: a block proposal phase (with the leader proposing); a first stage of voting (on the leader's proposal); a second stage of voting (on the outcome of the first stage of voting); and a commit phase (for nodes that witnessed successful outcomes of both voting stages).
8. A voting stage is successful if a supermajority (two-thirds) of nodes all vote for the same block. Their votes constitute a quorum certificate (QC).

[*]©2021–2022, Tim Roughgarden.

[†]Email: `tim.roughgarden@gmail.com`. The preparation of these notes was partially supported by a seed grant from the Columbia-IBM Center for Blockchain and Data Transparency. Thanks to Ittai Abraham and Andrew Lewis-Pye for helpful discussions related to this lecture. (All errors and omissions are, of course, the sole responsibility of the author.) These notes are a work in progress; corrections and feedback are welcome.

9. Tendermint satisfies consistency (when $f < n/3$) because an honest node commits to a block B only if it sees a successful second-stage vote, which happens only if more than $n/3$ honest nodes cast a first-stage vote for B. These honest nodes lock in on the block B and prevent the assembly of QCs for any blocks other than B.

10. Tendermint satisfies eventual liveness (when $f < n/3$) because, whenever there are two consecutive rounds post-GST with honest leaders, the second round concludes with all honest nodes committing a recently and honestly assembled block.

11. More recently developed consensus protocols use additional ideas to improve on the efficiency (e.g., latency, communication complexity, or responsiveness) of Tendermint without sacrificing optimal fault-tolerance in the partially synchronous model.

2 The Story So Far

Main result of lecture. This lecture provides a possibility result for state machine replication (SMR) in the partially synchronous model that matches (in terms of the number of Byzantine nodes tolerated) the impossibility result that we proved in Lecture 6. As a bonus, we'll use a well-known blockchain protocol—the Tendermint protocol—to prove the result.[1]

The partially synchronous model. The synchronous model (Lectures 2 and 3) was great for proving strong positive results (like the Dolev-Strong protocol for Byzantine broadcast) but made unrealistically strong assumptions about guaranteed message delivery. The asynchronous model (Lectures 4 and 5) made pleasingly minimal assumptions but, as shown by the FLP impossibility result, throws out the baby with the bathwater, with no good (deterministic) consensus protocols possible. This led us to the "sweet spot" partially synchronous model, which explicitly declares some periods of time "normal" (and thus well-modeled by the synchronous model) and other periods "under attack" (and thus appropriately modeled by the asynchronous model). Like the synchronous model, the partially synchronous model assumes a global shared clock.[2] The basic version of the model involves two parameters that constrain message delivery. The first is a parameter Δ that specifies the maximum delay (in time steps) that a message might suffer *in the synchronous phase.* This parameter is known up front, in the sense that the protocol's description (e.g., the length of a timeout) can depend on its value. The second is the global stabilization time (GST), which is the point at which the communication network transitions from the asynchronous setting to the synchronous setting. Precisely, a message sent at time step t is guaranteed to arrive by time step $\max\{t, GST\} + \Delta$. The GST is not known up front and can be arbitrarily large (but

[1]For example, the Terra and Cosmos networks are both built on top of Tendermint consensus. More generally, Tendermint is one of the most popular "off-the-shelf" consensus solutions for up-and-coming projects that are uninterested in rolling their own consensus protocol.

Bitcoin and (the original proof-of-work version of) Ethereum, meanwhile, use longest-chain consensus. We'll discuss this genre of consensus protocols at length in Lecture 8.

[2]Or, at least, clocks that drift at a rate no faster than an priori known bound.

finite)—it's the protocol's responsibility to provide guarantees no matter what GST might be.

Recall from Lecture 6 the traditional goals for a consensus protocol in the partially synchronous model:[3]

Traditional Goals in the Partially Synchronous Model

1. *Safety:* Safety holds always, even in the asynchronous phase (i.e., pre-GST).

2. *Eventual liveness:* Not long after the GST, safety and liveness both hold.[4]

Let's also remember the specific safety and liveness conditions used in the SMR problem (defined way back in Lecture 2). First, recall that in the SMR problem, clients submit transactions to a network of nodes that are responsible for running a consensus protocol, with each node maintaining a local append-only sequence of transactions (its "history"). Safety and liveness then mean:

Goal #1: Consistency. No two nodes ever disagree on the relative order of two different transactions. (Ideally they would stay perfectly in sync, but we allow some nodes to fall behind as long as they eventually catch up with the others.)

Goal #2: Liveness. Every transaction submitted to at least one node is eventually added to every node's local history. (Actually, in this and the next lecture we'll study a slightly weaker notion of liveness, see Section 6 for details.)

Last lecture we stated the following (where, as usual, n denotes the number of nodes and f the number of Byzantine nodes):[5]

Theorem 2.1 ([?]) *There exists a deterministic protocol for the SMR problem that satisfies consistency and eventual (post-GST) liveness in the partially synchronous model if and only*

[3]The FLP impossibility theorem tells us that we can't hope for both safety and liveness during the asynchronous phase.

[4]How long, exactly? There is interesting research on that question (see e.g. [?]), but we won't have time to discuss it. For our purposes, think of the number of time steps needed as bounded above by some polynomial function of Δ and possibly also the number of nodes n.

[5]Actually, in Lecture 6 we stated this result for the (single-shot) Byzantine agreement (BA) problem rather than the (multi-shot) SMR problem, but the same result holds true either way. The proof of the "only if" impossibility result in that lecture (for the BA problem) can be reworked to apply also to the SMR problem (as you should verify). For the "if" direction (proved for the SMR problem in this lecture), in the $f < n/3$ regime that we're interested in, the BA problem reduces to the SMR problem and so the possibility result translates immediately from the latter to the former. (A good exercise: show how to build, given an SMR protocol that guarantees consistency and liveness in the partially synchronous model (with $f < n/3$), a BA protocol that satisfies termination, agreement, and validity in the partially synchronous model (with the same f). Feel free to use the PKI assumption.)

if $f < n/3$.[6]

Theorem 2.1 holds with or without the PKI assumption. In this lecture we'll prove the "if" direction under the PKI assumption, but the same positive result can be achieved without it (see [?]).[7]

3 Tendermint: High-Level Ideas

While Tendermint is a 21st-century protocol [?] designed with blockchains in mind, the way that it works strongly resembles an iterated version of classical Byzantine agreement protocols from the 1980s and 1990s (such as in [?, ?]). This section introduces three of the main high-level ideas in the protocol, and the protocol is then detailed in Section 4.

The three ideas outlined in this section will probably seem reasonably natural to you, but keep in mind that there are many different ways in which you could turn those ideas into a precisely defined protocol. And the devil is in the details: most approaches would produce a buggy protocol that fails consistency, eventual liveness, or both.

In some areas of computer science, intuition tends to be a good guide. The design of distributed protocols is not one of those cases. You should be automatically suspicious of any proposal that is not precisely described (as we do for Tendermint in Section 4) or that does not come equipped with rigorous proofs of its alleged safety and liveness guarantees (as in Sections 5 and 6 for the Tendermint protocol).

3.1 Idea #1: Iterated Single-Shot Consensus

The first idea is to reduce the multi-shot SMR problem that we care about to single-shot consensus. Tendermint will basically iterate a Byzantine agreement-type protocol over and over again, with one invocation for each block of transactions (i.e., a batch of transactions, sequenced in some way). That is, nodes will agree on block #1 (via single-shot consensus), then on block #2 (ditto), and so on. Each node will be working single-mindedly on one block at a time.[8] Every message that a node sends will be annotated with the block number that its currently working on, and nodes will ignore all messages that concern any past or future blocks (with one small exception, explained below).

Looking ahead, one challenge will be that, in the asynchronous phase, different nodes may be participating in different invocations of the single-shot protocol—some nodes may

[6]Remember, whenever you hear a blockchain person or whitepaper refer to a "33% threshold"—generally to justify why a protocol isn't more robust to attacks than it already is—this theorem is where it comes from.

[7]Recall from Lecture 2 that, in the synchronous model with the PKI assumption, there is an SMR protocol that satisfies consistency and liveness even when 99% of the nodes are Byzantine. Thus Theorem 2.1 formally separates the synchronous and partially synchronous models, with some tasks achievable in the former but unachievable in the latter.

[8]Many BFT-type blockchain protocols that followed Tendermint pipeline these single-shot instances to achieve lower latency. For example, if there are two stages of voting per block (see Section 3.3), a node might be permitted to cast a second-stage vote for block #7 and a first-stage vote for block #8 at the same time.

be working to figure out block #9 (having already figured out the first eight blocks) while other nodes, to whom key messages have been massively delayed, may still be working on an already-decided-upon block like block #7.

3.2 Idea #2: Aggressive Restarts

To explain the second idea, zoom in on a particular block, say block #9. As in the Byzantine broadcast problem and its application in SMR protocols (see Lecture 2), there will be a "leader" node responsible for proposing a block to other nodes. There are two obvious issues that could interfere with honest nodes reaching agreement about block #9. The first is that the leader could be a Byzantine node, sending inconsistent information to different honest nodes. The second is that, before the global stabilization time, honest nodes' messages could be massively delayed. The solution is for the nodes working on block #9 to restart (with the next leader) after a short period of time if they don't see sufficient progress. The hope (which we'll formalize in Section 6) is that eventually—once GST has passed and the current leader is honest, if not earlier—the honest nodes will be able to come to agreement on block #9 and move on to block #10. One key challenge is that, due to inconsistent behavior by Byzantine nodes and/or big message delays, some but not all honest nodes may "see sufficient progress" by the restart time, and this should raise red flags about possible consistency violations. This issue is addressed by the last high-level idea.

3.3 Idea #3: Two Stages of Voting

The third and most clever idea in the Tendermint protocol is to use two stages of voting in each attempt by honest nodes to come to agreement on the next block.

Why do we need two stages of voting rather than one? The issue, mentioned above, is that different honest nodes may perceive different outcomes of a vote. For example, for a simple majority vote with the two options "commit to block B" and "try again with a new leader," some honest nodes may see over 50% of the votes for the first option and the other honest nodes over 50% supporting the second option. Again, each of two separate forces is powerful enough to cause different honest nodes to see different vote counts: inconsistent messages from Byzantine nodes (sending votes for one option to some honest nodes and the other option to the rest) and (at least pre-GST) big message delays (with some honest votes arriving on time and some delayed indefinitely).

Presumably nodes that perceive the vote as a success will perform one action (e.g., commit the proposed block as block #9 to their local history) while the others perform a different action (e.g., restart with the next leader). This could easily lead to a violation of consistency (e.g., if the next leader proposes a different block for block #9 and the as-yet-uncommitted nodes agree to it).

Fundamentally, the problem with one stage of voting is that there are only two possible outcomes, and so disagreeing honest nodes perceive "diametrically opposite" outcomes. With two stages of voting—and with the second-stage vote taken only if the first-stage vote succeeds—there are three possible outcomes (first stage fails, first stage succeeds but second stage fails, and both stages succeed). While it's still the case that different honest nodes may observe different outcomes, the hope is that the "intermediate" outcome allows an honest node to hedge its bets between the more extreme outcomes of committing to a block and restarting the agreement process with a brand-new block. Intuitively, if an honest node observes a first-stage voting success and a second-stage failure, it will "lock in" on a block that it's pretty sure will wind up being block #9, but without irrevocably committing to it. The node remains open to subsequent arguments that it should lock in on a different block instead, but will only do so if confronted with overwhelming evidence that it is behind the times and needs to catch up with the most recent information.

4 The Tendermint Protocol

4.1 Preliminaries

Fidelity of description. There are many variants of the Tendermint protocol. The variant described in this lecture is meant to be the most straightforward possible implementation that is faithful to the three big ideas in Tendermint (Section 3), subject to actually being correct. This choice means sacrificing some amount of efficiency—for example, we won't worry about the details of who sends which messages to whom (every message will be signed by its sender and broadcast to everybody) or about optimizing constant factors in the protocol's rate of block production. (Of course, these details can be important for an efficient concrete implementation of Tendermint.) Another consequence is that the proofs of consistency and liveness (in Sections 5 and 6, respectively) will not be as slick as for some other variants.

PKI assumption. The Tendermint protocol makes use of the PKI assumption, and begins with a commonly known list of nodes and their public keys. (Each node is assumed to know its own private key.) The PKI assumption is not necessary to achieve optimal fault-tolerance in the partially synchronous model (as originally shown by Dwork, Lynch, and Stockmeyer [?]), but it is a convenient and reasonably natural assumption in a blockchain setting.[9]

Rounds. In the partially synchronous model (see Section 2), all the nodes know and agree on at all times the current time step (with no communication needed), even in the asynchronous phase. Also, there is a known upper bound Δ on the maximum number of time steps that a message might get delayed during the synchronous phase (after GST).[10] In Tendermint, a *round* corresponds to 4Δ consecutive time steps, and represents one attempt at reaching agreement on a block before restarting (see Section 3.2). The first round starts at time 0 and ends at time 4Δ, the second starts at time 4Δ and ends at time 8Δ, and so

[9] It is especially natural for proof-of-stake blockchains (see Lecture 12), which typically require registration of a public key as a prerequisite to contributing to block production. "BFT-type" consensus protocols like Tendermint generally show up in such proof-of-stake blockchains.

[10] In a typical Tendermint instantiation, Δ might be set to roughly one second. (You can think of each time step as one millisecond, for example, in which case this choice corresponds to $\Delta = 1000$.)

on. Because all nodes always know the current time step (and the length of each round), all nodes always know the current round number (even in the asynchronous phase).[11] The protocol description depends on the value of the parameter Δ, and this is allowed because the value Δ is assumed to be known a priori. (By contrast, the protocol description *cannot* depend on the (unknown) global stabilization time.)

Rotating leaders. Each round will have a unique leader node responsible for proposing a block, with nodes taking turns as the leader. This plan should remind you of our SMR protocol in Lecture 2 for the synchronous model, with each leader acting as the sender in a Byzantine broadcast subroutine such as the Dolev-Strong protocol. (This lecture, without the benefit of synchrony, the Tendermint protocol can't afford to wait for a round to conclude with agreement on a block and instead restarts after a timeout.) Because the names of the nodes running the protocol and the current round are common knowledge (as we're in the permissioned model with a shared global clock), all nodes always know who the current leader is (e.g., node 1 in the first round, node 2 in the second round, and so on).

4.2 Quorum Certificates (QCs)

This section highlights a crucial definition (which is common to most "BFT-type" consensus protocols, not just Tendermint).

Votes. As suggested in Section 3.3, nodes will be voting on blocks. Such a vote has five attributes:

Attributes of a Vote

1. the identity of the voter (as proved via a cryptographic signature that could only have be created by that node);
2. the block (i.e., ordered sequence of not-yet-executed transactions) that the vote is for;
3. the block number (e.g., block #9);
4. the round number (e.g., round #117);
5. the voting stage (first or second).

Accordingly, you can think of a vote as a 5-tuple (i, B, h, r, s). (Here "h" stands for "height," which is synonymous with the block number.)

The first two attributes should be self-explanatory—votes are for specific blocks, and to avoid ballot-stuffing the protocol must be able to associate each vote with a specific node.

[11]By contrast, staying in sync on the current block number *does* require communication between nodes (e.g., sending and receiving votes), and for this reason different nodes may be working on different block numbers during the asynchronous phase (see also Section 3.1).

(Note that we're taking advantage of the PKI assumption here.) Section 3 foreshadowed the other three attributes. Because different instances of single-shot consensus shouldn't interfere with each other and different nodes may be working on different blocks (Section 3.1), votes should be annotated with the block number. Because reaching agreement on a given block may require multiple restarts (Section 3.2), a vote must be explicitly associated with the round in which it was cast. Finally, because each attempt at agreement (in a given round, for a given block) requires two stages of voting (Section 3.3), a vote must indicate which stage it belongs to.

Definition of QCs. Call a triple (h, r, s) a *referendum* (on the outcome of stage s of voting within round r and for block number h). Here's the key definition (where as usual, n denotes the number of nodes, which is common knowledge in the permissioned setting):

Definition 4.1 (Quorum Certificate (QC)) A *quorum certificate (QC)* is a set of votes from at least $\frac{2}{3}n$ distinct voters that are all for the same block in the same referendum.

In other words, all the votes in a QC agree in their last four components (and take on at least $\frac{2}{3}n$ different values in the first component). A QC represents a supermajority of support for a specific block (in a specific referendum). We will sometimes say that the QC *supports* that block. We can interpret a QC as a "successful vote" (for the supported block in the given referendum).

Properties of QCs. Next is a simple but important lemma about QCs, which will also mark the first entrance of the famous "33%."

Lemma 4.2 (QC Overlap Property) *Every pair of QCs overlaps in at least $n/3$ nodes.*

Proof: Let Q_1, Q_2 denote a pair of QCs. By definition, at most $n/3$ nodes are not represented in Q_1. Similarly, at most $n/3$ nodes are not represented in Q_2. This leaves at least $n-(n/3)-(n/3) = n/3$ nodes that must be represented in both Q_1 and Q_2. ■

Lemma 4.2 is true no matter how many nodes are Byzantine (all it is is some simple counting). But something special happens once the fraction of Byzantine nodes drops below one third:

Corollary 4.3 (Two QCs Overlap in an Honest Node) *If $f < n/3$, then every pair of QCs overlaps in at least one honest node.*

How is this helpful? As we'll see, honest nodes in the Tendermint protocol will be instructed to vote at most once in each referendum. Byzantine nodes may engage in double-voting (sending votes for a block B to some honest nodes and for a different block B' to the rest), but with $f < n/3$, they will be unable to create QCs for two different blocks in the same referendum.

Corollary 4.4 (At Most One QC Per Referendum) *Suppose that every honest node votes at most once per referendum and that $f < n/3$. Then, if Q_1 and Q_2 are QCs for the same referendum, Q_1 and Q_2 support the same block.*[12]

Proof: By the previous corollary, some honest node i is represented in both QCs. By assumption, that node does not vote more than once in the referendum. Thus, both QCs must support the same block (whichever block node i voted for in that referendum). ∎

Newer and older QCs. In the Tendermint protocol, every (honest) node i will maintain two local variables, a block B_i and a QC Q_i that supports B_i. (One exception: when a node first starts working on a new block number, it sets Q_i to null and B_i to the as-yet-unexecuted transactions that it knows about, ordered arbitrarily.) Intuitively, B_i indicates node i's current belief about what the next block should be.

As we'll see, a node may cast aside an old QC in favor of a new one (for the same block number). Precisely, a (non-null) QC Q_1 with referendum (h, r_1, s_1) is *more recent* than another (non-null) QC Q_2 with referendum (h, r_2, s_2) if: (i) Q_1 is from a later round (i.e., $r_1 > r_2$); or (ii) Q_1, Q_2 are from the same round but Q_1 is from a later stage (i.e., $r_1 = r_2$ and $s_1 > s_2$). Any (non-null) QC is considered more recent than a null value.[13,14]

4.3 Protocol Pseudocode

As mentioned earlier, each round of the Tendermint protocol consumes 4Δ time steps, where Δ denotes the maximum message delay following the global stabilization time. Every (honest) node will take actions only at time steps that are multiples of Δ (with messages possibly received in between consecutive such multiples). Accordingly, each round has 4 phases, which we'll explain first in pseudocode and then in English.

The pseudocode below is for a node i working on a particular block number h_i. The node ignores all messages received that are about different block numbers (with one small exception, detailed at the end of this section). Also, remember that every sent message should be signed by the sender. Messages that could not have plausibly been sent by an honest node are automatically ignored by honest nodes. This applies, for example, to messages missing a signature; round-r block proposals by any node that is not the leader of round r; and any message beyond what is expected (e.g., if a node hears more than one block proposal from the same leader, it ignores all but the first).

[12] They may differ in the specific identities of the $\geq \frac{2}{3}n$ voters, but this is harmless (as we'll see).

[13] For the edge case in which we're comparing two null values, it will be convenient to allow a node to replace either with the other.

[14] We're not worried about QCs that concern different block numbers, as they don't interact with each other. We're also not worried about the case in which $r_1 = r_2$ and $s_1 = s_2$, as Corollary 4.4 then guarantees that Q_1 and Q_2 support the same block (and thus may as well be the same QC).

(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 $\ell$ has received a height-$h_i$ QC more recent than $(B_\ell, Q_\ell)$ then
3          $B_\ell := B_j$, $Q_\ell := Q_j$    // update accordingly, to the most recent one $(B_j, Q_j)$
4      broadcast $(B_\ell, Q_\ell)$ to all nodes                  // annotated with $h_i, r$, signature

   // Second phase (executed at time $t = 4\Delta r + \Delta$)
5  if $i$ has received $(B_\ell, Q_\ell)$ from $\ell$ then                        // must be signed by $\ell$
6      if $Q_\ell$ 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 $(B_\ell, Q_\ell)$ 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
```

The round and leader are common knowledge. Let's walk through the pseudocode that dictates the behavior of the honest nodes. Consider an honest node i that is trying to figure out block number h_i (having already finalized and committed to blocks $1, 2, \ldots, h_i - 1$), and consider the beginning of a round r. (Which is time step $4\Delta r$, assuming that we start numbering rounds at 0.) Because we're working in the permissioned and partially synchronous setting (with a shared global clock), and assuming a commonly known leader rotation (e.g., round robin in order of the node ID), node i automatically knows the current round r and the identity of the leader ℓ of this round.

First phase. The first phase is relevant only if node i happens to be the round-r leader ℓ. In this case, the node is responsible for proposing a block to the other nodes (much as the sender in the Byzantine broadcast problem). Node i first checks if it has received any (height-h_i) QCs from other nodes that are more recent than its incumbent QC, and if so it updates its locally stored QC Q_i to the most recent one received (and the variable B_i to the block supported by Q_i). Node i then broadcasts a proposal for the block B_i (along with the supporting QC Q_i) to all nodes. (For ease of exposition, imagine that node i also sends such a message to itself, which arrives immediately.) Node i adds its signature to each of these messages, along with the block number h_i and the round number r.

Second phase. The second phase is meant for non-leaders to process any block proposals that might have been sent in the first phase. Each node i listens during the time steps $4\Delta r, 4\Delta r + 1, \ldots, 4\Delta r + \Delta$ for a block-QC pair sent by the leader ℓ of the current round. (If more than one is received, all but the first are ignored. Proposals by nodes other than ℓ are likewise ignored.) If no such pair is received by time step $4\Delta r + \Delta$ (due to a Byzantine leader or delayed messages), node i does nothing in the second phase. If node i receives such a pair (B_ℓ, Q_ℓ) in which Q_ℓ is at least as recent as its locally stored QC Q_i, the node jettisons its values of B_i and Q_i and replaces them with the newly received values B_ℓ and Q_ℓ. Node i does this whether or not the new block B_ℓ matches the old one B_i. In this case, the node also broadcasts the new values of (B_i, Q_i) to all nodes (to help them stay up-to-date), along with its first-stage vote for B_i.[15] (If Q_i is more recent than Q_ℓ, node i concludes that the leader ℓ must be out of date (or Byzantine), and so it sticks to its guns with the incumbent values of (B_i, Q_i) and skips this phase's referendum.)[16,17]

Third phase. In the third phase, nodes try to ascertain the outcome of the referendum that took place at the second phase of the round (without any bias from whatever vote they themselves cast in that referendum). If a node receives a supermajority—with at least two-thirds of the nodes represented, possibly including that node itself—of round-r stage-1 votes

[15] In a more efficient implementation, you could imagine this all-to-all communication step being replaced with an all-to-one step followed by a one-to-all step, with the leader responsible for collecting votes and then broadcasting the results. (This would reduce the number of messages in the phase from quadratic to linear, while possibly blowing up the size of each message.)

[16] Recall from Corollary 4.4 that, under our standing assumption that $f < n/3$, there can't be a "tie" in recency between Q_i and Q_ℓ—if they belong to the same referendum, they must support the same block.

[17] A good exercise is to show that Tendermint's consistency (Theorem 5.1) and eventual liveness (Theorem 6.2) properties remain true if honest nodes also vote for the leader's proposal (B_ℓ, Q_ℓ) whenever $B_\ell = B_i$ (even if Q_ℓ is less recent than Q_i).

for the same block B by the end of time step $4\Delta r + 2\Delta$, it considers B the conclusive winner of the referendum. (By Corollary 4.4, assuming $f < n/3$, there is at most one such winner.) Node i adopts block B as its locally stored block B_i and assembles the received votes into a QC supporting B. (By definition, these votes constitute a QC, seen first-hand, for block B in the referendum at the previous phase.) This is the most recent QC imaginable, so node i adopts it as the new value of Q_i. Finally, it notifies all the nodes by broadcasting (B_i, Q_i), along with a second-stage vote for B_i (annotated as usual with the block number h_i, the round number r, the stage number $s = 2$, and i's signature). If node i does not receive such a supermajority by the end of time step $4\Delta r + 2\Delta$, it does not cast a second-stage vote.

Fourth phase. In the fourth phase, nodes look for evidence of a successful referendum in the third phase (again, without any bias from whether they themselves participated in that referendum). If a node receives a supermajority of round-r stage-2 votes for the same block B by the end of time step $4\Delta r + 3\Delta$, it considers B the winner of that referendum. (Again, assuming $f < n/3$, there is at most one such winner. If node i does not receive such a supermajority by the end of time step $4\Delta r + 3\Delta$, it does nothing in this phase.) In this case, analogous to the third phase, node i adopts block B as its locally stored block B_i, assembles the received votes into a (seen first-hand and most recent possible) stage-2 QC supporting B, sets Q_i to this QC, and broadcasts (B_i, Q_i) to all the nodes. But now that B has survived not one but two stages of voting, node i takes the plunge and *permanently commits block B* as block number h_i in its local history. (Our proof of consistency in Section 5 will have to rule out the possibility of any other honest node ever committing any other block as block number h_i it its own history, even during the asynchronous phase.) Having committed to block number h_i, node i will switch to start working on block $h_i + 1$ in round $r + 1$, with its local variables B_i and Q_i reset appropriately (to the as-yet-unexecuted transactions the node knows about and null, respectively).

Final details. An honest node i working on block number h_i ignores messages about earlier blocks. It also ignores messages about future blocks, with one exception: whenever it receives a QC for a future block $h > h_i$ (as would be broadcast by an honest node in line 4, 8, 13, 18, or 24), it saves it in its back pocket for future use. First, these saved QCs potentially come in handy the next time i is a leader (they would be processed in lines 2–3). Second, at the end of each round, node i processes any saved stage-2 QCs that it has received for future blocks, subject to the constraint that it commits to a height-h block only after committing to blocks for all previous heights. (There's no point to participating after-the-fact in the corresponding single-shot consensus instances, so node i skips them and directly adopts the already-agreed-upon future blocks.)

Perspective. It's not at all obvious that the Tendermint protocol satisfies consistency or eventual liveness when less than a third of the nodes are Byzantine, and you should demand careful proofs of both those properties (as will be supplied in Sections 5 and 6). Frankly, it's hard to imagine coming up with this protocol without simultaneously writing out the proofs of its key properties (a great example of how protocol analysis can inform protocol design).

As we've said many times, intuition doesn't get you very far with distributed protocols—the devil is usually in the details, and getting those details right generally requires careful mathematical analysis.

5 Proof of Consistency

The goal of this section is to prove that the Tendermint protocol satisfies consistency, even in the asynchronous phase. The proof here sacrifices some degree of brevity in exchange for (relatively) easy line-by-line verifiability.

Theorem 5.1 (Consistency of the Tendermint Protocol) *In the Tendermint protocol, if $f < n/3$ and two honest nodes commit blocks B and B' to their local histories as the same block number h, then $B = B'$.*

Consistency rules out the epic fail of two honest nodes committing to two different versions of the same block number. In other words, as soon as a single honest node commits a block B to its local history as block number h, that block can be considered as "finalized"—no other honest node will ever consider any block other than B as the rightful occupant of block number h.

Theorem 5.1 *does* leave open the possibility that some honest nodes will be lagging behind others. The theorem promises that the tortoises, once they catch up to the hares, will reach exactly the same conclusions.

Proof of Theorem 5.1: We'll prove the theorem block by block, so zoom in on your favorite block number h. The plan is to prove that, if Q_1 and Q_2 are height-h stage-2 QCs (possibly from different rounds) supporting blocks B_1 and B_2, then $B_1 = B_2$. In other words, at no point in the protocol will there ever be two stage-2 QCs for a given block number that support different blocks. Because possession of a height-h stage-2 QC for a block B is a prerequisite for an honest node to commit to block B at the height h (as per the fourth phase of the pseudocode), this will imply that no two honest nodes ever commit to different blocks at height h.[18]

Let r denote the first round in which more than $n/3$ honest nodes contribute height-h stage-2 votes in support of a common block; denote this set of honest nodes by S and the block they voted for by B^*. (Because a QC requires votes from at least $\frac{2}{3}n$ distinct nodes and $f < n/3$, this event is a prerequisite for the creation of a height-h stage-2 QC. That is, no such QC could have possibly been produced at any round prior to r.) The basic idea is that the nodes of S, having seen at least one successful vote for B^*, will remain "locked in" on B^* (refusing to cast votes for other blocks) until confronted with convincing evidence that they're in the wrong; and because S is so big and a QC requires participation by so many nodes, such evidence can never be created.[19]

[18]You might be worried about honest nodes never committing to any block at height h; addressing that issue is the responsibility of the eventual liveness proof in Section 6.

[19]This is the payoff of two stages of voting, as promised in Section 3.3. Roughly, you can think of S as

Formally, we can complete the proof by showing that there will never be a height-h stage-2 QC that supports a block other than B^*. We proceed inductively. For the base case, there cannot be a QC for the referendum $(h, r, 2)$ that supports a block $B \neq B^*$—because $|S| > n/3$ and at least $\frac{2}{3}n$ nodes must contribute to a QC, such a QC would require participation from some (honest) node of S. That node would have then voted twice (once for B^* and once for B) in the same referendum $(h, r, 2)$. But as is evident from the protocol pseudocode, every honest node votes at most once in each referendum.

To continue the argument, let's take stock of where things stand at the end of round r and beginning of round $r + 1$ (at time step $4\Delta(r + 1)$):

1. By virtue of voting for the block B^* in the second stage of round-r voting (in line 14), every node of S was, at the start of time step $4\Delta r + 2\Delta$, in possession of a QC from the referendum $(h, r, 1)$ that supports the block B^*. (As usual, Corollary 4.4 then implies that every QC from this referendum must support B^*.)

2. At that same time, each such node $i \in S$ must have set its local variable B_i to B^* (and Q_i to its QC from referendum $(h, r, 1)$ that supports B^*); see lines 11 and 12.

3. By the base case argument, any QC that can be assembled from votes in the referendum $(h, r, 2)$ must support block B^*. Thus, no QC from this referendum can cause a node $i \in S$ to assign in line 16 its local variable B_i to anything other than B^*. (Such a node may or may not wind up updating Q_i in line 17 from a QC from referendum $(h, r, 1)$ to one from $(h, r, 2)$; if it does, it will wind up committing the block B^* to its local history in round r.)

Summarizing, at time $4\Delta(r+1)$: (i) for each $i \in S$ that has not already committed block B^*, $B_i = B^*$; (ii) for each $i \in S$ that has not already committed block B^*, Q_i is a QC supporting B^* from referendum $(h, r, 1)$ or later; (iii) every height-h QC from a referendum $(h, r, 1)$ or later supports the block B^*.

Moving onto round $r+1$, properties (i)–(iii) imply that no node of S will change its mind about supporting B^*. (This is obvious for nodes that have already committed B^* to their local histories, as they will never again vote in any height-h referendum.) For example, in the first phase, if $\ell \in S$, the "if" statement in line 2 cannot be satisfied with any QC that does not support B^*, so B_ℓ remains equal to B^* after this phase. Similarly, in the second phase, if $i \in S$, the "if" statement in line 6 cannot be satisfied by any QC that does not support B^*, and B_i remains equal to B^* after this phase. No such node will vote for a block other than B^* in the second phase, and so, because $|S| > n/3$, the referendum $(h, r + 1, 1)$ cannot produce a QC for any block other than B^*. Consequently, no node of S will vote for

the honest nodes who observed a winning outcome in the first stage (with different honest nodes potentially witnessing different outcomes, due to Byzantine behavior or delayed messages). The second-stage results are then used to convince a node of S that many other nodes agree with them (i.e., that S is big), in which case the node can proceed to commit the block B^*; if the second-stage results are not convincing (either because S is small or because messages are getting delayed in the asynchronous phase), the node can hedge and proceed to the next round, still in favor of but not yet committed to B^*.

a block other than B^* in the third phase (the "if" statement in line 10 cannot be satisfied for any block other than B^*), and the referendum $(h, r+1, 2)$ also produces no QCs for blocks other than B^*. Similarly, in the fourth phase, because the "if" statement in line 15 cannot be satisfied for any block other than B^*, every node $i \in S$ (that hasn't already committed B^*) concludes the phase with $B_i = B^*$. Thus, each of the four phases preserves properties (i)–(iii).[20]

Given that (i)–(iii) hold at the beginning of round $r+2$, history will repeat itself, with the exact same argument implying that (i)–(iii) also hold at the end of the round. By induction on the number of rounds, properties (i)–(iii) hold forevermore. By property (iii), and because round r is the earliest round at which a height-h stage-2 QC could possibly be produced, we can conclude that there will never be a height-h stage-2 QC for a block other than B^*. This implies that no honest node will ever commit to a block at height h other than B^*, completing the proof. ■

6 Proof of Liveness

Let's move on to the second property that we really care about, eventual liveness. (If all we wanted was consistency, we could just use a SMR protocol that literally never does anything!) Here "eventual" means that liveness should hold soon after the network resumes "normal operating conditions" (i.e., soon after the global stabilization time, one of the two key parameters in the partially synchronous model that we reviewed in Section 2).

The proof of Tendermint's consistency (Theorem 5.1) is not trivial, but it is relatively robust to variations in the protocol description—it relies primarily on the two stages of quorum certificates. The proof of Tendermint's liveness is more delicate and it depends heavily on some of the finer details in the protocol description.[21]

6.1 Defining Liveness

In Lecture 2, we defined liveness for the SMR problem (with clients submitting transactions to the nodes running the protocol) as:

Liveness (Strong Version)

If a transaction is known to at least one honest node, that transaction is eventually added to every honest node's local history.

This property actually doesn't hold for the version of the Tendermint protocol described in this lecture—if a transaction tx is known to only one honest node i, it is possible that i will never get any of its block proposals finalized and thus tx is effectively censored, never

[20]Nodes may update their locally stored QC to reflect more recent QCs supporting B^*, but this does not change properties (i)–(iii).

[21]For the same reason, buggy implementations of BFT-type protocols tend to cause liveness failures (and, in more severe cases, consistency violations as well).

added to any node's local history. (Proving this fact is an excellent test for checking if you thoroughly understand the Tendermint protocol.) We'll work instead with a slightly weaker definition of liveness.

Liveness (Weak Version)

If a transaction is known to *all* honest nodes, that transaction is eventually added to every honest node's local history.

We'd obviously rather satisfy the stronger requirement than the weaker one, but I don't mean to make too big of a deal of the distinction. You can imagine the honest nodes exchanging transactions that they've heard about via a gossip protocol (thereby reducing the strong version to the weak version), or directly modifying the Tendermint protocol so that it (eventually) satisfies the stronger liveness property.

6.2 Fast Forwarding Past Obvious Obstructions to Progress

Fast forwarding past the GST. At what point can we hope for guaranteed progress, with honest nodes continually adding new blocks to their local histories? In the partially synchronous model, one obvious obstruction to progress is message delays. It's possible that literally no message ever gets delivered before the global stabilization time, so presumably we'll need to fast forward past the GST to guarantee liveness.

Fast forwarding past Byzantine leaders. A second obvious obstruction to liveness is Byzantine leaders—when a Byzantine node is a round's leader, it can give the other nodes the silent treatment (making no block proposals) to ensure that the round is wasted. So presumably we'll need to fast forward to a round with an honest leader (and also after GST). In fact, this isn't enough—we need to fast forward to a pair of consecutive rounds that both have honest leaders. How can we be sure that there will ever be such a pair of consecutive rounds? Assume that leaders rotate in some round-robin order (with each node acting as the leader every n rounds). For there to never be a pair of consecutive honest leaders, there would need to be a Byzantine leader at least every other round. With round-robin leader rotation, this can happen only if at least 50% of the nodes are Byzantine. Our standing assumption is that less than one-third of the nodes are Byzantine, so we're good to go:

Lemma 6.1 (Inevitability of Consecutive Honest Leaders) *Assuming round-robin leader election and that $f < n/3$, there are infinitely many pairs of consecutive rounds with honest leaders.*

6.3 The Proof

Here's the claimed liveness guarantee for the Tendermint protocol:

Theorem 6.2 (Liveness of the Tendermint Protocol) *In the Tendermint protocol with round-robin leader rotation, if $f < n/3$ and a transaction tx is at some time step known to all honest nodes, that transaction is eventually added to every honest node's local history.*

To start the proof, suppose a transaction tx is known at time step t to every honest node. Fast forward to the first pair of consecutive rounds r_1, r_2 with honest leaders such that round r_1 begins at or after time step $\max\{t, GST + \Delta\}$. (Lemma 6.1 assures us that such a pair exists.) Let ℓ_1 and ℓ_2 denote the (honest) leaders of rounds r_1 and r_2, respectively. See also Figure XXX.

The rough idea is that the first honest leader ℓ_1 will synchronize all honest nodes—in effect, flushing the system of any foolishness that previous Byzantine leaders might have been up to—and the second honest leader ℓ_2 will lead them to commit a block that is guaranteed to contain tx (assuming that tx has not already been included in some previous block). There are a lot of details, however, and going through them should give you a visceral feel for how tricky consensus protocol design really is.

First, a lemma:

Lemma 6.3 (Near-Convergence of Local Block Numbers) *Every honest node begins round r_1 working on block number h or $h+1$, where h denotes the highest block number that any honest node is working on at time $t^* := 4\Delta r_1 - \Delta$.*

Proof: Consider the fourth phase of the round r_0 that immediately proceeds r_1, which takes place at time $4\Delta r_0 + 3\Delta = t^*$. By assumption, $t^* \geq GST$. At time t^*, no node (honest or otherwise) possesses a QC for a block number higher than h (as any QC requires some votes from honest nodes, and no honest node has yet voted for any block number higher than h).

Suppose honest node i is working on block number h at time t^*. This node must, at that time, possess stage-2 QCs for blocks $1, 2, \ldots, h-1$. All of these QCs would have been echoed to all nodes at time t^* if not earlier (in line 24 of some earlier round, or in one of lines 4, 8, or 18 of the current or some earlier round). Because $t^* \geq GST$, all these echoed QCs are received by all honest nodes by the end of round r_0. After the processing of these QCs in line 24 of round r_0, all honest nodes begin round r_1 working on block number h or higher. Because there are not yet any QCs for any block numbers higher than h, every honest node must begin round r_1 working on block number h or $h+1$. ■

Unfortunately, it is possible for a Byzantine node to force some honest nodes to work on block number h in round r_1 and others to work on block number $h+1$; the idea is to keep a height-h stage-2 QC secret until time t^* and then release it selectively to some but not all honest nodes. (Thinking this Byzantine strategy through is a good exercise.) We address this complication via a case analysis.

Case 1: Everyone's Working on Block Number $h+1$

Suppose every honest node begins round r_1 working on block number $h+1$. Because there are not yet any QCs for block numbers higher than h, all honest nodes begin the round with a null QC. Events then unfold as follows (using repeatedly that round r_1 is post-GST and hence all messages arrive within Δ time steps):

1. In the first phase of round r_1, the (honest) leader ℓ_1 proposes a block B that includes all not-yet-executed transactions that the leader knows about. If the target transaction tx has not already been included in some previous block, it will then be included in B.

2. In the second phase, all honest nodes receive the leader's proposal of block B (and so the protocol proceeds to line 6). Because there are not yet any QCs for block number $h + 1$, the "if" statement in line 6 is satisfied and all honest nodes proceed to execute lines 7–9.[22] Thus all honest nodes issue a first-stage vote for block B in line 9.

3. In the third phase, all honest nodes receive each other's first-stage votes for B. Because $f < n/3$, these votes constitute a stage-1 QC and all honest nodes proceed to execute lines 11–14. In particular, all honest nodes issue a second-stage vote for block B in line 14.

4. In the fourth phase, all honest nodes receive each other's second-stage votes for B. These constitute a stage-2 QC and all honest nodes proceed to execute lines 16–22. In particular, all honest nodes commit to block B as block number $h + 1$ in line 19. Because the target transaction tx is included in either B or some previous block, the proof for case 1 is complete.

In general, call a round *clean* if: (i) the round begins at or after GST; (ii) the round has an honest leader; (iii) all honest nodes begin the round working on the same block number h; and (iv) after the processing in lines 2–3, the (height-h) QC Q_ℓ stored locally at the leader ℓ is at least as recent as every QC Q_i stored locally by an honest node at the beginning of the round. The four-step argument above proves:

Lemma 6.4 (Clean Rounds Commit Proposed Blocks) *Every clean round concludes with all honest nodes committing to their local histories (as block number h) the block proposed by the leader in the first phase.*

In particular, condition (iv) ensures that, in the second phase, the "if" condition in line 6 is met at all honest nodes, who then issue first-stage votes for the block proposed to them by the (honest) leader in the first phase.

Case 2: Everyone's Working on Block Number h

Suppose every honest node begins round r_1 working on block number h. In particular, no honest node has received a height-h stage-2 QC by the start of this round (at time $t^* + \Delta$). We claim that this round is clean. Because properties (i)–(iii) hold by assumption, we only need to verify (iv).

[22] Recall that we interpret one null value as being at least as recent as another null value.

Let (B, Q) denote the most recent height-h (and stage-1) QC known to any honest node at time t^* and the block that this QC supports. (If there are no such QCs, define Q to be null and B to be the non-yet-executed transactions that the leader ℓ_1 knows about.) Analogous to the proof of Lemma 6.3, this block-QC pair will have been echoed and received by the round-r_1 leader ℓ_1 by the beginning of that round. Thus, in the first phase of round r_1, after the processing of lines 2–3, the (honest) leader ℓ_1 has a locally stored height-h block-QC pair (B^*, Q^*) for which Q^* at least as recent as Q.[23] Now consider an honest non-leader i. If i was working on a block number less than h in round r_0 (only catching up to the others in the fourth phase or line 24 of that round), it begins round r_1 with the null QC (which is no more recent than Q^*). If i was working on block number h at round r_0, the value of its locally stored QC Q_i at the beginning of round r_1 is the same as it was at time t^* (and is therefore no more recent than Q). Either way, the value of Q^* (after the processing of lines 2–3) is at least as recent Q_i (at the beginning of round r_1), which verifies (iv).

Knowing that round r_1 is clean, Lemma 6.4 implies that all honest nodes commit block B^* to their local histories as block number h. We're not done, however: because B^* may be an inherited block proposal from an earlier round, possibly even from a Byzantine leader, there is no guarantee that it includes the target transaction tx.

But the next round r_2 is also clean, with all honest nodes working on block number $h+1$ and with null QCs. (Remember, no height-$(h+1)$ QCs exist yet.) By Lemma 6.4, the round concludes with all honest nodes committing to the block B proposed by the (honest) leader ℓ_2 as block number $h+1$. Because B was proposed by an honest node in a time step after tx was known to all such nodes, it is guaranteed to include tx (assuming that tx isn't already in some previously confirmed block).

Case 3: The Leader Lags Behind

Suppose that the leader ℓ_1 begins the round r_1 working on block number h, and at least one honest node i begins the round working on block number $h+1$. Node i must then have in its possession a height-h stage-2 QC Q^* for a block B^* at the beginning of the round. Because (B^*, Q^*) was echoed at or before this time (in one of lines 4, 8, 18, or 24 of an earlier round), all honest nodes have received this height-h stage-2 QC by the end of round r_1. Thus, by the end of round r_1 (in line 24 if not earlier), every honest node will have committed to some height-h block.[24] Thus, round r_2 will be clean, with all honest nodes working on block number $h+1$ and with null QCs. (There are not yet any height-$(h+1)$ QCs: Round r_1 was the first in which any honest node would have been willing to vote in a height-$(h+1)$ referendum, but the block proposed in that round by ℓ_1 was for block number h.) As in case 2, Lemma 6.4 then implies that the target transaction tx will be included in some block (either the one produced in round r_2 or some earlier block).

[23]It is possible that ℓ_1 received, by the start of round r_1, a block-QC pair even more recent than (B, Q) (necessarily from a Byzantine node).

[24]Tendermint's consistency property (Theorem 5.1) implies that they all commit the same block, namely B^*.

Case 4: The Leader Is Ahead

Finally, suppose that the leader ℓ_1 begins the round r_1 working on block number $h + 1$ and that at least one honest node begins the round working on block number h. Assume that the target transaction tx has not been included in any of the first h blocks (as otherwise, we're done). The block B^* proposed by the (honest) leader ℓ_1 (for block number $h + 1$) in the first phase of this round will contain tx.[25]

At the start of round r_1, no height-$(h+1)$ QCs exist. Any such QCs produced in round r_1 must support the block B^*.[26] In particular, if any honest node commits to block number $h+1$ in round r_1, it must be to B^*. In this case, we're done. (Block B^* contains tx and all honest nodes will end up committing to block B^*, in line 24 of round r_2 if not earlier.)

If no honest node commits to block number $h + 1$ in round r_1, then all honest nodes begin round r_2 working on block number $h + 1$. This round is clean, for the exact same reasons that round r_1 in case 2 is clean.[27] By Lemma 6.4, round r_2 concludes with all honest nodes committing (as block number $h + 1$) the block proposed by ℓ_2 in the first phase of the round. This block is either the proposal B^* inherited from the previous round or a new block proposal B' assembled by ℓ_2 in the first phase (which, because ℓ_2 is honest, will contain tx).[28] Either way, the target transaction tx will be added to all nodes' local histories, completing the proof of liveness.

6.4 Chain Quality

Theorem 6.2 and its proof have interesting consequences for the "chain quality" of the sequence of finalized blocks, meaning the fraction of such blocks that are contributed by honest nodes.

Call a post-GST round with an honest leader *bad* if the round does not conclude with all honest nodes committing a block that was originally proposed by an honest node. In all four cases of the proof of Theorem 6.2, the block committed by honest nodes in the second (clean) round r_2 is a block that was proposed by an honest node.[29] Thus, every bad round

[25]The leader ℓ_1 proposes its own block rather than one inherited from previous rounds, because it begins round r_1 with its local variables newly initialized (in the fourth phase or line 24 of round r_0) and because there are not yet any $(h + 1)$-height QCs to process in lines 2–3.

[26]Honest nodes only cast first-stage votes for the block they heard from the current round's leader. With $f < n/3$ and no honest first-stage votes for blocks other than B^*, there will also be no honest second-stage votes for blocks other than B^*, and hence no (stage-1 or stage-2) round-r QCs for any such blocks.

[27]Only property (iv) needs to be verified; every honest non-leader node either begins round r_2 with a null QC or with a QC that it explicitly forwarded to the leader ℓ_2 of round r_2 in time for processing in lines 2–3.

[28]The proposed block will be B^* except in the edge case in which ℓ_2 was working on block h in round r_1, had its local variables reset in lines 21–22 or 24 of that round, and received no height-$(h+1)$ QCs to process in lines 2–3 of round r_2. (Honest nodes remember QCs for future blocks but do not pay any attention to leader proposals for future blocks.)

[29]Case 1 (the easiest case) only discussed round r_1, but the next round r_2 proceeds in exactly the same way. In cases 1–3, the block committed is the one assembled from scratch by the round-r_2 leader ℓ_2. In case 4, the block committed was assembled from scratch by either ℓ_2 or the (honest) leader ℓ_1 of the previous round r_1 (see footnote 28).

must be preceded by a round with a Byzantine leader. Because more than two-thirds of the rounds have honest leaders and less than one-third have Byzantine leaders (assuming round-robin leader rotation), more than one-third of the post-GST rounds conclude with the commitment of an honestly assembled block. Because less than one-third of the post-GST rounds conclude with the commitment of a block assembled by a Byzantine node (as any given round can lead to at most one confirmed block), more than half of the blocks finalized post-GST were originally proposed by honest nodes. Using *chain quality* to mean the fraction of confirmed blocks proposed by honest nodes, we thus have:

Theorem 6.5 (Chain Quality of the Tendermint Protocol) *After the global stabilization time, the chain quality of the Tendermint protocol is more than 50%.*

More generally, if at most an $\alpha < \frac{1}{3}$ fraction of the nodes are Byzantine, the chain quality (post-GST) is at least

$$\frac{1-2\alpha}{1-\alpha}.$$

We will see this expression again in the next lecture, in the context of longest-chain consensus.

7 Can We Do Better?

With the tricky proofs out of the way, let's take a step back and ask: Could we hope to do better than Tendermint? Here are a few senses in which the protocol is "optimal":

1. The upper bound on the number f of Byzantine nodes cannot be relaxed (without compromising elsewhere). We saw in Lecture 6 that when $f \geq n/3$, no consensus protocol satisfies consistency and eventual liveness in the partially synchronous model.
2. The assumption of partial synchrony cannot be relaxed to asynchrony (without compromising elsewhere). We studied the FLP impossibility theorem in Lectures 4–5, which implies that in the asynchronous setting, and even with only one Byzantine node, no deterministic protocol can guarantee both consistency and eventual liveness.
3. Holding the bound on f and the model of communication model (i.e., partial synchrony) fixed, we cannot strengthen "eventual liveness" to "liveness" (without sacrificing consistency). This again follows from the FLP impossibility theorem—during the asynchronous phase, you cannot have both consistency and liveness.

So is Tendermint the end of the story in state machine replication? Is there any reason to talk about any other SMR protocols?

Alternate trade-offs. One reason to consider alternative protocols is to achieve different trade-offs—different points on the Pareto curve. While no protocol will strictly dominate Tendermint in all of the dimensions listed above, there could be interesting protocols that are stronger in some respects and weaker in others.

For example, we could stick with the partially synchronous model and ask: Can we strengthen eventual liveness to liveness (always), while relaxing consistency to eventual consistency? In other words, could we escape the FLP impossibility theorem by giving up on consistency rather than liveness during the asynchronous phase? (In the synchronous phase, as usual, we should expect to have both.) This was not a particularly popular question in the 20th-century literature on consensus protocols, but it is exactly the alternative trade-off that protocols like Bitcoin and (the original version of) Ethereum make.

More efficient protocols. A second active and practically important direction is to develop consensus protocols that match the guarantees of Tendermint (in terms of fault-tolerance, consistency, and liveness) but have better performance. "Performance" means different things to different people, but some example metrics include the number of messages sent per round, the duration of a round, the number of times nodes need to carry out certain expensive cryptographic operations per round, and so on.

As promised at the beginning of this bootcamp on classical consensus protocols, we haven't paid much attention to any of these kinds of performance metrics. One reason is the lack of agreement among experts about which metrics are the most important. A second reason is that a detailed performance analysis would be a little out in the weeds for our purposes. This lecture series is fundamentally about blockchains, not consensus protocols per se, and there are a lot of other interesting aspects of blockchains that we want to have time for.

You should know, however, that there have been a number of advances in BFT-type protocols since Tendermint, and some of these appear likely to make it to production over the next few years. For example, Facebook/Meta's Diem project derived its consensus protocol from HotStuff [**?**], which from 30,000 feet can be viewed as a sort of pipelined, more efficient version of Tendermint. The Diem project has been dissolved, but its consensus protocol looks likely to resurface in other projects.

Another example is the Ethereum blockchain, which is slated to make a major upgrade in 2022 that would, in particular, incorporate some BFT-type ideas into its existing longest-chain consensus protocol. (We'll discuss basic longest-chain consensus in Lecture 8.) For example, if you look into "Casper FFG"—a "finality gadget" responsible for finalizing periodic checkpoints of the underlying longest-chain protocol—you'll again see something that resembles (at a high level) a pipelined version of Tendermint [**?**].

HotStuff and Casper-FFG have exactly the same fault-tolerance as Tendermint, and in particular there is again the magical 33% threshold on the fraction of Byzantine nodes that can be tolerated (subject to consistency and eventual liveness in the partially synchronous model). Having survived this bootcamp on consensus protocol basics (Lectures 2–7), you are well positioned to understand these recent advances.

Optimistic responsiveness. While on the topic of efficiency, one bummer about the Tendermint protocol is that each round takes 4Δ time steps, even if all messages are being delivered super-quickly. (E.g., maybe Δ corresponds to 1 second but it's a good network day, with all messages delivered within 10 milliseconds.) *Optimistic responsiveness* is the property that, whenever the current round's leader is honest, the time required to confirm a block scales with the actual (rather than worst-case) message delay [?]. This property is one of the claims to fame of the aforementioned HotStuff protocol [?], for example. Protocols with this property generally differ from Tendermint in that new leaders are chosen only on an as-needed basis (if nodes detect that something has gone wrong with the current leader), rather than automatically in round-robin order. The idea of as-needed-leader-election comes originally from the famous PBFT (for "practical BFT") consensus protocol [?]; in that context, a switch in leader is known as a "view change." Tendermint can be viewed as (pessimistically) enacting a view change in every round.

Onward to 21st-century consensus protocols, and the longest-chain consensus made famous by the Bitcoin protocol!

References