Foundations of Blockchains
Lectures #8: Longest-Chain Consensus
(IN PROGRESS)*

Tim Roughgarden†

1 The Upshot

1. Longest-chain consensus protocols as in Bitcoin and (the original version of) Ethereum are an important category of blockchain protocols (different from BFT-type protocols like Tendermint).

2. Longest-chain consensus starts with a genesis block and consists of a sequence of rounds, where in each round one node acts as the current block proposer.

3. Different implementations of longest-chain consensus (permissioned, proof-of-work, proof-of-stake) implement leader selection in different ways and impose different constraints on the block proposer's behavior.

4. Relative to BFT-type protocols, longest-chain consensus favors liveness over consistency.

5. Longest-chain consensus loses consistency under network attacks (in the partially synchronous model) and is analyzed primarily in the synchronous model.

6. An honest block proposer is instructed to propose a single block, with its predecessor set to the current end of the longest chain (breaking ties arbitrarily).

7. Byzantine block proposers can, among other things, deliberately create or perpetuate forks by proposing blocks with predecessors that are not the end of the longest chain.

8. If there is a sequence of consecutive rounds in which the Byzantine block proposers outnumber the honest block proposers by k, those Byzantine nodes can force the rollback of the last k blocks of the longest chain.

†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. These notes are a work in progress; corrections and feedback are welcome.

9. The last few blocks on the longest chain should always be regarded as tentative and still under negotiation.

10. A sequence of block proposers is w-balanced if, for every window of at least w consecutive rounds, the honest block proposers outnumber the Byzantine ones.

11. If block proposers are chosen randomly, with each round's choice independent and more likely than not to be an honest block proposer, the sequence of block proposers will be w-balanced with high probability (provided w is set appropriately).

12. The reason is that, with randomly selected block proposers, all sufficiently long sequences of consecutive rounds will have a near-proportional representation of honest nodes.

2 A Tale of Two Protocol Paradigms

Lecture 7 was about the Tendermint protocol (for state machine replication), and its optimal fault-tolerance in the partially synchronous model (with consistency always and liveness eventually). This was the culmination of our six-lecture bootcamp on classical consensus protocols. Most of what we talked about was figured out by brilliant computer scientists in the 1980s. (Tendermint is a 21st-century protocol, but heavily influenced by the classic protocols from the 1980s and 1990s.[1]) As you may know, the most famous blockchain protocol of them all (Bitcoin) is based on a consensys protocol that does not at all resemble those that we've been discussing thus far. In the blockchain world, there are currently two different prevalent paradigms for consensus protocols, "BFT-type" and "longest-chain" protocols.[2]

2.1 Category #1: BFT-type procols (e.g., Tendermint)

Protocols like Tendermint are often called *BFT* or *BFT-type protocols*; the "BFT" stands for "Byzantine fault tolerant." The first shared property of these protocols is that they all look similar from 30,000 feet, with multiple stages of voting and some analog of quorum certificates. This ensures that a nodes commits to a new block only after it can be sure (assuming not to many Byzantine nodes) that no other honest node will ever commit to any other block.

[1]The influence of these 30-year old ideas and protocols on modern blockchains (an application domain that didn't exist until the 21st century) is pretty amazing. Tendermint is exactly the consensus protocol that drives (for example) the Cosmos ecosystem, and more modern blockchain protocols (e.g., variants of HotStuff [**?**]) also reuse many of the same ideas. All of these protocols offer the same consistency and liveness guarantees as Tendermint—in particular, favoring consistency over liveness when the communication network is broken or under attack—with some extra ideas that improve performance.

[2]There are also blockchain consensus protocols that don't fall nearly into either camp. But you can classify a majority of major blockchain protocols as one of these two types.

By design, under appropriate assumptions (such as $f < n/3$, where f and n denote the number of Byzantine and total nodes, respectively), BFT-protocols to do not suffer from "forks"—there will never be two different blocks committed by honest nodes at the same block height. If there ever *is* a fork in a BFT-type protocol (either due to a buggy implementation or more Byzantine nodes than expected), it's not at all clear how to resolve the fork from within the protocol.[3]

Because BFT-type protocols favor consistency over liveness (when the network is poor or under attack), when they fail (for whatever reason), they typically fail by stalling (i.e., not confirming any new blocks of transactions for a prolonged period of time). You do not typically hear, for example, about double-spend attacks caused by the rollback of thought-to-be-finalized blocks.

2.2 Category #2: Longest-Chain Protocols (e.g., Bitcoin)

If you've watched even a 20-minute video on Bitcoin at some point in your life, you already know there's a second category of blockchain consensus protocols, namely *longest-chain protocols*. Longest-chain consensus was invented by Nakamoto and first described in the Bitcoin white paper [**?**].

Conceptually, longest-chain protocols are radically different from BFT-type protocols. First, there is no explicit voting—each block (along with an explicit pointer to a predecessor block) is proposed unilaterally by some node.[4] Second, because there's no waiting for quorum certificates, forks—two (potentially conflicting) blocks that claim a common predecessor—can easily arise in the normal course of business (either due one Byzantine node or network delays). With potentially frequent forks, such a protocol can function only if it has an in-protocol way of automatically resolving the ambiguity introduced by a fork. This is done by—wait for it—always resolving forks in favor of the longest chain of blocks. (We'll get into all the protocol details and nuances starting in the next section.) Third, by virtue of embracing forks and resolving them in-protocol, longest-chain protocols favor liveness over consistency when there's a communication breakdown between nodes. Accordingly, in practice longest-chain protocols tend to violate by violating consistency (with thought-to-be-confirmed blocks getting rolled back) rather than by stalling.[5] Such consistency violations can enable "double-spend" attacks. For example, suppose a blockchain transaction carries the payment for an expensive physical good like a Tesla. Once the Tesla dealer regards the transaction as confirmed, they let the buyer drive off in their new car. If that transaction later gets rolled back, the buyer effectively gets a full refund even though they still have the car.

[3]For example, the Tendermint Core documentation (`https://docs.tendermint.com/master/spec/consensus/consensus.html`) states: "For now, we leave the problem of reorg-proposal coordination to human coordination via internet media." (I.e., the advice is to spin up a Discord channel to discuss what do to next as a step toward recovery.)

[4]A block proposal can be interpreted as an implicut vote for its predecessor and all of its ancestors.

[5]For a real-world example, look into the many large-scale blockchain re-organizations that have been suffered by the Ethereum Classic blockchain protocol.

3 Longest-Chain Consensus

3.1 Preamble

The version of longest-chain consensus described in this lecture may differ from what you had in mind. Let me explain.

This lecture: permissioned setting, PKI assumption. Through this lecture, we will continue working in the permissioned setting with the PKI assumption (with an a priori known set of nodes running the protocol, each with a private key and a known-to-all corresponding public key). One reason for this is continuity with all of the lectures thus far. Another reason is that many of the novel ideas in and properties of longest-chain consensus manifest already in this setting.

That said, longest-chain consensus's biggest claim to fame is its shocking graceful extension to the "permissionless" setting in which the protocol has no idea what nodes are running it. Next lecture (Lecture 9) is all about permissionless longest-chain consensus (specifically, the "proof-of-work" version used in Bitcoin).

Looking ahead toward the Bitcoin protocol. As I've said many times, this lecture series is focused on principles of blockchain design and analysis rather than on specific blockchain protocols per se. And if you study Bitcoin for Bitcoin's sake, for example, you generally wind up conflating multiple independent innovations. We'll take care to unbundle these innovations in this lecture series, both because of the mental clarity that it affords and because the separate pieces can be remixed with other ideas to design other interesting blockchain protocols.

A consensus protocol lays down the law about which of the proposed blocks are the ones that count (and about their ordering). A conceptually distinct question is which nodes can participate (e.g., by proposing a new block or voting on a previously proposed block). It's easy to come up with answers to this question in the permissioned setting (with the PKI assumption)—for example, by letting all nodes vote and having nodes take round-robin turns acting as the block proposer. It's much harder in the permissionless setting, because the protocol has no idea which nodes are running it. The second big innovation in Bitcoin (in addition to longest-chain consensus) is an effective way of selecting block proposers in longest-chain consensus in the permissionless setting, using an idea known as "proof-of-work."[6]

In this lecture series, we'll keep these two questions (which blocks count? who participates in block production?) as separate as possible. They will interact a little bit at their edges, though, as we'll see (page **??**).

[6]The combination of longest-chain consensus with proof-of-work block proposer selection, as introduced in Bitcoin, is sometimes called "Nakamoto consensus."

Permission + PKI vs. proof-of-work vs. proof-of-stake implementations. Some readers may enter this lecture with the mindset of permissioned consensus (e.g., those coming in directly from Lectures 2–7) while others may be strongly influenced by famous permissionless blockchain protocols that they've read about (e.g., Bitcoin and Ethereum). I'm going to try to have my cake and eat it too, by addressing both audiences at the same time.

The plan is for the main narrative to focus squarely on longest-chain consensus in the permissioned setting (with the PKI assumption), but with frequent side comments and forward pointers about how to interpret that narrative in the content of permissionless blockchain protocols. We'll provide interpretations both for the proof-of-work implementation described in Lecture 9, and the "proof-of-stake" implementation covered in Lecture 12. Don't worry about it if you've never heard about approaches to "sybil-resistance" like proof-of-work or proof-of-stake—the main narrative of this lecture assumes nothing other than familiarity with permission protocols for state machine replication (as covered in the preceding lectures).

Remember the Dolev-Strong protocol? Another reason to keep looking forward toward permissionless implementations is that, in the setting of this lecture (permissioned setting with the PKI assumption), longest-chain consensus is strictly dominated by an SMR protocol that we analyzed way back in Lecture 2—the (iterated) Dolev-Strong protocol. That protocol has really impressive provably guarantees—consistency and liveness even when 99% of the nodes are Byzantine—provided you're willing to make a bunch of assumptions: the permissioned setting, the PKI assumption, and the synchronous model (with an a priori known bound Δ on the maximum message delay).

As we'll see (Section **??**), longest-chain consensus loses consistency in the partially synchronous setting and for this reason is studied primarily in the synchronous setting. Here, the protocol guarantees consistency and liveness provided at most *49%* of the nodes are Byzantine (see Theorems **??** and **??**). It's not trivial to design a protocol with such guarantees (nor is it easy to prove that longest-chain consensus does in fact offer them), but one has to concede that—in the permissioned setting, with the PKI assumption, in the synchronous model—there's really no reason to use longest-chain consensus instead of simply running the Dolev-Strong protocol once for each block (with rotating block proposals/senders).

Thus, it's only once we pass to the permissionless setting that longest-chain consensus allows us to reach goals that we don't otherwise know how to achieve. This lecture is therefore best viewed as a stepping stone toward the proof-of-work and proof-of-stake implementations of longest-chain consensus studied in Lectures 9 and 12, rather than as an end unto itself. But I promise that I'm not wasting your time—the key points in this lecture are all essential even for the reader who cares only about understanding permissionless longest-chain consensus.

3.2 Protocol Description

Our description of longest-chain consensus has three scenarios simultaneously in mind:

(PKI) Permissioned setting, PKI assumption.

(PoW) Permissionless setting, proof-of-work sybil-resistance.

(PoS) Permissionless setting, proof-of-stake sybil-resistance.

If you're unfamiliar with any permissionless blockchain protocols, ignore all comments about scenarios (PoW) and (PoS)—all will become clear in Lectures 9 and 12, respectively.

An underspecified version. Let's start with an abstract and underspecified description of longest-chain consensus, before interpreting it in each of the three scenarios above.

Longest-Chain Consensus (Abstract)

(1) Start with a hard-coded *genesis block* B_0.

(2) In each "round" $r = 1, 2, 3, \ldots$:

  (2a) Choose one node ℓ as the leader of round r.

  (2b) Node ℓ proposes a set of blocks, each specifying a single predecessor block.

We'll supply more details below, but already at this level of specification we can fruitfully visualize the data structure produced by the nodes running the protocol as an *in-tree*—a tree (in the sense of an acyclic connected graph) in which all edges are directly back toward the root (Figure **??**). For us, the genesis block acts as the tree's root, and each directed edge corresponds to one block's naming of its predecessor.[7] When the protocol is first fired up, this in-tree consists solely of the genesis block; as the rounds go by, nodes running the protocol continue to grow this in-tree (with each leader adding a new batch of vertices, each with out-degree 1). Note that, as promised, forks (a block named as the predecessor of multiple blocks, or equivalently a vertex with in-degree greater than 1) are embraced as part of normal operation.

The genesis block. Every longest-chain protocol has baked into it a *genesis block* that gets the party started. There are no transactions in the genesis block, but it is an eligible predecessor for any block that might be created later. Just as each node is born knowing the protocol code (and, in the PKI setting, its private-public key pair), each node should be thought of as born knowing the genesis block.

What's a "round"? Each iteration of longest-chain consensus—each with a single leader node who's granted the power to add to the current blockchain—is called a round. The meaning of a round depends on which scenario we're talking about. In scenarios (PKI) and (PoS), we'll assume a shared global clock (as in the synchronous model) and each round will correspond to a time interval of a fixed length. For example, round 1 might be the first 10 seconds, round 2 the next 10 sections, and so on. We've seen this idea before (in

[7] While the description thus far doesn't actually guarantee acyclicity, assumption (A4) below does.

scenario (PKI)), for example in the iterated Dolev-Strong protocol in Lecture 2 and the Tendermint protocol in Lecture 7.

Interestingly, in scenario (PoW), rounds will not correspond to fixed periods of time (except in a loose average sense), and more generally there's only minimal reliance on a shared notion of time. Rather, rounds are defined in a purely event-driven way. Readers who know how proof-of-work works—with block producers racing to be the first to solve a difficult cryptographic puzzle—may already see what I mean by "event-driven." Every time some block producer gets lucky and solves a puzzle (the event of interest), we'll call that the next round.

How is the leader chosen? Each round has a single node that acts as the leader. The abstract description above is silent on how this happens in step (2a), and the answer depends on the scenario that we're looking at.

The simplest scenario is (PKI), in which we can reuse our old idea from Lectures 2 and 7 of round-robin rotating leaders. E.g., if there are 100 nodes and it's currently round 117, node 17 is the leader. Assuming that each round has a fixed length and that all nodes share a global clock, they all automatically know what round it is, and in the permissioned setting they all then know which node is the leader at any given time.

You can, at a high level, think of scenario (PoS) as similar to scenario (PKI), except with leaders chosen uniformly at random rather than via a deterministic round-robin order. That's not quite right, because a node's probability of selection as leader will typically be proportional to the amount of stake (in the blockchain's native currency) that the node has locked up in a designated smart contract. E.g., if 10% of the total stake locked up in the contract is owned by node i, then in any given round, node i has a 1 in 10 chance of being chosen as the round's leader. (We'll talk about all the nuances of implementing this idea in Lecture 12.)

Scenario (PoW) is similar to (PoS) except that a node is effectively selected as a leader—meaning it's the first node to solve a hard cryptographic puzzle—with probability proportional to the amount of computational power that it contributes to the protocol (rather than to the amount of locked-up stake).

What can the leader do? Once a node is selected as the leader of a round (in step (2a)), in step (2b) that node gets to create some number of blocks. Here a *block* is, as usual, an ordered sequence of transaction (which were presumably submitted to nodes earlier by the clients participating in the SMR protocol). However many blocks that leader creates, it must specify a unique predecessor block for each of those blocks. (Any block that specifies zero or more than two valid predecessors is automatically considered invalid by all honest nodes. Looking ahead, an honest leader will be instructed to create exactly one block; it's Byzantine nodes that might create several.) The leader is always in a position to do this—if nothing else, it can use the (commonly known) genesis block as a predecessor.

As we'll see below, exactly what the leader is capable of doing will depend on which of the scenarios we're in (e.g., the leader is most highly constraint in scenario (PoW))—this is

part of longest-chain consensus in which there is inevitably interaction between the details of the consensus protocol and the method by which nodes are selected to participate.

The use of explicit predecessors is a departure from BFT-type protocols like Tendermint in which, assuming less than a third of the nodes are Byzantine, there will be only one block at each block height (because quorum certificates are required for block finalization, and by the quorum intersection property). Because there's only going to be one block #8, one block #9, one block #10, and so on, there's no need to explicitly name a predecessor—everyone knows that the predecessor is the (unique) block at the previous height. In longest-chain consensus, any leader can create a fork if it wants (e.g., by proposing multiple blocks, all with the same predecessor). Thus there may be multiple blocks at each block height, effectively competing for eventual finalization. This renders explicit predecessors necessary—if there are multiple blocks at height 8, a block at height 9 needs to specify which one it views as the preceding block.

4 Five Assumptions

The goal of this lecture is to show that longest-chain consensus satisfies consistency and liveness in the synchronous model provided at most 49% of the nodes are Byzantine. We'll carry out this analysis under five assumptions, detailed and discussed next.

4.1 Assumptions About the Genesis Block

The first assumption is a trusted setup assumption—meaning an assumption that we take on faith, kicking the can down the road as to how it might be enforced. Specifically, we assume that no nodes are privy in advance to the description of the protocol's genesis block—nodes only learn the name of that block at the moment the protocol commences. (E.g., the genesis block can't have been chosen by some Byzantine node in advance.)

Assumption (A1)

(A1) No node has knowledge of the genesis block prior to the deployment of the protocol.

As a trusted setup assumption, we'll make no effort to enforce it within the protocol itself (regardless of which of the three scenarios we're in). When deploying a longest-chain protocol, a good faith gesture is to include a genesis block that references verifiable information that could not have realistically been predicted well in advance of the protocol's deployment.[8] This assumption will show up in an important (but subtle) way in our proof of Theorem **??**.

[8]Nakamoto famously deployed the Bitcoin protocol with a genesis block that referenced a very recent headline of the *London Times* newspaper: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks."

4.2 Assumptions About Leader Selection

We'll also need to assumptions about how leader selection works in step (2a) of longest-chain consensus. These are not trusted setup assumptions, and we'll need to make sure that a concrete implementation of longest-chain consensus enforces them.[9]

Assumption (A2)

(A2) It is easy for all nodes to verify whether a given node is the leader of a given round.

The second assumption asserts each round's leader selection should be verifiable. This means two things: the round's selected leader can easily prove it is in fact the leader; and no other node can trick honest nodes into thinking it's actually the leader.

Happily, assumption (A2) takes care of itself in all the scenarios that we're interested in. In scenario (PKI), imagine that leaders rotate round-robin order, e.g. with node 17 automatically the leader of round 117 (assuming $n = 100$). Imagine further that every message sent by an (honest) node is signed by the sender and annotated with the round it belongs to. Because of the PKI assumption, node 17 and only node 17 will be in a position to send messages as the leader in round 117 (if a round-117 message that is supposed to be from the round's leader is not signed appropriately, all honest nodes ignore the message).

In scenario (PoS), leader selection will typically be a deterministic function of transactions that have been executed thus far and the set of public keys held by nodes running the protocol. All nodes can evaluate this function and then automatically know which node (or rather, which public key) is the current leader. (There will effectively be a pseudo-PKI assumption when we study scenario (PoS) in Lecture 12, so again the current leader and only the current leader will be in a position to send messages that will be accepted by the honest nodes and coming from the round's leader.)

In scenario (PoW), as we'll see in Lecture 9, leadership is literally defined by exhibiting a proof of it (in the form of a solution to a difficult cryptographic puzzle) in tandem with the choice of block and predecessor in step (2b). In effect, the way longest-chain consensus works in scenario (PoW), every message comes equipped with a self-contained proof that the node was authorized to send it.

We also need a second assumption about leader selection:

Assumption (A3)

(A3) No node can influence the probability with which it is selected as the

[9] For an analogy, we analyzed protocols like the Dolev-Strong protocol and Tendermint under the (crucial) assumption that signatures by honest nodes could not be forged by Byznatine nodes. That's fine, but in a concrete implementation of such a protocol, you still have to make sure that that assumptions do in fact hold (or at least boil down to other assumptions that you're already comfortable with). For the case of signatures, the answer is simply to use an off-the-shelf provable secure signature scheme such as ECDSA. (Granted, this security rests on a different assumption, that the amount of computation available to an attacker is insufficient to solve the discrete logarithm problem in a suitably chosen group.)

leader of a round in step (2a).

Intuitively, it would seem problematic if a Byzantine node could concoct a strategy that resulted in getting selected as the leader of every round. Ideally, leader selection should be completely unmanipulatable by the nodes running the protocol—in effect, with the choice of a leader just falling from the sky in each round.

Once again, in scenario (PKI) this assumption is easy to enforce using round-robin rotating leaders. Every node automatically knows the current round and therefore (by the permissioned assumption) the current round's leader—there's nothing anyone can do about it.

Enforcing assumption (A3) becomes non-trivial in the permissionless case, though in scenario (PoW) things work out surprisingly simply. We'll see in Lecture 9 that, under something known as the "random oracle assumption" about cryptographic hash functions, there exist cryptographic puzzles that can only be solved by repeated guessing (with each guess independent and equally likely to produce a solution). Under this assumption, you could think of all nodes as repeatedly throwing darts at a dartboard, attempting to hit the (really small) bullseye and thereby become the next round's leader. It's hopefully intuitive that a node's likelihood of being the first to hit the bullseye is proportional to its number of attempts (i.e., its computational power) and is otherwise unaffected by anything else that the node might do.

Enforcing assumption (A3) in scenario (PoS) is tricky and we'll talk about at length in Lecture 12. Intuitively, the difference between scenarios (PoW) and (PoS) is that, in the former, there is a natural external source of randomness (generated by the repeated dart-throwing). In the latter scenario, it would seem that the protocol must generate (pseudo)randomness on its own, based only on the information available in its hermetically sealed environment (i.e., the protocol code and the sequence of transactions executed thus far) and without the benefit of extrenal randomness.[10] This should sound dangerous—e.g., maybe a node can influence its future probability of selection through the transactions that it includes in its blocks? Over the past several years, designers of proof-of-stake blockchains have been continually coming with new ideas to better enforce assumption (A3)—more on this in Lecture 12.

Assumption (A2) and (A3) are both crucial whenever we argue (e.g., in Section 7) that the assumption of at most 49% Byzantine nodes (or computational power in scenario (PoW) or stake in scenario (PoS)) translates to an analogous assumption about the fraction of leaders selected in step (2a) that are Byzantine.

4.3 Assumptions About Block Production

The next assumption plays a fundamental role in our analysis of longest-chain consensus (e.g., in the proofs of Theorem **??** and **??**), and we'll need to take care to enforce it in any concrete implementation.

[10] An alternative approach (discussed in a future lecture) is to trust one or more third parties responsible for importing randomness from the outside world.

Assumption (A4)

(A4) Every block produced by the round-r leader must claim as its predecessor some block that belongs to a previous round.

In other words, if you trace predecessor pointers back from a block, you will encounter a sequence of blocks with strictly decreasing round numbers (ending in the genesis block, which we interpret as belonging to round 0). In particular, because this assumption rules out any cycles among the produced blocks, our visualization of the longest-chain consensus as an ever-growing in-tree is accurate. This assumption also automatically rules out the possibilities of two blocks from the same round appearing on a common chain (Figure **??**).

Your first reaction to assumption (A4) might be that it seems obvious—how can a block refer to something that doesn't exist yet? The assumption rules out two possibilities that would otherwise be problematic. First, remember that in step (2b) a (Byzantine) node might create multiple round-r blocks; assumption (A4) makes the non-trivial assertion that none of these round-r blocks can point to each other as predecessors. (The node can still create a bunch of round-r blocks, for example using some round-$(r-1)$ block as the predecessor for each of them.) Second, one possible strategy for a Byzantine node that is selected as the round-r leader is to delay the announcement of its choices in step (2b) until later. (We inevitably have to deal with this possibility, because it can be indistinguishable from an honest nodes that has had its messages delayed.) For example, imagine node 17 is a Byzantine node selected as the leader of round 117, and only in round 125 does it announce "by the way, here are my round-117 blocks and their predecessors," with some of the named predecessors created only in rounds 118–124. Assumption (A4) asserts that Byzantine nodes should be unable to do this.

Assumption (A4) is crucial to our analysis (as will be obvious in Sections **??**–**??**), but it's actually pretty trivial to enforce in all three scenarios that we're thinking about. In scenario (PKI), for example, the obvious approach is to require that each block proposal is accompanied by a round number and a signature from the leader of that round, and for honest nodes to ignore any block proposal in which the round number of the predecessor block is at least as large as that of the new block. The exact same idea can be used in scenario (PoS).

Scenario (PoW) is the interesting one, and in fact blocks will *not* be explicitly annotated with their round number. As we'll see in Lecture 9, typical proof-of-work-based implementations of longest-chain consensus requires nodes to commit to their choices in step (2b) before they're even notified that they're the round's leader. (Technically, solutions to cryptographic puzzles are regarded by valid as honest nodes only if they include an encoding of the node's step (2b) choices, and validity can only be checked after the proposed solution has been assembled.) Because they commit to their block proposals before round r has started (it starts only once they've validated their winning lottery ticket, including the block proposals encoded by the ticket), those blocks can only refer to predecessor blocks that exist at that time (which, by definition, were created in rounds prior to r).

Given that, in scenario (PoW), winning lottery tickets by definition much encode the

node's decisions in step (2b), why not further restrict the ticket format so that the node specifies exactly one (rather than many) round-r block, along with its predecessor? (As we'll see, honest nodes are anyways supposed to proposed exactly one block in a round of longest-chain consensus.) By incorporating this idea, typical implementations of longest-chain consensus in scenario (PoW) wind up enforcing a much stronger version of assumption (A4):

Assumption (A4) [Stronger Proof-of-Work Version]

(A4') The leader of a round produces exactly one block, and this block claims as its predecessor some block that belongs to a previous round.

The fundamental consistency and liveness guarantees of longest-chain consensus (Theorems **??** and **??**) depend only on assumption (A4), not on the stronger version satisfied by typical proof-of-work instantiations. This is exactly the kind of insight that you miss by studying the Bitcoin protocol for its own sake rather than by examining each of its ingredients in isolation. This point also demonstrates the value of crisply articulating the minimal assumptions required for the validity of an argument—as it turns out, many ideas that were originally developed to analyze the (PoW) scenario specifically can be reused directly for the other two scenarios as well.

4.4 Assumptions About Communication

You should be bothered by our final assumption, which is patently false:

Assumption (A5)

(A5) At all times, all honest nodes know about the exact same set of blocks (and predecessors).

Alternatively, you can think of assumption (A5) as asserting that we're working in the synchronous model with the maximum message delay Δ equal to zero—whenever, an honest node learn anything new, it can then instantly communication (by clairvoyance, in effect) the new information to all the honest nodes. We'll sometimes call this the "super-synchronous" or "instance communication" model (neither of which are standard terms).

Even when we work in the synchronous model with $\Delta = 1$, there will be periods of time in which different honest nodes know different things. For example, when an honest leader proposed a new block, it knows that that block is one time step earlier than the other honest nodes. Thus, unlike assumption (A1), we can't take (A5) on faith—it's simply not true. Unlike assumptions (A2)–(A4), we can't design a consensus protocol to enforce it—message delays are outside the control of the protocol. So what's the plan and the reasoning behind it?

1. We'll adopt assumption (A5) temporarily, and will relax it later. (We could relax it in this lecture, but the most sensible place to do so is in Lecture 9, in the context of

scenario (PoW).)[11]

2. The vast majority of interesting and non-trivial ideas in the analysis of longest-chain consensus are necessary even under assumption (A5).

3. These ideas are easier to understand in the safe confines provided by assumption (A5), unoccluded by other details.

4. All the guarantees we'll prove for longest-chain consensus under assumption (A5) will hold more generally in the traditional synchronous model, provided the maximum message delay Δ is small relative to the average length of a round.[12]

5. The analysis of this extension to the synchronous model (discussed in Lecture 9) is conceptually straightforward once the main ideas in this lecture (under assumption (A5)) have been properly internalized. (Warning: some amount of math is involved.)[13]

Still, you'd be right to ask: "Isn't the whole point of state machine replication to keep nodes in sync? And haven't you trivialized the problem by asserting that all nodes automatically know the same information?" That's a good criticism. Assumption (A5) doesn't trivialize liveness, but it does seem to trivialize consistency. But let's split "consistency" into two conceptually different properties: (i) consistency between different nodes at a given moment in time; and (ii) "self-consistency," meaning consistency between an honest node and a future version of itself. Property (i) is obviously trivialized by assumption (A5), so let's look at property (ii).

"Consistency with one's future self" means if an honest node i regards a block B as finalized at time t, then it should also regard it as finalized at all moments in time after t. (The order of finalized blocks should also remain the same.) If other words, "finalized" should really mean "finalized"—an honest node should never have to roll back a block that it thought was finalized. We'll sometimes refer to this property as *finality*.

In previous lectures, property (ii) was baked into the definition of the SMR problem—remember that every node maintains an append-only data structure of transactions. (Note

[11] In scenario (PKI) and in the synchronous model, we could modify longest-chain consensus so that each round is carried out by the Dolev-Strong protocol or some other Byzantine broadcast subroutine (with the round's leader playing the role of the sender). This achieves the same effect as assumption (A5), with every round delivering exactly the same information to every honest node. (The only difference between this version of longest-chain consensus and the SMR protocol from Lecture 2 based on iterated Dolev-Strong is how the two protocols handle the case of a Byzantine leader who sends conflicting blocks to different nodes—the former protocol keeps them all (resolving consequent forks in-protocol) while the latter discards them all (thereby preventing any forks from occurring).)

[12] For example, in the Bitcoin protocol the average time between consecutive blocks is, famously, 10 minutes. If we take Δ to be 10 seconds for communication of the internet, the ratio between Δ and the average round length is less than 2%.

[13] The main issue with non-zero message delays is that, in addition to deliberate forks caused by Byzantine leaders, there may also be inadvertent forks caused by honest nodes who didn't have enough time to hear about each other's progress. As long as such inadvertent forks are infrequent, as intuitively should be the case if the maximum message delay is small relative to the average round length, they have a minimal effect on the analysis in this lecture.

"append-only" is equivalent to "nothing gets rolled back.") And for BFT-type consensus protocols like Tendermint, the way we proved consistency for different nodes—by arguing via quorum certificates that there will never be two different blocks finalized at the same block height—immediately implies that no honest node would ever roll back an already-committed block. Longest-chain consensus, however, can be vulnerable to violations of finality in the form of rollbacks. Even under assumption (A5), it is not at all trivial to prove that the protocol satisfies such self-consistency under reasonable assumptions. And happily, the main ideas that prove finality (ii) under assumption (A5) (in Theorem **??**) can be largely reused to also prove consistency between nodes (i) when assumption (A5) is relaxed (to the traditional synchronous model, with a parameter Δ that is small relative to the average round length).

4.5 Recap

Here are the five assumptions, collected together for convenience:

Assumptions (A1)–(A5)

(A1) No node has knowledge of the genesis block prior to the deployment of the protocol.

(A2) It is easy for all nodes to verify whether a given node is the leader of a given round.

(A3) No node can influence the probability with which it is selected as the leader of a round in step (2a).

(A4) Every block produced by the round-r leader must claim as its predecessor some block that belongs to a previous round.

(A5) At all times, all honest nodes know about the exact same set of blocks (and predecessors).

You should be wondering about the roles of these assumptions in this lecture's analysis. The analysis will be modular, with two main steps. The first step identifies a sufficient condition on the sequence of leaders chosen in step (2a) of longest-chain consensus—a type of "balancedness property"—under which the protocol satisfies consistency and liveness (see Section **??**–**??**). The second step identifies conditions (on the amount of Byzantine participation and the method of leader selection, e.g. random leader selection) that guarantee the generation of a balanced leader sequence (perhaps with high probability). Assumptions (A2) and (A3) are about the integrity of the leader selection process; accordingly, they show up (somewhat implicitly) in the second step, as part of proving that Byzantine leaders will not be overly frequent.

The other three assumptions are all important for the first step—for ruling out the possibility that infrequent Byzantine leaders can cause an outsized amount of havoc on the protocol. The role of assumption (A1) is a bit subtle, while assumptions (A4) and (A5) will

be obviously crucial to the analysis. I think it will also be intuitively clear from this lecture's analysis that assumption (A5) is overkill and that similar argument should work (perhaps with a slightly messier proof) in the general synchronous model (with delay parameter Δ well less than the typical round length). Here's a table summarizing the roles of the five assumptions:

Assumption	Type	Used in Which Proofs?
(A1)	trusted setup	Theorem **??**
(A2)	to be enforced	implicit in Section 7
(A3)	to be enforced	implicit in Section 7
(A4)	to be enforced	Theorems **??**, **??**, and **??**
(A5)	to be relaxed	Theorems **??**, **??**, and **??**

5 Honest vs. Dishonest Behavior

Now that we understand what longest-chain consensus looks like at a high level, let's discuss what honest nodes are supposed to do (in step (2b), once selected as a leader) and how Byzantine nodes might deviate from that prescribed behavior.

Prescribed behavior for honest nodes. First, when selected as a leader, an honest node assembles a block from scratch as in our previous SMR protocols—by including all the as-yet-unexecuted transactions that it knows about (perhaps directly from a client, or perhaps from another node), in some arbitrary order. Note that an honest node proposed exactly one block each time it's selected as the leader of a round.

What about the predecessor? An honest node is instructed to look over the in-tree of all the blocks that it knows and set the new block's predecessor to the existing block that is furthest (in terms of number of hops) from the root (i.e., from the genesis block). In other words, a honest node adds its block to the end of the longest chain, making that chain still longer (Figure **??**). And what if there's a tie, with two or more equally long chains (Figure **??**)? For the purposes of our consistency and liveness analysis, we'll be pessimistic and assume that honest nodes break ties in an arbitrarily (even adversarially chosen) manner. You could imagine imposing a tie-breaking rule that honest nodes are supposed to follow (e.g., randomly, or in favor of the block heard about earlier), but guarantees as important as consistency and liveness should not be so fragile as to hinge on how honest nodes carry out tie-breaking.

Finally—this mean seem obvious, but it is something that Byzantine nodes might deviate from—honest nodes are expected to broadcast their chosen block and predecessor immediately after they discover that they're the leader of the current round.

Deviations by Byzantine nodes. A Byzantine node might deviate from the three instructions above in arbitrary ways—proposing more than one block or a block that doesn't contain all known transactions (or even an empty block), naming as a predecessor a block

other than the end of the longest chain, and delaying the announcement of block proposals until some later point in time.

The first type of deviation doesn't seem that bad—we've had to deal with the threat of Byzantine leaders proposing empty blocks ever since the SMR protocol based on iterative Dolev-Strong back in Lecture 2. Unlike in Lecture 2, however, there is not necessarily a one-to-one correspondence between the blocks that get finalized and the blocks that get proposed. Our liveness analysis in Section **??** will have to address this issue by arguing honest nodes regularly manage to assemble blocks that eventually get finalized.

For the purposes of the consistency and liveness analysis in this lecture, the third type of deviation (delayed block announcements) turns out not to matter (other than making the proofs more annoying). Looking it ahead, it *will* matter in Lecture 10 when we study the case of block producers competing for "block rewards." (As we'll see, this also relates to the "chain quality" discussion in Section **??**.) We'll see in that lecture that, perhaps countintuitively, there are scenarios in which a node can increase its block rewards (relative to honestly following the protocol) via a deviation that includes strategically delayed block announcements.

Deviations of the second type—the deliberate creation or perpetuation of forks through the extension of blocks other than the end of the longest chain—should be immediately worrying, and they will be the focus of this lecture's analysis. Why are they worrying? Honest nodes are effectively trying to coordinate on a single (longest) chain, and absent interference by Byzantine nodes the longest chain would keep growing longer and longer. A block that does not the longest chain not only fails to make the longest chain longer, it also threatens to switch which chain is the longest, which would lead to a rollback of blocks on the previously longest chain.

For example, suppose there are 10 rounds in a row with Byzantine leaders. These Byzantine nodes can collaborate and grow a fork of length 10 starting from 9 blocks back, to roll back the last 9 blocks of what had been the longest chain (Figure **??**). For generally, a sequence of k consecutive Byzantine leaders are in a position to roll back $k-1$ blocks. Still more generally, if there is a sequence of leaders in which the Byzantine leaders outnumber the honest leaders by k, the Byzantine nodes can roll back the last $k-1$ blocks of what had been the longest chain (why?). For example, if in a window of 1000 rounds there are 510 Byzantine leaders and 490 honest leaders, the Byzantine nodes are in a position to roll back the last 19 blocks of what was the longest chain at the start of the window.

Takeaways. What can we learn from these examples? Three things:

1. With longest-chain consensus, you should always regard the last several blocks of the longest chain as tentative—not-yet-finalized and still under negotiation. In other words, a block should be considered finalized only once it it ensconced sufficient deeply on the longest chain, already extended by a number of other blocks. How deep on the longest chain does the block need to be? That's an important question, which we'll start tackling head-on in the next section.

2. This example points out a fundamental obstruction to proving any kind of finality result for longest-chain consensus: if you can't rule out sequences of leaders in which the number of Byzantine leaders outnumber the number of honest leaders by k, then you can't expect any kind of finality to hold for any of the last $k-1$ blocks on the longest chain. The best-case scenario would be that this is the *only* obstruction, meaning that the converse also holds: if you *can* rule out such windows, then finality *does* hold for all the blocks on the longest chain other than the last $k-1$ of them. Great news: we'll be able to prove exactly this best-case scenario!

3. This example shows that there's no hope of proving anything about longest-chain consensus unless strictly less than half of the nodes are Byzantine. To see this, imagine that 51% of the nodes are Byzantine. Assume that leader selection is done in a way that every node gets their fair share of turns. Then, 51% of the rounds will have Byzantine leaders. In a window of 1000 consecutive leaders you're likely to see something like 510 Byzantine nodes and 490 honest nodes, and these Byzantine nodes are then in a position to roll back the last 19 blocks of the once-longest chain. In a window of 10000 nodes, if there are 5100 Byzantine leaders and 4900 honest leaders, the Byzantine nodes will be able to roll back the last 199 blocks of the once-longest chain. And so on. Given that Byzantine leaders can outnumber honest leaders by an arbitrarily large amount (as the sequence length grows large), no block is ever safe from being rolled back—the Byzantine nodes effectively have dictatorial control over which blocks wind up on the longest chain.

   So, a necessary condition for longest-chain consensus to have any provable guarantees is that more than half of the nodes run the protocol honest and correctly; the best-case scenario would be that this condition is also sufficient (for provable consistency and liveness). Great news again: this is exactly what we'll prove!

6 Which Blocks Are Finalized?

Given the power of the potential deviations (specifically, deliberate forking) by Byzantine nodes identified in the previous section, we've homed in on the coolest statement that might conceivably be true about longest-chain consensus: that whenever more than 50% of the nodes running the protocol are honest, all blocks on the longest chain other than the last k can be considered finalized. (Along with the other usual desired properties, like liveness.) And is exactly the main punchline of the analysis in this lecture.

6.1 The Parameter k

The parameter k—the number of blocks at the end of the longest chain that are still considered unfinalized and under negotiation—will obviously be an important one for us. On the one hand, we'd like it to be as small as possible (so that blocks of transactions get finalized as

soon as possible after they are created); on the other, intuitively, larger (more conservative) values of k seem more likely to allow for provable consistency guarantees.

Let's introduce some notation to reason about this parameter: for an in-tree G of block (rooted at the genesis block) and a nonnegative integer k, define:

$$B_k(G) := \text{the longest chain of } G\text{, with the last } k \text{ blocks removed.} \tag{1}$$

For example, for the in-tree G in Figure **??**, $B_2(G)$ is the chain $B_0 \leftarrow B_2 \leftarrow B_4$ while $B_3(G)$ is the chain $B_0 \leftarrow B_2$.

There are two possible points of confusion. First, if you look at the description of longest-chain consensus in Section 3, you'll notice that there's no parameter k—the evolution of the protocol is completely independent of what value of k you might have in mind. Rather, the parameter k dictates how to *interpret* the evolution of the protocol—how many blocks at the end of the longest chain you should regard as "still tentative" in some sense.

The parameter k is in the eye of the beholder, and can therefore be sent client-side, and differently for different clients. (We're using "client" here in the usual tech sense—the user-facing application that interacts with the blockchain—rather than in the strict SMR sense.) For example, imagine a blockchain protocol based on longest-chain consensus that also has a native currency (like Bitcoin), and imagine that you're a seller who accepts payments in this currency. If you're a high-volume seller of a low-cost product, like cups of coffee, you might be content to hand over a cup of coffee to a customer after that customer's payment transaction has been added to the longest chain and extended by a single block ($k = 1$). You'd run some risk of the transaction getting rolled back (and the customer effectively getting a free cup of coffee), but perhaps you view minimizing customer wait time as more important. If you're selling Teslas, on the other hand, probably you want to make the customer wait for awhile (maybe $k = 100$), so that you can be confident that their payment can be regarded as finalized, never to be evicted from the longest chain.

The second point of confusion, which you'd be right to wonder about, it whether the notation $B_k(G)$ is even well defined in a mathematical sense. In (1), who am I to write "*the* longest chain," when we all know that they're might be multiple chains tied for the longest? Figure **??** depicts an in-tree G with two longest chains. Here, $B_2(G)$ actually is well defined—no matter which of the two longest chains you choose, after lopping off the last 2 blocks you get the same thing (the chain $B_0 \leftarrow B_2$). But $B_1(G)$ is not well defined, because if you lop off only 1 block from the end you're still left with different chains ($B_0 \leftarrow B_2 \leftarrow B_4$ and $B_0 \leftarrow B_2 \leftarrow B_5$). In general, we say that "$B_k(G)$ is well defined," we mean that it doesn't matter which longest chain you choose, after lopping off the last k blocks you get the same thing. Equivalently, all longest chains agree on all their blocks except possibly for their last k. Said still another way, all longest chains share a common prefix—if they all have length ℓ, then the prefix of the first $\ell - k$ blocks is shared across them.

6.2 Balanced Leader Sequences

In our analysis, the first order of business (Theorem **??**) will be to prove this "common prefix property": provided more than 50% of the nodes are honest and k is sufficiently

large, $B_k(G)$ is guaranteed to be well defined. From there, the goal will be to prove finality, meaning that the blocks of $B_k(G)$ can be considered finalized (Theorem **??**), and liveness, meaning that $B_k(G)$ continues to grow over time, with blocks assembled by honest nodes regularly added.

The plan. Our analysis will be modular. To explain, consider a sequence of rounds of longest-chain consensus and their corresponding leaders. If you think about it, there's no need to differentiate between which honest node is selected as a leader (each is following the protocol and thus behaving identically) nor which Byzantine node is selected as a leader (we always assume that Byzantine nodes are acting in cahoots, so it doesn't matter which one is chosen). We can therefore think of the leaders of a sequence of rounds as a bunch of "H"s (for honest) and "A"s (for adversarial). For example, we argued in the previous section that in any sequence of rounds in which there are k more "A"s than "H"s, the Byzantine nodes are in a position to cancel the last $k-1$ blocks of the current longest chain. The plan for the modular analysis is then:

1. (Definition 6.1) State a definition that articulates a condition that a given leader sequence may or may not meet.

2. (Sections **??**–**??**) Prove that, as long as the sequence of leaders generated in longest-chain consensus satisfies this condition (and the parameter k is chosen appropriately), the protocol satisfies all the properties that we want (consistency, liveness, etc.).

3. (Section 7) Prove that, as long as more than half of the nodes running the protocol are honest, this condition is satisfied (perhaps with high probability).

Chaining together the second and third steps shows exactly what we want: as long as more than half the nodes are honest and the parameter k is set appropriately, longest-chain consensus enjoys provable consistency and liveness guarantees.

Balanced leader sequences. Here's the key definition (with respect to a parameter w, which stands for "window" and is a positive integer):

Definition 6.1 (w-Balanced Leader Sequence) A sequence $\ell_1, \ell_2, \ell_3, \ldots \in \{H, A\}$ is *w-balanced* if, in every window $\ell_i, \ell_{i+1}, \ldots, \ell_{j-1}, \ell_j$ of length at least w, the H's outnumber the A's.

The bigger w is, the easier Definition 6.1 to satisfy (because there are fewer windows to worry about). Thus will generally be interested in the smallest w for which a leader sequence is w-balanced.

For example, any leader sequence that includes at least one Byzantine node fails to be 1- or 2-balanced (in a window of length 1 or 2 that contains that node, the honest nodes do not outnumber the Byzantine nodes). A sequence in which every pair of Byzantine leaders are separated by at least two honest nodes is 3-balanced (why?). A sequence generated by

the round-robin rotation through n nodes, with $f < n/3$ Byzantine nodes, is n-balanced (why?). (Homework: what about with the weaker condition $f < n/2$?) A leader sequence in which the average frequency of Byzantine nodes exceeds that of honest nodes—e.g., with $f > n/2$ and either round-robin or random leader selection—will not be w-balanced for any w (why?).[14]

Implementing the second step of the plan. Let's connect Definition 6.1 back to the second takeaway from Section 5. We saw in that section how, in a window in which there are k more Byzantine nodes than honest nodes, Byzantine nodes can cancel the last $k-1$ blocks of the current longest chain. The best-case scenario is that this obstruction to finality is the *only* obstruction to finality, meaning that as long as there are no windows in which the number of Byzantine nodes outnumber the honest nodes by more than k, then all but the last $k-1$ blocks of the longest chain can safely be considered finalized.

In a w-balanced leader sequence, meanwhile, there cannot be any window in which the Byzantine nodes outnumber the honest nodes by $w/2$ or more (why?). The dream, then, would be that whenever Definition 6.1 is satisfied with some parameter w, all but the last $(w/2)-1$ blocks of the longest chain can be considered finalized. And this is exactly what we're going to prove in Theorem **??** (under the assumptions (A1)–(A5) from Section 4)! Specifically, we'll use the w-balancedness condition in in the proof of the common prefix property—that the notion $B_k(G)$ above is well defined, with all longest chains agreeing on all but possibly their final k blocks—in Theorem **??** (see Section **??**), with the finality guarantee in Theorem **??** then following easily in Section **??**. We'll also use the w-balanced condition in the proofs of liveness (Theorem **??** in Section **??**) and chain quality (Theorem **??** in Section **??**).

Implementing the third step. The second step of our modular analysis shows that if you leader sequence is balanced—for whatever reason—then you're good, with all the consistency and liveness properties that you might want. But why should the leader sequence be balanced in the first place? We already argued that it *won't* be balanced if more than half the nodes are Byzantine, but why should it be balance if more than half the nodes are honest?

In Section 7 we'll drill down on the version of longest-chain consensus in which each leader is chosen independently and uniformly at random. (This is a natural design already in scenario (PKI), but more importantly it's completely essential to the permissionless implementations of longest-chain consensus in Lectures 9 and 12 for scenarios (PoW) and (PoS), respectively.) The punchline of that analysis (Theorem **??**) will be that, as long as less than half the nodes are Byzantine, random leader selection generates a balanced leader sequence with high probability. The intuition is that—by the Law of Large Numbers, essentially—honest nodes will have (almost) proportional representation in every sufficiently large window

[14] This is actually a reassuring sanity check: we said that our plan was to show that Definition 6.1 is a sufficient condition for the consistency and liveness of longest-chain consensus (provided the parameter k in the definition of $B_k(G)$ is sufficiently large), and we already argued that the protocol does *not* have these properties when more than half the nodes are Byzantine, so in that case it had better be that Definition 6.1 can't possibly be satisfied!

of consecutive leaders. (E.g., if 51% of the nodes are honest, then in all sufficiently large windows there should be somewhere between 50.5% and 51.5% honest nodes.)

How balanced a sequence, exactly—how big do the window lengths need to be before proportional representation kicks in? (Remember that the more balanced it is—i.e., the smaller w is—the smaller we can take k and the faster blocks can be finalized.) The answer depends on various parameters (the fraction of nodes that are Byzantine, the duration you're interested in, and the acceptable failure probability), and we'll get into the details in the next section. For typical parameter values, the analysis suggests taking k (the number of additional blocks needed before considering a block to be final) in the low-to-mid double-digits. More aggressive values are often used in practice—for example, for the Bitcoin protocol the famous rule of thumb is to take $k = 6$. (Though when selling a Tesla, k should definitely be taken to something larger than that!)

7 Random Leaders Are Balanced

7.1 Random Leaders and Probabilistic Guarantees

Tim Roughgarden: What we saw at the end of the last video is that the key that unlocks all of the good properties of longest chain consensus, it really boils down to how balanced is the sequence of leaders. Just to remind you, what does it mean that a leader sequence is W balanced? It means that if you look at any interval of W or more consecutive leaders, over half of them should be honest nodes. For example, if you're 100 balanced, that means if you look at any stretch of 100 consecutive leaders, at least 51 of them should be honest. Similarly, if you look at any stretch of 200 consecutive leaders, at least 101 of them should be honest, and so on.

We'll see in the next few sections proofs that, as long as the sequence of leaders generated in longest-chain consensus is w-balanced (see Definition 6.1), the protocol satisfies consistency and liveness (with the parameter k chosen appropriately, and under assumptions (A1)–(A5)). When can we count on a balanced sequence of leaders? And for what balance parameter w?

In this section we'll investigate, in scenario (PKI), and the version of longest-chain consensus in which, every round, the leader is selected uniformly at random from the set of all nodes (i.e., with each node equally likely to be chosen). This is easy to do in the permissioned setting, but it also extends amazingly gracefully to the permissionless setting (scenarios (PoW) and (PoW), see Lectures 9 and 12).

Even in the permissioned setting, given that we want a balanced leader sequence, random leaders sound like a pretty good idea. For example, imagine that there are 3000 nodes, 1000 of which are Byzantine. If we cycle through the nodes in some fixed round-robin order, for all we know all 1000 Byzantine nodes appear consecutively in the ordering. In this case, the leader sequence generated is 2001-balanced but not w-balanced for any $w < 2001$ (why?). If instead each leader is selected randomly, then any given leader has a two-out-of-three chance of being honest. You might for course get a few Byzantine leaders in a row by

random chance, but getting something like 100 Byzantine leaders in a row—an event with probability $(1/3)^{100}$—would presumably happen so rarely that we could ignore the possibility.

On the other hand, no matter how big w is, there is some positive—if astronomically small—probability of seeing w Byzantine leaders in a row. And this would be true even if there was only one Byzantine node! Thus with randomly selected leaders, though no hope of proving any guarantees for longest-chain consensus that hold with certainty. (For example, any block might eventually get rolled back if the protocol winds up choosing a sufficiently long sequence of consecutive Byzantine leaers.) The best-case scenario would be to prove that consistency and liveness hold with high probability (meaning probability close to 1, like 99.9%). Such probabilistic guarantees are typical of permissionless implementations of longest-chain consensus (including Bitcoin). If the failure probability is sufficiently small (e.g., less than the probability of your neighborhood getting hit by an asteroid in the next 24 hours), then probabilistic guarantees are for all practical purposes as good as deterministic guarantees.

7.2 Intuition for the Analysis

Taking the third step of this lecture's analysis (Section **??**–**??**)—that a balanced leader sequence is all you need for consistency and liveness—on faith for now, let's investigate the extent to which a sequence of random leaders is balanced. We'll keep the discussion a bit on the intuitive and informal side; it wouldn't be hard to turn this discussion into a rigorous proof, but that wouldn't be the best use of our time.

Let α the fraction of nodes that are Byzantine. We saw in Section 5 that there's no hope of proving anything unless $\alpha < \frac{1}{2}$, so let's go ahead and assume that from here on out. The basic idea is:

1. A consequence of the large of large numbers is the hopefully intuitive fact that, if you repeat an experiment with success probability p a large number of times, the long-run fraction of successes you'll see is (almost always) going to be roughly p.

2. Identifying a successful experiment with the selection of an honest leader, this means that after enough repeated trials the long-run fraction of honest leaders is (almost always) going to be roughly $1 - \alpha$.

3. Because $\alpha < \frac{1}{2}$, this proportional representation of honest leaders means that the honest leaders should outnumber the Byzantine leaders in every sufficiently large window.

The parameter w in the w-balancedness condition will control how large we need to take the parameter k—the number of subsequent blocks required to regard a block as finalized—to guarantee consistency and liveness. Time-to-finalization is an important performance characteristic of a blockchain protocol, so we'd like a more quantitative understanding of just how balanced a sequence of random leaders is likely to be.

The exact guarantee on the parameter w will depend on just how far α is from $\frac{1}{2}$—the closer it is to α, the bigger the w we'll need to take to ensure sufficiently proportional representation of honest leaders.

For example, imagine that α is 49%. In a length-100 window of consecutive leaders, we would expect 51 honest nodes and 49 Byzantine nodes. There will be some variance, of course—sometimes there will be 55 honest leaders, sometimes there will be 46 (which would violated 100-balancedness), and so on. So we would not expect a random leader sequence to be 100-balanced when $\alpha = 0.49$.

In a length-1000 window, meanwhile, we would expect 510 honest nodes and 490 Byzantine nodes—now we have a margin of error of 20 between the two, and it intuitively seems that we should have a better shot at seeing a majority of honest nodes. Sometimes there might 515 honest nodes, sometimes 505, sometimes 523, sometimes 498 (which would violate 1000-balancedness), and so on. So we might expect a random leader sequence to be 1000-balanced with somewhat high (but not 99.9%) probability. Similarly, with length-10000 windows, we have a buffer of 200 between the expected number of honest and Byzantine nodes to absorb the inevitable variations, and it would seem still more unlikely to have an unluckily low number (< 5001) of honest nodes. If this intuition is correct, given a desired failure probability δ, we should be able to take the window length large enough (as a function of α and δ) to guarantee a failure probability of at most δ.

7.3 Quantitative Analysis of Random Leader Selection

Exponentially small bounds. The intuition above is quite accurate. But if you really want some concrete numbers, you have to do some actual math. And if you do the math, the very cool thing that you discover is that not only is the probability of a w-balancedness violation decreasing with the window size, it's decreasing *exponentially* quickly. More formally:

$$\mathbf{Pr}[\text{a given length-}w\text{ window is at least half Byzantine}] \leq e^{-cw}, \tag{2}$$

where c is some constant (independent of w). (The constant c depends on α—the closer α is to $\frac{1}{2}$, the smaller the c and the slower the decrease in failure probability.) If you want to keep a concrete number in mind, think of c as being (for example) 0.1. Also, the "e" in (2) denotes the base of the natural logarithm, 2.718....

An exponentially small failure probability like the one in (2) is good news: every time you bump up the window length w by 1, the failure probability drops by another constant factor. This is the key property behind the assertion that random leader sequences are w-balanced for reasonably small values of w (with high probability). (Remember, w controls the time to finality, and we want to take it to be as small as possible!)

Applying the Union Bound. The failure probability in (2) concerns a specific window (e.g., the leaders of rounds 101, 102, ..., 140). Looking back at Definition 6.1, we see that it asserts something about *every* window of sufficiently large length (e.g., at least 40 leaders in a row), not just one window. To bridge that gap, we can use the Union Bound, which states that probability that any bad event happens is at most the sum of their individual

probabilities:[15]

$$\mathbf{Pr}[\text{any of events } E_1, E_2, \ldots, E_k \text{ occur}] \leq \sum_{i=1}^{k} \mathbf{Pr}[\text{event } E_i \text{ occurs}]\,.$$

Or, in terms of the complementary event:

$$\mathbf{Pr}[\text{none of events } E_1, E_2, \ldots, E_k \text{ occur}] \leq 1 - \sum_{i=1}^{k} \mathbf{Pr}[\text{event } E_i \text{ occurs}]\,. \tag{3}$$

For us, the individual events E_i correspond to the windows of length at least w (with E_i occurring if and only if the corresponding window is at least half Byzantine leaders). The key point is that, by definition, a leader sequence is w-balanced if and only if none of these events occur.

Now imagine that we're interested in some stretch of T consecutive rounds of longest-chain consensus (which could represent a day, a month, etc.). The number of windows (of length at least w, or otherwise) can be crudely bounded by T^2 (with only T choices for the first and for the last round of the window). Because there are at most T^2 E_i's and (by (2)) $\mathbf{Pr}[E_i] \leq e^{-cw}$ for every i, the Union Bound (3) tell us that

$$\mathbf{Pr}[\text{a sequence of } T \text{ randomly selected leaders is } w\text{-balanced}] \leq T^2 \cdot e^{-cw}. \tag{4}$$

Solving for w given choices of T and δ (and α). So, is the number in (4) big or small? The answer depends on our choice of w. Fix a failure probability δ that we're comfortable with, such as $\delta = .01$. The bound in (4) then tells us how to set w, by replacing the right-hand side by δ and then solving for w (by taking logarithms of both sides and rearranging). When the dust settles, the punchline is:

a sequence of T random leaders is w-balanced with probability at least $1 - \delta$, provided

$$w \geq c_2(\ln T + \ln \tfrac{1}{\delta}).$$

Here, c_2 is a constant independent of w and δ. (Like c_1, it does depend on α—the closer α is to $\frac{1}{2}$, the bigger the constant factor c_2.)

The exponentially small probability in (2) thus translates to a lower bound on w that involves only the *logarithms* of the two key parameters, the duration T of interest and the (reciprocal of the) acceptable failure probability. The most important fact about logatithms is that they grow really slowly—for example, if we're talking about the base-2 logarithm, the log of 1000 is roughly 10, the log of one million is roughly 20, the log of one billion is roughly 30, and so on. This means that we take the duration T to be quite long and the failure probability δ quite small while still getting away with palatable values of w.

[15] Pictorially, the area of the union of a bunch of regions is at most the sum of the regions' areas (Figure **??**). Equally holds if and only if the regions do not overlap.

Recap.

1. As we'll see in Sections **??**–**??**, the number of blocks k required before finalizing a block in longest-chain consensus is controlled by the smallest w for which the generated leader sequence is w-balanced (with $k = (w/2) - 1$).

2. Randomly chosen leaders are w-balanced with high probability, where the parameter w depends on the fraction α of Byzantine nodes, the duration T of interest, and the desired failure probability δ (with w increasing with α, T, and $1/\delta$).

3. Because the lower bound on w increases only logarithmically quickly with T and $1/\delta$, for typical parameter values (say $\alpha = 0.33$, $T = 1000$, and $\delta = .01$) it's safe to take w (and hence k) in the double digits. (A more detailed analysis can be used to improve this conservative lower bound by a constant factor [**?**].)

7.4 Toward Permissionless Consensus

Longest-chain consensus is already interesting to study in the permissioned setting (scenario (PKI)), but what's really special about it is how easily it extends to the permissionless case. Looking ahead to Lectures 9 and 12, let's examine exactly how the permissioned assumption was used in the analysis of this section.

Rereading the informal argument in Section 7.2, we see that only one thing matters for the analysis: that every round, the probability that a Byzantine nodes is selected is less than 50%. (This then leads to (almost) proportional representation of honest nodes in sufficient long windows, all of which will then have a majority of honest nodes.)

Key Property for Analysis of Random Leaders

There is a constant $\alpha < \frac{1}{2}$ such that, in every round, the probability that a Byzantine node is selected as the round's leader is at most α. (Also, each leader should be chosen independently of any of the previous ones.)

If this property holds—for whatever reason, whatever the concrete implementation of longest-chain consensus—the "proportional representation" argument remains valid.

Now, in scenario (PKI), it's easy to think of a combination of assumptions and protocol design choices that together ensure that the key property above holds: if less than half the nodes are Byzantine and each leader is selected independently and uniformly at random from the set of all nodes, then the probability that a given round has a Byzantine leader is less than 50%.

In permissionless scenarios (PoW) and (PoS), where the set of nodes running the protocol is unknown, it's not obvious how to sample a node uniformly at random. *But*, if we can find a combination of assumptions and (permissionless) protocol design decisions that ensure the key property above, we'll be good to go (the generated leader sequence will be balanced with high probability, which as we'll see in the following sections is sufficient for consistency and liveness).

References