Foundations of Distributed Consensus and Blockchains

(Preliminary Draft)

Elaine Shi

Dedicated to the memory of my beloved mother, Honghua Ding (1952-2017), the kindest and most talented person I've known.

献给我最爱的母亲丁虹华 (1952-2017)

Preface

This is a preliminary (read: still somewhat rough) draft. I will be continually updating it as I teach this course in the next few years. Please always check back for the latest version.

Acknowledgements: I would like to thank my student Andrew Miller who first got me interested in distributed consensus about five years ago. I am indebted to my colleague Rafael Pass, who, since I joined Cornell, blocked out time from his busy life to discuss consensus with me, and who also suggested to write this textbook. Thanks especially to Hubert Chan, my best friend and collaborator, for discussing distributed consensus, research, life, universe, and everything with me, throughout the past almost two decades, and for always understanding that I meant "left" when I said "right".

Many people have provided me valuable feedback on the first draft of the textbook, including Elisaweta Masserova, Daniel Lavie, Tal Moran, Qixing Huang, Joerg Kliewer, Krishna Podder, Ling Ren, Kartik Nayak, Bryan Tantisujjatham, and Kai Zou. I am still working on addressing some of this feedback. A few of the chapters benefited from scribe notes from the students in my graduate-level course "CS6432 Distributed Consensus and Blockchains". Last but not the least, many thanks to Srini Devadas, Yan Gao, Jacob Leshno, Bruce Maggs, and Dan Boneh for their moral support, help, and/or for discussions about distributed consensus.

This book is in part supported by an NSF grant under the number CNS-1561209.

If you have any comments or feedback about the book or exercises, please do not hesitate to email me!

Elaine (Runting) Shi, Summer, 2020

Contents

Chapter 1

Distributed Consensus: from Aircraft Control to Cryptocurrencies

Back in the 1970s, it became clear that computers were going to be used in aircraft control. Since this was a mission-critical system, it was important to replicate it on multiple machines. But how do we make sure that the multiple machines share a consistent view and make consistent decision?

To understand this problem better, NASA sponsored the Software Implemented Fault Tolerance (SIFT) project [WLG+89], whose goal was to build a resilient aircraft control system that tolerated faults of its components. The famous work of Lamport et al. [LSP82] which introduced the well-known "Byzantine Generals" problem, came out of this project. This work laid the foundation of distributed consensus. Since then, distributed consensus has come a long way and has been deployed in many application settings.

In the past couple of decades, companies like Google and Facebook have adopted distributed consensus as part of their computing infrastructure, e.g., to replicate mission-critical services such as Google Wallet and Facebook Credit.

Starting in 2009, Bitcoin and various subsequent cryptocurrencies came around and became popular. The cryptocurrencies achieved a new breakthrough in distributed consensus, since they showed, for the first time, that consensus is viable in a decentralized, permissionless environment where anyone is allowed to participate.

In this course, you will learn the mathematical foundations of distributed consensus as well as how to construct consensus protocols and prove them

secure. We will motivate distributed consensus with a modern narrative, and yet we will cover the classical theoretical foundations of consensus. We will cover both classical, permissioned consensus protocols, as well as modern, permissionless consensus protocols such as Bitcoin.

A note on the terminology. Throughout this course, we will formally define several consensus abstractions, including notably, Byzantine Broadcast, and blockchains (i.e., state machine replication). On the other hand, we use the term "consensus" in an informal and generic manner to refer to any related abstraction that captures the act of reaching agreement in a distributed system.

Although multiple fault models have been studied in the past (e.g., fail-stop, crash, omission, and Byzantine faults), in this course, we focus on the hardest case, i.e., Byzantine faults. A node or system component that exhibits Byzantine fault can behave arbitrarily maliciously and need not follow the prescribed consensus protocol. Multiple Byzantine nodes can collude with each other and coordinate in their attack. Unless otherwise noted, whenever we say a node/player is "corrupt" or "malicious", we specifically mean Byzantine faults.

Unless otherwise noted, we assume that corruption is *static*, i.e., the adversary decides which set of nodes to corrupt a-priori before the protocol execution starts.

The consensus protocols we will cover sometimes make use of cryptographic primitives such as digital signatures or collision-resistant hashing.

Intended audience. Chapters and sections marked with "※" are meant as more advanced contents: they are recommended to be taught in a graduate-level course or in a special-topics course on distributed consensus. Other chapters can be taught at the senior undergraduate level assuming a general background of discrete mathematics.

Chapter 2

Preliminaries

We briefly review some simple cryptographic primitives and probability tools that will become useful in constructing and reasoning about consensus protocols. We will only cover the basic concepts and not go into detailed constructions and proofs; and the reader could use this chapter for reference purposes. For a more extensive course on cryptography, we refer the reader to several other textbooks [BS, PS, KL07].

Throughout the course, we will use $\{0,1\}^*$ to mean a string of arbitrary length; we use $\{0,1\}^\ell$ to mean a string of ℓ bits; and we use $[n]$ to mean $\{1, 2, \ldots, n\}$.

2.1 Negligible Function and Security Parameter

In cryptographic schemes such as encryption and digital signatures, we often need to choose a secret key of length λ. The longer the key length λ, the more secure the scheme is. Ideally, we want the following desired property: as we increase the key length a little, the probability that a polynomial-time adversary can break the cryptographic scheme drops very sharply. This way, we can get sufficient security with key lengths that are not too long.

We typically use a so-called *negligible function* to capture such a sharply dropping function. We say that $\mathsf{negl}(\lambda)$ is a *negligible function* iff for any fixed polynomial function $p(\lambda)$, there exists λ_0 such that for any $\lambda > \lambda_0$, $\mathsf{negl}(\lambda) < 1/p(\lambda)$. In other words, a negligible function is one that drops off faster than any inverse-polynomial function. For example, $\exp(-\log^2 \lambda)$, $\lambda^{-\log \lambda}$, and $2^{-\lambda}$ are all negligible functions. If $\mathsf{negl}(\lambda)$ is a negligible function and $\mathsf{poly}(\lambda)$ is a polynomial function in λ, then $\mathsf{negl}(\lambda) \cdot \mathsf{poly}(\lambda)$ is a negligible function in λ.

The parameter λ is often said to be a *security parameter*. Throughout this textbook, we will use λ to denote the security parameter. For example, λ may refer to the length of a secret key, or the length of a hash function's outcome depending on the context. We will assume that any adversary trying to attack our consensus protocols (possibly controlling a subset of corrupt players) is restricted to running in time that is polynomial in λ.

2.2 Collision Resistant Hash Functions

Later in our course, we will construct blockchain protocols where a distributed set of nodes jointly maintain an ever-growing, linearly-ordered log of transactions. It will be convenient if we can use a short digest to "uniquely" identify a blockchain (i.e., a linearly ordered log) or any prefix of it. For this purpose, we can take a so-called hash function and hash the blockchain to a short digest. For any function that maps a long string to a short digest, obviously a "collision" must exist, i.e., there must exist two different inputs that map to the same digest. However, if a hash function is cryptographically secure, it is computationally infeasible for any polynomial-time adversary to find such collisions. This way, the short "hash" cryptographically binds to the blockchain. Below, we define a hash function more formally which essentially conveys the above intuition.

Given a hash function $h : \{0,1\}^* \to \{0,1\}^\lambda$, a *collision* is a pair (x, x') such that $x \neq x'$ but $h(x) = h(x')$. A collision-resistant hash family $\mathcal{H}$ is a family of hash functions, such that if we draw a hash function from the family at random, it is hard for a polynomial-time adversary to find a collision.

Formally, let $\mathcal{H} = \{h_s : \{0,1\}^* \to \{0,1\}^\lambda\}_{s \in \{0,1\}^\lambda}$ be a family of hash functions indexed by $s \in \{0,1\}^\lambda$. We say that $\mathcal{H}$ is a collision-resistant hash family, iff:

- h_s can be computed in polynomial time;
- for any adversary $\mathcal{A}$ running in time polynomial in λ, there is a negligible function $\mathsf{negl}(\cdot)$, such that for every λ,

$$\Pr\left[s \xleftarrow{\$} \{0,1\}^\lambda : \mathcal{A}(s) \text{ outputs a collision for } h_s \right] \leq \mathsf{negl}(\lambda)$$

  where $\xleftarrow{\$}$ means "randomly sample".

Note that since the domain is larger than the range, a collision must exist by the pigeon-hole principle. However, the point here is that a computationally bounded adversary cannot find collisions except with negligible probability.

In practice, we often use a single hash function, such as SHA256 [sha], as a collision-resistant hash function. SHA256 is a function that hashes long messages into 256-bit digests. It is commonly believed that finding collisions for SHA256 is difficult.

2.3 Random Oracles and Proof-of-Work

In practice, hash functions such as SHA256 are believed to give random-looking outcome for each distinct query. For this reason, we often pretend that a cryptographic hash function (e.g., SHA256) is an idealized object called a *random oracle*.

A random oracle $H : \{0,1\}^* \to \{0,1\}^\lambda$ captures a function that is chosen completely at random. Whenever we query the function H with a fresh input $x \in \{0,1\}^*$ that has not been queried before, H generates a random answer from $\{0,1\}^\lambda$ and returns it. Whenever a repeat query x is made, H simply returns the same answer it gave before for the input x.

We sometimes also use random oracles whose output range is not exactly $\{0,1\}^\lambda$ for some λ. For example, we sometimes use $H : \{0,1\}^* \to [n]$ whose output range is $[n]$.

Remark 1. A single deterministic function like SHA256 in fact cannot realize a random oracle; nonetheless, we often pretend that it's a random oracle as a heuristic. If a cryptographic scheme employs random oracles and is proven secure assuming the existence of random oracles, we often replace the random oracle with a concrete hash function in practice and hope that it is still secure. This heuristic has been adopted in various contexts, and we have some empirical evidence why this could be a good approach (if used carefully).

We often use cryptographic hash functions such as SHA256 as a Proof-of-Work (PoW) random oracle. In this case, not only are we assuming that every fresh invocation of the function returns a random answer, we are also assuming that every invocation consumes some computational resources, and that there is no obvious short-cut one can exploit to speed up the computation. If one would like to learn the outcome of the function at many places, there should be no better way than brute-force evaluating the function at all of these inputs.

2.4 Digital Signatures

In our consensus protocols later, often times, when Alice says "I propose that we agree on the bit 0", Bob might later on want to convince Charles that Alice indeed said this. If Bob is malicious, however, he could potentially alter Alice's message and try to convince Charles that Alice suggested to agree on 1. Digital signature schemes can help us here. Alice can cryptographically sign the message she sends, producing a so-called "digital signature". The digital signature can ensure to a recipient (e.g., Charles), that the message indeed was signed by Alice, and that the message was not altered by Bob.

Formally, a digital signature scheme consists of three algorithms, Gen, Sign, and Vf:

- $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$: Gen takes in the security parameter λ and generates a public- and secret-key pair denoted pk and sk respectively.

- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: the signing algorithm Sign takes in a secret key sk, a message $m \in \{0,1\}^*$, and outputs a signature σ.

- $\{0,1\} \leftarrow \mathsf{Vf}(\mathsf{pk}, m, \sigma)$: the verification algorithm takes in the public key pk, a message m, a signature σ, and outputs either 1 indicating "accept" or 0 indicating "reject".

A digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ is said to be *correct* if for every pk, sk pair in the support of the Gen algorithm, for any $m \in \{0,1\}^*$, $\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$.

A digital signature scheme is secure, iff, roughly speaking, no polynomial-time adversary can forge signatures without the secret key, on any fresh messages whose signatures it has not seen. More formally, a digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ is said to be secure, iff for every polynomial-time $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every λ,

$$\Pr\left[\mathsf{pk}, \mathsf{sk} \xleftarrow{\$} \mathsf{Gen}(1^\lambda), (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk}) : \begin{array}{l} \mathcal{A} \text{ did not query } m \\ \wedge\ \mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

In the above, the notation $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$ means that $\mathcal{A}$, who obtains the public key pk, can interact with a signing oracle which uses the secret key sk to sign any message that is queried by $\mathcal{A}$, and gives the signature to $\mathcal{A}$. In other words, even if $\mathcal{A}$ can obtain signatures on messages of its choice, it still cannot forge a signature on any fresh, unqueried message.

Ideal signatures. In our course, we often assume the ideal signature model where the adversary simply cannot forge signatures. Formally, this means that we often argue our consensus protocols secure, ignoring the negligible probability with which the adversary can break the signature scheme.

2.5 Chernoff Bound ※

The Chernoff bound is often used to prove that the sum of independent random variables is highly concentrated around its mean. In this course, we will use the following version of Chernoff bound.

Theorem 1 (Chernoff bound)**.** *Let* $\mathbf{X} := \sum_{i=1}^{n} \mathbf{X}_i$ *where each* $\mathbf{X}_i = 1$ *with probability* p_i *and* $\mathbf{X}_i = 0$ *with probability* $1 - p_i$*. Further, all* $\mathbf{X}_i$*'s are independent. Let* $\mu := \sum_{i=1}^{n} p_i$*. Then, we have the following:*

- **Upper tail:** $\Pr[\mathbf{X} \geq (1+\delta)\mu] \leq \exp(-\frac{\delta \cdot \min\{\delta,1\} \cdot \mu}{3})$ *for all* $\delta > 0$*;*
- **Lower tail:** $\Pr[\mathbf{X} \leq (1-\delta)\mu] \leq \exp(-\frac{\delta^2 \mu}{2})$ *for all* $0 < \delta < 1$.

Chapter 3

Byzantine Broadcast and the Dolev-Strong Protocol

3.1 Introduction

3.1.1 The Byzantine Generals' Problem

In a seminal paper by Lamport et al. [LSP82], the problem of consensus is illustrated with the following example.

"Imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement."

Suppose that there are n generals, and one of them is called the *commanding general*. The commanding general would like to propose an order that is either ATTACK or RETREAT to all generals, such that

1. All loyal generals reach the same decision; and
2. If the commanding general is loyal, then all loyal generals will obey the commanding general's order.

Lamport et al. [LSP82] named this problem the Byzantine Generals problem; and it is also commonly referred to as *Byzantine Broadcast* (BB). Note that if the commanding general is guaranteed to be loyal, then the problem is trivial: the commanding general could send its order to all other generals, and all other generals could simply obey (assuming that they can

verify that the order indeed came from the commanding general). Of course, even the commanding general can be a traitor, and in this case, it can propose different orders to different generals; and thus the aforementioned naïve solution would result in inconsistent decisions.

So is it still possible for the loyal generals to agree upon an attack plan by communicating with each other despite the influence of corrupted generals?

3.1.2 A Modern Variant

Here is a more modern variant. Suppose that during the Covid-19 pandemic, the program committee of the Blockchain'20 conference try to decide whether next year's Blockchain conference will be held virtually online or in person. The chair of the program committee is called the *program chair*. The program chair would like to convey a suggestion to the entire program committee, that is either "virtual" or "physical". Since the program committee are all in quarantine, they decide to reach agreement over the Internet by sending emails back and forth to each other. We may assume that an email sent sometime today will be received and read by the recipient at the beginning of tomorrow.

Now, there is a problem: a subset of the program committee are unhappy with the Blockchain'20 conference since their papers had got rejected earlier from the conference. These unhappy committee members may be secretly plotting to disrupt agreement and prevent the next Blockchain'20 conference from happening. Even the program chair herself may be secretly unhappy with the Blockchain conference. While the happy committee members will faithfully follow the protocol rules, the unhappy members can misbehave arbitrarily, and send arbitrary messages.

Can we devise a protocol such that

- all the happy committee members can reach a common agreement; and moreover,
- if the program chair is happy, then every happy committee member should output the chair's original suggestion?

3.1.3 Analogy to Reliable Distributed Systems

The above imaginary situations serve as an analogy with a computer system in which one or more components can fail. More precisely, the problem of distributed consensus in a distributed system is that while some of the nodes in the system might act in an arbitrary manner, the correctly functioning nodes

still need to agree on a common value among themselves. Real-life consensus protocols (e.g., Bitcoin) typically need to repeatedly reach consensus over time, such that nodes jointly maintain an ever-growing, linearly-ordered log of transactions, sometimes called a *blockchain*. In our course, we shall start with Byzantine Broadcast which allows nodes to reach agreement once. This is important for understanding the foundations of distributed consensus. Later in our course, we will indeed cover how to define and construct "blockchains", i.e., a repeated consensus abstraction.

3.2 Problem Definition

Let us now formalize the problem more precisely. We will henceforth refer to the generals (or committee members) as *nodes*, and refer to the commanding general (or the program chair) as the *designated sender*, or simply *sender* for short.

Consider a distributed network of n nodes numbered $1, 2, \ldots, n$ respectively. We often use the notation $[n] := \{1, 2, \ldots, n\}$ to denote the set of all nodes. Without loss of generality, we may assume that *the sender is named node* 1.

We refer to the nodes that follow the prescribed protocol throughout as *honest* nodes. A subset of the nodes, however, can be *corrupt*. The corrupt nodes need not follow the prescribed protocol, and they may send/transmit arbitrary messages at arbitrary times, omit sending messages, stop or take an incorrect step. All the corrupt nodes can form a coalition and share information with each other, and perform a coordinated attack. For this reason, it is often instructive to imagine that all the corrupt nodes are controlled by a single *adversary*. We stress that a-priori, we do not know which nodes are corrupt — had we known, the problem would also be trivialized (see Exercise 4). In other words, our Byzantine Broadcast protocol must work no matter which subset of nodes are corrupt, as long as the total number of corruptions is upper bounded by $f < n$.

We assume that every pair of nodes can communicate with each other, and moreover every node has a public and secret key pair corresponding to a digital signature scheme, and all nodes' public keys are common knowledge. In the Dolev-Strong protocol below, every message will be signed by its creator. We use the notation $\langle m \rangle_i$ a pair (m, σ) where σ is a valid signature on the message m that can be verified under node i's public key.

Remark 2. For simplicity, unless otherwise noted, we assume that the set of corrupt nodes is chosen at the start of the protocol (but the protocol is

unaware of which nodes are corrupt). This model is often referred to as the *static* corruption model. In the distributed systems and cryptography literature, an alternative model, called *adaptive* corruption, is also extensively studied. In the adaptive model, an adversary can decide which nodes to corrupt in the middle of the protocol's execution, after having observed messages sent by honest nodes. The Dolev-Strong protocol is in fact also secure against adaptive corruptions although we do not explicitly discuss it in this chapter.

3.2.1 Synchronous Network

We assume that the protocol takes place in a *synchronous* network, i.e., when honest nodes send messages, the honest recipients are guaranteed to receive them within a bounded amount of time, say, one minute. No matter how long the message delay is, we can define it as one *round*. Therefore, we shall assume that the protocol execution proceeds in rounds. At the beginning of each round, all nodes receive incoming messages from the network. They then perform some local computation, and send out new messages. The following is guaranteed by the network:

> Synchrony assumption: *If an honest node sends a message in round r to an honest recipient, then the recipient will receive the message at the beginning of round $r + 1$.*

3.2.2 Definition of Byzantine Broadcast

At the beginning of a protocol, the designated sender receives an input bit $b \in \{0, 1\}$. The nodes then run some protocol; at the end of the protocol, every honest node outputs a bit. A Byzantine Broadcast (BB) protocol is supposed to satisfy the following two requirements (no matter how corrupt nodes behave):

- **Consistency** : If two honest nodes output b and b' respectively, then $b = b'$.
- **Validity** : If the sender is honest and receives the input bit b, then all honest nodes should output b.

Note that consistency alone would be trivial to achieve without the validity requirement: the protocol can simply require that every node output a canonical bit 0. Therefore, the problem definition is only non-trivial/meaningful if we required both properties simultaneously.

Remark 3. In this chapter, we shall assume that the signature scheme is ideal, i.e., no signature forgery can happen. Moreover, our Dolev-Strong protocol described below will be deterministic. Therefore, we may assume that we want the above requirements to be satisfied deterministically. In later chapters, we shall consider randomized protocols, in which case it may be sufficient for the above properties to hold with all but negligible probability.

3.3 A Naïve (Flawed) Protocol

We first look at a natural but naïve approach. As mentioned earlier, it would not work if everyone simply followed the bit heard from the designated sender: if the designated sender is corrupt and sends different bits to different nodes, then consistency can be violated.

What is the next most natural idea? Perhaps it would be a voting-based approach. For convenience, we shall assume that all nodes sign all messages before sending them, and only messages with valid signatures from purported senders are viewed as valid. All invalid messages are discarded immediately without being processed.

Now, imagine that the sender signs and sends its input bit to everyone in the first round. In the second round, everyone votes on the bit they heard from the sender. If no bit is heard or both bits are heard, they vote on a canonical bit 0. Now, if a node hears majority nodes vote for b, then it outputs b. We describe this simple voting-based protocol more formally below where we use the notation $\langle m \rangle_i$ to denote the message m along with a valid signature from node $i \in [n]$:

A naïve majority voting protocol

- Sender (i.e. node 1) receives the bit b as input.
- **Round** 1: Node 1 sends $\langle b \rangle_1$ to every node (including itself).
- **Round** 2: Every node $i \in [n]$ does the following: if a single bit $\langle b' \rangle_1$ is received, send the vote $\langle b' \rangle_i$. Else send the vote $\langle 0 \rangle_i$.
- **Round** 3: If no bit or both bits received more than $n/2$ votes from distinct nodes, then output 0. Else output the bit that received more than $n/2$ votes from distinct nodes.

Does this protocol work? Keep in mind that corrupt nodes can behave arbitrarily, including sending different votes to different nodes. It turns out

that this naïve protocol is not secure under even a single corrupt node. Below we describe an attack in which the designated sender is the only corrupt node, and everyone else is honest, and we show that consistency can be violated.

The attack. Assume that $n = 2k+1$ is odd and only the sender, i.e., node 1, is corrupt. Let us divide the $2k$ honest nodes into two disjoint sets denoted S_0 and S_1, each of size k. In the first round, the sender sends $\langle 0 \rangle_1$ to the set S_0, and it sends $\langle 1 \rangle_1$ to the set S_1. In the second round, nodes in S_0 votes for 0, and nodes in S_1 votes for 1. Now, the corrupt sender skillfully votes for 0 to the set S_0 and it votes for 1 instead to the set S_1. Observe that nodes in S_0 receive majority votes for 0 whereas nodes in S_1 receive majority votes for 1, and thus they will output inconsistently.

So how can one design a *secure* Byzantine Broadcast protocol?

Exercise 1. Here is another simple idea. Round 1 and round 2 are the same as the naïve majority voting protocol. In other words, in round 1, the sender signs its input bit and sends it to everyone. In round 2, everyone votes for the bit it has heard from the sender. If no bit or both bits were heard, vote for the canonical bit 0. In round 3, if some bit b has gained the votes of very node, then output b; otherwise, output 0.

Does this protocol achieve Byzantine Broadcast? If so, please explain why. If not, please describe an explicit attack that either breaks consistency or validity. In your attack, use as few corrupt nodes as possible.

3.4 The Dolev-Strong Protocol

The celebrated Dolev-Strong protocol [DS83] solves the Byzantine Broadcast problem.

Recall that the nodes are numbered $1, 2, \ldots, n$, and we will assume that the designated sender is numbered 1. We denote the sender's input as b. We also assume that each node i maintains a set extr_i, also referred to as the node's "extracted set," of the distinct valid bits that have been chosen so far. For brevity, we will use the notation $\langle b \rangle_S$ to denote the message b attached with a valid signature on b verifiable under the public keys of nodes in $S \subseteq [n]$. In our protocol below, f denotes an upper bound on the number of corrupt nodes.

The Dolev-Strong protocol

Initially, every node i's extracted set $\mathsf{extr}_i = \emptyset$.

- **Round** 0: Sender sends $\langle b \rangle_1$ to every node.
- **For each round** $r = 1$ **to** $f + 1$:

  For every message $\langle \tilde{b} \rangle_{1,j_1,j_2,\ldots,j_{r-1}}$ node i receives with r signatures from distinct nodes including the sender:

  - If $\tilde{b} \notin \mathsf{extr}_i$: add $\tilde{b}$ to extr_i and send $\langle \tilde{b} \rangle_{1,j_1,\ldots,j_{r-1},i}$ to everyone — note that here node i added its own signature to the set of r signatures it received.
- **At the end of round** $f + 1$**:** If $|\mathsf{extr}_i| = 1$: node i outputs the bit in extr_i; else node i outputs 0.

3.4.1 Intuition

So why is $f + 1$ rounds necessary for the Dolev-Strong protocol?

Suppose that all nodes have to output at the end of round f instead of $f + 1$. We construct an attack as follows. In round 0, the corrupt sender sends $\langle 1 \rangle_1$ to all honest nodes. Thus, all honest nodes will add 1 to their extracted sets in round 1. Now, recall that there can be f corrupt nodes, including the sender. At the beginning of round f, the corrupt nodes make a single honest node v receive the bit 0 along with all f signatures, but does not deliver this message to every other honest node. In this case, v will end up with 2 bits in its extracted set whereas all other honest nodes have only 1 bit in their extracted sets in round f; and thus v will be inconsistent with all other honest nodes.

The above attack is not possible if the algorithm is run for one more round, i.e., if the nodes output at the end of round $f + 1$ instead. Intuitively, this is because a bit b tagged with $f + 1$ signatures must have been signed by at least 1 honest node, say, node i. However, when the honest node i signed the bit b earlier, say, in round $r < f + 1$, it must have propagated a batch of $r + 1$ signatures on b (including its own) to all other honest nodes, and therefore all other honest nodes will have received b along with $r + 1$ signatures by the beginning of round $r + 1 \leq f + 1$, and will have added the bit b to their extracted sets (if not earlier).

3.4.2 Analysis

Equipped with the above intuition, we now formally prove that the Dolev-Strong protocol satisfies both consistency and validity.

Lemma 1. *Let $r \leq f$. If by the end of round r, some honest node i has $\tilde{b}$ in extr_i, then by the end of round $r+1$ every honest node has $\tilde{b}$ in its extracted set.*

Proof. We know that node i has the bit $\tilde{b}$ in extr_i by the end of round r. This bit $\tilde{b}$ must have been added to extr_i in some earlier round. Suppose $t \leq r \leq f$ is the round in which the bit $\tilde{b}$ first got added to extr_i. According to the protocol, it must be that in round t, node i received $\langle \tilde{b} \rangle_{1,j_1,\ldots,j_{t-1}}$ with t distinct signatures including one from the sender. Moreover, none of these signatures must come from node i itself because if i had signed $\tilde{b}$ earlier, it would have added $\tilde{b}$ to extr_i earlier. Therefore, node i then must have sent $\langle \tilde{b} \rangle_{1,j_1,\ldots,j_{t-1},i}$ to every other node in round t. Now, by our synchrony assumption, all other honest nodes will receive this message with $t+1$ distinct signatures at the beginning of round $t+1 \leq f+1$ and will, therefore, add $\tilde{b}$ to their extracted sets in round $t+1$ (if it has not already been added). □

The above lemma says that if some honest node has included some bit $\tilde{b}$ in round $r < f+1$, then all honest nodes will have included the same bit in the *immediate next* round. To prove consistency, we need to show that if some honest node has included a bit $\tilde{b}$ in the final round $f+1$, then all honest nodes must have included it in the *same* round. The last round, i.e., round $f+1$, is where the magic happens such that *common knowledge* is reached.

Lemma 2. *If some honest node i has $\tilde{b}$ in extr_i by the end of round $f+1$, then every honest node has $\tilde{b}$ in its extracted set by the end of round $f+1$.*

Proof. We consider the following two cases:

1. **Case 1 (Node i first added $\tilde{b}$ to its extracted set in round $r < f+1$):** By Lemma 1, once $\tilde{b}$ is in extr_i at the end of round r, then every honest node will have $\tilde{b}$ in its extracted set by round $r+1 \leq f+1$.

2. **Case 2 (Node i first added $\tilde{b}$ to its extracted set in round $f+1$):** For this case to happen, node i must have received $f+1$

distinct other nodes' signatures on $\tilde{b}$ at the beginning of round $f+1$. Since at most f nodes are corrupt, at least one honest node must have signed $\tilde{b}$ in an earlier round $r < f+1$. Thus, by Lemma 1, every honest node, including node i, would have added $\tilde{b}$ to its extracted set by the end of round $r+1 \leq f+1$.

□

Using the two lemmas above, we now state the theorems that establish the desired properties for the Dolev-Strong protocol.

Exercise 2. Prove that the Dolev-Strong protocol satisfies validity, that is, if the sender is honest, then every honest node would output the sender's input bit.

Theorem 2 (The Dolev-Strong protocol [DS83])**.** *The Dolev-Strong protocol achieves Byzantine Broadcast in the presence of up to $f \leq n$ corrupt nodes.*

Proof. By Lemma 2, all honest nodes must have the same extracted set at the end of round $f+1$, and thus consistency is achieved. Proving validity is left as a homework exercise (see Exercise 2). □

3.4.3 Further Discussions

Dolev and Strong [DS83] also proved that any *deterministic* protocol solving Byzantine Broadcast (allowing ideal signatures) must incur at least $f+1$ rounds. In practice, $f+1$ rounds may be too expensive for large-scale applications. Fortunately, this $f+1$ round complexity lower bound can be circumvented by using randomness in the protocol design. We will explore randomized protocols later in the course.

3.5 The Muddy Children Puzzle

There is a cute puzzle called the "muddy children puzzle" that bears a remote resemblance to the Dolev-Strong protocol.

There are n children playing in the playground, and $k \leq n$ of them acquire mud on their forehead. After playing, the teacher gathers the children, and declares, "one or more of you have mud on your forehead". Every one can see if others have mud on their forehead, but they cannot tell for themselves.

The teacher says, "at this moment, if you know you have mud on your forehead, please step forward". The teacher waits for a minute and no one steps forward. The teacher says again, "second call: at this momement, if you know that you have mud on your forehead, please step forward.". This goes on for multiple rounds until some children step forward. In each round, the teacher calls for those who know that they have mud on their forehead to step forward.

Question: in which round will some children step forward? Note that the children do not communicate with each other. They know that at least one of them has mud on their forehead, and they know the current round number.

The puzzle can be solved by induction. The case $k = 1$ is easy. If Alice is the only kid with mud, then she would step forward in the first round: she sees that no one else has mud, so it must be herself. Now, consider the case $k = 2$. Say, Alice and Bob are the two kids with mud. In this case, no one steps forward in the first round, because Alice sees that Bob has mud, and she cannot be sure if she has mud too; and the same reasoning applies to Bob. However, knowing that no one stepped forward in the first round, Alice and Bob now know that at least two kids have mud (otherwise the only kid with mud would have stepped forward in the first round). Now, in the second round, Alice sees only one other kid with mud, so she knows that she must be the other. The same reasoning applies to Bob. Therefore, in the second round, both Alice and Bob step forward.

This argument can be carried out inductively, and one can show that if k kids have mud, then all k muddy kids will step forward in round exactly k.

In this puzzle, common knowledge is reached in round k. Therefore, it is somewhat reminicient of the Dolev-Strong protocol in which common knowledge is reached in round $f + 1$.

3.6 Additional Exercises

Exercise 3. In our lecture, we defined a one-bit version of Byzantine Broadcast, where the sender wants to distribute a single bit. We may considered a multi-valued variant (called Multi-Valued Byzantine Broadcast) in which the sender receives an ℓ-bit value $m \in \{0,1\}^\ell$, and it wants to propagate this value to every one else. Consistency requires that all honest nodes output the same value; and validity requires that if the sender is honest, all honest nodes output the sender's input value m.

Please design a protocol for achieving Multi-Valued Byzantine Broadcast,

and prove it secure. Your protocol should be able to tolerate $f < n$ number of corruptions.

Exercise 4. Suppose that, as the protocol designer, we already know exactly which subset of nodes are corrupt. Describe a 1-round protocol that achieves Byzantine Broadcast. Here the set of corrupt nodes is provided as input to the protocol. This exercise shows why the problem of Byzantine Broadcast is trivialized if the set of corrupt nodes is known a-priori.

Exercise 5. In our description of the Dolev-Strong protocol earlier, the protocol needs to be parametrized with the parameter f, i.e., the maximum number of corruptions. What if we do not know an upper bound on f a-priori and any number of nodes can potentially be corrupt? Can we set $f = n - 2$ in this case?

Exercise 6. In this problem, you are asked to help analyze two buggy implementations of the Dolev-Strong protocol.

a) Lanie is asked to implement the Dolev-Strong protocol for her all-time favorite course on distributed systems. Lanie is absent-minded when implementing the Dolev-Strong protocol. She forgot to check that the batch of r signatures expected in round r must contain the signature from the designated sender. Describe an explicit attack that can break Lanie's implementation. In your attack, use as few corrupt nodes as possible. Does your attack violate the consistency or validity property?

b) Another student, Elanna, also made a mistake in her implementation, but a different mistake from Lanie. Elanna has a bug in implementing the digital signature scheme. As a result, an attacker can efficiently forge signatures of honest players even without their private signing keys. Please describe an attack that breaks Elanna's Byzantine Broadcast implementation.

Chapter 4

Byzantine Broadcast without Digital Signatures (Lower Bound)

In the previous chapter, we learned the Dolev-Strong protocol [DS83], which solves Byzantine Broadcast under any $f < n$ number of corruptions. The Dolev-Strong protocol relies on digital signatures and the existence of a Public-Key Infrastructure. Typically, we call a Public-Key Infrastructure (PKI) a setup assumption, because there needs to be a trusted channel for distributing the nodes' public keys. A natural question arises:

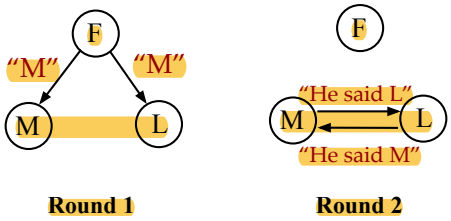
Can we achieve Byzantine Broadcast without digital signatures and without a Public-Key Infrastructure (PKI)?

Even if we are happy to assume cryptographic hardness assumptions and the existence of digital signatures, the above question is intriguing in the following senses:

- First, setting up and managing a PKI is notoriously challenging in practice. For example, the infrastructure for distributing and managing certificates for the world-wide-web has suffered from numerous attacks in the past.
- Second, even in scenarios where a PKI is feasible to set up, sometimes we would like to deploy consensus protocols that achieve high-performance. For example, in data center settings, we might hope to achieve micro-second confirmation delay, in which case we cannot afford to spend even millisecond on computing digital signatures.

Intuitively, digital signatures help us a lot in the design of consensus protocols, because digital signatures enable non-repudiation. If a node i sends to node j a message m along with its signature on m, node j can later on convince others that node i indeed has signed m. Indeed, the Dolev-Strong protocol critically relies on this ability to reach consensus.

To understand why non-repudiation might be useful, we can also look at the following informal example. Suppose that Cranberry Melon University (CMU) would like to elect a new president. In Cranberry Melon University, no one wants to be a university president because it is too much work. The current president is Far, and there are two running candidates, Mor and Les. Far wants to get one of Mor or Les elected. He makes a proposal to Mor and Les, and suppose that the proposal says "Mor". When Mor hears the proposal, not wanting to get herself elected, she relays to Les, "I heard that Far suggested Les". When Les hears the proposal, he relays to Mor honestly, "I heard that Far suggested Mor".



In the above, both Far and Les were honest, but Mor deviated from the protocol to avoid getting herself elected. If the protocol did not use any digital signature, then Mor's message to Les is plausible and from Les's perspective, there are two possible interpretations:

- *Interpretation 1:* Far is corrupt and both Mor and Les are honest. Far sent differing proposals to Mor and Les.

- *Interpretation 2:* Far is honest and proposed "Mor" to both recipients, but Mor did not relay Far's message honestly.

Les would not be able to tell these two scenarios apart without digital signatures. Had digital signatures been used, however, Mor would not have been able to relay a forged message from Far since she wouldn't be able to forge a signature on Far's behalf. Therefore, the above scenario could have been prevented with digital signatures.

4.1 Impossibility of Consensus with 1/3 Corruptions without Digital Signatures

Although the above is *not* a formal proof that setup assumptions (e.g., PKI) and digital signatures help in consensus, it does convey some degree of intuition. In the remainder of this chapter, we will formally prove why setup assumptions such as PKI and digital signatures do help with consensus.

The pairwise authenticated channels model. We consider how to achieve BB in a network where every pair of nodes can communicate through a pairwise channel. Moreover, we assume that when an honest node receives a message, it knows which node sent the message, i.e., the authenticity of the message-sender can be ascertained.

Like in the previous chapter, we shall consider a synchronous network, where honest nodes' messages can be delivered in exactly one round.

Impossibility of BB under 1/3 corruption without setup assumptions. The following theorem was first proven by Pease, Shostak, and Lamport [PSL80], and later the proof was simplified by Fischer, Lynch, and Merritt [FLM85].

Theorem 3 (Impossibility of BB under 1/3 corruption without setup assumptions [PSL80,FLM85])**.** *In a plain pairwise authenticated channel model without any setup assumptions such as a PKI, Byzantine Broadcast (BB) is impossible if at least $n/3$ nodes are corrupt.*

It is interesting to contrast this impossibility result with the Dolev-Strong protocol we learned earlier. With Dolev-Strong, we can achieve BB under an arbitrary number of corruptions, but the protocol required a PKI and digital signing. It turns out that without these assumptions, a result like Dolev-Strong would have been impossible.

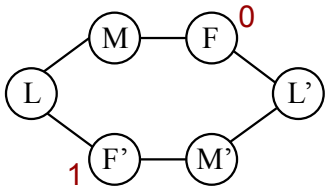
Remark 4 (A remark on the terminology)**.** Due to historical reasons, in the distributed systems literature, the problem of achieving BB assuming the existence of a PKI and digital signatures is often referred to as "authenticated broadcast". Achieving BB in the plain, pairwise authenticated channels model without setup is often referred to as "unauthenticated broadcast". Unfortunately, the use of the term "authenticated" differs among the distributed systems and cryptography communities. In this textbook, we adopt the standard terminology from the cryptography community in describing our modeling assumptions.

4.2 Proving the Lower Bound

We present a simplified proof of the lower bound suggested by Fischer et al. [FLM85]. Our intuitive explanation earlier (recall the protocol between Far, Mor, and Les) was a "triangle", but that was not a formal proof. Interestingly, we shall see that the formal proof relies on a "hexagon" argument rather than a triangle.

Proof of Theorem 3. Suppose for the sake of contradiction that there exists a protocol Π that achieves BB under the pairwise authenticated channels model, for $n = 3$ and $f = 1$.

Although the protocol Π is meant to be executed among three nodes, we can nonetheless imagine a hypothetical experiment where Π is executed among six nodes, named F, L, M, F', L', M' respectively[1]. The nodes F and F' act as honest senders, each receiving the input bit 0 and 1 respectively. The other nodes L, M, L', M' also run honestly as non-senders.



The first thing to observe is that this hypothetical experiment is well-defined. As far as each node is concerned, it has two neighbors and a pairwise link with each of the two neighbors — just like in a 3-node execution. Now, there are several ways to interpret this hypothetical experiment as we describe shortly below. In all interpretations, there are only 3 nodes, and one of them is corrupt and "simulating all the remaining nodes in its head" (see Remark 5 about "simulation"). By our assumption, the protocol Π satisfies consistency and validity when 1 out of 3 nodes is corrupt, we can then apply the consistency and validity properties of Π to the different interpretations. Since all interpretations are in fact *different ways to "explain" the same hypothetical experiment*, the conclusions we reach should be internally consistent, had such a protocol Π existed (i.e., satisfying both consistency and validity when 1 out of 3 is corrupt). However, by applying consistency and validity rules

[1] Yes, indeed Π is intended as a 3-party protocol, but the 6-node execution is well-defined, and thus we can reason about it as a thought experiment. This might be unintuitive at first, but to me it is the elegant idea in the proof.

to each interpretation, we will in fact reach a contradiction, thus ruling out the possibility that such a protocol Π could exist.

- *Interpretation 1.* There are in fact three nodes, F, L, and M. F is corrupt and it is simulating all the nodes F, L', M', and F' in its head (see Remark 5 about "simulation"); whereas L and M are honest. Using this interpretation, we conclude that L and M must output the same bit in the hypothetical experiment.
- *Interpretation 2.* There are in fact three nodes, F, L, and M. L is corrupt and it is simulating all the nodes L, L', F', and M' in its head; whereas F and M are honest. Using this interpretation, by validity, we conclude that M should output F's input bit, that is 0.
- *Interpretation 3.* There are in fact three nodes, F', L, and M. M is corrupt, and it is simulating all the nodes M, M', F, and L' in its head; whereas F' and L are honest. Using this interpretation, by validity, we conclude that L should output the input bit of F', that is, 1.

Thus we have reached a contradiction about what L and M would output in the hypothetical experiment.

Remark 5 (About "simulation")**.** What does it mean for a node F to "simulate the nodes F, L', M', and F' in its head"? This means that node F acts as an "emulator" that emulates the execution of the (interactive) programs of the simulated nodes — note also that because of this, F is not running the honest protocol. In fact, the "simulation" technique is often used in proofs in distributed systems theory, cryptography, as well as complexity theory. You will see this simulation technique later in our course too.

Why is it that the above lower bound proof does NOT hold assuming the existence of a PKI and digital signatures?

Answer: for example, in the first interpretation, F would not be able to simulate L' in its head because F does not know the secret signing key of L'.

4.3 Additional Exercises

Exercise 7. The above proof rules out the existence of a BB protocol, absent a PKI and digital signatures, for the case when $n = 3$ and $f \geq 1$. Please argue that without any setup assumptions such as a PKI, no protocol can achieve BB for $n = 6$ and $f \geq 2$. Similarly, please show the same lower bound for the case $n = 4$ and $f \geq 2$.

Exercise 8. In the above proof, we have ruled out the existence of *deterministic* BB protocol (allowing ideal signatures) that can defend against 1/3 corruptions without any setup assumptions such as a PKI.

However, often times, we may want to consider *randomized* protocols in which nodes can flip random coins and use these random coins as inputs during the protocol. A randomized protocol Π is said to achieve BB with probability p iff regardless of the corrupt coalition's strategy, it must be that with probability p over the choice of the randomized execution, consistency and validity are both respected.

Please prove that in the plain pairwise authenticated channels model without setup, there does not exist a protocol that achieves BB with probability greater than 2/3.

Exercise 9. We have shown that without a PKI and digital signatures, it is impossible to achieve BB under 1/3 or more corruptions. Typically the PKI is called a *setup assumption* because we essentially need a trusted distribution channel to distribute the nodes' public key a-priori. Digital signatures can be viewed as a *cryptographic hardness assumption*.

Upon careful examination, our lower bound proof actually rules out the existence of BB under 1/3 or more corruptions absent any *setup assumption* such as PKI — and the impossibility still holds even if we allow the use of cryptographic primitives such as digital signatures and encryption schemes, and even if we assume that the adversary is indeed polynomially bounded and cannot break cryptography!

Please reflect upon this and intuitively explain why.

Chapter 5

Byzantine Broadcast without Digital Signatures (Upper Bound)

Previously, we have learned that the Dolev-Strong protocol can achieve Byzantine Broadcast (BB) under any number $f \leq n - 2$ of corruptions; however, the Dolev-Strong must rely on a Public-Key Infrastructure (PKI) and digital signatures (see Chapter 3). We have also proven than when setup assumptions such as PKI are not allowed, BB is impossible if there are at least $f \geq n/3$ corrupt nodes (see Chapter 4).

In this chapter, we will show that the 1/3 lower bound in Chapter 4 is tight: suppose that strictly fewer than $n/3$ nodes are corrupt, there indeed exists a BB protocol that does not rely on any setup assumption or cryptography.

5.1 Protocol

We assume that the n nodes are numbered $1, 2, \ldots, n$, and let $[n] := \{1, 2, \ldots, n\}$.

The protocol works by voting. Every node locally maintains a *sticky bit* whose value is chosen from the set $\{0, 1, \perp\}$. If a node's sticky bit is either 0 or 1, it reflects its current belief of the bit that's agreed upon. If the sticky bit is $\perp$, it roughly means that node currently does not have any belief. At the end of the protocol, everyone outputs their local sticky bit.

Initially, the designated sender's sticky bit is its input bit, and everyone else's sticky bit is set to $\perp$. In the first iteration, the designated sender is

the leader, and in every other iteration $r \neq 1$, a random node $L_r := H(r)$ is chosen as the leader using a public hash function $H : \{0,1\}^* \to [n]$ — we will think of this hash function as a random oracle[1]

In every iteration r, the following happens. The leader L_r proposes a bit b to everyone: if the leader L_r's sticky bit is not $\perp$, it proposes the sticky bit; else it proposes a random bit from $\{0,1\}$. Now, everyone votes on a bit: to vote on a bit, simply echo the corresponding bit to everyone. If the node's sticky bit is not $\perp$, it votes on the sticky bit; else, it votes on the bit b proposed by the leader L_r or chooses an arbitrary bit if L_r proposed no bit or both bits. If a node hears $2n/3$ nodes vote on the same bit b' in the current iteration, it updates its sticky bit to b', else it updates its sticky bit to $\perp$. The protocol repeats for sufficiently many iterations and at the end, everyone outputs their sticky bit.

The protocol is shown a little more formally below. Without loss of generality, we assume that node 1 is the designated sender. We let $L_1 := 1$ and for $r \neq 1$, let $L_r := H(r)$ where $H : \{0,1\}^* \to [n]$ is a public hash function.

Initially, the designated sender's sticky bit is its input bit, and everyone else's sticky bit is set to $\perp$.

For every iteration $r = 1, 2, \ldots, k$:

- Round 0: leader L_r sends a proposed bit b to everyone where b is chosen as follows:
  - If L_r's sticky bit is not $\perp$, choose b to be the sticky bit;
  - Else L_r chooses $b \in \{0,1\}$ at random.
- Round 1: everyone votes on a bit by sending the bit to everyone; the bit voted on is chosen as follows:
  - If the node has a non-$\perp$ sticky bit, choose the sticky bit;
  - Else choose the bit proposed by L_r — if L_r proposed no bit or both bits, pick a bit arbitrarily.
- Round 2: everyone tallies the votes it has received: if at least $2n/3$

[1]Essentially, whenever a node wants to know what the outcome of $H(m)$ is, H chooses a random answer from $[n]$ if the message m has not been queried before, otherwise it returns the same answer previously returned for m. See also Chapter 2 on the notion of a "random oracle".

nodes voted on the same bit b', update its sticky bit to be b'; else update its sticky bit to be $\perp$.

Output: Finally, everyone outputs its own sticky bit's value.

5.2 Analysis

Henceforth in our analysis, we shall assume that strictly fewer than $n/3$ nodes are corrupt.

The following lemma (Lemma 3) says that at the end of any iteration, it cannot be that some honest node's sticky bit is 0, and another honest node's sticky bit is 1 — however, it is possible that some honest node's sticky bit is 0 (or 1) and another honest node's sticky bit is $\perp$. It is easiest to understand why with an example. Suppose there are 4 nodes and $1 < 4/3$ node is corrupt. For an honest node to have the sticky bit 0 at the end of an iteration, it must be that it has received at least 3 votes on 0. This means that at least two honest nodes voted on 0. Similarly, if another honest node has the sticky bit 1 at the end of the same iteration, it must be that at least two honest nodes voted on 1. However, since there are only 3 honest nodes, this would imply that there is an honest node who voted on both 0 and 1 in the same iteration, but this is impossible because in our protocol, an honest node votes on only one bit in any iteration. Below we formalize this argument and prove it for general parameters.

Lemma 3. *In any iteration, it cannot be that some honest node i sees at least $2n/3$ nodes vote on a bit $b \in \{0, 1\}$, and some honest node j sees at least $2n/3$ nodes vote on the other bit $1 - b$ (note that j can be the same as i or different).*

As a direct corollary, we have that at the end of every iteration, it cannot be that some honest node has a sticky bit $b \in \{0, 1\}$, and another honest node has the sticky bit $1 - b$. Note that it is possible for an honest node to have a sticky bit $b \in \{0, 1\}$ and another honest node to have $\perp$.

Proof. Suppose that in some iteration r, an honest node i heard that at least $2n/3$ nodes, denoted the set S_i, vote on b; and an honest node j heard that at least $2n/3$ nodes, denoted the set S_j, vote on $1 - b$. Now, $|S_i \cap S_j| \geq n/3$, and thus there must be an honest node in $S_i \cap S_j$; but this honest node voted on both b and $1 - b$ which leads to a contradiction. □

We say that some iteration r is *lucky*, iff 1) its leader L_r is honest, and 2) if L_r proposes the bit b in iteration r, then no honest node's sticky bit at

the beginning of iteration r is $1 - b$. The following lemma (Lemma 4) shows that if there is a lucky iteration, then all honest nodes will output a bit $b \in \{0, 1\}$ at the end of the protocol (i.e., their sticky bit cannot be $\perp$ at the end), and moreover they'll all output the same bit. To complete our proofs, it would then suffice to show that a lucky epoch must exist with extremely high probability — we prove this in the subsequent Lemmas 5 and 6.

Lemma 4. *Suppose that iteration $r \leq k$ is lucky and L_r proposes $b \in \{0, 1\}$ in iteration r. Then, every honest node's sticky bit at the end of the protocol must be $b \in \{0, 1\}$.*

Proof. In iteration r, every honest node must vote on $b \in \{0, 1\}$. Thus, every honest node will see at least $2n/3$ nodes vote on b in Round 2 of the iteration. By Lemma 3, no honest node can see at least $2n/3$ nodes vote for $1 - b$ in iteration r. Thus, every honest node will set its sticky bit to b by the end of iteration r, and thus in iteration $r + 1$, all honest nodes will vote on b too, and so on. □

Lemma 5. *Suppose that the hash function H is chosen independently of the choice of corrupt nodes. Then, for every $1 \leq r \leq k$, even when conditioned on whether the iterations prior to r are lucky or not, the r-th iteration is lucky with probability at least $1/3$.*

Proof. The following statements hold even when conditioned on whether previous iterations are lucky or not. In every iteration r, if L_r is honest and its sticky bit is not $\perp$ at the beginning of iteration r, the iteration is guaranteed to be lucky due to Lemma 3. Conditioned on L_r being honest and its sticky bit being $\perp$ at the beginning of iteration r, due to Lemma 3, the iteration is lucky with probability at least $1/2$ (depending on L_r's coin that chooses the proposed bit). Therefore, no matter whether previous iterations were lucky or not, conditioned on L_r being honest, iteration r is lucky with probability at least $1/2$.

Notice also that every iteration r has an honest leader with probability $2/3$ since its leader is randomly chosen using the hash function H. We thus conclude that even when conditioned up whether previous iterations are lucky or not, iteration r is lucky with probability at least $\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$.

□

Lemma 6. *There is a lucky iteration with probability $1 - (\frac{2}{3})^k$.*

Proof. Imagine that for each iteration $r = 1, 2, \ldots, k$ sequentially, we flip a coin which decides with probability at least $1/3$ that iteration r is lucky. Clearly, the probability that all iterations are unlucky is at most $(\frac{2}{3})^k$. □

Remark 6. The probability that there isn't a lucky iteration drops *exponentially fast* w.r.t. k. For example, if $k = 40$, the probability that there isn't a lucky iteration is only 9×10^{-8}. Because of this exponentially sharp tail, slightly informally, we can set k to be a reasonable parameter, such that the failure probability $(\frac{2}{3})^k$ is as small as the probability that encryption schemes get cracked.

Combining Lemmas 4 and 5, we may conclude the following theorem:

Theorem 4 (Consistency)**.** *With probability* $1 - (\frac{2}{3})^k$*, all nodes output the same decision.*

Theorem 5 (Validity)**.** *If the designated sender (i.e., node* 1*) is honest, then all honest nodes output node* 1*'s input bit.*

Proof. Follows directly from Lemma 4 and the definition of the protocol. □

5.3 Additional Exercises

Exercise 10. Suppose that we want to make sure that consistency is achieved with probability at least $1 - \delta$ for some small $\delta \in (0, 1)$. How should we set the number of iterations k in the above protocol? Your answer should express k as a function of δ.

Remark 7 (On the round complexity of the protocol)**.** Dolev and Strong proved that no *deterministic* protocol (even allowing ideal signatures) can achieve BB in fewer than $f + 1$ rounds [DS83]. Notice that the protocol in this section is randomized, and its round complexity is related to the failure probability δ, and does not depend on n or f. In subsequent chapters, we will see more examples how randomization can circumvent the $f + 1$ round complexity lower bound that pertains to deterministic protocols.

Chapter 6

Blockchain and State Machine Replication

So far in our lectures, we have considered *single-shot* consensus. In practice, however, more often than not, it is not enough to reach consensus just once. Practical applications of consensus often require reaching consensus repeatedly over time. For example, in modern cryptocurrency systems such as Bitcoin and Ethereum, the underlying core abstraction is for a distributed set of nodes to maintain an *ever-growing public ledger* which records the sequence of all transactions that have taken place so far.

In this section, we will define a repeated consensus abstraction called a *blockchain*. The notion of a blockchain was classically called *state machine replication* in the long line of work in the distributed systems literature. In fact, before Bitcoin and Ethereum, state machine replication (or blockchain) protocols have been deployed by companies like Google and Facebook for more than a decade to replicate their computing infrastructue. The modern name "blockchain" was born together with Bitcoin and became popularized soon after.

6.1 Modeling Network Delay More Generally

So far in our textbook we have considered a "strongly" synchronous network model, where honest nodes' messages are delivered to honest recipients in the immediate next round. Starting in this section, we often adopt a more relaxed (i.e., general) model, where we assume that honest nodes' messages can take at most Δ rounds to be delivered to an honest recipient. The parameter Δ is often called the (maximum) *network delay*. More precisely, if

an honest node sends a message m to an honest recipient in round r, then the recipient will have received the message by the beginning of round $r + \Delta$ if not earlier[1].

Defining this more general network model will lend to our discussions of various network timing assumptions in subsequent chapters. Note that the protocols we learned so far in this course, such as Dolev-Strong [DS83], are strongly synchronous protocols: essentially every Δ delay is renamed to be one round, and nodes only perform actions every Δ amount of time. Of course, not all consensus protocol must abide by this strongly synchronous restriction, even in the case when Δ is a-priori known.

6.2 Defining a Blockchain Protocol

In a blockchain protocol, a set of distributed nodes aim to agree on an ever-growing, linearly-ordered log of transactions. Rather than agreeing on one transaction at a time, often times the protocol would want to use *batching* to improve throughput — a block is exactly a batch of transactions (possibly attached with protocol metadata) and therefore a chain of blocks would be called a *blockchain*.

Roughly speaking, a blockchain protocol must satisfy two important security properties, *consistency* and *liveness*. Consistency requires that all nodes have the same view of the linearly ordered log — but since their network speeds may differ, we shall allow some nodes' logs to potentially grow a little faster than others. More specifically, in our formal definition below, we shall require that honest nodes' logs be prefixes of each other; but we do not require that all honest nodes' logs are exactly the same length in the same round. Liveness requires that if some honest node receives some transaction tx in some round r, then tx will appear in every honest node's "finalized log" by the end of round $r + T_{\text{conf}}$ where T_{conf} is often called the "confirmation time"[2]. Whenever a transaction appears in a node's finalized log, it is treated as having been confirmed, i.e., the transaction cannot be undone later. Below we define a blockchain abstraction more formally.

We assume that there are in total n nodes. The nodes receive transactions from an external environment, where transactions are represented as bit-strings that possibly need to abide by certain validity rules (e.g., with valid

[1] Alternatively, one can also model time as continuous rather than consisting of discrete rounds, this modeling choice is non-essential in understanding the results we shall present.

[2] $\mathsf{tx} \in \{0,1\}^*$ is a payload string. In some applications, there may be some validity or well-formedness rule on tx.

signatures from the coin's owner). The nodes each maintain a growing linearly ordered log of transactions. Henceforth let LOG_i^r denote node i's log in round r — LOG_i^r is also called a *finalized log*, i.e., every transaction or event contained in LOG_i^r cannot be undone later. A blockchain protocol must satisfy the following two requirements:

- *Consistency:* for any honest nodes i and j, and for any round numbers t and r, it must be that $\mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^r$ or $\mathsf{LOG}_i^t \succeq \mathsf{LOG}_j^r$. Here $\mathsf{LOG} \preceq \mathsf{LOG}'$ means that the former log is a prefix of the latter or they are the same.

- T_{conf}*-liveness:* If an honest node receives some transaction tx as input in some round r, then by the end of round $r + T_{\text{conf}}$, all honest nodes' local logs must include tx.

Clarifications on the definition. We make some further clarifications regarding the above definition:

- In the above consistency definition, the two honest nodes i and j are allowed to be the same, or different; and consistency must hold regardless.

- As we stipulated earlier, each (honest) node's local log is growing over time and *can never shrink* — this requirement is baked into the syntax and not reflected in the above consistency or liveness notions.

- The confirmation time T_{conf} can be a function of the number of nodes n, the maximum network delay Δ, and possibly other parameters. It is straightforward why T_{conf} might be a function of Δ since this is the maximum delay it takes for honest nodes to deliver messages to each other. Why can T_{conf} also depend on n? For example, jumping slightly ahead, we shall see how to construct a blockchain protocol through sequential composition of one-shot Byzantine Broadcast (BB) — in this case, if we instantiate the BB protocol with Dolev-Strong, then, as we have learned, the number of rounds will depend on n.

- If the blockchain protocol is deterministic (possibly in the ideal signature model), we would require that the above consistency and liveness properties hold deterministically. However, we will encounter randomized blockchain constructions later. If the protocol is randomized, we often require that regardless of corrupt nodes' strategy, the consistency and liveness properties must hold with probability $1 - \delta$ over the choice

of the randomized execution. The term $\delta \in (0, 1)$ is often referred to as the failure probability, and we typically want δ to be tiny.

- Note that the blockchain definition itself does not specify how the application-layer should process the messages included in the blockchain. The application layer can specify application-dependent validity rules for the format of the message to be included in each block: for example, a typical rule is that the message needs to be a set of transactions with valid signatures.

  Rules for dealing with double-spending are also application-specific decisions and therefore we do not include them in the blockchain abstraction. For example, if two or more transactions spending the same coin both appear in a node's finalized log, the application level can say, only the first one of them will be treated as valid, and the later ones will be discarded by the application semantics. In this case, when is it safe for a merchant to ship the goods to a buyer? The merchant should make sure that the corresponding transaction tx^* appears in the finalized log; and morever, there is no transaction before tx^* in the finalized log that spends the same coin.

6.3 Construction of a Blockchain Protocol from Byzantine Broadcast

In the remainder, we will show that assuming the existence of a PKI and digital signatures, we can construct a blockchain protocol by sequential composition of Byzantine Broadcast (BB). For convenience, we shall assume that the n nodes are numbered $0, 1, \ldots, n-1$. Here we will rely on the Multi-Valued variant of BB (see Exercise 3 of Chapter 3), and we assume that each BB instance runs in R number of rounds.

The blockchain construction is described below, where a new instance of BB is run every r rounds. Each instance of BB agrees on a block, and all blocks are sequentially concatenated to form a log.

Blockchain from sequential composition of BB

- In every round kR that is a multiple of R, i.e., where $k = 0, 1, 2, \ldots$, spawn a new BB protocol whose designated sender is defined to be $L_k := (k \mod n)$. Henceforth, the BB protocol spawned in round kR is denoted BB_k.

The designated sender $L_k := (k \mod n)$ of BB_k collects every transaction tx it has received as input, but that have not been included in its current log, i.e., $\mathsf{tx} \notin \mathsf{LOG}_{L_k}^{kR}$, and inputs the concatenation of all such transactions into BB_k.

- At any time, suppose $\mathsf{BB}_0, \mathsf{BB}_1, \ldots, \mathsf{BB}_{k'}$ have finished and their outputs are $m_1, m_2, \ldots, \mathsf{m}_{k'}$ respectively. The node's current output log is defined as the concatenation $m_1||m_2||\ldots||m_{k'}$ where "||" denotes concatenation.

Theorem 6. *Suppose that the BB protocol adopted realizes Multi-Valued Byzantine Broadcast for a network of n nodes and tolerating up to f corruptions, then the above blockchain construction satisfies consistency and $O(Rn)$-liveness also for the same n and f, where R denotes the round complexity of BB.*

Proof. Consistency of the blockchain is guaranteed due to consistency of the BB protocol. For liveness, observe that if a transaction tx is input to some honest node i in round r, then, it takes at most $(n+1)$ additional BB instances till the i becomes the designated sender again (note that the extra $+1$ is because in the worse case, the current BB instance has already started and i is the sender of the current BB instance). The next time i becomes the designated sender again in a BB instance, either tx is already included in i's log, or i will input tx (along with other transactions) into the BB, and by validity of the BB, every honest node's output in this BB will include tx.

What is the confirmation time of the above blockchain protocol?

Answer: in the above, since it takes at most $O(n)$ instances till the honest node i becomes the designated sender again in a BB instance, and every honest node's log includes tx after that BB instance, it is easy to see that the confirmation time is $O(Rn)$. □

Obviously, this is not the best approach to construct a blockchain protocol. However, it helps us to understand the relationship between BB and blockchains from a feasibility (but not necessarily efficiency) perspective.

Remark 8. The above construction of a blockchain protocol from BB does not require introducing any additional assumptions beyond those already used by the BB. That is, if the underlying BB requires a PKI and signatures, then the blockchain would require the same. If the underlying BB does not rely on setup assumptions, the blockchain protocol would need no setup

too. However, in the next section, our construction of BB from blockchain requires the existence of a PKI and digital signatures.

6.4 Discussions

If blockchain is implied by BB from a feasibility perspective in the PKI setting, do we really need blockchain as a separate abstraction?

The answer is yes, and there are many reasons we care about blockchain as a separate abstraction. First, in practice, it is rarely a good idea to implement blockchains by sequentially composing BB. Although there are indeed better ways to sequentially compose BB than what's described above, the sequential composition approach makes it difficult to perform cross-instance pipelining, thereby making it difficult for performance optimizations. Not surprisingly, almost all blockchain protocols that have been deployed in practice (e.g., variants of PBFT [CL99] and Paxos [Lam98], and Bitcoin [Nak08, GKL15]) are "direct blockchain constructions" where there is no clear boundary that separates the blockchain into independent one-shot instances.

Besides performance concerns, the blockchain abstraction also seems useful for defining additional properties such as fairness and incentive compatibility [PS17a].

Chapter 7

A Simple Blockchain Protocol — Streamlet

In this section, we shall construct an extremely simple blockchain protocol called Streamlet which defends against $f < n/3$ number of corruptions. The Streamlet protocol was recently introduced by Chan and Shi [CS20a, CS20b] as a "unified blockchain protocol for pedagogy and implementation"[1]

The entire protocol is very natural and works assuming the existence of a PKI and digital signatures. Informally speaking, it works as follows where each epoch is assumed to be 1 second long (or any suitable parameter whose choice will be discussed later):

- **Propose-vote.** In every epoch:
  - The epoch's designated leader proposes a block extending from the longest notarized chain it has seen (if there are multiple, break ties arbitrarily).
  - Every node casts votes on the first proposal from the epoch's leader, as long as the proposed block extends from (one of) the longest notarized chain(s) it has seen. A vote is a signature on the proposed block.
  - When a block gains votes from at least $2n/3$ distinct nodes, it becomes *notarized*. A chain is *notarized* if all blocks in it are.

[1]The Streamlet protocol in some sense subsumes classical candidates such as Paxos [Lam98], PBFT [CL99], and RAFT [OO14], and is much simpler than these classical protocols. For this reason, we do not seperately cover these classical consensus protocols. See also Section 7.5 for more historical notes.

- **Finalize.** If in any notarized chain, there are three adjacent blocks with consecutive epoch numbers, the prefix up to the second of the triple is considered *final*. When a block becomes final, the block and its entire prefix appears in the node's local finalized log.

In this chapter, we will prove that this extremely simple protocol

1. achieves consistency no matter what the actual network delays are, and even if the network can get partitioned at times;
2. achieves liveness when network conditions are good, i.e., during a period of time in which honest nodes can send messages back and forth to each other within a 1-second round-trip time.

The conditions under which liveness holds are often referred to as *a period of synchrony* [DLS88]. Obviously, if the network is partitioned and nodes cannot deliver messages to each other, the protocol cannot make progress. The Streamlet protocol guarantees consistency nonetheless no matter how bad or adversarial the network conditions are.

7.1 The Streamlet Protocol

We now describe the protocol more formally and prove its security.

7.1.1 Epoch and Leader Rotation

Epochs: The protocol runs sequentially in synchronized epochs that are 1 second long. The 1 second can be replaced with other reasonable parameters, and we will explain how to choose the parameter later. Every player starts epoch 1 at the same time, and then after 1 second, every player enters epoch 2 at the same time, and so on.

Epoch leader: We assume that the n nodes are numbered $1, 2, \ldots, n$ respectively, and let $[n] := \{1, 2, \ldots, n\}$. Every epoch e will elect a random leader using a public hash function $H : \{0,1\}^* \to [n]$ which is modeled as a random oracle. Specifically, epoch e's leader is computed as $H(e)$.

7.1.2 Blocks and Blockchain

Block: Each block is a tuple of the form (h, e, txs) where

- h is called the *parent hash*, i.e., a hash of the prefix of the chain;

- e is called the *epoch number* of the block, which records the epoch in which the block is voted on; and
- tx is a payload string (e.g., it may record a set of transactions to be confirmed). In some applications, we may require that tx satisfies certain validity or well-formedness rules.

A special block of the form $(\bot, e = 0, \bot)$ is called the *genesis block* which is the first block of every valid blockchain.

Blockchain: A blockchain chain is a sequence of blocks starting with the genesis block $\mathsf{chain}[0] := (\bot, e = 0, \bot)$, and with *strictly increasing* epoch numbers. For a blockchain chain to be valid, it must be that for every non-genesis block $\mathsf{chain}[\ell]$ where $\ell > 0$, $\mathsf{chain}[\ell]$ contains a parent hash equal to $H^*(\mathsf{chain}[0..\ell-1])$, where H^* denotes a hash function that was chosen from a collision-resistant hash family during a setup phase.

In a valid blockchain, we often say that a block $\mathsf{chain}[\ell]$ *extends from* the parent chain $\mathsf{chain}[0..\ell-1]$.

Remark 9 (The hash-chain data structure)**.** Assuming that the adversary did not find any hash collisions during the lifetime of the protocol. Then, the hash-chained data structure guarantees that given a block, its prefix is uniquely determined.

Remark 10 (A practical optimization)**.** Although we write $H^*(\mathsf{chain}[0..\ell])$ for conceptual simplicity, in practice, H^* is typically implemented as an incremental hash (rather than having to hash the entire prefix chain for every block). In other words, each block's parent hash h may be computed simply as a hash of the parent block, which contains a hash of its own parent, and so on.

7.1.3 Votes and Notarization

A vote on a block is simply the node's signature on the block. A collection of at least $2n/3$ signatures from distinct nodes on the same block is called a *notarization* on the block.

A valid blockchain is considered to be notarized by a node i if i has observed a notarization for every block (except the genesis block).

7.1.4 Protocol

Although not explicitly stated, we shall make an implicit echoing assumption:

Implicit echoing: upon observing a new transaction or message, a node always echos the transaction or message to everyone else.

We use the notation $\langle m \rangle_i$ to denote a message m, along with node i's signature on m. With this in mind, we now describe the Streamlet protocol.

The Streamlet blockchain protocol

For each epoch $e = 1, 2, \ldots$:

- **Propose:** At the beginning of epoch e, epoch e's leader L does the following: let chain be (any one of) the longest notarized chain(s) that the leader has seen so far, let $h := H^*(\mathsf{chain})$, and let txs be the set of unconfirmed pending transactions.

  The leader L sends to everyone the proposed block $\langle (h, e, \mathsf{txs}) \rangle_L$ extending from the parent chain chain.

- **Vote:** During the epoch e, every node i does the following. Upon receiving the first valid proposal $\langle (h, e, \mathsf{txs}) \rangle_L$ from epoch e's leader L, vote for the proposed block iff it extends from one of the longest notarized chains it has seen at the time.

  To vote for the proposed block (h, e, txs), node i simply sends to everyone $\langle (h, e, \mathsf{txs}) \rangle_i$.

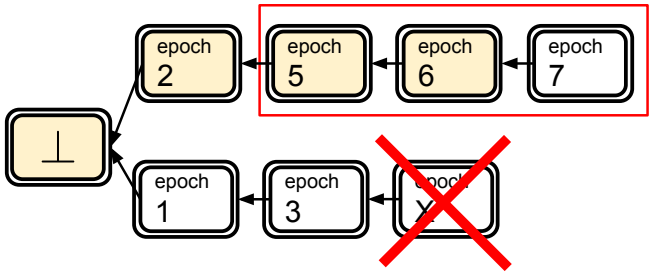
Finalize: On seeing three adjacent blocks in a notarized blockchain with consecutive epoch numbers, a node can finalize the second of the three blocks, and its entire prefix chain.

Note that when a block is *finalized* by a node i, the block and its entire prefix chain shows up the i's local log; all the transactions contained in the block and its prefix are confirmed and can never be undone later on.

Basically the entire protocol follows a propose-vote paradigm, with a somewhat natural but also "magical" finalization rule. We give an example of the finalization rule below to aid understanding.

Example. To aid understanding, we illustrate the finalization rule in the following figure. In this example, all blocks are notarized, and in the top chain, we have three adjacent blocks with consecutive epoch numbers $5, 6$, and 7. In this case, we can finalize the entire prefix chain[2] "$\perp$ - 2 - 5 - 6".

[2]When we refer to the example in the figure, we often use the block's epoch number to name the block.



To argue consistency, we will later prove that there cannot be a conflicting block notarized at the same *height* (i.e., distance from the genesis block) as the block with epoch 6, and thus the bottom chain "⊥ - 1 - 3" basically cannot grow further.

7.2 Consistency

For simplicity, henceforth we shall assume that both the signature scheme and the collision-resistant hash function H^* are ideal, i.e., there are no signature forgeries and no hash collisions.

We give a proof for the above example — and the full, formal proof basically follows by changing the parameters in our argument to general ones.

In the above example, we want to show that there cannot be any other conflicting block notarized at the same height as the block 6.

Suppose for the sake of contradiction that indeed some other conflicting block got notarized at the same height as block 6, e.g., the block with epoch number X. The following lemma says that X must either be greater than 7 or smaller than 5 (see also Remark 9). We will use the term "in honest view" to mean that some honest node observes it at some time in the protocol.

Lemma 7 (Unique notarization per epoch)**.** *Suppose that $f < n/3$. Then, for each epoch e, there can be at most one notarized block with the epoch e in honest view.*

Proof. Suppose that two blocks B and B', both of epoch e, got notarized in honest view. It must be that at least $2n/3$ nodes, denoted the set S, signed the block B, and at least $2n/3$ nodes, denoted the set S', signed the block B'. Since there are only n nodes in total, $S \cap S'$ must have size at least $n/3$, and thus at least one honest node is in $S \cap S'$. According to our protocol, every honest node votes for at most one block per epoch. This means that $\mathsf{B} = \mathsf{B}'$. □

Because of Lemma 7, the conflicting block notarized at the same height as block 6 must have epoch either greater than 7 or smaller than 5. We will therefore consider the two cases one by one, and the proof for both cases are almost the same.

Case 1: $X < 5$**.** Since block X got notarized, it means that more than $n/3$ honest nodes, denoted S, voted for block X and not only so, at the time of the voting (that is, during epoch $X < 5$), they must have observed block 3 notarized. Now the honest nodes in S will not vote for block 5 during epoch 5, since it fails to extend a longest notarized chain seen, which is block 3 or longer. Since $f < n/3$, this means that block 5 can never get notarized in honest view. This leads to a contradiction.

Case 2: $X > 7$**.** Since block 7 is notarized, more than $n/3$ honest nodes (denoted the set S) must have seen a notarized block 6 by the time they vote for block 7 (i.e., by the end of epoch 7). As a result, by in epoch $X > 7$, the set S of nodes must have seen block 6 notarized and will not vote for block X, since block X now fails to extend the longest notarized chain seen (which is block 6 or longer). Therefore block X cannot gain $2n/3$ votes, and it cannot be notarized, which is a contradiction.

The above consistency proof was for our specific example earlier, but it can easily be generalized to the following statement.

Theorem 7 (Consistency)**.** *If some honest node sees a notarized chain with three adjacent blocks* $\mathsf{B}_0, \mathsf{B}_1, \mathsf{B}_2$ *with consecutive epoch numbers* $e, e+1$*, and* $e+2$*, then there cannot be a conflicting block* $\mathsf{B} \neq \mathsf{B}_1$ *that also gets notarized in honest view at the same height as* B_1*.*

> **Exercise 11.** Please prove Theorem 7 by generalizing the parameters of the argument above. Your proof should not rely on network timing assumptions — see the paragraph below.

The consistency proof holds regardless of network timing. By examining the entire proof, it is not hard to see that the consistency proof holds no matter what the actual network message delays are. Of course, if the network is partitioned and honest nodes' messages are being held up, then we cannot guarantee progress. As explained in the following section, liveness ensues during "periods of synchrony", i.e., during a period of time in which honest nodes can deliver messages back and forth within a round-trip time of 1 second (i.e., equal to the epoch length).

7.3 Liveness

A *period of synchrony* is a duration of time in which honest nodes can deliver messages to each other back and forth in a round-trip-time of at most 1 second (or whatever the epoch duration is). Another way to state it is that the network's maximum delay Δ is half a second (i.e., half the epoch length) during a period of synchrony.

Chan and Shi [CS20a] prove the following liveness theorem:

Theorem 8 (Liveness [CS20a])**.** *If after some point of time, the network enters a period of synchrony, and moreover, there are 5 consecutive epochs* $e, e+1, \ldots, e+4$ *all with honest leaders, then, by the beginning of epoch* $e+5$*, every honest node's log must grow by at least one new block (that was not there at the beginning of epoch* e*), and moreover, this new block was proposed by an honest leader.*

Discussion. Recall that earlier in our protocol, we need a notarized chain to have three adjacent blocks with consecutive epoch numbers to finalize blocks. You might wonder why in the liveness theorem above, we require 5 consecutive epochs with honest leaders (during a period of synchrony) to make progress. At a very high level, we first need a couple honest leaders to "undo" the damage that corrupt leaders have done, so that the next three honest leaders could make sure to have three consecutive blocks notarized. The proof by Chan and Shi [CS20a] formalizes this very informal intuition.

Confirmation time and how randomization helps. Since the epoch leaders are randomly chosen by a hash function, with $\Theta(1)$ probability, any fixed 5 consecutive epochs would all have honest leaders. In other words, during a period of synchrony, transactions can be confirmed in *expected constant* number of rounds. This is really interesting because Dolev and Strong [DS83] showed that any deterministic consensus protocol must incur at least $f+1$ number of rounds. Therefore, we can conclude that randomization critically helps with the round efficiency of consensus protocols. In fact, we will explore the round complexity of consensus more in later chapters.

Reading assignment. We won't have time to go over the full liveness proof. Please read the proof by Chan and Shi [CS20a] as a homework assignment.

7.4 The Partial Synchronous Model and Choosing the Epoch Length

Although we did not make it explicit, Streamlet is in fact what's commonly referred to a *partially synchronous* protocol. The partially synchronous model was first proposed by Dwork, Lynch, and Stockmeyer in a groundbreaking work [DLS88]. Unlike a synchronous protocol which relies on network synchrony to achieve consistency, a partially synchronous must safeguard consistency no matter how long the network delay is. When deploying a partially synchronous protocol, we still need to choose a delay parameter Δ and the protocol needs to be preconfigured with this parameter. But no matter what the choice is and whether the actual network respects the Δ-bound, consistency must be guaranteed nonetheless — for this reason, partially synchronous protocols are said to be arbitrarily *partition tolerant*. A partially synchronous consensus protocol must also respect liveness during periods of synchrony, i.e., when the Δ-bound is indeed respected. In practice, we often choose Δ based on what we believe to be the network's performance under normal conditions (without having to worry about what Δ might be in worst-case scenarios, e.g., during outages or attacks).

Remark 11 (Two ways to define partial synchrony [DLS88])**.** Dwork et al. [DLS88] in fact proposed two ways to model a partially synchronous network.

- *The "period of synchrony" model.* This is the model we have adopted so far, i.e., consistency should hold regardless of the network delay and liveness is required to hold only during sufficiently long periods of synchrony. We stress that the protocol does not know a-priori when the period of synchrony will begin.

- *The "unknown-Δ" model.* Another partially synchronous model introduced by Dwork et al. [DLS88] is the "unknown-Δ" model: in this case the protocol is not preconfigured any Δ parameter. In other words, we want a *universal* protocol that works regardless of the actual Δ; that is, both consistency and liveness should always hold even though the protocol does not know the actual Δ, but the confirmation delay T_{conf}, of course, may depend on the actual Δ (i.e., the faster the network, the faster the confirmation).

Without going into details, Dwork et al. [DLS88] showed that the above two models are equivalent from a feasibility perspective, i.e., it is possible to

construct one type of partially synchronous protocol from another. For this reason, both models are referred to as the partially synchronous model.

In the distributed consensus literature, the "unknown-Δ" model is adopted more often in theoretical explorations, whereas the "period-of-synchrony" model is used more often for practical constructions since it typically results in simpler and cleaner protocols. Please read the elegant work of Dwork et al. [DLS88] to learn about the two definitional approaches.

Remark 12 (Resilience of partially synchronous consensus)**.** It turns out that Streamlet achieves *optimal resilience* under partial synchrony. In the later chapters, we will prove that no partially synchronous consensus protocol can tolerate $n/3$ or more corruptions. This elegant lower bound was first described by Dwork et al. [DLS88] as well.

7.5 Historical Anecdotes

Historically, Paxos [Lam98], PBFT [CL99], and their numerous variants [KAD+07, GKQV10,Bur06,JRS11,BSA14] have been the mainstream practical approach for distributed consensus. As mentioned in the previous chapter, almost all practical implementations have adopted direct blockchain constructions, rather than composition of single-shot consensus. Partly, the latter makes it difficult to perform cross-instance pipelining and performance optimization.

The classical mainstream approach [CL99, Lam98, KAD+07, GKQV10, Bur06, JRS11, BSA14] typically adopts a simple *normal path* that follows a natural propose-vote paradigm. However, when the leader misbehaves, the classical approach would go through a complicated recovery path (often called *view change*) to rotate the leader. It is often difficult to precisely understand the recovery path construction and implement it correctly, and flaws have been demonstrated later on in well-known protocols.

For decades, researchers have made it a top priority to simplify these consensus protocols, and various attempts have been made [OO14, Lam01, VRA15]. Very recently, due to interests in decentralized cryptocurrencies, various blockchain projects made independent efforts to simplify consensus protocols [BZ17, CPS18b, CPS18a, YMR+18, HMW, Shi19b]. Nonetheless, this line of work is difficult to navigate partly due to the use of disparate terminology.

Inspired the proposals of various blockchain projects, a couple very recent efforts [Shi19b, CS20a] called for a unified protocol for pedagogy and implementation. The Streamlet protocol presented in this chapter is the arguably (among the) simplest and most natural candidate(s) known to date.

7.6 Additional Exercises

Exercise 12. The Dolev-Strong protocol we learned earlier worked in a synchronous model, and consistency holds when the network always guarantees the delivery of honest messages within one round. Show that the Dolev-Strong protocol is *not* consistent under partial synchrony. For example, you can demonstrate a counter-example in which one or more honest nodes drop offline for some number of rounds and then come back online. The messages sent by or destined for these nodes will be delivered late due to the outage. Show that these honest nodes might not be consistent with other honest nodes.

Exercise 13. Consider an execution of the Streamlet protocol where $n = 99$. Suppose that strictly fewer than $n/3$ nodes are corrupt. For each of the following scenarios, please answer whether such a scenario can possibly occur during the execution. If so, please provide an explanation of how such a scenario can occur. If not, please say why. For each scenario, if it is possible to occur, please also state which blocks are considered final by the relevant nodes.

a) The two *notarized* chains $\bot - 1 - 3 - 5$ and $\bot - 2 - 4 - 6$ will both be in honest view.

b) An honest node P observes a notarized chain of the form $\bot - 3 - 4 - 5$, and another honest node Q observes a notarized chain of the form $\bot - 1 - 2 - 4 - 5 - 6$.

c) An honest node P observes a notarized chain of the form $\bot - 3 - 4 - 5$, and another honest node Q observes a notarized chain of the form $\bot - 1 - 2 - 7 - 8 - 9$.

Exercise 14. As a software engineer, Victor is given a task to implement a consensus protocol for a core service of his company. Victor decided to implement the Streamlet protocol. However, his manager told him that it is ok to assume that strictly less than $\frac{1}{4}$ of the nodes can be corrupt. (Recall that in the original protocol, we assume that strictly less than $\frac{1}{3}$ can be corrupt.) Being an expert on Streamlet, Victor wants to modify the protocol to make use of this stronger assumption in order to maximize system efficiency. What is the minimum number of votes we need to notarize a block in this case? (Recall that in the original protocol, we need $\frac{2}{3}$ fraction of the nodes to vote to get a notarization.) Please prove consistency[3].

[3]Thanks to Yiwen (Victor) Song for suggesting this exercise.

Chapter 8

Lower Bound for Partial Synchrony

In the last chapter, we learned about the *partially synchronous* model, and studied a partially synchronous blockchain protocol, called Streamlet, that resists fewer than $n/3$ faults. One natural question arises:

Can a partially synchronous consensus protocol tolerate $n/3$ or more malicious corruptions?

It turns out that the answer is NO, i.e., no partially synchronous protocol can achieve consensus under at least $n/3$ corruptions. We shall prove it in this lecture.

8.1 Problem Definition

In the last lecture, we mentioned that Dwork et al. [DLS88] suggested two partially synchronous models: 1) the unknown-Δ model; and 2) the period-of-synchrony model. They also showed that the two models are equivalent from a feasibility perspective.

In this lecture, we will prove our lower bound using the unknown-Δ model; and the same lower bound in fact applies to the period-of-synchrony model too. We leave it as a homework exercise (see Exercise 18) to prove the same lower bound for the period-of-synchrony model.

We will prove our lower bound for a one-shot consensus abstraction. Recall that in earlier lectures we considered Byzantine Broadcast (BB) as a one-shot consensus abstraction when studying the feasibility of consensus in a synchronous network. Unfortunately, BB is unrealizable in a partially

synchronous network even if at most one node can be corrupt (see Exercise 17). Instead we consider an agreement version of the problem henceforth referred to as Byzantine Agreement (BA). In BB, there is a designated sender who receives an input bit and wants to broadcast this input bit to everyone else. In BA, *everyone* receives an input bit, and they would like to run a protocol to agree on a bit. The following requirements must be satisfied — note that the key difference from BB is that the new validity requirement:

- *Consistency.* If two honest nodes output b_1 and b_2 respectively, it must be that $b_1 = b_2$.
- *Validity.* If all honest nodes receive the same input bit b, then all honest nodes' must output b.
- *T-liveness.* Every honest node must have produced an output after T rounds where T may be a function of n, the *actual* network delay Δ, and other parameters. Here the actual network delay is determined a-posteriori from the actual execution trace since there is no a-priori promised Δ.

There are a few important things to note about the definition:

1. First, always keep in mind that a partially synchronous protocol is unaware of Δ and in this sense, it must a universal protocol that works for all Δ.
2. Related to the above point, the confirmation delay T is a function of the *actual* delay Δ (and other parameters) determined a-posteriori from the actual execution trace.
3. Recall that when we defined *synchronous* Byzantine Broadcast earlier in Chapter 3, we did not define a liveness property. This is because in the synchronous setting, we may assume that the protocol always terminates and outputs after an a-priori fixed number of rounds $T(n, \Delta)$ where Δ is a-priori known. In the partially synchronous setting, since Δ (or GST) is unknown, we cannot know a-priori how long the protocol needs to run to get any output — it depends on the actual network delay.
4. In the above, we implicitly required that the security properties, including consistency, validity, and liveness to hold deterministically. Sometimes, the BA protocol may be randomized, and in such cases we say that a protocol achieves BA with probability $1 - \delta$ iff regardless of

the corrupt nodes' strategy, with probability $1 - \delta$ over the choice of the randomized execution, the above security properties are respected. We often want the failure probability $\delta \in (0, 1)$ to be tiny.

Given a partially synchronous blockchain protocol, one can construct a partially synchronous BA protocol as below — as long as fewer than $n/3$ nodes are corrupt: nodes run a blockchain protocol, and each node signs its input bit and posts the signed bit to the blockchain (by inputting the signed bit to the blockchain protocol). As soon as at least $2n/3$ bits signed by distinct nodes have appeared in a node's finalized chain, the node outputs the majority bit among the first $\lceil 2n/3 \rceil$ bits (signed by distinct nodes). In the case of a tie, output the canonical bit 0.

Exercise 15. Prove that as long as fewer than $n/3$ nodes are corrupt, the above BA protocol indeed satisfies consistency, validity, and T-liveness in the unknown-Δ partially synchronous model for some appropriate choice of T, as long as the underlying blockchain protocol satisfies consistency and liveness (defined in Chapter 6) in the unknown-Δ model.

Therefore, the existence of a partially synchronous blockchain protocol implies the existence of a partially synchronous BA protocol, as long as fewer than $n/3$ nodes are corrupt.

It turns out the reverse direction is true — and this time without imposing a constraint on the number of corrupt nodes f. In other words, given a partially synchronous BA protocol resilient against f corrupt nodes, we can construct a partially synchronous blockchain protocol resilient against f corrupt nodes. Informally, we can divide the execution into epochs and have a leader propose a block of transactions during each epoch; then nodes invoke a BA instance per epoch to agree on a block, and the input to the BA is what the epoch's leader has proposed. The log of each node is formed by concatenating all the BAs' outputs sequentially. Because of the above, proving a resilience lower bound for partially synchronous BA is equivalent to proving the same resilience lower bound for a partially synchronous blockchain protocol.

8.2 Impossibility of Partial Synchronous Consensus under $n/3$ Corruptions

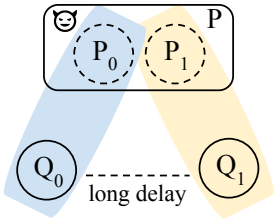
We now prove the main theorem of this lecture stated below.

Theorem 9 (Impossibility of partially synchronous consensus under $n/3$ corruptions [DLS88])**.** *Let T be an arbitrary function on n and Δ. Suppose that at least $n/3$ nodes are corrupt, then, no partially synchronous protocol can realize BA with liveness parameter T.*

Proof. We prove that no partially synchronous protocol can achieve BA for $n = 3$ and $f = 1$. Suppose that this is not true and there exists a partially synchronous protocol Π that achieves BA for $n = 3$ and $f = 1$.

Consider an execution of the protocol with three nodes named P, Q_0, and Q_1. Among the three, Q_0 and Q_1 are honest and receive the input bits 0 and 1 respectively. Now, P is corrupt, and instead of following the honest protocol, it simulates the actions of two nodes named P_0 and P_1 in its head. Both the simulated P_0 and P_1 run the honest protocol, P_0 interacts with Q_0 and P_1 interacts with Q_1. Furthermore, P_0 is given the input bit 0 and P_1 is given the input bit 1.

Now, imagine that the message delay between Q_0 and Q_1 are more than $T(n = 3, \Delta = 1) + 5$. However, messages between P_0 and Q_0 are always delivered in the immediate next round, and so are messages between P_1 and Q_1.



Since P is the only corrupt node, by consistency, we know that Q_0 and Q_1 must output the same bit.

However, before round $T(n = 3, \Delta = 1) + 5$, P_0 and Q_0's joint view is identically distributed as the case when Q_1 has crashed but both P_0 and Q_0 are honest, receive the input bit 0, and moreover, the actual network delay $\Delta = 1$ between P_0 and Q_0. By T-liveness, P_0 and Q_0 will produce an output in $T(n = 3, \Delta = 1)$ rounds[1]. By validity, Q_0 must output 0. With a symmetric argument, we conclude that Q_1 must output 1. This leads to a contradiction since earlier we concluded that Q_0 and Q_1 must produce the same output bit. □

Remark 13. The above lower bound holds even when allowing setup (e.g., PKI, random oracles) and making cryptographic hardness assumptions.

[1]Note that here T takes in the actual Δ which is 1 in this case.

Remark 14. The above lower bound rules out the existence of any partially synchronous protocol that achieves BA with probability 1 under $n/3$ corruptions. We can in fact generalize the above impossibility proof and rule out any *randomized*, partially synchronous protocol that achieves BA with probability more than $2/3$, assuming that at least $n/3$ nodes are corrupt. This is left as a homework exercise (see Exercise 16).

8.3 Additional Exercises

Exercise 16. Theorem 9 rules out the existence of any deterministic, partially synchronous protocol that achieves BA in the presence of at least $n/3$ corrupt nodes.

Prove that there does not exist a *randomized*, partially synchronous protocol that achieves BA with probability more than $2/3$ in the presence of at least $n/3$ corrupt nodes.

Exercise 17. Recall that in Byzantine Broadcast (BB), a designated sender has an input bit b, and it would like to propagate this input bit to everyone else. *Consistency* and *liveness* are defined in the same manner as the BA definition earlier in this lecture. *Validity* requires that if the designated sender is honest, then all honest nodes must output the sender's input bit.

A partially synchronous BB is unaware of Δ, but its liveness parameter may depend on the network's actual Δ.

Prove that there does not exist a partially synchronous BB protocol even for $f = 1$. You may choose to prove it for either the unknown-Δ model or the period-of-synchrony model.

Exercise 18. Prove the equivalent of Theorem 9 but now for the partially synchronous protocols in the period-of-synchrony model. Note that in the period-of-synchrony model (see Chapter 7), T-liveness is redefined such that honest nodes are only required to produce an output if the network eventually enters a period of synchrony, and moreover honest nodes must have produced an output T rounds after the start of the period of synchrony.

Exercise 19. Prove that BA is impossible if at least $n/2$ nodes are corrupt, even in the synchronous model. In other words, unlike the BB abstraction, we need to assume honest majority to construct BA even under synchrony.

Chapter 9

Round Complexity of Deterministic Consensus ※

In this lecture, we will prove that any *deterministic* Byzantine Broadcast protocol (allowing ideal signatures) must incur at least $f + 1$ rounds. This is a celebrated result first proven by Dolev and Strong [DS83], and it shows that the Dolev-Strong protocol achieves optimal round complexity. We will present a re-exposition [lec] of Dolev and Strong's proof [DS83].

9.1 Weakly Valid Byzantine Agreement

To rule out a deterministic BB protocol with fewer than $f + 1$ rounds, we actually prove a round complexity lower bound for a weaker abstraction called *weakly valid* Byzantine Agreement (BA). Recall that in Chapter 8 we defined the agreement version of single-shot consensus, called Byzantine Agreement (BA). In BA, every node receives an input bit, and would like to agree on an output bit. We want *consistency*, i.e., all honest nodes' outputs be the same; and *validity*, i.e., if all honest nodes receive the same input bit b, then they must all output b. Henceforth, this notion of validity is referred to as *strong validity*.

We now define a more relaxed version, called weakly valid BA. The only difference from the earlier (strongly valid) BA is that the validity condition is weakened: we now require only the following:

> *Weak validity:* if all nodes are honest and receive the same input bit b, then they must all output b too.

As it turns out, if there is an R-round BB protocol that tolerates f

corruptions, then there is an R-round weakly valid BA protocol also tolerating f corruptions. Proving this is straightforward and left as a homework exercise. This tells us that proving a round complexity lower bound for weakly valid BA would immediately lead to the same round complexity lower bound for BA.

Exercise 20. Prove that if there is a deterministic R-round BB protocol that tolerates f corruptions, then there is a deterministic R-round weakly valid BA protocol also tolerating f corruptions.

9.2 Proving the Lower Bound for $f = 2$

For simplicity, we will focus on the special case when $f = 2$ and later on argue why the proof readily extends to more general choices of f.

More concretely, we begin by proving the following statement:

Lemma 8. *Suppose that $n \geq 4$. Then, no 2-round deterministic protocol can realize weakly valid BA in the presence of 2 corruptions.*

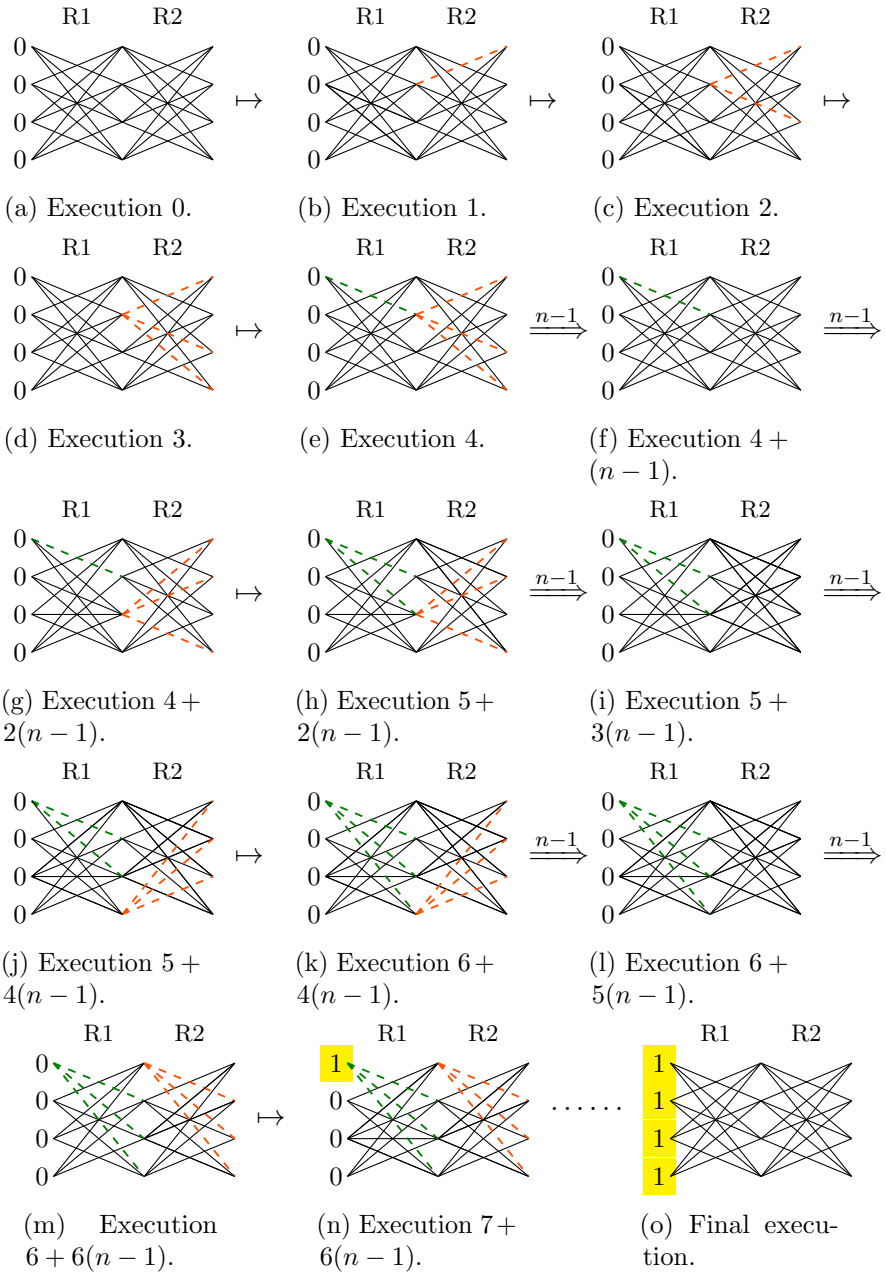
9.2.1 Overview of the Proof

Simplifying assumptions. Since we are concerned about the round complexity of the protocol, without loss of generality, we may assume that in every round, everyone sends a message to everyone else (but not the node itself).

Proof idea: constructing a sequence of executions. Suppose for the same of reaching a contradiction, that there exists a 2-round protocol Π that realizes BB in the presence of 2 corruptions.

We will consider a sequence of executions (depicted in Figure 9.1) of this protocol Π and reach a contradiction. Specifically, in the first execution, all nodes are honest and receive the input 0 whereas in the last execution, all nodes are honest and receive the input 1. We want to conclude that the nodes' outputs in the first execution must be equal to their outputs in the last execution, thereby reaching a contradiction.

In every execution in this sequence, every node runs the honest protocol; however, a network adversary may erase some of the messages. The messages erased are denoted by dashed lines in Figure 9.1. Alternatively, instead of viewing the attack as being conducted by a network-only adversary, we can imagine that a subset of the nodes are corrupt: corrupt nodes suppress a



(a) Execution 0. (b) Execution 1. (c) Execution 2.

(d) Execution 3. (e) Execution 4. (f) Execution $4 + (n-1)$.

(g) Execution $4 + 2(n-1)$. (h) Execution $5 + 2(n-1)$. (i) Execution $5 + 3(n-1)$.

(j) Execution $5 + 4(n-1)$. (k) Execution $6 + 4(n-1)$. (l) Execution $6 + 5(n-1)$.

(m) Execution $6 + 6(n-1)$. (n) Execution $7 + 6(n-1)$. (o) Final execution.

Figure 9.1: Sequence of executions. Dashed lines denote communication that is dropped. The arrow $\mapsto$ denotes a single transition to the next execution in this sequence (also called a "small step"); the arrow $\overset{n-1}{\Longrightarrow}$ denotes a collection of $n-1$ transitions (also called a "big step").

subset to all of the messages it ought to send but otherwise follow the honest protocol. Thus, if a node i's message is partially or completely being erased in some round (i.e., there exist dashed edges outgoing from i) then node i is treated as corrupt.

Invariants. The sequence of executions in Figure 9.1 are constructed with two important invariants in mind.

1. *Neighboring constraint:* for any pair of adjacent executions, there exists at least one node that is honest in both executions, and moreover its view has not changed in between these executions.
2. *Corruption constraint:* in every execution, for each round, there is at most one node whose outgoing messages can be erased. In other words, in Figure 9.1, only one node can have outgoing dashed edges in each round. Recall that one way to interpret the missing messages is that their sender is corrupt and dropped the messages — thus by construction, the total number of corrupt nodes is upper bounded by f.

The corruption constraint makes sure that in every execution, at most f nodes are corrupt and thus consistency and weak validity should be respected in all executions. Now, if we also have the neighboring constraint, we can then make sure that honest nodes' outputs are consistent in any two adjacent executions — therefore, the honest nodes' outputs must be consistent in the first and last executions as well.

9.2.2 Sequence of Executions for $f = 2$

We now walk through the sequence of executions (Figure 9.1) for the special case $f = 2$, at the end of which we will reach the contradiction we need.

Notation. We first introduce some notation that will help us:

1. The n nodes are numbered $1, 2, \ldots, n$, and we use the notation $[n] := \{1, 2, \ldots, n\}$.
2. We use the notation $i \xrightarrow{r} j$ to denote the r-th round message from node i to node j where $r \in \{1, 2\}$ and $i, j \in [n]$.
3. Similarly, for $S, S' \subseteq [n]$, $S \xrightarrow{r} S'$ denotes the set of all round-r messages from any node in S to any node in S'.
4. We use $*$ to denote wildcard, i.e., $* := [n]$. We use $j \in [n]$ to denote a singleton set $\{j\}$ whenever convenient.

Sequence of executions. We now go over the sequence of executions in Figure 9.1.

- To start with, in Execution 0, everyone receives the input bit 0 and no message is erased. In this case, everyone should output 0 due to weak validity.

- In Execution 1, message $2 \xrightarrow{2} 1$ is erased. Now, observe nodes 3 and 4 are honest in both Execution 0 and Execution 1, and moreover, their views have not changed in between the two executions. Since these nodes output 0 in Execution 0, they must output 0 in Execution 1 as well. Since only one node is corrupt in Execution 2, by the consistency requirement, all honest nodes (i.e., everyone except node 2) should output 0 in Execution 1.

- In Executions 2 and 3, we erase the messages $2 \xrightarrow{2} 3$ and $2 \xrightarrow{2} 4$ one by one. In both Executions 2 and 3, only node 2 is corrupt. Moreover, we respect the neighboring constraint as we remove the edges one by one. We thus conclude in a similar fashion that in both Executions 3 and 4, every honest node should output 0.

- In Execution 4, we erase the message $1 \xrightarrow{1} 2$ and therefore in Execution 4, only 1 and 2 are corrupt. Since in Execution 3, node 2 does not send any second-round messages, erasing $1 \xrightarrow{1} 2$ does not change node 3 or 4's view in these two adjacent executions. Therefore by a similar argument, we conclude that all honest nodes output 0 in Execution 4.

- Now, we restore the messages $2 \xrightarrow{2} *$ one by one, leading to Execution $4 + (n-1)$. It is not hard to verify that each step respects both the neighboring and the corruption constraints. Thus all honest nodes must output 0 in these executions.

- Now one by one, we erase the second-round message $3 \xrightarrow{2} *$ leading to Execution $4 + 2(n-1)$. Both above constraints are respected in each step.

- At this moment, we may erase the message $1 \xrightarrow{1} 3$, leading to Execution $5 + 2(n-1)$.

- We now continue this process as shown in Figure 9.1. When we reach Execution $7 + 6(n-1)$, node 1 no longer sends any message and thus we can flip its input to 1 without affecting any honest nodes' views.

- At this moment, we follow the reverse process that reverses the steps taken from Execution 0 to Execution $7 + 6(n-1)$ to add back all edges. At this point, we have flipped node 1's input in comparison with Execution 0, and everything else remains unchanged.
- Now, using the procedure we took to flip the input of node 1, we can flip the input of every other node too, eventually reaching the final execution where all nodes are honest and receive the input 1. Since in all steps we respect the neighboring constraint as well as the corruption constraint, we may conclude that even in the final execution all honest nodes must output 0. However, this violates the weak validity requirement and thus we have reached a contradiction.

Exercise 21. How many executions are there in the above sequence?

9.3 Extending the Argument for General Choices of f

Given the above example for $f = 2$, we are now ready to generalize the proof to arbitrary choices of $f \leq n - 2$, leading to the following theorem:

Theorem 10 (Round complexity lower bound for deterministic consensus [DS83])**.** *Suppose that $n \geq f + 2$. Then, no deterministic protocol of f or fewer rounds can realize weakly valid BA in the presence of f corruptions.*

To prove the above theorem. we define a recursive algorithm that generates a sequence of executions that respects both the neighboring and corruption constraints, where each execution can be described as a communication graph G with f rounds similar to Figure 9.1.

Remark 15. Recall that earlier in Section 9.2, we ignored self-destined messages and assume that nodes do not send messages to themselves. Here, it is a little simpler to describe our recursive algorithm that generates a sequence of executions, assuming that nodes can send messages to themselves too.

Our recursive algorithm in Figure 9.1 makes use of the following recursive function calls:

- **Delete**(i, r): Suppose that currently the graph G has a complete set of edges in any round $r' \geq r$ (henceforth this is said to be the *input*

Delete(i, r):

If $r = f$:
  For $j \in [n]$: *DeleteOne*(i, j, f)

Else:
  For $j \in [n]$:
  - **Delete**$(j, r+1)$
  - *DeleteOne*(i, j, r)
  - **Restore**$(j, r+1)$

  Delete$(i, r+1)$

Restore(i, r):

If $r = f$:
  For $j \in [n]$: *RestoreOne*(i, j, f)

Else:
  Restore$(i, r+1)$
  For $j \in [m]$:
  - **Delete**$(j, r+1)$
  - *RestoreOne*(i, j, r)
  - **Restore**$(j, r+1)$

Main():
For $i \in [m]$:
- **Delete**$(i, 1)$
- Flip i's input bit and **output**
- **Restore**$(i, 1)$

DeleteOne(i, j, r):
  Delete $i \xrightarrow{r} j$ and **output**

RestoreOne(i, j, r):
  Restore $i \xrightarrow{r} j$ and **output**

Figure 9.2: **Recursive algorithm for defining a sequence of executions.** All algorithms modify a global graph. Initially, when **Main** is invoked, the graph is a complete communication graph of R rounds with all-0 inputs. Whenever **output** is called, the current graph's state is output as the next execution in the sequence.

assumption for **Delete**). **Delete**(i, r) removes from G all edges $i \xrightarrow{r'} *$ for any round $r' \geq r$. The sequence of intermediate graphs encountered during the process are output.

- **Restore**(i, r): Suppose that in the current graph G, all edges $\{i \xrightarrow{r'} *\}_{r' \geq r}$ are missing but all other edges in round $r' \geq r$ are complete (henceforth this is said to be the *input assumption* for **Restore**). Now, **Restore**(i, r) restores in G all edges $i \xrightarrow{r'} *$ for any $r' \geq r$. The sequence of intermediate graphs encountered during the process are output.

Through mechanical checking, it is easy to verify the syntactic correctness of the above recursive algorithm, that is,

1. **Delete** or **Restore** do realize their intended functionalities described earlier;

2. If **Delete** or **Restore** is invoked upon a graph that satisfies its respective input assumption, then all recursive calls it makes to **Delete** and **Restore** will satisfy the respective input assumptions.

3. Suppose that the **Main** function is invoked for the complete communication graph with all-0 input. Then, the calls to **Delete** and **Restore** made by **Main** satisfy the respective input assumptions. Furthermore, the final graph G is the complete communication graph with all-1 input.

It remains to prove that the sequence of graphs generated by the above algorithm respect the two invariants, i.e., the neighboring constraint and the corruption constraint. If this is true, then we can conclude that honest nodes' output are the same in all executions, thus leading to a contradiction. We now prove that indeed both constraints are satisfied. Recall that the above recursive algorithm outputs a graph only inside *DeleteOne*, *RestoreOne*, and inside the **Main** function.

Lemma 9 (Corruption constraint)**.** *The sequence of graphs output by the above recursive algorithm respects the corruption constraint.*

Proof. It suffices to prove that whenever $\mathit{DeleteOne}(i, j, r)$ is called, all nodes other than i must have complete round-r edges. The proof is done through mechanical checking.

Last round: $\mathit{DeleteOne}(i, j, f)$ can only be invoked from within $\textbf{Delete}(i, f)$. By the input assumption of **Delete**, when $\textbf{Delete}(i, f)$ is invoked, all last-round edges are complete. Therefore, whenever $\mathit{DeleteOne}(i, j, f)$ is called, all nodes other than i must have complete last-round edges.

All other rounds $r < f$: $\mathit{DeleteOne}(i, j, r)$ can only be invoked from within $\textbf{Delete}(i, r)$. $\mathit{DeleteOne}(i, j, r)$ must be invoked upon a graph G that satisfies the following 1) edges in rounds $r + 1$ and greater are all complete, except that outgoing edges from j in rounds $r + 1$ and greater are all missing; and 2) except for node i, all other nodes must have complete round-r messages. □

Lemma 10 (Neighboring constraint)**.** *The sequence of graphs output by the above recursive algorithm respects the neighboring constraint.*

Proof. **Last round:** Since $n \geq f + 2$, and the corruption constraint is satisfied, there are at least 2 honest nodes. However, each time $\mathit{DeleteOne}(i, j, f)$ $\mathit{RestoreOne}(i, j, f)$ is invoked, it removes or restores only one last-round edge. So there must be one honest node whose view is unaffected by the operation.

All other rounds $r < f$: *DeleteOne*(i, j, r) can only be invoked from within **Delete**(i, r). The line **Delete**$(j, r+1)$ deletes all edges in rounds $r+1$ or greater outgoing from j. At this moment, *DeleteOne*(i, j, r) obviously does not affect any honest nodes' view. A similar argument holds for *RestoreOne*(i, j, r).

Inside Main: the line **Delete**$(i, 1)$ deletes all edges outgoing from i. Obviously at this moment, changing i's input bits do not affect any honest nodes' view. □

We leave the following natural questions as homework exercises.

Exercise 22. How many executions are there in the above sequence generated by our recursive algorithm?

Exercise 23. Exactly where would the above proof break if the communication graph had $f + 1$ rounds?

Chapter 10

Round Complexity of Randomized Consensus ※

We know by now that any deterministic BB protocol (possibly in the ideal signature model) must incur at least $f + 1$ rounds where f denotes the maximum number of corruptions. We also learned in Chapter 7 that randomized protocols can achieve blockchain or BB in expected constant number of rounds assuming PKI and digital signatures. In this lecture, we prove a $2n/(n - f) - 1$ lower bound on the round complexity of randomized BB protocols — the proof is due to the elegant work of Garay et al. [GKKO07]. To help understand the result, we may consider a couple interesting special cases of the statement:

- The lower bound is constant when a constant fraction of the nodes are corrupt. Although asymptotically, a constant-round lower bound seems trivial, in this case the statement does provide a concrete constant as the lower bound which is interesting.

- In the special case when $f = n - 2$ nodes are corrupt, we get a linear round complexity lower bound.

10.1 Round Complexity of Randomized BB

We will state and prove our lower bound for the synchronous model, assuming that when honest nodes send messages, the messages are delivered to honest recipients in the immediate next round. Note that a lower bound for a synchronous network makes the result stronger.

Theorem 11 (Round complexity lower bound for randomized BB [GKKO07])**.** *Suppose that $f \leq n-2$ and $\delta < \frac{n-f}{4n}$. No randomized protocol that terminates in fewer than $2n/(n-f)-1$ rounds can achieve Byzantine Broadcast with probability $1-\delta$ in the presence of f corruptions.*

Proof. For simplicity we shall assume that the number of honest nodes $h = n - f$ is an even number, and that n is divisible by $h/2$.

Suppose for the sake of reaching a contradiction, that there is a protocol that defends against f corruptions, completes in fewer than $2n/h - 1$ number of rounds, and achieves BB with probability $1 - \delta$. Let us consider the following execution of the protocol:

- We partition the n nodes into disjoint sets of $h/2$ nodes each, denoted $S_1, S_2, \ldots, S_d$ respectively where $d = 2n/h$.
- All nodes execute the honest protocol; however, there is a network adversary that drops certain messages such that only the following messages get delivered (within one round): 1) messages between nodes in the same set; and 2) messages between nodes in adjacent sets. All other messages are dropped. For example, when a node in S_1 sends a message to a recipient in S_1 or S_2, the message will be delivered in the immediate next round. However, if a node in S_1 sends a message to a recipient in S_3, the message gets dropped.
- The designated sender is in S_1.

Now, for every $i \in [d-1]$, we can interpret the execution in the following way: the h nodes in $S_i \cup S_{i+1}$ are honest, and all other nodes are corrupt and they are ignoring certain incoming messages and dropping certain outgoing messages. Note that in this interpretation, we no longer have a network adversary that drops messages; instead, the messages could have been ignored/dropped by corrupt nodes themselves.

Clearly, there are $d-1$ interpretations in total. Under the first interpretation, and due to the validity properties of BB, we conclude that with probability $1-\delta$, nodes in S_1 and S_2 must all output the designated sender's input (recall that the designated sender is in S_1). For $i > 1$, consider the i-th interpretation: due to the consistency property of BB, we may conclude that with probability $1-\delta$, nodes in $S_i \cup S_{i+1}$ agree on their outputs.

By the union bound, with probability at least $1 - (d-1) \cdot \delta > 1/2$, the nodes in S_d must output the designated sender's input bit. Recall that the designated sender is in S_1, and it takes d rounds for any information to propagate from anyone in S_1 to anyone in S_d. Since BB completes in at most

$d-1$ rounds, by the time the protocol terminates, the view of nodes in S_d is independent of the designated sender's input — in other words, information in S_1 cannot possibly propagate to S_d in fewer than $d-1$ rounds. Therefore, one of the two input bits, denoted b', must cause the nodes in S_d to output differently from b' with probability at least $1/2$. Thus we have reached a contradiction. □

Remark 16. The above round complexity lower bound holds even allowing trusted setup (e.g., PKI, random oracles) and cryptographic assumptions.

10.2 Survey of Recent Results

Although for the honest majority setting, it is long known how to construct expected constant round BB [FM97, KK09]; for the corrupt majority setting, say, when 51% of the nodes are corrupt, it was not known for a very long time how to construct BB with sublinear (in n) round complexity. Fortunately, a couple recent works made progress along this front:

1. Chan et al. [CPS20] showed that assuming trusted setup and standard cryptographic assumptions, there is a protocol that completes in $\Theta(\log \frac{1}{\delta} \cdot \frac{n}{n-f})$ number of rounds and achieves BB with $1-\delta$ probability in the presence of $f < n$ corruptions.

2. Wan et al. [WXSD20] showed a BB protocol that completes in expected $O((\frac{n}{n-f})^2)$ number of rounds and tolerates any $f < n$ corruptions, assuming PKI and digital signatures. Their protocol achieves expected constant number of rounds even when, say, 99% of the nodes are corrupt.

Chapter 11

Communication Complexity of Consensus ※

In the previous couple of chapters, we proved lower bounds on the round complexity of Byzantine Broadcast (BB). Besides the protocol's round complexity, another important metric is the protocol's *communication complexity*, i.e., the total number of bits sent by the union of the honest nodes.

We will assume a *pairwise channel* model, i.e., every pair of nodes have dedicated links for sending messages to each other. The communication complexity of a protocol is the total number of bits honest nodes send over all pairwise channels. This means that if an honest node sends the same message m to all nodes, it is counted as sending $n \cdot |m|$ total bits where n is the number of nodes and $|m|$ denotes the length of the message m.

Moreover, in some protocols, the amount of information an node sends may depend on other nodes' behavior, including what corrupt nodes send. For deterministic protocols, we define communication complexity as the total number of bits honest nodes send under the *worst-case* scenario (assuming that at most f nodes are corrupt). For randomized protocols, communication complexity is typically stated in the following way parametrized with a failure probability $\delta \in (0, 1)$: the union of honest nodes send at most C bits with $1 - \delta$ probability no matter which (up to f) nodes are corrupt and no matter what corrupt nodes' strategy is.

In this lecture, we will first prove a quadratic lower bound on the communication complexity of *deterministic* BB protocols [DR85]. We will then discuss the communication complexity of *randomized* BB protocols.

11.1 Communication Lower Bound for Deterministic Consensus

Dolev and Reischuk proved the following elegant result:

Theorem 12 (Dolev and Reischuk [DR85])**.** *Any* deterministic *Byzantine Broadcast protocol that tolerates up to $f \leq n-2$ corruptions must incur communication complexity at least $\lfloor f/2 \rfloor^2$.*

Proof. Suppose for the sake of reaching a contradiction that there is a BB protocol that has communication complexity smaller than $\lfloor f/2 \rfloor^2$.

Without loss of generality, we may assume that if a node receives no message during the protocol execution, it outputs either 0 or 1 (see Remark 19). Therefore, either at least $n/2$ nodes output 0 when receiving nothing during the entire protocol, or at least $n/2$ nodes output 1 when receiving nothing. Without loss of generality, suppose that it is the former case (if it's the other case, we can simply do a symmetric argument).

We can choose $\lfloor f/2 \rfloor$ nodes that are not the designated sender and would output 0 if nothing is received during the protocol — call this set of nodes V. We will number the n nodes as $1, 2, \ldots, n$, and let $U := \{1, 2, \ldots, n\} \backslash V$. The designated sender is in U. Consider the following execution:

- The designated sender in U receives the input bit 1.
- Nodes in U behave honestly.
- Nodes in V are corrupt and perform the following message omission attack: they drop all messages they would send to nodes in V and send messages only to those in U. Furthermore, every node in V ignores the first $\lfloor f/2 \rfloor$ messages they receive from nodes in U. Otherwise, nodes in V follow the honest protocol.

Since the protocol has smaller than $\lfloor f/2 \rfloor^2$ communication complexity, it means that at least one node in V, denoted p, receives fewer than $\lfloor f/2 \rfloor$ messages from nodes in U, and thus p will ignore all messages it receives from nodes in U. Since at most $\lfloor f/2 \rfloor$ nodes are corrupt in the above execution, by validity, all nodes in U output 1.

Now we will view the same execution with a different interpretation to reach a contradiction. The alternative interpretation is the following:

- Everyone in V is corrupt except p. Everyone in $V \backslash \{p\}$ drops all messages to everyone in V and send messages only to nodes in U; it

also ignores the first $\lfloor f/2 \rfloor$ messages received from nodes in U as well as any message from p. Otherwise these nodes follow the honest behavior.

- The up to $\lfloor f/2 \rfloor$ nodes in U that ever need to send message to p are corrupt but all other nodes in U are honest. The corrupt nodes in U drop all messages they would have sent to p but otherwise follow the honest protocol.

In this alternative interpretation, at most $\lfloor f/2 \rfloor \cdot 2 - 1 \leq f$ nodes are corrupt. By consistency, the honest node p should output 1 since all honest nodes in U output 1 (this was concluded in the earlier interpretation). However, notice that p does not receive any message at all in the alternative interpretation, and by construction, nodes in V output 0 if they receive no message in the protocol. Thus we have reached a contradiction. □

Remark 17. In the above proof, in the second interpretation, the designated sender may be corrupt too. This is why we use consistency rather than validity in the second interpretation.

Remark 18. In fact, the above proof shows something stronger: any deterministic BB must incur at least $\lfloor f/2 \rfloor^2$ *message complexity*, where message complexity is defined as the total number of messages the union of honest nodes send in the worst case. Again, if a node sends the same message to all n nodes, it is counted n times towards the message complexity.

Remark 19. A (deterministic) protocol defines a "program" to be executed by every node. For each node, we can test what its output would be if it does not receive any messages at all during the entire duration of the protocol. There are three possible behaviors: it outputs 0, 1, or nothing. Without loss of generality, we may assume that every node should either output 0 or 1 if no message is received — if a node outputs nothing when no message is received, it means that protocol guarantees that the node receives a message (otherwise the protocol would not be well-defined). In this case we can simply redefine its output to be 0.

Exercise 24. We can tighten the above proof and show that in fact, any deterministic BB protocol that tolerates $f \leq n - 2$ corruptions must incur at least $\lceil f/2 \rceil \cdot \lfloor f/2 \rfloor$ communication complexity. Please describe the tightened proof.

11.2 Communication-Efficient Randomized Consensus

We now describe a random committee election technique that allows us to improve not just the communication complexity of Byzantine Broadcast, but also its round complexity. The technique works assuming that $\frac{1}{2} + \epsilon$ fraction of the nodes are honest, for an arbitrarily small constant $\epsilon \in (0, \frac{1}{2})$.

Without loss of generality, we assume that the nodes are numbered $1, 2, \ldots, n$, and that 1 is the designated sender. Let $[n] := \{1, 2, \ldots, n\}$. We will use a public hash function $H : \{0,1\}^* \to [n]$ (modeled as a random oracle) to elect a small committee of size k. The designated sender is always elected. The remaining $k-1$ committee members are computed as $H(2)$, $H(3)$, ..., $H(k)$. Note that a node can be elected multiple times into the committee, and therefore the committee is a multi-set.

We now run a Byzantine Broadcast protocol, say, the Dolev-Strong protocol, among the committee. This will allow all committee members to agree on a bit. Finally, everyone who is outside the committee contacts all committee members and asks each committee member what the outcome is. It then outputs the bit that is the majority vote among the committee members (and in the presence of a tie, outputs a bit arbitrarily).

We describe the protocol more formally below. Let $\delta \in (0,1)$ be an intended failure probability for the BB protocol, and recall that $\frac{1}{2} + \epsilon$ fraction of the nodes are honest, where $\epsilon \in (0, \frac{1}{2})$ is an arbitrarily small constant.

- At the beginning of the protocol, a public hash function H is chosen and we elect a committee which is a *multiset* of size $k := \lceil 16 \log(\frac{1}{\delta})/\epsilon^2 \rceil + 1$ whose members are elected as follows:
  - first, the designated sender, i.e., node 1, is elected;
  - for $i = 2$ to k, elect the node $H(k)$.

  This step is done locally by all nodes and does not incur any communication.

- The committee members run a possibly inefficient Byzantine Broadcast protocol, say, the Dolev-Strong protocol. ($\star$)

  At the end of the protocol, every committee member has an output b. Let $R(k)$ denote the round complexity of Dolev-Strong when run among k nodes. This step continues for $R(k)$ rounds.

- In the $(R(k)+1)$-st round: for each node i outside the committee, every committee member send its output bit to i. If more than $k/2$ committee members return the same bit b' to i, node i outputs b'; else it outputs an arbitrary bit. $(\star)$

$(\star)$: If a node is elected μ times into the committee, it acts as μ nodes in the steps marked $(\star)$.

Lemma 11. *Suppose that the hash function is chosen at random after an adversary picks which nodes to corrupt, and moreover suppose that $\frac{1}{2}+\epsilon$ fraction of nodes are honest where $\epsilon \in (0, \frac{1}{2})$ is an arbitrarily small constant. Then,*

$$\Pr\left[\textit{majority in the committee are honest}\right] \geq 1-\delta$$

Proof. Let $\mathbf{X}$ be the number of honest nodes among the $k-1$ randomly elected committee members. Clearly, $\mathbf{E}[\mathbf{X}] = (\frac{1}{2}+\epsilon)\cdot(k-1)$.

Now, given that $k = 16\log(\frac{1}{\delta})/\epsilon^2 + 1$, it is not difficult to verify that $(1-\epsilon/2)\cdot\mathbf{E}[\mathbf{X}] > k/2$. We have that

$$\begin{aligned}\Pr\left[\mathbf{X} \leq k/2\right] &\leq \Pr\left[\mathbf{X} \leq (1-\epsilon/2)\cdot\mathbf{E}[\mathbf{X}]\right] \\ &\leq \exp(-(\epsilon/2)^2\cdot\mathbf{E}[\mathbf{X}]/2) \qquad (\bigstar) \\ &\leq \exp(-\epsilon^2\cdot(k-1)/16) = \delta\end{aligned}$$

where the inequality marked $\bigstar$ is due to the Chernoff bound. □

Theorem 13. *The above protocol achieves Byzantine Broadcast with probability $1-\delta$.*

Proof. Due to Lemma 11, it suffices prove that consistency and validity hold for every execution where the majority of the committee are honest. Henceforth we may focus only on executions where the majority of the committee are honest.

We first prove consistency. The honest committee members output consistent decision due to the consistency property of the inefficient BB (i.e., Dolev-Strong). Let b be the bit output by the honest committee members. It remains to show that any honest node outside the committee will output b too. Since the majority of the committee are honest, b must be the majority vote among the committee.

Due to the above argument, to prove validity, we just need to show that the honest committee members' output satisfies validity. This follows from the validity of the inefficient BB among the committee. □

Efficiency of the protocol. The communication and round efficiency of the above protocol depends only on the protocol's failure probability δ and the honest margin ϵ away from a half. In particular, the efficiency measures do not depend on the total number of nodes n. Here, by employing randomness, we circumvented the $\lfloor f/2 \rfloor^2$ communication complexity lower bound [DR85] (described earlier in this chapter) as well as the $f+1$ round complexity lower bound [DS83] (described in Chapter 9) for *deterministic* protocols.

Exercise 25. It turns out that the random committee election technique does not work for a corrupt majority setting. Please explain why.

Exercise 26. In Lemma 11, we assumed that the adversary decides which nodes to corrupt prior to the start of the protocol execution, and specifically in this case, before the hash function H is randomly chosen. This is commonly referred to as the *static* corruption model. Unless otherwise noted, most of this course assumes the static corruption model for simplicity.

However, now let us consider a stronger adversary. Suppose that the adversary is *adaptive*, i.e., it is allowed to corrupt nodes in the middle of the protocol execution, after having observed the messages nodes have sent so far during the protocol. Of course, the adversary must still abide by the corruption budget, say, at most $(\frac{1}{2} - \epsilon)n$ nodes can be corrupt.

Is our protocol in this section secure in the presence of such an adaptive adversary? Please explain why.

11.3 Survey of Recent Results

In Exercise 26, we showed why the protocol in this section does not defend against an adaptive adversary. Therefore, a natural question arises:

Is there a BB protocol with sub-quadratic communication complexity, secure even in the presence of adaptive corruptions?

It turns out that the answer to the above question is somewhat subtle, and it depends on the precise power of the adaptive adversary. Abraham et al. [ACD+19] recently showed that BB with sub-quadratic communication is possible (assuming trusted setup and standard cryptographic assumptions),

provided that the adversary is *weakly* adaptive; but it is impossible if the adversary is *strongly* adaptive. The difference in power between a weakly adaptive and a strongly adaptive adversary is explained briefly below:

- A weakly adaptive adversary can observe the messages honest nodes send in a round, adaptively corrupt a subset of the nodes (subject to a total corruption budget); and moreover, the newly corrupt nodes can send additional messages of the adversary's choice in the same round. However, the adversary cannot perform "after-the-fact message removal" and retroactively erase the messages a newly corrupt node already sent prior to being corrupt in the same round.

- By contrast, a strongly adaptive adversary can examine the messages honest nodes would have sent in some round, adaptively corrupt a subset of the nodes in the same round, erase the messages they would have sent and make the newly corrupt send arbitrary other messages instead.

We refer the readers to Abraham et al. [ACD+19] to a detailed description of the results.

Chapter 12

Asynchronous Consensus: The FLP Impossibility

So far in this course, we have assumed a *round-based* execution model in which nodes are invoked every round to perform some action, including receiving messages, performing local computation, and sending new messages. Based on this round-based execution model, we consider two types of networks, *synchronous* and *partially synchronous* networks, depending on whether there is an a-priori known bound on the maximum network delay.

We can implement the round-based execution model if nodes have synchronized clocks. If, say, every second is a round, then nodes can register timeout callbacks such that they are invoked every second. Having synchronized local clocks can be a strong assumption depending on the application scenario.

Remark 20. In the partially synchronous model, it is known that this the assumption on having synchronized clocks can be relaxed: if nodes have clocks with bounded drift, it is possible to synchronize their clocks and avoid the drift using a standard clock synchronization procedure [DLS88, CPS18a, CL99].

In this lecture, we will look at a new model called the *asynchronous* model where we remove the assumption for nodes to have local clocks. In an asynchronous consensus protocol, every node can only be invoked upon receiving some message from the network, at which point it performs some computation and decides some messages to send. The node's program is not allowed to register any timeout callbacks. For example, an asynchronous protocol cannot say, if no message is received in three seconds, perform some action (whereas this is allowed in synchronous and partially synchronous protocols).

12.1 Definitions: Asynchronous Consensus and Execution Model

Suppose that there are n nodes, each node starts with an input bit. The n nodes run a weakly valid Byzantine Agreement protocol at the end of which each node outputs a bit. We want the following guarantees:

- *Consistency.* If two honest nodes output b and b' respectively, it must be that $b = b'$.
- *Weak validity.* If all nodes are honest and they all receive the same input bit b, then they must all output b too.
- *Liveness.* All honest nodes output eventually (assuming that messages are delivered eventually).

12.2 Impossibility of Asynchronous, Deterministic Consensus

The question we want to answer is, *can we achieve consensus in such an asynchronous network model?*

In this lecture, we will show that no *deterministic* protocols can accomplish this purpose. Not only so, perhaps somewhat surprisingly at first sight, the impossibility holds even if we are guaranteed that there is at most one node crash during the entire execution (and there is no malicious behavior at all). This result is also known as the FLP impossibility, since it was proven by Fischer, Lynch, and Paterson [FLP85] in 1985. The FLP impossibility result is arguably one of the most famous theorems in distributed computing!

Theorem 14 (Impossibility of deterministic, asynchronous consensus under a single crash fault [FLP85])**.** *There does not exist a deterministic, asynchronous protocol that realizes weakly valid Byzantine Agreement in the presence of one crash fault.*

In the above, "one crash fault" means that at most one node can crash and stop participating altogether, at an unannounced time, during the protocol's execution. Otherwise all nodes follow the honest protocol.

12.3 Proving the FLP Impossibility

The proof we present below is mostly faithful to Fischer, Lynch, and Paterson's original and beautiful proof [FLP85].

Suppose that there exists an asynchronous protocol Π that realizes weakly valid Byzantine Agreement. We will reason about Π below and reach a contradiction.

12.3.1 Terminology

Configuration, event, schedule. A *configuration* defines the state of the execution; it consists of: 1) all nodes' internal states; and 2) an event buffer that stores the set of pending messages to be delivered and their recipients. We assume that each event in the event buffer is of the form $e := (p, m)$ where m denotes a message and p denotes the node that is supposed to receive m. In an asynchronous protocol, a node is invoked only upon receiving a message. Suppose the execution is in configuration C, at this point delivering the event $e := (p, m)$ means that we make node p receive m; and node p may now perform computation, update its internal state (including possibly outputting a bit), and add new events to the event buffer representing messages p wants to send. After these actions, the execution enters a new configuration C'. Henceforth, if e is an event in the event buffer of C, we say that e can be applied or is applicable to C, and we use the notation $C' := e(C)$ to denote that the configuration C' is reached after applying e to C.

A *schedule* σ is a sequence of events.

Valency. Let C be a configuration and let V be the set of output values of configurations reachable from C. C is bivalent if $|V| = 2$. C is univalent if $|V| = 1$, let us say 0-valent or 1-valent according to the corresponding output value. Due to the liveness definition, V cannot be empty.

Recall that the *initial configuration* is determined by all nodes' input bits. A schedule σ from some configuration C is said to be *deciding*, iff after applying σ to C, some node has output a bit.

12.3.2 Proof Roadmap

Suppose that there is an asynchronous, weakly valid BA protocol Π that tolerates a single crash fault. At a very high level, the proof will contain two major steps.

1. First, we show that some initial configuration must be bivalent for Π.
2. Next, we show that one bivalent configuration must lead to yet another.

Now, if both of the above are true, then essentially we can start with a bivalent configuration, and keep extending it into a new bivalent configuration.

We can do this for infinitely long, leading to an infinite execution path that remains bivalent throughout. Thus Π cannot satisfy liveness on this particular execution path, leading to a contradiction.

12.3.3 Existence of a Bivalent Initial Configuration

Lemma 12. *The protocol* Π *must have a bivalent initial configuration.*

Proof. Suppose not. Then Π must have both 0-valent and 1-valent initial configurations by weak validity. Let us call two initial configurations adjacent if they differ only in the input bit of a single node p. Any two initial configurations are joined by a chain of initial configurations, each adjacent to the next. Hence, there must exist a 0-valent initial configuration C_0 adjacent to a 1-valent initial configuration C_1. Let p be the node in whose input value they differ. Now consider some deciding schedule from C_0, denoted σ, which does not involve p. Note that in this schedule, only node p is prevented from receiving messages, i.e., it is as if p has crashed. By liveness of Π under one crash fault, such a deciding schedule σ must exist. Now, σ can be applied to C_1 also, and corresponding configurations in the two runs are identical except for the internal state of the node p. It is easily shown that both runs eventually result in the same output value. However, since C_0 is 0-valent and C_1 is 1-valent, a successor of C_0 cannot have the same output value as a successor of C_1. Thus we have reached a contradiction. □

12.3.4 One Bivalent Configuration Leads to Another

We say that a schedule σ does not involve a node p if no message is delivered to p in σ. The following lemma roughly says that two schedules that do not involve a common node, and both applicable to the same starting configuration, can be applied commutatively, and yet lead to the same configuration.

Lemma 13 (Commutativity)**.** *Suppose that from some configuration* C*, the schedules* σ_1 *and* σ_2 *lead to configurations* C_1 *and* C_2*, respectively. If the sets of nodes involved in* σ_1 *and* σ_2*, respectively, are disjoint, then* σ_2 *can be applied to* C_1 *and* σ_1 *can be applied to* C_2*, and both lead to the same configuration* C_3 *(see Figure 12.1).*

Proof. The result follows at once from the system definition, since σ_1 and σ_2 do not interact. □

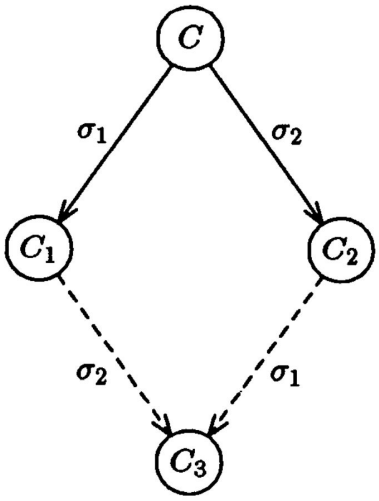


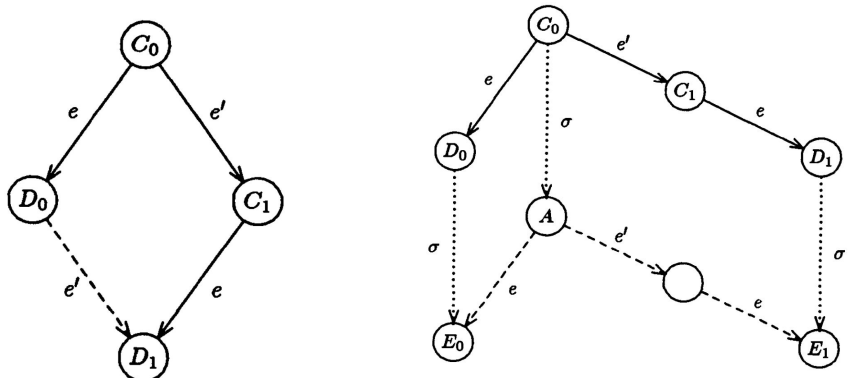
Figure 12.1: Two schedules that do not involve a common node are commutative — see Lemma 13.

This above commutativity lemma will be a key idea we use to prove that "one bivalent configuration leads to another" — let us now prove this. The following lemma says, informally, that if a protocol starts from a bivalent configuration and there is a message e that is applicable to that configuration, then the set of configurations reachable through any sequence of messages where e is applied last contains a bivalent configuration. More intuitively, it says that if you delay a pending message for an arbitrarily amount of time, there will be one configuration in which you receive that message and end up in a bivalent state [flp].

Lemma 14. *Let C be a bivalent configuration of protocol Π, and let $e = (p, m)$ be an event that is applicable to C. Let $\mathfrak{C}$ be the set of configurations reachable from C without applying e, let $\mathfrak{D} = e(\mathfrak{C}) = \{e(E)|E \in \mathfrak{C}\}$ (note that e must be applicable to every configuration in $\mathfrak{C}$). Then, $\mathfrak{D}$ contains a bivalent configuration.*

Proof. Suppose that $\mathfrak{D}$ contains no bivalent configurations, so every configuration $D \in \mathfrak{D}$ is univalent. We proceed to derive a contradiction.

First, we show that $\mathfrak{D}$ must contain both 0-valent and 1-valent configurations. To prove this, let E_b be a b-valent configuration reachable from C where $b \in \{0, 1\}$. Note that for either bit $b \in \{0, 1\}$, E_b must exist since C is bivalent. If $E_b \in \mathfrak{C}$, let $F_b = e(E_b) \in \mathfrak{D}$. Otherwise, e was applied in reach



(a) Case 1 of Lemma 14.

(b) Case 2 of Lemma 14.

Figure 12.2: Graphical illustration of Lemma 14.

E_b and so there exists $F_b \in \mathfrak{D}$ from which E_b is reachable. In either case, F_b is b-valent since F_b is not bivalent (since $F_b \in \mathfrak{D}$ and $\mathfrak{D}$ contains no bivalent configurations), and one of E_b and F_b is reachable from the other. Since $F_b \in \mathfrak{D}$ for $b \in \{0, 1\}$, $\mathfrak{D}$ contains both 0-valent and 1-valent configurations.

Call two configurations neighbors if one results from the other in a single step. By an easy induction, there exist neighbors $C_0, C_1 \in \mathfrak{C}$, such that $D_b = e(C_b)$ is b-valent for $b \in \{0, 1\}$. Without loss of generality, assume $C_1 = e'(C_0)$ where $e' = (p', m')$.

Case 1. If $p' \neq p$, then $D_1 = e'(D_0)$ by Lemma 13. This is impossible, since any successor of a 0-valent configuration is 0-valent. (See Figure 12.2a.)

Case 2. If $p' = p$, then consider any finite deciding schedule σ from C_0 which does not involve p. Let $A := \sigma(C_0)$. By Lemma 13, for either $b \in \{0, 1\}$, σ is applicable to D_b, and it leads to an b-valent configuration $E_b = \sigma(D_b)$. Also by Lemma 13, $e(A) = E_0$ and $e(e'(A)) = E_1$ (see Figure 12.2b). Hence, A is bivalent. But this is impossible since the schedule σ which leads to A is deciding (by assumption), so A must be univalent.

In each case, we reached a contradiction, so $\mathfrak{D}$ contains a bivalent configuration. □

Finishing the proof of Theorem 14. We now show that given the protocol Π, we can construct an infinite schedule that traverses only bivalent states. First, there must be an bivalent initial configuration C according

to Lemma 12. By the liveness of Π, there must exist events in the event buffer, and let e be an event that is applicable to C. By Lemma 14 there is a bivalent configuration C' reachable from C in which e is applied the last. Now, we can apply the same argument to C' to extend the schedule, and this can go on infinitely. This means that there is a schedule that cause nodes to loop forever and not output a decision, and thus Π cannot satisfy liveness.

Discussions. The proof of FLP's impossibility shows that for any asynchronous protocol that preserves consistency and weak validity, there must exist a run in which nodes loop forever without outputting a decision, thus violating liveness. One natural question is why the proof works only for deterministic protocols but not randomized ones. The reason is that in a probabilistic protocol, the never-ending run could be an extremely rare run, i.e., with very high probability, the never-ending run does not happen.

Indeed, we can use randomness to overcome the FLP impossibility and design consensus protocols in the asynchronous setting. We will discuss this in the next lecture.

Does the FLP impossibility suggest that we cannot have practical consensus protocols in the real world? Of course not! Clearly, we have had many successful applications of distributed consensus in the past. To get around this impossibility, we can rely on (somewhat) synchronized clocks and/or randomness. Nonetheless, for decades, the FLP impossibility has provided guidance and served as an important sanity check for designers of consensus protocols.

Chapter 13

A Randomized Asynchronous Consensus Protocol ※

In the previous lecture, we learned that asynchronous *deterministic* consensus is not possible, even when we are promised that there is at most one crash fault. One way to overcome this impossibility is to rely on randomness.

In this lecture, we will learn a randomized asynchronous consensus protocol that is secure in the presence of $f < n/3$ corruptions.

13.1 Assumption: A Common Coin Oracle

Our protocol will make use of a Public-Key Infrastructure and digital signatures; not only so, we introduce a new assumption called a CommonCoin. Imagine that there is a trusted oracle, called CommonCoin, that tosses a random coin for the participants of the consensus protocol whenever needed. More specifically, CommonCoin allows nodes to query a random coin for each epoch number e. Once at least $2n/3$ nodes have queried CommonCoin(e), the oracle flips a random coin b and returns b to every CommonCoin(e) query in the past, and it returns the same b upon receiving every CommonCoin(e) query in the future too. Note that the messages between the nodes and the CommonCoin oracle can also be delayed arbitrarily.

We stress that the coin flips are *unpredictable* in advance: specifically, the adversary cannot predict the outcome of the e-th coin flip until more than $n/3$ honest nodes have called CommonCoin(e).

One might wonder how to realize such a CommonCoin functionality. One straightforward approach is to rely on a trusted service that provides a public random beacon. In fact, it is also known how to realize such a CommonCoin

through either Verifiable Secret Sharing or threshold signatures schemes. For simplicity, we do not plan to go into the details on how to realize CommonCoin; we shall just assume CommonCoin as given in our lecture.

13.2 Randomized Asynchronous Consensus

Recall that in Byzantine Agreement (BA), every node is given an input bit. They would like to agree on a bit through a consensus protocol. We say that an asynchronous protocol realizes BA with probability $1 - \delta$ iff with $1 - \delta$ probability over the choice of the randomized execution, the following properties hold:

- *Consistency.* If two honest nodes output b and b' respectively, it must be that $b = b'$.
- *Validity.* If all honest nodes receive the same input bit b, then they must all output b too.
- *Liveness.* All honest nodes output eventually (assuming that messages are delivered eventually).

Note that here we would like to achieve the standard validity notion; whereas in the previous lecture, we used a weak validity notion for proving the FLP impossibility (since using a weak notion makes the impossibility result stronger).

Protocol. We will present a variant of the protocol by Cachin et al. [CKS05]. We assume that there are n nodes numbered $1, 2, \ldots, n$, and node i's public key is pk_i. The notation $\{m\}_{\mathsf{pk}_i^{-1}}$ denotes the message m along with a signature from node i on m. In the protocol below, a pre-vote means a message of the form $\{\mathtt{pre}, e, b\}_{\mathsf{pk}_i^{-1}}$ signed by some node i whereas a *properly justified* pre-vote means the pre-vote as well as a justification for why the pre-vote was cast. The same terminology is used for main-votes too. The protocol is parametrized with E, i.e., the total number of epochs. As we will show later, the protocol's success probability is related to the parameter E.

A Randomized Asynchronous Consensus Protocol

Epoch 0: Initially, every node i does the following: send an epoch-0 pre-vote of the form $\{\mathtt{pre}, 0, b\}_{\mathsf{pk}_i^{-1}}$ to everyone where b denotes its input bit.

For each epoch $e = 1, 2, 3, \ldots, E$**:** every node i performs the following and moves onto the next epoch:

- **Pre-vote.** If $e = 1$, wait to collect at least $2n/3$ epoch-0 pre-votes from distinct nodes. If among them, more than $n/3$ epoch-0 pre-votes contain the bit b (breaking ties arbitrarily), send the pre-vote $\{\texttt{pre}, e, b\}_{\mathsf{pk}_i^{-1}}$ to everyone and attach a justification that contains a set of at least $2n/3$ epoch-0 pre-votes among which more than $n/3$ are for the bit b.

  Else, if $e > 1$, wait to receive at least $2n/3$ properly justified main-votes of epoch $e-1$ from distinct nodes, and let $b^* := \mathsf{CommonCoin}(e)$.

  - If among them there is a main-vote for $b \in \{0, 1\}$, then send the pre-vote $\{\texttt{pre}, e, b\}_{\mathsf{pk}_i^{-1}}$ to everyone and attach to the pre-vote a justification that contains at least $2n/3$ epoch-$(e-1)$ pre-votes for b (this can be found in the justification for the epoch-$(e-1)$ main-vote for b).
  - Else, it must be that all main-votes received are signed `abstain` votes. In this case, let $b := b^*$ and send the pre-vote $\{\texttt{pre}, e, b\}_{\mathsf{pk}_i^{-1}}$ to everyone; attach to the pre-vote a justification containing at least $2n/3$ epoch-$(e-1)$ `abstain` votes.

- **Main-vote.** Wait to receive at least $2n/3$ properly justified epoch-e pre-votes from distinct nodes, and based on these pre-votes, set v as follows:

$$v = \begin{cases} 0 & \text{if all pre-votes are for 0} \\ 1 & \text{if all pre-votes are for 1} \\ \texttt{abstain} & \text{o.w.} \end{cases}$$

  Now, send to everyone $\{\texttt{main}, e, v\}_{\mathsf{pk}_i^{-1}}$ tagged with a justification for the main-vote. A proper justification contains the set of at least $2n/3$ pre-votes that triggered the main-vote (no need to include the justifications for the pre-votes).

Output. At any time, if at least $2n/3$ main-votes for the same epoch e and the same bit b have been received from distinct nodes, output b and continue participating in the protocol.

13.3 Consistency

We will assume that the signatures are ideal in our proofs.

Fact 1. As long as messages are eventually delivered, honest nodes do not get stuck while waiting to collect messages, i.e., they can always advance epochs eventually.

Proof. Follows from the fact that honest nodes always wait for $2n/3$ messages from distinct nodes before moving onto the next action item, and moreover, CommonCoin always waits for $2n/3$ epoch-e queries before giving everyone answers. Since there are fewer than $n/3$ corrupt nodes, when all honest nodes have completed a common action item , they are guaranteed to move onto the next. □

Henceforth, we use the term "in honest view" to mean "in the union of honest nodes' views".

Lemma 15 (Consistency within an epoch)**.** *The following must be true for every epoch e:*

1. *Suppose $v, v' \in \{0, 1, \texttt{abstain}\}$ and $v \neq v'$. It cannot be that for both v and v', there is at least $2n/3$ epoch-e main-votes in honest view.*
2. *Suppose $b, b' \in \{0, 1\}$ and $b \neq b'$. It cannot be that for both b and b', there is at least $2n/3$ epoch-e pre-votes in honest view.*
3. *If there is a collection of at least $2n/3$ epoch-e main-votes for the bit $b \in \{0, 1\}$ in honest view, then, some honest node has seen at least $2n/3$ epoch-e pre-votes for b.*

Exercise 27. The above Lemma 15 has a simple proof. Please prove it yourself. Hint: similar to the proof of Lemma 7 of Chapter 7.

Lemma 16 (Consistency across epochs)**.** *If there is a collection of at least $2n/3$ epoch-e main-votes for the bit $b \in \{0, 1\}$ in honest view, then for any $e' > e$, all honest nodes' main-votes must be for the bit b.*

Proof. Consider the epoch $e + 1$, we claim that

1. only pre-votes for the same bit b can be properly justified; and
2. all honest nodes will indeed cast a properly justified pre-vote for b.

To see the first claim, note that an epoch-$(e+1)$ pre-vote for $1-b$ can only be justified either with at least $2n/3$ epoch-e `abstain` votes, or with at least $2n/3$ epoch-e pre-votes for $1-b$. Both are ruled out by the fact that there are at least $2n/3$ epoch-e main-votes for b in honest view, and by Lemma 15. To see the second claim, notice that by the first claim, all epoch-$(e+1)$ pre-votes in honest view must be for the bit b and thus an honest node can only cast a main-vote for b in epoch $e+1$. By Fact 1, every honest node will indeed cast a main-vote in epoch $e+1$.

The second claim above means that there is a collection of at least $2n/3$ properly justified epoch-$(e+1)$ main-votes for the bit b in honest view. Therefore by induction, the proof extends to all epochs $e' > e$.

□

Theorem 15 (Consistency)**.** *The above protocol satisfies consistency.*

Proof. Let e be the smallest epoch such that there exists a bit $b \in \{0,1\}$ and at least $2n/3$ epoch-e main-votes for b in honest view. Now, if some honest node outputs $b' \in \{0,1\}$, it must be due to observing at least $2n/3$ main-votes of the same epoch e' for the bit b'. By the definition of e, it must be that $e' \geq e$. If $e' > e$, by Lemma 16, all honest nodes must cast main-votes for b in epoch e', and now by Lemma 15, it must be that $b' = b$. If $e' = e$, it holds by Lemma 15 that $b' = b$. □

13.4 Liveness

An epoch $e+1$ is said to be *lucky* iff:

(a) either there is a collection of at least $2n/3$ epoch-e pre-votes from distinct nodes for some bit $b \in \{0,1\}$ in honest view and $\mathsf{CommonCoin}(e+1)$ returns the same b; or

(b) there isn't a collection of at least $2n/3$ epoch-e pre-votes from distinct nodes for either bit b in honest view.

Lemma 17. *In a lucky epoch $e+1$, let $b = \mathsf{CommonCoin}(e+1)$. There cannot be a properly justified epoch-$(e+1)$ pre-vote for $1-b$ in honest view. Moreover, all honest nodes must cast pre-votes and main-votes for the bit b in epoch $e+1$.*

Proof. A proper justification for an epoch-$(e+1)$ pre-vote for $1-b$ must contain at least $2n/3$ epoch-e pre-votes for $1-b$. This is not possible if epoch $e+1$ is type-(a) lucky because in this case there is at least $2n/3$ epoch-e

pre-votes for b and recall also Lemma 15 holds. This is also not possible if epoch $e+1$ is type-(b) lucky by definition.

By Fact 1, all honest nodes must cast pre-votes and main-votes for the bit b in epoch $e+1$. □

Lemma 18. *For any $e > 1$, the probability that both epochs e and $e+1$ are unlucky is at most $1/2$, even when conditioned on whether epochs before e are lucky or not.*

Proof. The following statements hold even when conditioned on whether epochs before e are lucky or not.

Note that $\mathsf{CommonCoin}(e)$ can only be decided when more than $n/3$ honest nodes have called $\mathsf{CommonCoin}(e)$ — let S denote a set of more than $n/3$ honest nodes that are the first to call $\mathsf{CommonCoin}(e)$. Each $i \in S$ must have collected at least $2n/3$ main-votes of epoch $e-1$.

- **Case 1:** Among all these main-votes collected by S, there is at least one main-vote is for a bit $b \in \{0,1\}$. This main-vote must carry a set of at least $2n/3$ epoch-$(e-1)$ pre-votes for b as justification. By Lemma 15, there cannot be $2n/3$ epoch-$(e-1)$ pre-votes for $1-b$ in honest view. Only at this moment, is the coin $\mathsf{CommonCoin}(e)$ actually flipped, and there is $1/2$ probability that it is not equal to b.

- **Case 2:** All these main-votes collected by S are `abstain` votes. In this case, the set S will cast a pre-vote for $b^* := \mathsf{CommonCoin}(e)$ in epoch e. It is not hard to see that there cannot be at least $2n/3$ epoch-e pre-votes for $1-b^*$ in honest view then. Note also at this moment (i.e., when $\mathsf{CommonCoin}(e)$ is first flipped), no honest node has called $\mathsf{CommonCoin}(e+1)$ yet. When $\mathsf{CommonCoin}(e+1)$ is flipped in the future, it has $1/2$ probability of being b, and if it turns out to be b^*, it means that epoch $e+1$ will be lucky.

□

Theorem 16 (Liveness)**.** *Recall that E is the total number of epochs. The above protocol achieves liveness with $1 - 2^{\lfloor (E-1)/2 \rfloor}$ probability.*

Proof. Lemma 17 shows that once there is a lucky epoch, all honest nodes will eventually output. Lemma 18 shows that the probability that there isn't a lucky epoch is at most $1 - 2^{\lfloor (E-1)/2 \rfloor}$. The theorem now follows in a straightforward manner. □

Exercise 28. Prove that the above protocol satisfies validity.

13.5 Termination

So far in our protocol, all nodes continue to participate forever even after outputting a bit. A standard "early termination" technique can allow nodes to terminate after outputting: basically, when outputting a bit b, a node i also sends a special message $\{\texttt{terminate}, b\}_{\mathsf{pk}_i^{-1}}$ attached with a justification consisting of at least $2n/3$ main-votes for b from the same epoch that triggered i to output b. This special message can serve as node i's pre-vote for b and main-vote b in all epochs.

13.6 Additional Exercises

Exercise 29. Prove that no asynchronous protocol can realize Byzantine Agreement with probability more than $2/3$ under $n/3$ or more corruptions. This means that our protocol earlier in this section achieves *optimal resilience*.

Exercise 30. Suppose that I want the protocol to achieve Byzantine Agreement with $1 - \mathsf{negl}(\lambda)$ probability for a negligible function $\mathsf{negl}(\cdot)$ and some security parameter λ. How should I set the number of epochs E as a function of λ?

Chapter 14

Bitcoin and Nakamoto's Blockchain Protocol

Back in the 1970s, the study of distributed consensus was motivated by the need to build reliable aircraft control systems replicated on multiple computers [WLG+89]. Later on, distributed consensus protocols became widely deployed in companies such as Google and Facebook. These companies need to replicate their mission-critical infrastructure such as Google Wallet or Facebook Credit, and thus the challenge of achieving consistency naturally arises. In all of these classical scenarios, consensus is typically deployed on a *small scale*, involving three to dozens of machines. Participation is *closed*, i.e., only a preconfigured, known set of nodes can join the protocol — such environments are often referred to as *permissioned* environments.

Bitcoin [Nak08] came around in 2009 and gained popularity rapidly. As the Wikipedia page explains it [wik]:

> "Bitcoin is a cryptocurrency. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries. ... Bitcoin was invented in 2008 by an unknown person or group of people using the name Satoshi Nakamoto [Nak08] and started in 2009 when its source code was released as open-source software".

At the core of Bitcoin is a *blockchain* protocol (defined in Chapter 6) that allows a set of distributed nodes to agree on an ever-growing, linearly-ordered log of transactions. In fact, the term "blockchain" was popularized due to Bitcoin.

Bitcoin is not just an empirical success, it is also a scientific breakthrough! Specifically, Bitcoin's blockchain protocol, often called Nakamoto's

blockchain [Nak08], is the first to demonstrate the feasibility of reaching consensus in a *permissionless* environment. In a permissionless environment, anyone is free to join the consensus protocol at any time. Since there is no a-priori knowledge of the identities of the participants, participants must communicate through *unauthenticated channels*.

To reach consensus in such a permissionless environment, one big challenge is the so-called "Sybil attack". Since the communication channel is unauthenticated, anyone can impersonate anyone else; and a single machine can also impersonate many machines, e.g., in an attempt to outnumber the honest players and disrupt the consensus. Exactly because of this reason, the classical insight had always been that consensus is impossible in such a permissionless environment without even authenticated communication channels. Indeed, with some effort, one can formalize this intuition and mathematically prove that absent any other assumptions, consensus is impossible in such a permissionless environment [PS17b].

Of course, the mathematical impossibility did not stop Bitcoin. Nakamoto's blockchain protocol circumvented this impossibility by leveraging Proof-of-Work (PoW). The idea is that players need to solve computational puzzles to cast votes. Roughly speaking, a player's voting power is proportional to its computational power. Moreover, the blockchain protocol guarantees consistency and liveness as long as *the majority of the mining power in the system is honest*.

In this lecture, we will describe how Nakamoto's blockchain works, and prove its security.

14.1 Nakamoto's Ingenious Idea in a Nutshell

Block format and notations. In Nakamoto's protocol, each honest node maintains a blockchain denoted chain at any point of time. The first block in the blockchain, denoted $\mathsf{chain}[0]$, is a canonical block called the *genesis*. Every other block $\mathsf{chain}[i]$ where $i > 0$ is of the format $\mathsf{chain}[i] := (h_{-1}, \eta, \mathsf{txs}, h)$, containing the hash of the previous block denoted h_{-1}, a puzzle solution η, a payload string txs which may contain a set of transactions to be confirmed, and a hash h of the present block. We will use the notation:

- We use $\mathsf{chain}[-\ell]$ to denote the ℓ-th to last block in chain. For example $\mathsf{chain}[-1]$ denotes the last block and $\mathsf{chain}[-2]$ denotes the second to last block, and so on;
- We use $\mathsf{chain}[:\ell]$ to denote the prefix $\mathsf{chain}[0..\ell]$.

- We use $\mathsf{chain}[:-\ell]$ to denote the prefix of chain except for the last ℓ blocks.
- We use $|\mathsf{chain}|$ to denote the length of chain, i.e., the total number of non-genesis blocks in chain.
- We often use the notation "$\_$" to denote a wildcard field that we do not care about.

Remark 21 (Bitcoin's genesis block)**.** Bitcoin's genesis block embedded the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". According to Bitcoin's Wiki page [gen], "this was probably intended as proof that the block was created on or after January 3, 2009, as well as a comment on the instability caused by fractional-reserve banking.".

Mining. Given some blockchain chain, let its last block be $(\_, \_, \_, h^*)$. To "mine" a new block off chain, let txs be the outstanding transactions — a miner would try random puzzle solutions $\eta \in \{0,1\}^\lambda$ and check if

$$\mathsf{H}(h^*, \eta, \mathsf{txs}) < D_p$$

where H denotes a Proof-of-Work (PoW) oracle (implemented as a hash function), D_p is some appropriate difficulty parameter. In Bitcoin, D_p is chosen such that in expectation it takes all miners combined 10 minutes to mine a new block. We will elaborate on how to choose D_p later in Section 14.3. If some puzzle solution η produces a hash outcome that is smaller than D_p, then the tuple $(h^*, \eta, \mathsf{txs}, \mathsf{H}(h^*, \eta, \mathsf{txs}))$ forms a valid block extending from chain; and chain is often said to be the parent chain of the newly mined block. Nodes propagate any new block they have mined.

Roughly speaking, we assume that the PoW function H behaves like a random function, and there is no algebraic shortcut one can exploit when evaluating H. In other words, there is no better way to find puzzle solutions than brute-force trying many different solutions. This is why mining is a computationally expensive process.

Because each block contains a hash of the previous block, the entire chain is bound together by the cryptographic hash. In other words, assuming that no hash collisions are found, then a block uniquely binds to its entire prefix.

Longest chain. One of the most beautiful ideas in Nakamoto's construction is the longest chain idea. Miners always try to mine a block off the *longest chain* it has seen. At any time, all but the last K blocks in the longest chain are considered *final*. In other words, if a transaction tx is embedded K blocks deep in the present longest chain (i.e., at least K blocks away from

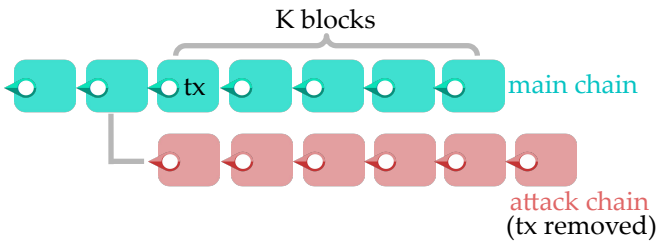


Figure 14.1: The adversary aims to undo transaction tx by mining a longer fork than the main chain. If the adversary has only minority of the mining power, statistically speaking, it is extremely unlikely that the adversary can win the race against the main chain. Let K be how deep tx is embedded in the main chain: as K increases, our confidence in tx's finality increases very sharply.

the end), we may treat the transaction as finalized. Moreover, the larger the K, the more confident we are about tx's finality.

To intuitively understand why, it helps to look at Figure 14.1. Suppose tx is contained in the block $\mathsf{chain}[-K]$ where chain denotes the longest chain observed thus far. Imagine that tx corresponds to the payment the adversary made to a Ferrari dealer to purchase a Ferrari. Once the car has been shipped to the adversary, the adversary may want to undo tx and reverse its payment. Can the adversary succeed in such an attack?

Informally, to undo tx, the adversary would have to mine an attack fork off some prefix $\mathsf{chain}' \preceq \mathsf{chain}[:-K]$; not only so, the attack fork must be longer than the main chain for it to win — but keep in mind that the main chain is growing too. Thus the adversary must, within a fixed time window, mine at least K more blocks than the honest nodes, to win this race. If the adversary controls only minority of the mining power, it is statistically unlikely that it can succeed; and further, the larger the K, the exponentially smaller the adversary's chance of success!

The above is not a formal proof why Nakamoto's blockchain preserves consistency, but we will formally prove it in Chapter 17.

14.2 Nakamoto's Blockchain: Formal Description

We will formally describe a stripped-down version of the full Nakamoto consensus protocol implemented in Bitcoin. One simplification we make is

to pretend that the total mining power in the system is known and fixed. In the Bitcoin's implementation, this assumption is not true, and therefore the protocol relies on a difficulty adjustment mechanism to adjust the difficulty of the computational puzzles being solved based on how much mining power is present in the recent past. Our description will omit this difficulty adjustment mechanism, and the resulting simplified protocol is often called the "barebone" Nakamoto's blockchain.

We will assume a synchronous network where honest nodes' messages must be delivered within at most Δ delay to honest recipients (see Chapter 6).

Modeling PoW puzzles. We will use $(\mathsf{H}, \mathsf{H.ver})$ to denote a PoW scheme where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ is the PoW's work function; and $\mathsf{H.ver} : \{0,1\}^* \to \{0,1\}^\lambda$ is the corresponding verification function. Recall that H requires the caller to expend work and evaluate a hash function, and $\mathsf{H.ver}$ is used to check if a purported puzzle solution is correct.

Without loss of generality, we may assume that all nodes have equal computational power and we use n to denote the total number of nodes — if a node has more computational power, it can be viewed as multiple nodes. This way, saying that "the majority of nodes are honest" equates to saying that "the majority of the computational power is honest". Every node can only query the PoW function H at a bounded rate. We may assume that in every round, each node can invoke H at most once — this is without loss of generality since we can always rename the time it takes to evaluate H as one round. However, we do not impose any limit on calls to the verification function $\mathsf{H.ver}$.

Remark 22. In Nakamoto's protocol, $\mathsf{H.ver}$ is only called when messages (specifically, blocks) are received from the network. In practice, since the network has limited bandwidth, $\mathsf{H.ver}$ is called significantly fewer times than H (even when an adversary may flood the network with fake blocks). This is why we do not charge calls to $\mathsf{H.ver}$.

Barebone Nakamoto's blockchain. Nodes always try to mine blocks off the *longest* valid chain they have observed thus far. Once a block is mined, the miner propagates it to others. At any time, the longest chain a node has observed with the last K blocks removed is considered as the current finalized log. We now describe the protocol more formally.

Although not explicitly noted below, we make an *implicit echoing* assumption: whenever a node hears a fresh message from the network previously

unseen or receives a new transaction as input, it echos the message or transaction to everyone else. This assumption makes sure that if any honest node sees a message in round r, then all honest nodes will have observed it by round $r + \Delta$.

Nakamoto's blockchain

- Nodes that are newly spawned start with initial chain containing only a special genesis block: $\mathsf{chain} := (0, 0, \bot, \mathsf{H}(0, 0, \bot))$.
- Whenever a node hears a message chain' from the network, if incoming message chain' is a valid blockchain and it is longer than its current local blockchain chain, replace chain by chain'. We define what it means for a chain to be valid later. Checking the validity of chain' can be done using only $\mathsf{H.ver}$ queries.
- In every round, try to mine a new block off the longest chain seen so far (denoted chain) as follows. Let txs be the outstanding transactions observed so far that are not contained in the current chain. Now parse $\mathsf{chain}[-1] := (\_, \_, \_, h_{-1})$, pick a random solution $\eta \in \{0,1\}^\lambda$, and issue query $h = \mathsf{H}(h_{-1}, \eta, \mathsf{txs})$. If $h < D_p$, then append the *newly mined* block $(h_{-1}, \eta, \mathsf{txs}, h)$ to chain and send $\mathsf{chain}||(h_{-1}, \eta, \mathsf{txs}, h)$ to everyone. The parameter D_p determines how difficult it is to mine a block, and how to choose D_p will be explained in Section 14.3 below.
- At any time, a node's *finalized log* is defined to be $\mathsf{chain}[: -K]$, i.e., the longest chain observed so far removing the last K blocks. We need to choose K to be sufficiently large such that the probability of breaking consistency is extremely small (see Theorem 17 for a rigorous statement).

Valid chain. We say a block $\mathsf{chain}[i] = (h_{-1}, \eta, \mathsf{txs}, h)$ is *valid with respect to a predecessor block* $\mathsf{chain}[i-1] = (h'_{-1}, \_, \_, h')$ if the following conditions hold: $h_{-1} = h'$, $h = \mathsf{H}(h_{-1}, \eta, \mathsf{txs})$, and $h < D_p$. A chain of blocks chain is *valid* iff:

1. $\mathsf{chain}[0] = (0, 0, \bot, \mathsf{H}(0, 0, \bot))$ is the genesis block, and
2. for all $i \in [\ell]$ where $\ell := |\mathsf{chain}|$, $\mathsf{chain}[i]$ is valid with respect to $\mathsf{chain}[i-1]$.

14.3 Choosing the Mining Difficulty Parameter

How should we choose the mining difficulty parameter? In Bitcoin, the difficulty parameter is chosen such that on average, all miners combined take 10 minutes to mine the next block. Of course, 10 minutes seem awfully long, especially given that one also has to wait for a transaction to be embedded K blocks deep for it to be confirmed. In practice, many consider $K = 6$ to be secure enough — this means it could easily take *an hour* for a transaction to confirm! From a confirmation delay perspective, it seems desirable to make the puzzles less difficult such that blocks are confirmed more frequently — but would this be safe?

It turns out that we cannot arbitrarily lower the puzzles' difficulty; doing so could break the consistency of the consensus protocol. One way to think of the matter is the following: since the network delay can be up to Δ, whenever a new block is mined, there is a Δ gap in which the new block is being propagated on the network to the honest nodes, and during this Δ gap, the honest nodes are not doing any useful work[1]! On the other hand, the adversary may not need to suffer from the same Δ delay (e.g., the adversary controls a mining farm where blocks are transmitted over dedicated links). This gives the adversary an advantage when it tries to mine an attack fork like in Figure 14.1. One can think of the advantage in terms of the honest mining power that is *discounted* by the network's delay Δ.

To understand how much honest mining power is discounted by Δ, we give an *informal* back-of-the-envelope calculation — this calculation is only to convey intuition and it should *not* be interpretted as a formal proof. Let p be the probability that a single node mines a block in any fixed round. Suppose that there are n nodes, 51% of which are honest. The probability that the honest nodes combined can mine a block in a round is roughly $1 - (1-p)^{0.51n} \approx 0.51pn \ll 1$. The expected number of rounds till a new honest block is mined is roughly $\frac{1}{0.51pn}$. Now, it takes Δ rounds to propagate the block. Suppose that all of the honest nodes' work is wasted during the Δ rounds, then roughly speaking, every $\frac{1}{0.51pn}$ rounds, we end up wasting Δ rounds. In this sense the discount ratio is roughly

$$\frac{\frac{1}{0.51pn}}{\frac{1}{0.51pn} + \Delta} = \frac{1}{1 + 0.51pn\Delta} \approx 1 - 0.51pn\Delta$$

[1] Let us ignore the tiny probability that honest nodes mine two consecutive blocks during the Δ interval.

Exercise 31. With the above back-of-the-envelope calculation, the term $(1 - 0.51pn\Delta) \cdot (0.51n)$ can be regarded as the *discounted honest mining power*. Now, suppose that $0.49n > (1 + \epsilon) \cdot (1 - 0.51pn\Delta) \cdot (0.51n)$, i.e., the corrupt mining power exceeds the discounted honest mining power by some constant $\epsilon \in (0, 1)$ margin. Describe how the adversary can succeed in mining an attack fork like in Figure 14.1 with probability almost 1, despite the fact that it controls only 49% of the mining power.

The above exercise suggests that for Nakamoto's consensus protocol to maintain consistency, more precisely speaking, we need not just honest majority in mining power, but a slightly more stringent condition, that is,

> *the honest mining power, even when discounted by the network delay Δ, must exceed the corrupt mining power!*

Setting the puzzles to be more difficult makes the discount factor smaller.

Formal requirements on the mining difficulty. We will formally articulate a set of requirements on the mining difficulty — under this set of parameters, we shall be able to formally state and prove the security properties of Nakamoto's blockchain.

Our goal is to set the D_p parameter, called the mining difficulty parameter, in the protocol formally described in Section 14.2. D_p can be chosen in the following way:

- first, choose an appropriate probability $p \in (0, 1)$ as described below;
- once p is fixed, we choose D_p such that the probability that any player mines a block in a round is $p \in (0, 1)$. This can be achieved[2] by setting $D_p := p \cdot 2^\lambda$ such that for all (h, txs), $\Pr_\eta[\mathsf{H}(h, \eta, \mathsf{txs}) < D_p] = p$.

We choose the parameter $p \in (0, 1)$ such that it satisfies the following conditions where n is the total number of nodes, and Δ denotes the maximum network delay:

1. $\nu := 2pn\Delta < 0.5$; and
2. Honest mining power, even when discounted by the network delay, must outnumber corrupt mining power by an appropriate constant margin.

[2]For simplicity, we shall assume that $p \cdot 2^\lambda$ is an integral number.

Formally, let $\phi \in (0,1)$ be an arbitrarily small constant, and let ρ denote the fraction of corrupt nodes. We require that

$$\frac{1-\rho}{\rho} \geq \frac{1+\phi}{1-\nu} \tag{14.1}$$

Note that the second requirement can be equivalently interpreted as $(1-\rho)\cdot(1-\nu) \geq (1+\phi)\cdot\rho$, where $1-\rho$ is the fraction of honest mining power and ρ is the fraction of corrupt mining power. The term $1-\nu = 1-2pn\Delta$ is the discount in the honest mining power. The constant 2 here differs from our earlier back-of-the-envelope calculation — but it turns out that this is the constant needed for the formal proofs we present in Chapter 17[3].

If $\Delta = 0$, i.e., all messages are received instantly without any delay, then no discount would be incurred — in this case, Equation (14.1) simply boils down to requiring that the honest mining power exceed the corrupt mining power by an arbitrarily small constant margin $\phi \in (0,1)$. The larger the network delay Δ, the more disadvantageous it is to the honest nodes, and thus the more honest fraction we will need to ensure consistency.

14.4 Properties of Nakamoto's Blockchain

To state the formal guarantees attained by Nakamoto's blockchain, we shall assume that Nakamoto's blockchain protocol is executed for $\mathsf{poly}(\lambda)$ number of rounds where λ is the security parameter. This is a reasonable assumption for cryptographic protocols, where we typically assume that the adversary is polynomially bounded in the security parameter λ. We say that $\mathsf{negl}(\lambda)$ is a *negligible function* if for any fixed polynomial function $p(\lambda)$, there exists λ_0 such that for any $\lambda > \lambda_0$, $\mathsf{negl}(\lambda) < 1/p(\lambda)$. In other words, a negligible function is one that drops off very sharply as we increase the security parameter λ. If a protocol's failure probability is negligible in the security parameter λ, then by increasing the security parameter λ a little, we can make the failure probability extremely small.

Nakamoto's blockchain, when instantiated with appropriate parameters stated in Section 14.3, satisfies the following theorem, which we shall prove in Chapter 17. Below we will state the theorem formally first, and then we will give intuitive explanations for each property.

Theorem 17. *Suppose that $K = \omega(\log \lambda)$ and let $\epsilon \in (0,1)$ be an arbitrarily small constants. There exists a negligible function $\mathsf{negl}(\cdot)$, such that with $1-$*

[3]It is possible that the constant 2 can be tightened with tighter proofs, but doing so is beyond the scope of this course.

$\mathsf{negl}(\lambda)$ *probability over the choice of the randomized execution of Nakamoto's blockchain, the following properties hold:*

- **Chain growth lower bound.** Let $\alpha := (1-\rho)np$ denote the expected number of honest nodes that mine a block in each round. For any round r_0 and any duration $t \geq \frac{K}{\alpha}$, let chain^{r_0} be some honest node's longest chain in round r_0 and let chain^{r_0+t} be some honest node's longest chain in round $r_0 + t$ (the two honest nodes can be the same or different). It must be that

$$|\mathsf{chain}^{r_0+t}| - |\mathsf{chain}^{r_0}| \geq (1-\epsilon)(1-2pn\Delta)\alpha t$$

- **Chain quality.** Let chain be the longest chain of some honest node sometime during the protocol execution: it must be that for any K consecutive blocks $\mathsf{chain}[j..j+K]$ in this longest chain, more than $\mu := 1 - \frac{1+\epsilon}{1+\phi}$ fraction of the blocks are mined by honest nodes.

- **Consistency.** Let chain^r denote some honest node's longest chain in round r and let chain^t denote some honest node's longest chain in round $t \geq r$ (note that the two honest nodes can be the same or different). It must hold that

$$\mathsf{chain}^r[:-K] \preceq \mathsf{chain}^t$$

  where $\mathsf{chain} \preceq \mathsf{chain}'$ means that the former is a prefix of the latter or they are the same chain.

Below we elaborate on these properties and provide some intuition for each of them.

14.4.1 Chain Growth Lower Bound

Intuitively, chain growth lower bound says that honest nodes' chains must grow steadily over time. Of course, the corrupt nodes could completely stop mining, and therefore we can only guarantee that honest nodes' chains grow at a rate proportional to the honest nodes' total mining power which is α. However, keep in mind that the network has maximum delay Δ, and every time a block is mined, Δ rounds can be wasted just transmitting the block to others. For this reason, the actual chain growth rate we can guarantee is only $(1-2pn\Delta)\alpha$, where the honest mining power α is further discounted by the factor $1-2pn\Delta$.

Why do we care about chain growth? Because chain growth is necessary for achieving liveness (see Section 6), i.e., transactions submitted must be

included in honest nodes' finalized logs fairly soon. It is not hard to see why chain growth is necessary for liveness, but it turns out that chain growth alone is not sufficient for ensuring liveness. For example, it could be that although the blockchain grows, every block is mined by corrupt players, and corrupt players may not include outstanding transactions in their mined blocks or they may selective drop certain transactions. For this reason, we also need chain quality which ensures that every now and then, some block mined by honest nodes makes its way into the blockchain.

14.4.2 Chain Quality

Chain quality says that in every window of consecutive K blocks in honest nodes' longest chains, it must be that more than $\mu := 1 - \frac{1+\epsilon}{1+\phi}$ fraction of them are mined by honest nodes. Chain quality is necessary for ensuring liveness, that is, transactions submitted must be included in honest nodes' finalized logs fairly soon. If (non-zero) chain quality holds, intuitively, it means that every now and then, an honest block makes its way into the blockchain. Since honest miners always include all outstanding transactions in the blocks they mine, liveness can be ensured (see also Exercise 33).

Does Nakamoto's blockchain achieve ideal chain quality? Although liveness only needs non-zero chain quality, it is natural to ask if the protocol provides *fairness*. If Nakamoto's protocol were completely "fair", the fraction of honest blocks ought to be $1 - \rho$. Henceforth the expression $1 - \rho$ is referred to as "ideal chain quality". Does Nakamoto's protocol provide ideal chain quality?

To understand this, let us try to gain some intuition about the chain quality parameter μ in Theorem 17. For simplicity, let us assume that $\Delta = 0$, and moreover, equality is taken in Equation (14.1). In this case, we simply have that

$$\frac{1-\rho}{\rho} = 1 + \phi$$

Since we can take $\epsilon \in (0, 1)$ to be very small, for a back-of-the-envelope calculation, we simply ignore ϵ. In this case, the chain quality parameter in Theorem 17 would roughly be

$$\mu \approx 1 - \frac{1}{1+\phi} = \frac{1-2\rho}{1-\rho}$$

For example,

- Suppose that the fraction of honest mining power $1 - \rho = 2/3$. Then, the chain quality μ guaranteed by Theorem 17 is roughly $1/2$, whereas ideal chain quality would be $2/3$.
- Suppose that the fraction of honest mining power $1 - \rho$ is sightly greater than $1/2$. Then, the chain quality μ guaranteed by Theorem 17 is slightly greater than 0, whereas ideal chain quality ought to be $1/2$.

  Is this mismatch due to looseness of the theorem, or is it that Nakamoto's blockchain is inherently not fair?

It turns out that Nakamoto's blockchain is *not fair*! A well-known attack, called the *selfish mining* attack [mtg, ES14], shows that if a coalition with roughly $\rho < 1/2$ fraction of mining power deviates from the honest protocol, it can, in the best-case scenario, control roughly $\rho/(1-\rho)$ fraction of the blocks! We will further explain the selfish mining attack in Chapter 15.

14.4.3 Consistency

The consistency property in Theorem 17 is stated w.r.t. nodes' longest chains, but not w.r.t. nodes' finalized logs. Recall that in Nakamoto's consensus, at any time, a node's finalized log is its longest chain but chopping off the trailing K blocks. If we want to prove that Nakamoto's blockchain realizes the blockchain abstraction defined earlier in Chapter 6, we need to prove the consistency property defined in Chapter 6, which is stated w.r.t. nodes' finalized logs. In other words, we want to prove that, with extremely high probability over the choice of the randomized execution, the following should hold: if LOG_i^r and LOG_j^t are the finalized logs of two honest nodes i and j in rounds r and t respectively, it must be that either $\mathsf{LOG}_i^r \preceq \mathsf{LOG}_j^t$ or $\mathsf{LOG}_i^r \succeq \mathsf{LOG}_j^t$. We leave this as a homework exercise.

Exercise 32. Recall that in the Nakamoto's blockchain, a node's finalized log is always its longest chain but removing the trailing K blocks. Suppose that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the consistency property stated in Theorem 17 is satisfied.

Prove that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the following holds: if LOG_i^r and LOG_j^t are the finalized logs of two honest nodes i and j in rounds r and t respectively, it must be that either $\mathsf{LOG}_i^r \preceq \mathsf{LOG}_j^t$ or

$\mathsf{LOG}_i^r \succeq \mathsf{LOG}_j^t$.

14.4.4 Liveness

In Exercise 32, we proved that Nakamoto's blockchain satisfies consistency as defined in Chapter 6. To prove that Nakamoto's protocol realizes a blockchain, we also need to show that it satisfies the liveness property as defined in Chapter 6.

Exercise 33. Suppose that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, chain growth lower bound and chain quality as stated in Theorem 17 are satisfied.

Prove that with $1 - \mathsf{negl}(\lambda)$ probability over the choice of the randomized execution of Nakamoto's blockchain, the following holds: suppose that an honest node receives some transaction tx as part of its input in some round r, then, by round $r + \Theta(K/\alpha + \Delta)$, tx must appear in every honest node's finalized log.

Combining Exercise 32 and 33, we may conclude that Nakamoto's protocol indeeed satisfies the blockchain abstraction defined earlier in Chapter 6.

Chapter 15

The Selfish Mining Attack and Incentive Compatibility

Traditionally, consensus was deployed by a single organization like Google or Facebook to replicate their mission-critical computing infrastructure (e.g., Facebook Credit, Google Wallet, etc.). In such settings, consensus is used to achieve *fault tolerance*, and incentive for participation is a non-issue. Further, if nodes fail or get compromised with somewhat independent probability, *honest majority* would be a very reasonable assumption.

Excitingly, with cryptocurrencies such as Bitcoin and Ethereum, consensus moved to a large-scale, decentralized setting. Many new challenges arose in this new setting, such as scalability, incentives, and governance. One particularly intriguing issue is incentives. In Bitcoin and Ethereum's Proof-of-Work-based consensus, it is rather costly for nodes to participate and contribute to maintaining a global public ledger. To incentivize participation, Bitcoin gives rewards to the miner of each block. Roughly speaking, for a miner to receive mining rewards, it needs to include a public key pk inside any block it mines, so that the rewards for mining the block can be credited to pk. In Bitcoin, the per-block mining reward includes two parts:

1. *Block reward*: when Bitcoin started first, a fixed block reward of 50 bitcoins was given to the miner of each block. After every 210,000 blocks are mined (approximately every 4 years), the block reward halves and will keep on halving until the block reward per block becomes 0 (approximately by year 2140).

2. *Transaction fees*: every transaction in Bitcoin can specify a fee to pay to the miner that includes the transaction.

Since miners make some money for mining each block, to maximize one's payoff, a selfish miner should want to mine as many blocks as possible. When everyone participates honestly in the protocol, it is not hard to see that all hashpower is "equal", that is, a miner with ρ fraction of the mining power gains ρ fraction of the rewards in expectation. However, selfish miners need not follow the honest protocol. If by deviating, they can increase their own payoff, they will have strong incentives to deviate!

It turns out that the Nakamoto's consensus protocol is indeed vulnerable to a well-known incentive attack called the *selfish-mining* attack. The attack was first suggested on the Bitcoin forum [mtg], and later Eyal and Sirer extended the analysis in an elegant work [ES14]. Alarmingly, these analyses showed that, if all blocks had equal rewards, then a miner who wields $1/3$ of the mining power could, in the best case, reap close to $1/2$ of the mining rewards. Similarly, a miner who wields 49% of the mining power could, in the best case, reap 96% of the rewards!

The big problem seems to be the following: if everyone has incentives to deviate, then *honest majority* is no longer an assumption that one can take for granted; and without honest majority, Nakamoto's consensus will no longer provide the desired consistency and liveness guarantees!

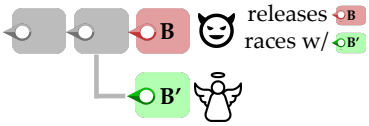
It is therefore desirable to have protocols that are *incentive compatible*. What we want, is for the protocol to not just incentivize participation, but also incentivize honest participation. In other words, we want honest behavior to be an *equilibrium* that reinforces itself. If all other miners' are behaving honestly, it should be that everyone's best response is to behave honestly too!

In this lecture, we will first describe the selfish mining attack for Nakamoto's consensus, we will then informally survey a work called by Pass and Shi [PS17a] which shows how to introduce a simple tweak to Nakamoto's protocol, and prove that the new protocol, called Fruitchain, defends against selfish mining attacks and achieves incentive compatibility.

15.1 The Selfish Mining Attack

When honest miners mine a block, they are supposed to release the block immediately and send it to others. In a selfish mining attack [mtg, ES14], when a selfish miner mines a block $\mathbf{B}$, it withholds the block $\mathbf{B}$ until some honest miner also mines a block, denoted $\mathbf{B}'$, which is at the same length as $\mathbf{B}$. When this happens, the selfish miner immediately releases the withheld block $\mathbf{B}$. If it can make $\mathbf{B}$ transmit faster than $\mathbf{B}'$, it can potentially convince

(a) A selfish miner mines a block **B** and withholds it.



(b) When an honest miner mines an equal-length block $\mathbf{B}'$, the selfish miner immediately releases **B** and races with $\mathbf{B}'$.

Figure 15.1: The selfish mining attack.

other honest nodes to mine off **B** rather than $\mathbf{B}'$ from now on. In this attack, effectively, the selfish miner's block **B** successfully erased honest nodes' work in mining $\mathbf{B}'$. In fact, the selfish miner can perform the same attack with every block it mines. For every block the selfish miner mines, it can use the block to erase roughly one block worth of work from honest nodes.

We can do an informal back-of-the-envelope calculation to gain more intuition. For simplicity, let's assume that the network's delay $\Delta = 0$. Suppose that the selfish miner controls $\rho < 1/2$ fraction of the mining power. During a very long window in which a total of T blocks are mined, the number of blocks mined by the selfish miner is roughly ρT whereas the number of blocks mined by honest nodes is roughly $(1-\rho)T$. Recall that every block the selfish miner mines, it can erase one block mined by honest nodes. Therefore, in the final blockchain, there would only be $(1-\rho)T - \rho T = 1 - 2\rho T$ blocks mined by honest nodes during this time window; and the total number of blocks (during this time window) that make their way into the blockchain is $1 - 2\rho T + \rho T = 1 - \rho T$. Hence, the fraction of the blocks in the chain mined by honest nodes (i.e., the chain quality) is roughly,

$$\frac{1-2\rho}{1-\rho}$$

The fraction of blocks controlled by the selfish miner is therefore $1 - \frac{1-2\rho}{1-\rho}$. For example, if $\rho = 1/3$, then the selfish miner controls roughly $1/2$ of the blocks; and if $\rho = 49\%$, then the selfish miner controls roughly 96% of the blocks.

Interestingly, the above calculation also matches the chain quality guarantee stated in Theorem 17 of Chapter 14 (assuming that $\Delta = 0$). This means that the above simple attack is in fact optimal for Nakamoto's consensus.

Discussion: tie-breaking. In the above selfish mining attack, we assumed that in the event of equal-length forks (e.g., **B** and $\mathbf{B}'$), the protocol breaks

ties by picking the block that arrives first. Furthermore, we assumed that the selfish miner can always successfully race against honest block $\mathbf{B}'$ and propagate its own withheld block $\mathbf{B}$ ahead of $\mathbf{B}'$. If the selfish miner colludes with a network relay whose job is to deliver blocks to miners, such an attack would be feasible. In practice, if the selfish miner does not have so much influence over the network transmission, its advantage will be discounted.

One naïve idea is to introduce a better tie-breaking rule in Nakamoto's consensus. For example, a natural idea is to pick a *random* fork if equal-length forks are encountered. Indeed, random tie-breaking mitigates the advantage of the selfish miner; but even with random tie-breaking, one can show that a selfish miner that controls a constant fraction of the mining power can improve its gains by a constant (multiplicative) factor with the aforementioned selfish mining attack.

Discussion: has there been a real-world selfish mining attack? It is natural to ask if a selfish mining attack has taken place on real-world cryptocurrencies such as Bitcoin and Ethereum. To the best of my limited knowledge, no one seems to know a conclusive answer, partly due to the difficulty of measuring decentralized systems like Bitcoin. It is quite possible that an attack has taken place stealthily. On the other hand, for Bitcoin, big stake-holders now dominate the mining game — to profit in mining, one would need to invest significantly in dedicated ASICs capable of fast hash computations. Since the big stake-holders have vested interest in the health and longer-term prosperity of Bitcoin, it is also possible that this deters them from launching selfish mining attacks.

15.2 Fruitchain: an Incentive-Compatible Blockchain ※

No matter whether selfish mining has actually taken place in the real world, it seems reassuring if we could design the consensus protocol to discourage selfish mining attacks. In other words, the consensus protocol should incentivize honest behavior, and that honest behavior should be an equilibrium that reinforces itself.

A recent work by Pass and Shi, called Fruitchain [PS17a], aims to achieve this goal. Fruitchain is obtained by making a somewhat small tweak to Nakamoto's protocol. Pass and Shi show that Fruitchain provably defends against any form of selfish-mining attacks. More specifically, *an adversary controlling minority of the mining power, cannot increase its gains by more*

than a $\delta \in (0,1)$ factor, no matter how it deviates from the prescribed protocol — moreover, this holds even when δ is an arbitrarily small constant. In other words, Fruitchain achieves a coalition-resistant Nash equilibrium.

At a very high level, in Fruitchain, any mining attempt by calling the hash function H can result in either a *fruit* or a *block*. Typically, it should be much easier to mine fruits than blocks. In Fruitchain, the fruits contain transactions whereas the blocks contain fruits. The very high-level idea is that although an adversary can perform a selfish-mining attack and erase honest nodes' effort in mining blocks, it cannot erase honest nodes' efforts in mining fruits; and in Fruitchain, the mining rewards are distributed to the fruits rather than blocks. All honest fruits mined, can be picked up by some honest block in the near future since non-zero chain quality holds for the underlying blockchain. Moreover, we require a fruit to refer to some recently stabilized block, and this provides a timestamping mechanism for measuring how "fresh" the fruit is. To prevent an adversary from accumulating many fruits and releasing them altogether to outnumber honest fruits during some time-frame, we require that only relatively fresh fruits can be included in a block. This makes sure that fairness holds for every sufficiently long time-window, and not in aggregate across the entire duration of the protocol — note that this is important because rewards are distributed to miners periodically and not just at the very end of time.

We refer the curious reader to the Fruitchain work [PS17a] to find out further details.

Chapter 16

A Simple, Deterministic Longest-Chain-Style Protocol

Our discussions in the past couple of lectures have focused on Nakamoto's blockchain. Our next big goal is to prove Nakamoto's blockchain secure, i.e., we eventually would like to prove Theorem 17 of Chapter 17. As we have explained, Nakamoto's blockchain departs significantly from classical consensus protocols, and arguably one of the most novel ideas is the *longest chain* idea.

In this lecture, we take a slight detour: instead of directly diving into the formal proofs for Nakamoto's blockchain, we will start with a warmup that helps to illustrate why the "longest chain" idea works. We will describe a simple deterministic longest-chain-style protocol [Shi19a] that is inspired by Nakamoto, and we will go over its proof which is also simple. Unlike Nakamoto's consensus, our deterministic longest-chain protocol is described for a permissioned setting where everyone's public key is well-known; further, the protocol secures only against fewer than $n/3$ corruptions. The proofs for this protocol can be viewed as a simplified, deterministic variant of the analysis of Nakamoto's blockchain. The proofs for the latter involve stochastic reasoning and are somewhat more complicated — we shall dedicate the next chapter to formally analyzing Nakamoto's blockchain.

16.1 Deterministic Longest-Chain-Style Consensus Protocol

To make things simplest possible, rather than constructing a blockchain, we describe a protocol for a one-shot abstraction, namely, weakly valid Byzantine Broadcast. There are n nodes numbered $0, 1, \ldots, n-1$ respectively, and suppose that all of their public keys $\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_{n-1}$ are well-known. Without loss of generality, we may assume that node 0 is the designated sender and it wants to broadcast a bit to everyone else. We assume a *synchronous* network where honest nodes' messages can be delivered to honest recipients in the immediate next round.

A weakly valid Byzantine Broadcast is defined in almost the same way as in Chapter 3, except that we weaken the validity requirement and only require the following:

> *Weak validity:* if *all* nodes are honest, then everyone outputs the designated sender's input bit.

Our protocol goes round by round, and in each round i, the node i mod n is eligible to vote on either bit. Nodes always pick the bit that has gained more votes so far to vote on — henceforth this bit is said to be the *more popular* bit. At the end of the protocol, everyone outputs their more popular bit.

To formalize the protocol, we first define a few notions below.

Leader. In every round $r \in \{0, 1, \ldots, n-1\}$, node r is the leader. Thus each round has a unique leader.

Valid votes. A valid vote for a bit $b \in \{0, 1\}$ is a tuple (b, r, σ) where $r \in \{0, 1, \ldots, n-1\}$ is a round number (also called the vote's *timestamp*), and σ is a valid signature under pk_r for the tuple (b, r). This means that in round r, only node r's vote counts as valid.

More popular bit. At any time, a node's more popular bit is the bit for which it has seen more valid votes, breaking ties arbitrarily. Two votes with the same (b, r) but different σ are treated as the same vote.

Protocol. We now present the protocol below.

A deterministic longest-chain-style protocol

- *Round 0:* let b be the sender's input bit. The sender signs $(b, 0)$, and let σ be the resulting signature. The sender sends the vote $(b, 0, \sigma)$ to everyone.
- *For each round* $r = 1, 2, \ldots, n-1$*:*
  - Receive all messages from the network and discard every vote whose timestamp is r or greater (discarded votes do not contribute to the node's view in the protocol).
  - If the current node is the leader of this round, perform the following steps (otherwise skip). Let $\tilde{b} \in \{0, 1\}$ be the more popular bit. Now, sign $(\tilde{b}, r)$ and let σ be the resulting signature. Send to everyone the resulting vote $(\tilde{b}, r, \sigma)$, as well as all valid votes the node has seen for $\tilde{b}$ so far. If there are multiple most popular bits, break ties arbitrarily.
- At the beginning of round n: every node outputs its more popular bit $\tilde{b} \in \{0, 1\}$ breaking ties arbitrarily.

Resemblance to longest chain. In our protocol, in every round, the round's leader votes on the more popular bit seen thus far. This is akin to Bitcoin's idea of having nodes vote on the longest chain seen so far. Here, to make the protocol and proofs even simpler, we don't do the chaining like in Nakamoto's protocol.

Remark 23. The protocol above has linear round complexity and secures only against fewer than $n/3$ corruptions. In a purely theoretical sense, this seems to be a step backwards in comparison with Dolev and Strong's protocol in Chapter 3. However, we are interested in formally analyzing this protocol, because its proofs nicely capture the core ideas in the analysis of Nakamoto's blockchain, but removes the probabilistic reasoning in the latter.

16.2 Analysis

We formally analyze this simple, longest-chain-style protocol. We will assume the ideal signature model in the analysis.

Lemma 19 (Vote growth lemma)**.** *Suppose that the number of corrupt nodes is strictly less than $n/3$. At the beginning of round n, every honest node's*

more popular bit has strictly more than $2n/3$ votes.

Proof. A round r is said to be an honest-leader round if its leader ($r \mod n$) is honest. We prove that if round r is the i-th honest-leader round, then at the beginning of round $r+1$, every honest node must have observed at least i votes for its more popular bit.

We can prove this by induction. The base case is obvious: the statement holds trivially for the 0-th honest-leader round.

Now, we prove the inductive step. Suppose that the statement holds for every $i \leq k-1$, we now show that it holds for the k-th honest-leader round too. Let t denote the k-th honest-leader round. By the induction hypothesis, in round t, the honest leader ($t \mod n$) must have seen at least $k-1$ votes for its more popular bit $\tilde{b}$ at the time. The honest leader ($t \mod n$) now creates a new vote on $\tilde{b}$ and shares all votes on $\tilde{b}$ it has seen with others as well as the new one. Thus, by the beginning of the next round, every honest node must have seen at least k votes for its more popular bit[1].

The lemma follows by observing that there are strictly more than $2n/3$ honest-leader rounds. □

Theorem 18 (Consistency)**.** *At the beginning of round n, if some honest node has seen strictly more than $2n/3$ votes for the bit $\tilde{b}$, then no honest node can have seen strictly more than $2n/3$ votes for the bit $1-\tilde{b}$.*

Note that if the above statement holds, then the protocol satisfies consistency.

Proof. Suppose that the theorem is not true, i.e., there are strictly more than $2n/3$ votes for both bits in the union of the honest nodes' views. The total number of distinct votes must exceed $4n/3$.

Note that an honest-leader round increases the total number of votes by exactly 1, and a corrupt-leader round increases the total number of votes by at most 2. Let $f < n/3$ denote the number of corrupt nodes; then the total number of votes can be at most $n - f + 2f = n + f < 4n/3$. Thus we have reached a contradiction. □

Theorem 19 (Weak validity)**.** *If all nodes are honest, then they all output the input bit of the designated sender.*

Proof. The proof should be straightforward. Suppose that everyone is honest and the designated sender receives the input bit b. In round 0, the designated

[1]Note that at the beginning of the next round, an honest node's more popular bit may not be $\tilde{b}$; but since every honest node will have seen k votes on $\tilde{b}$, for $1-\tilde{b}$ be the more popular bit, it must have gained k votes or more.

sender casts a vote for b. Henceforth, in every round, the leader of the round will create a new vote for b and distribute all votes thus far including the new one. At the end of the protocol, everyone will output b. □

Resemblance to the proof for Nakamoto's blockchain. In the above consistency proof, the core idea is to argue that every honest node's more popular bit must accumulate votes fairly quickly, such that at the end of the protocol, every honest node's more popular bit must have many votes — we call this property **vote growth**. To show consistency, we argue that it cannot be the case that both bits gain votes that quickly, because every honest node casts exactly 1 vote when it is the leader, and every corrupt node can cast at most 2 votes when it is the leader. This imposes an upper bound on the total number of distinct votes there can be.

In our analysis of the Nakamoto's protocol in the next lecture, we will use a similar strategy to prove consistency: we will show that in all likelihood, honest nodes chains grow steadily over time — this property will be called **chain growth**. To prove consistency, at a very high level, we will argue that in all likelihood, it cannot be the case that two parallel chains both grow quickly since the total computation power in the system is bounded.

In this sense, understanding the simple protocol in this lecture will help understand the analysis for Nakamoto's blockchain.

16.3 Additional Exercises

Exercise 34. Describe the simplest modification you can think about to the above protocol, such that it achieve the strong validity notion, that is, if the designated sender is honest, everyone should output its input bit. Prove that your modified protocol realizes Byzantine Broadcast as defined in Chapter 3.

Chapter 17

Analysis of Nakamoto's Blockchain ※

In this lecture, we will present a formal analysis of the barebone Nakamoto's blockchain protocol. Specifically we will prove Theorem 17 of Chapter 14.

17.1 Ideal-World Protocol

Nakamoto's blockchain relies on a PoW hash function. Because the hash function's outcome is random and sufficiently long, it should be the case that except with negligible probability, the adversary is never able to predict future hash values, or cause hash collisions. Assuming that this is the case, we may abstract away the details of the hash function, and imagine that mining is performed in an idealized world. This ideal-world protocol captures the core stochastic process we care about, and our proofs will be presented for this ideal-world protocol. For convenience, we refer to the protocol in Chapter 14 as the real-world protocol.

In the ideal world, we shall imagine that every block contains only the payload string txs; and the other fields h_{-1}, η, and h (see Chapter 14) are now abstracted away. Instead of calling the PoW hash function H to mine blocks, all nodes mine blocks by calling a so-called "ideal functionality" henceforth denoted $\mathcal{F}_{\text{tree}}$. One can think of $\mathcal{F}_{\text{tree}}$ as a trusted party that internally maintains a valid tree of blocks; initially, tree contains only the genesis block. $\mathcal{F}_{\text{tree}}$ answers two types of requests from nodes:

- Upon receiving $\texttt{mine}(\mathsf{chain}, \mathsf{txs})$: $\mathcal{F}_{\text{tree}}$ checks if chain is a valid blockchain in tree. If so, $\mathcal{F}_{\text{tree}}$ flips a coin that comes up heads with probability p.

If the coin flip is successful, $\mathcal{F}_{\text{tree}}$ records $\mathsf{chain}||\mathsf{txs}$ in the set tree, and returns success.

- Upon receiving $\texttt{verify}(\mathsf{chain})$: $\mathcal{F}_{\text{tree}}$ checks if chain is a valid blockchain in tree; if so, return true; else return false.

Honest nodes play according to the following rules in the idealized protocol. Just like the real-world protocol in Chapter 14, here we also make an *implicit echoing* assumption, that is, whenever a node hears a fresh message from the network previously unseen or receives a new transaction as input, it echos the message or transaction to everyone else. This assumption makes sure that if any honest node sees a message in round r, then all honest nodes will have observed it by round $r + \Delta$.

- Every node maintains the longest blockchain seen thus far denoted chain.
- In every round, every honest node first receives all incoming messages on the network. For any received message chain': If $\mathcal{F}_{\text{tree}}.\texttt{verify}(\mathsf{chain}') = 1$ and chain' is longer than the current local chain, then let $\mathsf{chain} := \mathsf{chain}'$ and broadcast chain'.
- Let txs be the outstanding transactions observed so far but not contained in chain. Now, query $\mathcal{F}_{\text{tree}}.\texttt{mine}(\mathsf{chain}, \mathsf{txs})$: if this mining query is successful, the node propagates $\mathsf{chain}||\mathsf{txs}$ to everyone and replaces its chain with $\mathsf{chain} := \mathsf{chain}||\mathsf{txs}$.
- The *finalized log* at any point of time is chain but removing the last K blocks.

We will prove Theorem 17 of Chapter 14 for the above ideal-world protocol; and we claim that the same guarantees extend to real-world protocol of Chapter 14. The fact that this should hold is not surprising; nonetheless formally proving this statement requires a bit of (somewhat tedious) work — and we refer the reader to Pass et al. [PSS17] for a formal statement and proof relating the security of the ideal-world protocol to the security of the real-world protocol.

Garay et al [GKL15] and Pass et al. [PSS17] first presented formal analyses of Nakamoto's consensus. Pass and Shi [PS17b] presented a simplified proof for pedagogical purposes. Our proof below is mostly faithful that of Pass and Shi [PS17b], but with simpler notations.

17.2 Notations

In our proofs below, we adopt the same notations as Chapter 14, and we often use $\mathsf{negl}(\cdot)$ to denote an appropriate negligible function. Any time we use the phrase "with negligible probability", unless otherwise stated, it means negligible in the security parameter λ.

Recall that $\alpha := p \cdot (1-\rho)n$ denotes the expected number of honest nodes that mine a block in each round; and we shall use $\beta := p \cdot \rho n$ to denote the expected number of corrupt nodes that mine a block in each round.

17.3 Convergence Opportunities

We now define a useful pattern called convergence opportunities which we shall later use in both our chain growth lower bound proof as well as consistency proof. Intuitively, a convergence opportunity is a Δ-period of silence in which no honest node mines a block, followed by a round in which a single honest node mines a block, followed by another Δ-period of silence in which no honest node mines a block. We formalize this notion below.

Convergence opportunity. Given an execution, we say that $[T-\Delta, T+\Delta]$ is a convergence opportunity iff

- For any $t \in [\max(0, T-\Delta), T)$, no honest node mines a block in round t;
- A single node honest mines a block in round T;
- For any $t \in (T, T+\Delta]$, no node honest in round t mines a block.

Henceforth let $N_H := (1-\rho)n$ denote the number of honest nodes. Let T denote the round in which a single honest node mines a block during a convergence opportunity. For convenience, we often use T to refer to the convergence opportunity. We say that a convergence opportunity T is contained within a window $[t' : t]$ if $T \in [t' : t]$.

Henceforth, let $\mathbf{C}[t' : t]$ be a random variable denoting the number of convergence opportunities contained within the window $[t' : t]$.

Many convergence opportunities. We now show that convergence opportunities happen sufficiently often.

Lemma 20 (Number of convergence opportunities)**.** *For any positive constant η and any κ that is a super-logarithmic function in λ, except with*

some $\mathsf{negl}(\lambda)$ *probability over the choice of the execution, the following holds: for any* $t_0, t_1 \geq 0$ *such that* $t := t_1 - t_0 > \frac{\kappa}{\alpha}$*, we have that*

$$\mathbf{C}[t_0 : t_1] > (1-\eta)(1-2pn\Delta)\alpha t$$

where $\alpha := p \cdot (1-\rho)n$ *denotes the expected number of honest nodes that mine a block in each round.*

Proof. It suffices to prove the lemma for any fixed choice of t_0 and t — if we can do so, the lemma then follows by taking a union bound over polynomially many choices of t_0 and t. We now focus on the coins $\mathcal{F}_{\text{tree}}$ flips for honest nodes upon their `mine` queries — henceforth we refer to these coins as honest coins for short.

- Let $\mathbf{X}$ denote the total number of heads in all the honest coins during $[t_0, t_1]$. Due to the Chernoff bound, for any constant $0 < \epsilon < 1$, it holds that
$$\Pr[\mathbf{X} < (1-\epsilon) \cdot \alpha t] \leq \exp(-\Omega(\alpha t))$$
Henceforth, let $L := (1-\epsilon) \cdot \alpha t$ for a sufficiently small constant ϵ.

- Let $\mathbf{Y}_i = 1$ iff after the i-th heads in the honest coin sequence during $[t_0, t_1]$, there exists a heads in the next $N_H\Delta$ coin flips. Notice that all of the $\mathbf{Y}_i$'s are independent — to see this, another way to think of $\mathbf{Y}_i$ is that $\mathbf{Y}_i = 0$ iff the i-th coin flip and the $(i+1)$-th coin flip are at least $N_H\Delta$ apart from each other.

  Let $\mathbf{Y} := \sum_{i=1}^{L} \mathbf{Y}_i$. We have that
$$\mathbf{E}[\mathbf{Y}] \leq (1-(1-p)^{N_H\Delta}) \cdot L \leq pN_H\Delta \cdot L = \alpha\Delta L$$
  By Chernoff bound, it holds that for any $\epsilon_0 > 0$,
$$\Pr[\mathbf{Y} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\Omega(L)) = \exp(-\Omega(\alpha t))$$
  More concretely, the inequality above arises from the Chernoff bound (see Section 2.5 of Chapter 2); there are 2 cases:

  - If $\delta := \frac{\epsilon_0}{\alpha\Delta} < 1$, we have that $\Pr[\mathbf{Y} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\delta^2\alpha\Delta L/3) = \exp(-\frac{\epsilon_0^2 L}{3\alpha\Delta}) \leq \exp(-\frac{\epsilon_0^2 L}{3}) = \exp(-\Omega(L))$. In the above, the step $\exp(-\frac{\epsilon_0^2 L}{3\alpha\Delta}) \leq \exp(-\frac{\epsilon_0^2 L}{3})$ follows because $\alpha\Delta < 2pn\Delta < 1$ by our assumption.

– If $\delta := \frac{\epsilon_0}{\alpha\Delta} \geq 1$, we have that $\Pr[\mathbf{Y} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\delta\alpha\Delta L/3) = \exp(-\epsilon_0 L/3)$.

- Let $\mathbf{Z}_i = 1$ iff before the i-th heads in the honest coin sequence during $[t_0, t_1]$, there exists a heads in the previous $N_H\Delta$ coin flips. Similarly as before, all of the $\mathbf{Z}_i$'s are independent. Let $\mathbf{Z} := \sum_{i=1}^{L} \mathbf{Z}_i$. We have that

$$\mathbf{E}[\mathbf{Z}] \leq (1 - (1-p)^{N_H\Delta}) \cdot L \leq pN_H\Delta \cdot L = \alpha\Delta L$$

By the Chernoff bound, it holds that for any $\epsilon_0 > 0$,

$$\Pr[\mathbf{Z} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\Omega(L)) = \exp(-\Omega(\alpha t))$$

- Observe that for any fixed execution,

$$\mathbf{C}[t_0 : t_1] \geq \mathbf{X} - \mathbf{Y} - \mathbf{Z}$$

Recall that our parameter choices (see Section 14.3 of Chapter 14) imply that $\alpha\Delta \leq pn\Delta < \frac{1}{4}$. For any execution where the aforementioned relevant bad events do not happen, we have that for any $\eta > 0$, there exist sufficiently small positive constants ϵ_0 and ϵ such that the following holds:

$$\begin{aligned}
&\mathbf{X} - \mathbf{Y} - \mathbf{Z} \geq (1 - 2\alpha\Delta - 2\epsilon_0)L \\
=&(1 - 2\alpha\Delta - 2\epsilon_0) \cdot (1 - \epsilon) \cdot \alpha t \geq (1 - \eta)(1 - 2\alpha\Delta) \cdot \alpha t \\
\geq&(1 - \eta)(1 - 2pn\Delta) \cdot \alpha t
\end{aligned}$$

The proof concludes by observing that there are at most $\exp(-\Omega(\alpha t)) = \exp(-\Omega(\kappa))$ fraction of bad executions that we could have ignored in the above.

□

The above lemma bounds the number of convergence opportunities for any fixed window. By taking a union bound, we can conclude that except for a negligible probability mass of bad executions, in all good executions, it must hold that any sufficiently long window has many convergence opportunities.

17.4 Chain Growth Lower Bound

To prove the chain growth lower bound, we observe that for any fixed execution, whenever there is a convergence opportunity, the shortest honest chain must grow by at least 1 (see Fact 2). Since earlier, we proved that except with negligible probability over the choice of the execution, there are many convergence opportunities, it naturally follows that honest chains must grow rapidly. We now formalize this intuition.

Fact 2. For any fixed execution, any t_0, any $t_1 \geq t_0$, and any honest chains chain^{t_0} and chain^{t_1} in rounds t_0 and t_1 respectively, it holds that

$$\mathbf{C}[t_0 + \Delta : t_1 - \Delta] \leq |\mathsf{chain}^{t_1}| - |\mathsf{chain}^{t_0}|$$

Proof. By definition, if t is a convergence opportunity in the execution, then the shortest honest chain at the end of round $t + \Delta$ must be longer than the longest honest chain at the beginning of round $t - \Delta$. The remainder of the proof is straightforward. □

Lemma 21 (Chain growth lower bound)**.** *For any positive constant ϵ', and any κ that is a super-logarithmic function in λ, except with some $\mathsf{negl}(\lambda)$ probability over the choice of the execution, the following holds: for any t_0 and any $t \geq \frac{\kappa}{\alpha}$, let chain^{t_0} be any honest chain at time t_0 and let chain^{t_0+t} be any honest chain at time $t_0 + t$, then*

$$|\mathsf{chain}^{t_0+t}| - |\mathsf{chain}^{t_0}| > (1 - \epsilon')(1 - 2pn\Delta)\alpha t$$

Proof. It suffices to prove the above lemma for any fixed t_0 and t since if so, we can take a union bound over the polynomially many choices of t_0 and t. Ignore the negligible probability mass of executions where bad events pertaining to Lemma 20 take place. For every remaining good execution, due to Fact 2 and Lemma 20, it holds that for every positive constant ϵ', there is a sufficiently small positive constant ϵ such that for sufficiently large λ and thus sufficiently large $\kappa(\lambda)$,

$$\begin{aligned}
&|\mathsf{chain}^{t_0+t}| - |\mathsf{chain}^{t_0}| \\
>&(1 - \epsilon)(1 - 2pn\Delta)\alpha(t - 2\Delta) \\
=&(1 - \epsilon)(1 - 2pn\Delta)\alpha t - 2(1 - \epsilon)(1 - 2pn\Delta)\alpha\Delta \\
\geq&(1 - \epsilon')(1 - 2pn\Delta)\alpha t
\end{aligned}$$

where the last inequality is due to the fact $\alpha\Delta < 2pn\Delta < 0.5$, and moreover $\alpha t = \Theta(\kappa)$. □

17.5 Chain Quality

Intuitively, we will prove chain quality by comparing the number of adversarially mined blocks with the honest chain growth lower bound. If corrupt nodes mine fewer blocks than the minimum honest chain growth, we can thus conclude that there cannot be too many corrupt blocks in an honest node's chain. We formalize this intuition below. Below, if an honest node called $\mathcal{F}_{\text{tree}}.\texttt{mine}(\mathsf{chain}[:-2], \mathsf{chain}[-1])$ and the query was successful, we say that $\mathsf{chain}[-1]$ is mined by an honest node (or an honest block). Otherwise, we say that $\mathsf{chain}[-1]$ is mined by the adversary (or is an adversarial block).

Fact 3 (Total block upper bound)**.** For any positive constant ϵ and any κ that is a super-logarithmic function in λ, except with some $\mathsf{negl}(\lambda)$ probability over the choice of the execution, the following holds: for any r and t such that $np(t-r) \geq \kappa$, the total number of blocks successfully mined during $(r, t]$ by all nodes (honest and corrupt alike) is upper bounded by $(1+\epsilon)np(t-r)$.

Proof. For any fixed choice of r and t, as long as $np(t-r) \geq \kappa$, the above statement holds by a straightforward application of the Chernoff bound. The fact then holds by applying a union bound over all possible choices of r and t. □

Upper bound on adversarial blocks. Given a fixed execution, let $\mathbf{A}[t_0 : t_1]$ denote the number of blocks mined by corrupt nodes during the window $[t_0 : t_1]$; let $\mathbf{A}^t$ denote the *maximum* number of adversarially mined blocks in any t-sized window.

Recall that $\beta := p \cdot \rho n$ denotes the expected number of corrupt nodes that mine a block in each round.

Fact 4 (Upper bound on adversarially mined blocks)**.** For any constant $0 < \epsilon < 1$, for any κ that is a super-logarithmic function in λ, except with $\mathsf{negl}(\lambda)$ probability over the choice of the execution, the following holds: for any $t \geq \frac{\kappa}{\beta}$, $\mathbf{A}^t \leq (1+\epsilon)\beta t$.

Proof. It suffices to prove that for any fixed t_0, for any positive constant ϵ, except with negligible probability, it holds that $\mathbf{A}[t_0 : t_0 + t] \leq (1+\epsilon)\beta t$ — if we can show this, the rest of the proof follows by taking a union bound over the choice of t_0. To prove the above for any fixed t_0, it suffices to apply the Chernoff bound in a straightforward manner. □

Lemma 22 (Chain quality)**.** *For any positive constant ϵ and any κ that is a super-logarithmic function in λ, except with $\mathsf{negl}(\lambda)$ over the choice of the execution, the following holds for $\mu := 1 - \frac{1+\epsilon}{1+\phi}$: for any honest chain and any consecutive $K \geq \kappa$ blocks $\mathsf{chain}[j+1..j+K]$ blocks in chain, at least μ fraction of these K blocks are mined by honest nodes.*

Proof. Consider a fixed execution. Let r be any round, let i be any node honest in round r. Consider an arbitrary honest chain $\mathsf{chain} := \mathsf{chain}_i^r$ belonging to i in round r, and an arbitrary sequence of K blocks $\mathsf{chain}[j+1..j+K] \subset \mathsf{chain}_i^r$, such that $\mathsf{chain}[j]$ is not adversarial (either an honest block or genesis); and $\mathsf{chain}[j+K+1]$ is not adversarial either (either an honest block or $\mathsf{chain}[j+K]$ is end of chain_i^r). Note that if a sequence of blocks is not sandwiched between two honest blocks (including genesis or end of chain), we can always expand the sequence to the left and right to find a maximal sequence sandwiched by honest blocks (including genesis or end of chain). Such an expansion will only worsen chain quality.

As we argued above, without loss of generality we may assume that $\mathsf{chain}[j+1..j+K]$ is sandwiched between two honest blocks (or genesis/end-of-chain). By definition of the ideal-world protocol, all blocks in $\mathsf{chain}[j+1..j+K]$ must be mined between r' and $r'+t$, where r' denotes the round in which the honest (or genesis) block $\mathsf{chain}[j]$ was mined, and $r'+t$ denotes the round in which $\mathsf{chain}[j+K+1]$ is mined (or let $r'+t := r$ if $\mathsf{chain}[j+K]$ is end of chain_i^r).

We ignore the negligible probability mass of executions where bad events related to chain growth lower bound, adversarial block upper bound, or total block upper bound take place.

- Now, due to chain growth lower bound, for any positive constant ϵ_0, we have that

$$t < \frac{K}{(1-\epsilon_0)(1-2pn\Delta)\alpha}$$

- Due to total block upper bound (Fact 3), it holds that $t \geq \Theta(\frac{\kappa}{np})$. Due to the adversarial block upper bound (Fact 4), for any positive constant $\epsilon'' > 0$, there exist sufficiently small positive constants ϵ' and ϵ_0, such

that

$$\begin{aligned}&\mathbf{A}[r' : r' + t] \leq \mathbf{A}\left[r' : r' + \frac{K}{(1-\epsilon_0)(1-2pn\Delta)\alpha}\right]\\ \leq &\frac{(1+\epsilon')\beta K}{(1-\epsilon_0)(1-2pn\Delta)\alpha} \leq \frac{(1+\epsilon')(1-2pN\Delta)K}{(1-\epsilon_0)(1-2pn\Delta)(1+\phi)}\\ \leq &\frac{(1+\epsilon'')K}{1+\phi}\end{aligned}$$

- Therefore, the fraction of honest blocks in this length K sequence is lower bounded by

$$1 - \frac{1+\epsilon''}{1+\phi}$$

□

17.6 Consistency

Fact 5 (Adversary must expend work to deny a convergence opportunity)**.** Consider a fixed execution: let t denote a convergence opportunity in which a single honest node mines a block denoted B^* at length ℓ. It holds that for any honest chain chain in round $t' \geq t + \Delta$, chain must be at least ℓ in length; moreover, if $\mathsf{chain}[\ell]$ is either mined by a corrupt node, or $\mathsf{chain}[\ell] = \mathsf{B}^*$.

Proof. By the definition of a convergence opportunity, no honest node will mine blocks at length ℓ after $t + \Delta$, and no honest node could have mined a block at length ℓ before $t - \Delta$ since otherwise the honest block mined during the convergence opportunity must be at length at least $\ell + 1$. Finally, since B^* is the only honest block mined during $[t - \Delta, t + \Delta]$, it holds that there is no other honest block at length ℓ in the execution. □

Lemma 23 (Consistency)**.** *For any κ that is a super-logarithmic function in λ, except with negligible probability over the choice of the execution, the following holds: for any round r and any round $t \geq r$, let chain^r be any honest chain in round r and let chain^t be any honest chain in round t; then it must hold that*

$$\mathsf{chain}^r[: -\kappa] \preceq \mathsf{chain}^t$$

Proof. Suppose for the sake of reaching a contradiction that $\mathsf{chain}^r[: -\kappa]$ is not a prefix of chain^t. Let $\mathsf{chain}^r[: -\ell]$ be the longest common prefix of chain^r and chain^t where $\ell > \kappa$. Let $\mathsf{chain}^r[: i] \preceq \mathsf{chain}^r[: -\ell]$ be the longest

prefix that ends at an honestly mined block, i.e., $\mathsf{chain}^r[i]$ is the first honest block to the left of $\mathsf{chain}^r[-\ell]$ (and including $\mathsf{chain}^r[-\ell]$); and let $s-1$ be the round in which $\mathsf{chain}^r[i]$ was mined. It holds that all blocks in $\mathsf{chain}^r[i+1:]$ and $\mathsf{chain}^t[i+1:]$ must be mined in or after round s.

Henceforth, let $\tau := r - s$. By total block upper bound, it must be that $\tau > \frac{\kappa}{2pn}$, which is large enough to make sure that our failure probabilities later will be negligibly small.

Case 1: when $t - r \leq \frac{\phi}{2}\tau$. Observe that all convergence opportunities that come in or after round s must be at length greater than i and moreover they must be at different lengths. Combining this observation and Fact 5, it must be the case that $\mathbf{C}[s : r - \Delta] \leq \mathbf{A}[s : t]$, since otherwise, there must be an honest block B mined during a convergence opportunity between $[s, r-\Delta]$, and B must appear in both chain^r and chain^t. Below we prove that except with negligible probability over the choice of the execution, it must be that $\mathbf{C}[s : r - \Delta] > \mathbf{A}[s : t]$ — if we can do so, then we reach a contradiction, and thus we can conclude the proof.

Below we ignore the negligible probability mass of executions where relevant bad events take place. By Lemma 20, for any positive constant ϵ_c, it holds that[1]

$$\mathbf{C}[s : r - \Delta] > (1 - \epsilon_c)(1 - 2pn\Delta)\alpha(\tau - \Delta)$$

By Fact 4, for any positive constant ϵ_a, it holds that

$$\mathbf{A}[s : t] < (1 + \epsilon_a)\beta \cdot (1 + \frac{\phi}{2})\tau$$

Thus for any positive constants ϕ, and as long as $0 < 2pn\Delta < 0.5$, there exist sufficiently small constants $\epsilon_c, \epsilon_a, \epsilon_1$ such that the following holds for sufficiently large κ:

$$\mathbf{C}[s : r - \Delta] > (1 - \epsilon_c)(1 - \nu)\alpha(\tau - \Delta) \tag{17.1}$$

$$> (1 - \epsilon_1)(1 - \nu)\alpha\tau \tag{17.2}$$

$$> (1 - \epsilon_1)(1 + \phi)\beta\tau \tag{17.3}$$

$$> (1 + \epsilon_a)\beta \cdot (1 + \frac{\phi}{2})\tau > \mathbf{A}[s : t] \tag{17.4}$$

where (17.2) stems from the fact that $\alpha\tau = \Theta(\kappa)$ and $\alpha\Delta = O(1)$; and (17.3) stems from our parameter choices, i.e., the discounted honest mining power exceeds the corrupt mining power by a constant margin (see Equation (14.1) of Chapter 14).

[1] The choice of ϵ_c affects the choice of the negligible function.

Case 2: when $t - r > \frac{\phi}{2}\tau$. By Fact 5, for every length ℓ' corresponding to the length of some convergence opportunity during $[s, r-\Delta]$, either $\mathsf{chain}^r[\ell']$ or $\mathsf{chain}^t[\ell']$ must be mined by a corrupt node. Also, recall that except with $\mathsf{negl}(\lambda)$ probability,

$$\mathbf{C}[s : r - \Delta] > \mathbf{A}[s : r + \frac{\phi}{2}\tau]$$

This means that except with negligible probability, there must be some ℓ^* corresponding to the length of a convergence opportunity during $[s, r-\Delta]$, such that if $\mathsf{chain}^r[\ell^*]$ or $\mathsf{chain}^t[\ell^*]$ is mined by a corrupt node, it must be mined after $r + \frac{\phi}{2}\tau$. This means that $\mathsf{chain}^r[\ell^*]$ cannot be mined by a corrupt node. By Fact 5, $\mathsf{chain}^t[\ell^*]$ must be mined by a corrupt node after $r + \frac{\phi}{2}\tau$, since otherwise it must be that $\mathsf{chain}^t[\ell^*] = \mathsf{chain}^r[\ell^*]$.

However, by chain growth lower bound, for an arbitrarily small consant ϵ, except with $\mathsf{negl}(\lambda)$ probability, in any round $r + \frac{\phi}{2}\tau$ or later, even the shortest honest chain must have length more than

$$\widetilde{\ell} := \ell^* + (1-\epsilon) \cdot \frac{1}{2}\phi\tau \cdot (1 - 2pn\Delta)\alpha$$

Note that this also means that chain^t must have length at least $\widetilde{\ell}$, since it is an honest node's longest chain in round $t > r + \frac{\phi}{2}\tau$. Moreover, since honest nodes always mine off the longest chain, no honest node will mine off any chain whose length is smaller than $\widetilde{\ell}$ in round $r + \frac{\phi}{2}\tau$ or later. This means that all blocks in $\mathsf{chain}^t[\ell^* : \widetilde{\ell}]$ must be mined by corrupt nodes. However, due to chain quality, this cannot happen except with negligible probability. □

Chapter 18

Proof of Stake (Brief Overview)

Nakamoto's consensus first showed how to achieve consensus in a permissionless, decentralized environment with open enrollment. This was amazing, but the community soon realized that Proof-of-Work (PoW) based approaches are undesirable partly due to the enormous energy waste induced by the protocol. According to the bitcoin energy consumption tracker at Digiconomist, as of June, 2019 Bitcoin consumed 66.7 terawatt-hours per year. That's comparable to the total energy consumption of the Czech Republic, a country of 10.6 million people!

To avoid the enormous waste, many blockchain projects put their hope on a new paradigm called Proof-of-Stake (PoS). In PoW-based consensus, nodes have voting power proportional to their mining power, and consistency and liveness are guaranteed if the (super-)majority of the mining power is honest. Analogously, in PoS, nodes have voting power proportional to *the amount of cryptocurrency they hold*, and we hope to guarantee consistency and liveness as long as, roughly speaking, *the (super-)majority of the stake participating in consensus behaves honestly.* Of course, as we discussed in Chapter 15, honest majority shouldn't be taken for granted in a decentralized setting and a good protocol ought to incentivize honest behavior.

So how can we construct PoS consensus protocols for decentralized cryptocurrencies? Various academics [KRDO17, DPS19] and blockchain projects (e.g., Ethereum, Dfinity, Algorand) have such attempts. In this lecture, we will give an informal overview of how to construct PoS consensus.

Interestingly, while PoS also supports "open enrollment", at any snapshot of time, it actually employs a "permissioned" consensus instance! This is

because at any snapshot of time, the set of consensus nodes are well-defined, and their public keys are known (since players have to demonstrate possession of the cryptocurrency to become an eligible participant).

At a very high level, PoS consensus systems start with an *initial consensus committee whose identities are well-known.* Then, during the life-time of the system, the *consensus committee is reconfigured periodically* to reflect the latest stake distribution (since cryptocurrencies may switch hands over time). Reconfiguration of the consensus committee requires common knowledge — therefore, typically, when the current committee is running consensus, we use this opportunity to not only agree on transactions, but also to reach agreement on the next consensus committee.

Bootstrapping the initial consensus committee. For bootstrapping the system, we need to assume that there is an initial set of consensus nodes whose identities are publicly known. This initial set of consensus nodes can be established in multiple ways in practice. One approach that has been adopted by several blockchain projects is through an initial coin sale such that interested parties can become stake-holders and participate in consensus. Another approach is to bootstrap the cryptocurrency system with PoW, and when miners have mined some cryptocurrency, the system then transitions to PoS (e.g., Ethereum has adopted this approach). To enable such transition, miners can "stake in" the cryptocurrencies they mined through PoW or purchased through an online exchange. A stake-in operation can be accomplished by sending an explicit message to the blockchain — at this moment, the blockchain is sill running PoW — indicating that the player is willing to "freeze" their coins and become a PoS participant. Typically, cryptocurrency that is staked in will be frozen for a while and cannot be spent when the corresponding stake-holder is participating in consensus. The concern here is that if a player has sold all of its stake, it may be incentivized to engage in a history-rewriting attack, and it is no longer incentivized to participate honestly and protect the robustness of the cryptocurrency system.

Committee reconfiguration. As mentioned, since cryptocurrencies switch hands, we need to periodically reconfigure the consensus committee to allow old stake-holders to exit and allow new ones to join. At any time, any player can purchase cryptocurrencies and bid to become a member of the next consensus committee. Such bidding can take various forms, and one common approach is to use the same stake-in mechanism mentioned earlier, i.e., players send explicit stake-in messages to the current blockchain to

indicate their intent to lock up their cryptocurrencies, and participate in the next consensus period. Effectively, the current consensus protocol is not just used for confirming transactions, but also to agree on the next consensus committee.

Choice of permissioned consensus protocol. As mentioned, interestingly, PoS is in some sense returning to our classical roots, since it adopts permissioned consensus protocols at any snapshot of time. Indeed, the community's joint push towards the PoS paradigm has created much momentum for perfecting permissioned consensus protocols. Many elegant works have appeared in the past decade [BZ17, CPS18b, CPS18a, YMR+18, HMW, Shi19b, CS20a, PS17c, KRDO17, DPS19, CM16]. At a very high level, most of these recent protocols aim to achieve one or more of the following goals:

- **Simplicity.** As mentioned in Chapter 7, it turns out that the classical mainstream approaches such as PBFT [CL99], Paxos [Lam98] and their numerous variants [KAD+07, GKQV10, Bur06, JRS11, BSA14, OO14] have been viewed as too complicated, and thus many projects focused on developing simpler, and more streamlined consensus approaches [BZ17, CPS18b, CPS18a, YMR+18, HMW, Shi19b, CS20a]. One of the simplest approaches, called Streamlet, was described in Chapter 7.

- **Robustness.** As we have learned in this course, synchronous consensus can tolerate more faults than partially synchronous protocols, since the latter is subject to a 1/3 resilience lower bound which we learned in Chapter 8. Several blockchain projects preferred synchronous consensus protocols for this reason; but it turns out that all classical synchronous protocols lack in terms of robustness. As a recent work by Guo et al. [GPS19] pointed out, alarmingly, all classical synchronous consensus protocols are *underspecified* and *unimplementable* in practice: if a node ever experiences even a short-term outage and receives messages out of sync, it is treated as faulty and will no longer enjoy consistency and liveness. Classical synchronous protocols provide no way for such nodes to join back and continue to participate! Since blockchain protocols have been running for more than a decade, no one can guarantee 100% up time for so long (e.g., even Gmail has outages every now and then). Thus, if classical synchronous protocols are adopted, everyone will eventually experience some (possibly short-term) outage and become faulty! Note that in partially synchronous protocols, nodes that suffer from short-term outages are not penalized since a short-term outage

can be viewed as a really long network delay. However, as mentioned, the drawback with partial synchrony is the 1/3 resilience barrier.

Realizing that this is a problem, several works [PS17c, GPS19, CPS18b] suggested new modeling techniques, and showed that it is possible for consensus protocols to support sporadic participation (i.e., allow nodes to come and go), and still be able to achieve security under honest majority (as opposed to the 1/3 resilience of partially synchronous protocols). Interestingly, notice that Bitcoin's Nakamoto consensus allows sporadic participation and achieves security under honest majority — but it requires PoW.

- **Performance.** As mentioned earlier, multiple blockchain projects found synchronous consensus desirable due to its better resilience. However, besides lack of robustness, classically, synchronous consensus protocols are also considered slow because the protocol's performance typically depends on an a-priori determined network delay parameter. For example, if the network's average delay is expected to be 1 second, one might want to conservatively set this delay parameter to be 10 seconds — since consistency can be violated if the network violates synchrony assumptions. We say that a protocol is *responsive* if it confirms transactions as fast as the network makes progress, independent of any a-priori configured delay parameter. Classically, most partially synchronous or asynchronous protocols are responsive; but again they can defend only against 1/3 corruptions. Recent works [PS18] proposed a new notion of performance called *optimistic responsiveness* which aims to achieve responsiveness almost all the time in practice, without being subject to the 1/3 lower bound pertaining to partial synchrony or asynchrony. For example, a couple recent works [PS18, CPS18b], constructed protocols that guarantee consistency and (slower) liveness as long as the majority of nodes are honest; but assuming that a designated leader and 3/4 of the nodes are honest and online, the protocols can confirm transactions at raw network speed.

Bibliography

[ACD+19] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 317–326, 2019.

[BS] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. `http://toc.cryptobook.us/book.pdf`.

[BSA14] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *DSN*, pages 355–362, 2014.

[Bur06] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, 2006.

[BZ17] Vitalik Buterin and Vlad Zamfir. Casper. `https://arxiv.org/abs/1710.09437`, 2017.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, July 2005.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[CM16] Jing Chen and Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

[CPS18a] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. Cryptology ePrint Archive, Report 2018/981, 2018. https://eprint.iacr.org/2018/981.

[CPS18b] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. Cryptology ePrint Archive, Report 2018/980, 2018. https://eprint.iacr.org/2018/980.

[CPS20] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Sublinear-round byzantine agreement under corrupt majority. In *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, pages 246–265, 2020.

[CS20a] Benjamin Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. https://eprint.iacr.org/2020/088, 2020.

[CS20b] Benjamin Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains (blog post). https://decentralizedthoughts.github.io/2020-05-14-streamlet/, 2020.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. In *Financial Crypto*, 2019.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *Siam Journal on Computing - SIAMCOMP*, 12(4):656–666, 1983.

[ES14] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, 2014.

[FLM85] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of*

Distributed Computing, PODC '85, pages 59–70. Association for Computing Machinery, 1985.

[flp] A brief tour of flp impossibility. `https://www.the-paper-trail.org/post/2008-08-13-a-brief-tour-of-flp-impossibility/`.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *SIAM Journal of Computing*, 1997.

[gen] `https://en.bitcoin.it/wiki/Genesis_block`.

[GKKO07] Juan Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 11 2007.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[GKQV10] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 363–376, New York, NY, USA, 2010. ACM.

[GPS19] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, pages 499–529, 2019.

[HMW] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview seriesconsensus system. `https://dfinity.org/tech`.

[JRS11] Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In

Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks, DSN '11, pages 245–256, 2011.

[KAD+07] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 45–58, 2007.

[KK09] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, February 2009.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Crypto*, 2017.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 1998.

[Lam01] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, December 2001.

[lec] Lecture notes for 6.852: Distributed algorithms fall, 2009. MIT Open Courseware, `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-852j-distributed-algorithms-fall-2009/lecture-notes/MIT6_852JF09_lec05.pdf`.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 1982.

[mtg] mtgox. `https://bitcointalk.org/index.php?topic=2227.msg29606#msg29606`.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.

[PS] Rafael Pass and Abhi Shelat. A course in cryptography. `https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf`.

[PS17a] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.

[PS17b] Rafael Pass and Elaine Shi. Rethinking large-scale consensus (invited paper). In *CSF*, 2017.

[PS17c] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Asiacrypt*, 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[sha] `https://en.wikipedia.org/wiki/SHA-2`.

[Shi19a] Elaine Shi. Analysis of deterministic longest-chain protocols. In *CSF*, 2019.

[Shi19b] Elaine Shi. Streamlined blockchains: A simple and elegant approach (A tutorial and survey). In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, pages 3–17, 2019.

[VRA15] Robbert Van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3), February 2015.

[wik] `https://en.wikipedia.org/wiki/Bitcoin`.

[WLG+89] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Milliar-Smith, R. E. Shostak, and C. B. Weinstock. *SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control*, page 560–575. IEEE Computer Society Press, Washington, DC, USA, 1989.

[WXSD20] Jun Wan, Hanshen Xiao, Elaine Shi, and Srini Devadas. Expected constant round byzantine broadcast under dishonest majority. `https://eprint.iacr.org/2020/590`, 2020.

[YMR+18] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.