Foundations of Blockchains
Overview of Lectures #2–7:
Bootcamp on Classical Consensus[*]

Tim Roughgarden[†]

1 The Upshot

1. Lecture 2 introduces the synchronous model (with reliable message delivery) and covers the amazingly fault-tolerant Dolev-Strong protocol for state machine replication.

2. Lecture 3 presents our first impossibility result, showing that the guarantees of the Dolev-Strong protocol are provably unachievable without some type of "trusted setup" assumption (like the in-advance distribution of nodes' public keys).

3. Lectures 4 and 5 introduce the asynchronous model, which makes no assumptions about message delivery (other than eventual delivery), and proves what is arguably the most famous impossibility result in distributed computing: deterministic consensus is impossible in the asynchronous setting.

4. Lecture 6 introduces the partially synchronous model as a sweet spot between the synchronous and asynchronous models, proves that consensus is impossible in this model when more than 33% of the nodes can deviate from the protocol, and compares these results to a famous principle from distributed systems, the CAP theorem.

5. Lecture 7 describes the Tendermint protocol (the basis of the Cosmos and Terra networks, among others) and proves that it achieves optimal fault-tolerance in the partially synchronous setting.

2 Preamble

These notes present capsule summaries of Lectures 2–7, which provide a bootcamp on some of the amazing work that was done in distributed computing on consensus in the 1980s.

[†] 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.

(And, in Lecture 7, a case study of a modern instantiation of these ideas via the well-known Tendermint blockchain protocol.) This work may be "old," but it remains super-important today (hence all the time we'll devote to it). For an analogy, think about algorithms—concepts like dynamic programming and NP-completeness are more than a half-century old, and yet they continue to very much inform modern algorithm design. So too does this classical work on consensus protocols inform modern blockchain protocol design.

The bootcamp covers several possibility results, meaning we'll talk about some concrete consensus protocols and rigorously prove that they have desirable guarantees. We'll also go through several proofs of impossibility results, which show that good consensus protocols do not exist unless certain assumptions are made. These are all famous results with cool proofs, so I encourage you to spend some quality time with this material (and don't get frustrated if you get lost now and then). This bootcamp is the most mathematically intense portion of the lecture series; the rest of the series is also geared toward technical people (e.g., those with some computer science training) but is significantly less proof-heavy.

3 Capsule Summaries

3.1 Purposes of Summaries

The following brief lecture summaries serve four purposes.

1. They make explicit the story told by these six lectures, and the logic behind their sequencing.

2. They serve as a pitch to entice you to spend some serious time absorbing all the cool protocols, theorems, and proofs in these lectures.

3. For those of you who only want to dabble in classical consensus, the summaries will indicate which parts of which lectures should be the most interesting and valuable to you.

4. I'm not under any delusions and realize that the last thing some of you want to spend time on is hard math. For those of you that plan to skip the bootcamp and move on to other parts of the lecture series, these brief descriptions will at least provide you with a summary of what you're going to missing and the bootcamp's high-level takeaways.

3.2 Lecture 7: Tendermint

The overarching narrative will be clearest if I summarize the bootcamp lectures out of order. The culmination of these six lectures is a detailed case study of a real-world blockchain protocol called the Tendermint protocol. This protocol is widely used and is a key building block of, for example, the Cosmos and Terra ecosystems.

"Principles over protocols" is one of the key themes of this lecture series, but this will be one of the few instances where we talk in detail about how a specific real-world protocol works. Our interest in the Tendermint protocol stems as much from the provable guarantees that it enjoys (best-possible fault-tolerance in the partially synchronous model, subject to consistency and eventual liveness) as its many real-world use cases. Tendermint will also serve as the modern embodiment of several key ideas developed in the 1980s and 1990s for classical state machine replication (SMR) protocols. We'll take care to rigorously prove the Tendermint protocol's consistency and liveness guarantees, as these proofs are the best way to understand on a deep level why the protocol works the way that it does.

3.3 Lecture 6: The Partially Synchronous Model

Before stating and proving rigorous guarantees for the Tendermint protocol in Lecture 7, we'll need to do some work figuring out the right model and the right properties to focus on. We'll analyze the Tendermint protocol in what's known as the "partial synchronous model." Intuitively, in this model the underlying communication network alternates between phases of "normal operation" (with reliable message delivery) and "attack mode" (with unbounded message delays, due to outages or denial-of-service attacks). If you're focused on blockchains and want to remember only one model of a communication network, this should be the one.

Ideally, a protocol should work well in periods of normal operation, not break too badly when under attack, and recover quickly once the network returns to normal operation. One encoding of these intuitive goals (for the SMR problem) is to insist on consistency (nodes never disagree on transaction ordering, even when under attack) and "eventual liveness" (meaning transactions may hang while the network is under attack, but they will be processed once normal operation resumes).

The central technical result in Lecture 6 is an impossibility result due to Dwork-Lynch-Stockmeyer, stating that there's no hope of designing a good SMR protocol (i.e., one that guarantees consistency and eventual liveness) in the partially synchronous model unless at least 67% of the nodes are honest (i.e., correctly run the protocol). This impossibility result, in tandem with the guarantees we'll prove for the Tendermint protocol in Lecture 7, shows that there's a magical threshold of 33% in the partially synchronous model: good SMR protocols exist *if and only if* the fraction of faulty nodes is less than this threshold. Ever see "33%" (or "67%") mentioned as a critical threshold in a blockchain whitepaper (e.g., Solana, proof-of-stake Ethereum, Polkadot, Algorand, etc.) or in a discussion of various "layer-1" protocols? This impossibility result is the exact source of that ubiquitous magical threshold, and in Lecture 6 you'll see a full proof of it (along with ample intuition about why it is true).

The final goal of Lecture 6 is to relate our discussion of what's possible and impossible in the partially synchronous model to a famous principle from distributed systems known as the CAP theorem ('C' for "consistency," 'A' for "availability," and 'P' for "partition-tolerance").

3.4 Lecture 2: The Dolev-Strong Protocol (Synchronous Model)

If the partially synchronous model of Lecture 6 is the best one to remember, due to its accurate articulation of the trade-offs that real-world blockchain protocols must grapple with, why bother with anything else? Well, if I started out Lecture 2 with a formal definition of the partially synchronous model, you'd walk away in disgust, muttering "Why such a fancy definition for such a simple concept as network reliability?" The correct (and historically accurate) way to understand the partially synchronous model is as an interpolation between two more obvious models of communication. One of those is called the synchronous model and is the subject of Lectures 2 and 3. The polar opposite is the asynchronous model, which will be the subject of Lectures 4 and 5.

The synchronous model assumes reliable message delivery, meaning that whenever one node running a protocol sends a message to another node running that protocol, that message will be delivered within a bounded amount of time. Formally, there will be a bound Δ on the maximum number of time steps that can elapse before a message is guaranteed to have been received. (The parameter Δ is assumed known up front, meaning that it can be baked into the protocol's code.) This is a semi-reasonable model of a communication network in times of normal operation. For example, if the Internet is having a particularly good day, it resembles the synchronous model with the parameter Δ set to a couple of seconds.

The good news about the synchronous model is that it enables some pretty impressive SMR protocols. The main goal of Lecture 2 is to explain the design and analysis of such a protocol, known as the Dolev-Strong protocol (from the early 1980s). This protocol is for a single-shot consensus problem known as "Byzantine broadcast," but we'll also see in Lecture 2 how to reduce the SMR problem (which is multi-shot consensus) to the Byzantine broadcast problem. This reduction extends the Dolev-Strong protocol to an SMR protocol with all the properties that we might want. What's really extraordinary about the Dolev-Strong protocol is that it tolerates an arbitrarily large number of dishonest nodes. Even if 98 out of 100 nodes are malicious and acting in cahoots, there's nothing they can do to trick the remaining 2 honest nodes into getting out of sync. The bad news about the synchronous model is that its assumptions are simply too strong to provide accurate guidance for the design of Internet-scale protocols.

You won't see the Dolev-Strong protocol referenced very often in blockchain whitepapers and discussions, but it's worth studying for several reasons. One, if you ever *do* find yourself in a setting where the synchronous model is appropriate (perhaps in a private network), you'll know how to achieve a pretty amazing set of guarantees. Second, studying this (relatively simple) protocol will act as a good warm-up to more complicated protocols that we'll see later (like Tendermint)—we'll get a sense for what typical consensus protocols look like and how they can make use of tools such as digital signatures. Finally, it's a famous result—definitely part of the distributed computing greatest hits reel—and readers who identify as card-carrying computer scientists should enjoy learning how and why it works.

3.5 Lecture 3: FLM Impossibility (Synchronous Model, no PKI)

Lecture 3 concerns a famous impossibility result which, like the one in Lecture 6, rules out good SMR protocols when at least 33% of the nodes can deviate from the protocol. Unlike the Lecture 6 impossibility result (for the partially synchronous model), the one here applies even in the (easier) synchronous model. But wait, why doesn't this contradict the guarantees promised by the Dolev-Strong protocol, which hold even when almost all the nodes are malicious?

This lecture drills down on a key assumption made by the Dolev-Strong protocol, typically called the PKI (for "public key infrastructure") assumption. The assumption is that all the nodes running the protocol have their own public key-private key pairs and that all nodes' public keys were somehow exchanged before the commencement of the protocol. In other words, all nodes begin the protocol with the ability to verify a signature by any other node.

While the PKI assumption is fairly palatable in a blockchain context, it winds up being quite interesting to study what's possible without it and other "trusted setup" assumptions. And the 33% impossibility result of this lecture pops up already in the synchronous setting when there are no trusted setup assumptions. The proof of this impossibility result—due to Fischer-Lynch-Merritt and called the "hexagon argument," for reasons that will become clear—is impressively slick. For those of you who like cool proofs, Lecture 3 is likely to be your favorite one of the bootcamp. Conversely, if you're not into proofs and wanted to skip one of these six lectures, Lecture 3 should probably be the one. It's a famous and enlightening result that belongs to the distributed computing canon (like the other impossibility results in the bootcamp), but it is not 100% essential to the bootcamp's overarching narrative.

3.6 Lectures 4 and 5: FLP Impossibility (Asynchronous Model)

The synchronous model allows consensus protocols with amazing guarantees (at least under the PKI assumption) but makes assumptions that are simply too strong for protocols that operate over the Internet. We therefore have no choice but to weaken our assumptions about the underlying communication network. One natural idea is to assume *nothing* about the communication network, other than the minimal property that every message is delivered eventually. This is the idea behind the "asynchronous model."

The good news about the asynchronous model is that, with essentially no assumptions about the communication network, any consensus protocol with non-trivial provable guarantees in the model would automatically be impressively robust. The bad news is that this model throws out the baby with the bathwater, in the sense that consensus is provably impossible (at least for deterministic protocols). This result, due to Fischer-Lynch-Paterson, is perhaps the most famous impossibility result in all of distributed computing. Its proof will be the hardest one that we'll see in this bootcamp, and probably the hardest one in the whole lecture series. We'll get started on the proof in Lecture 4 (after formally defining the asynchronous model) and finish it off in Lecture 5.

The key takeaway from the FLP impossibility result is that the asynchronous model simply doesn't make enough assumptions—for us, the point of a model is to help us brainstorm about the space of protocols and hopefully identify some that work well in both theory and practice. With no good (deterministic) protocols in this model, there's not much brainstorming to be done! The FLP impossibility result thus completes the justification for introducing a third model (the partially synchronous model) in Lecture 6; while slightly less natural than the synchronous or asynchronous model, it acts as a sweet spot between the two, with assumptions that are weak enough to be relevant to Internet-scale protocols and strong enough so that practical consensus protocols with provable guarantees actually exist.

4 Looking Further Ahead: Longest-Chain Protocols

BFT-type protocols. We will refer to blockchain protocols that resemble the consensus protocols from the 1980s and 1990s (like Tendermint) as *BFT-type* protocols. (Here BFT stands for "Byzantine fault tolerant.") Several of the major "layer-1" networks use BFT-type blockchains, including Solana, Terra, Cosmos, and Algorand. When Facebook/Meta was considering rolling out its own blockchain (for the Diem cryptocurrency), they were planning to base it on a BFT-type protocol called HotStuff. This bootcamp thus prepares you well to understand how these BFT-type blockchain protocols work.

Longest-chain protocols. You've probably noticed that I've failed to mention the two most famous blockchain protocols of them all, Bitcoin and Ethereum. These two protocols achieve consensus in a radically different way and are examples of *longest-chain* protocols. The idea of longest-chain consensus does not come from the classical consensus literature and was invented specifically for the Bitcoin protocol. (Several blockchain protocols that came later, including Ethereum, also adopted longest-chain consensus.[1]) Our first order of business after our classical consensus bootcamp is to develop a deep understanding of longest-chain consensus and its pros and cons relative to BFT-type consensus. We'll get started on this topic in Lecture 8.

5 Conclusion (and Blockchain Failure Modes)

At the conclusion of this bootcamp, you will have accomplished two really important things. First, you'll have mastered the basic concepts behind one of the two major design paradigms for blockchain protocols (BFT-type protocols). Second, and possibly even more important, you'll be equipped with the mental model, the formal definitions, and the language to discuss, assess, and rigorously compare different blockchain consensus protocols (including longest-chain protocols and anything else that protocol designers manage to come with).

Fundamental safety-liveness trade-offs. For example, the impossibility results proved in this bootcamp show that all blockchain consensus protocols must make compromises. Different protocols choose different compromises. For example, if you read or hear that a particular blockchain protocol "favors safety," it's probably a BFT-type protocol; as we'll see, such protocols guarantee consistency (i.e., safety) in the partially synchronous setting, even when under attack. If you hear that a protocol "favors liveness," it's probably in reference to a longest-chain protocol (which, as we'll see, typically continues to produce new blocks of transactions even when under attack).

Blockchain failure modes. The fundamental trade-offs that any blockchain protocol must face (e.g., between consistency and liveness when under attack) explain why different protocols fail in very different ways. When you hear about a BFT-type protocol like Solana

[1] In 2022, Ethereum is slated to incorporate several elements of BFT-type protocols into its new consensus protocol.

or Cosmos failing (perhaps due to a bug and/or crushing load), it's because the protocol has stalled and stopped processing any new transactions. By contrast, while you don't generally hear about a longest-chain protocol like Bitcoin or Ethereum stopping altogether, you do hear about "re-orgs" (i.e., chain reorganizations) in which a large number of transactions that had been considered confirmed get rolled back in favor of an alternative transaction sequence.[2] The reason you hear about BFT-type protocols stalling when under stress is exactly because they favor safety over liveness, and the reason you hear about re-orgs in longest-chain protocols is exactly because they make the opposite trade-off.

[2] Or at least, you hear about them for longest-chain protocols that are less secure than Bitcoin and Ethereum, including Ethereum Classic.