State Machine Replication(SMR) in Asynchronous Network setting

Instructor: Nitin Awathare

September 12, 2024

So far, we've assumed that we are in synchronous network, meaning the message delivery is guaranteed with in known bounded time, say Δ. In this lectures we will introduce the asynchronous communication model (a counter part of the synchronous model) and prove the FLP impossibility theorem, one of the most famous results in distributed computing. This is likely the longest and trickiest proof in the series, and mastering it will be a significant achievement, placing you in rarified company.

One simple way to relax the problematic subassumption (ii) in the synchronous model is to allow messages to be delayed by up to a fixed number of time steps, say between 1 and 100. However, this doesn't effectively generalize the synchronous model. For instance, in the Dolev-Strong protocol, where messages are exchanged at precise time steps, adjusting for potential delays by having non-senders communicate only at time steps 100, 200, etc., merely mirrors the original protocol's behavior. This approach is unsatisfactory for two reasons: it fails to address the core challenge of handling unexpected outages and attacks on the Internet, and it leads to inefficient protocols that spend most of their time idle. For example, even if all messages arrive by time step 1, nodes would still wait until time step 100 to act, just to ensure that all intended recipients have received their messages.

A practical protocol shouldn't let its speed be dictated by the worst-case message delay; it should be efficient when the network is fast. This intuition underlies the partially synchronous model (discussed forthcoming Lectures). Before delving into that, let's explore what's possible with minimal assumptions about message delivery reliability.

The minimal message delivery assumption is the eventual delivery of the message, irrespective of when it will get delivered. This perfectly fit for an asynchronous model. The asynchronous model of communication represents the polar opposite of the synchronous model, with the two subassumptions replaced by nonassumptions. First, there will be no shared global clock, and thus no (even approximately) shared notion of time. Second, messages may suffer arbitrarily long (finite) delays—a protocol designed for the asynchronous model must be ready for anything.

In this network setting, our goal is to demonstrate that consensus is impossible even if all messages eventually arrive. There is no set limit on message delivery time; a message might arrive after billions of others sent later. With this informal understanding of the asynchronous model, let's now define it formally.

Formal description of Asynchronous Model We can represents the Asynchronous Model formally as follows:

- Let M represent the pool of outstanding (undelivered) messages, initially set to $\{(r, \perp)\}_{r=1}^{n}$
- while(TRUE):
  - An arbitrary message $(r, m) \in M$ is delivered to recipient r under the eventual delivery constraint.
  - Recipient r can add any number of messages to M.

Think of the iterations of the main while loop as time steps, but note that in the asynchronous model, nodes have no sense of how many iterations have occurred. A message (r, m) sent by a node includes the recipient r (one of the n nodes) and a payload m (any content). The model is event-driven, with nodes sending messages only in response to receiving new ones. Every message (r, m) will eventually be delivered to r, but the delivery order is arbitrary. To design a consensus protocol with guarantees regardless of message delivery order, envision each iteration's message as being selected by an 'adversary' aiming to disrupt the consensus.

If the message pool M starts empty, no messages get delivered or sent. To initiate activity, initialize M with one dummy message $(r, \perp)$ for each node r, ensuring all nodes eventually participate. To allow repeated participation, assume that when node r receives a message (r, m), it either terminates or sends another dummy message $(r, \perp)$ to M, enabling continuous participation.

In the asynchronous model, the protocol is event-driven and activates when a node receives a new message. At that moment, the node only knows two things: (i) its initial private input, and (ii) the sequence of messages it has received so far. A protocol specifies the messages a node should send in response to the latest received

message, based solely on (i) and (ii). It cannot rely on unknown information, such as the message sequences received by other nodes. If two protocol runs result in the same message sequence for a node (with the same private input), the node will behave identically in both runs, producing the same output.

In previous lectures, we explored the difficulties of designing consensus protocols with multiple Byzantine nodes, which can deviate from the protocol in a coordinated effort to undermine it. In the asynchronous model, Byzantine nodes gain a powerful ally: adversarial message delivery. A robust protocol must withstand conspiracies between Byzantine nodes and adversarial message delivery, where delays might facilitate Byzantine strategies. Essentially, both the actions of Byzantine nodes and the timing of message deliveries are controlled by a single malicious entity, making protocol design in the asynchronous model particularly challenging.

To further explore the asynchronous model and the impossibility of consensus within it, we introduce the Byzantine Agreement (BA) problem, an extension of the previously discussed Byzantine Broadcast (BB) problem. A module of BA can be used to design a BB protocol, and vice versa.

1 Byzantine Agreement

The Byzantine agreement (BA) problem is a one-time consensus problem, similar to Byzantine broadcast (BB) but without a distinguished sender—all nodes have the same role. In BA, each node i has a private input v_i from a known set V of possible values. Unlike BB, where only the sender has a private input, in BA all nodes have private inputs (e.g., transaction orderings in a blockchain). For the impossibility result discussed in this lecture, we focus on the case where $V = \{0, 1\}$. However, proving impossibility for cases where $|V| \geq 2$ is straightforward (consider this as part of your homework). As usual, "private" means that initially, no node knows any other node's input.

A solution to the Byzantine agreement problem must satisfy termination, agreement (safety), and validity (liveness). Termination and agreement are defined as in Byzantine broadcast. Validity differs slightly: if all honest nodes start with the same private input, their outputs should match their inputs, ensuring that Byzantine nodes can't cause deviation from this common input. These properties are defined formally as follows:

- **Termination -** Every honest node i eventually halts with some output $w_i \in V$.
- **Agreement -** All honest nodes halt with the same output
- **Validity -** If all honest nodes have $v_i = v^*$, then their outputs will be $w_i = v^*$.

Note that, we introduce the Byzantine Agreement (BA) problem instead of proving an impossibility result directly for Byzantine Broadcast (BB) or State Machine Replication (SMR) because, in the asynchronous model, BB is trivially unsolvable—honest non-senders can't tell if a Byzantine sender is silent or if messages are just delayed. The BA problem captures the core reasons why consensus is impossible in the asynchronous model. The FLP impossibility for BA implies SMR's impossibility in this model, which is the key result. Consider how an SMR protocol guaranteeing consistency and liveness with $f = 1$ could be adapted to a BA protocol satisfying termination, agreement, and validity, assuming $n \geq 3$.

We're all set to explore the landmark result in the history of distributed systems: the FLP impossibility theorem. Let's dive in!

2 The FLP Impossibility

The theorem states that, even with just one Byzantine node, no deterministic protocol can solve the Byzantine agreement problem in the asynchronous model.

Similar to the hexagon proof, we'll proceed by contradiction. We'll assume there exists a deterministic protocol π for the Byzantine agreement problem that guarantees termination, agreement, and validity in the asynchronous model with $f = 1$. We aim to show that if π satisfies agreement and validity upon termination, there will be cases where it runs indefinitely, leading to a contradiction, proving that π can't exist.

In other words, for a protocol π, we need to demonstrate an infinite sequence of events where the protocol does not terminate. To do this, we define a configuration as a snapshot of the protocol's state, containing:

- the current state of the message pool M
- the private input of each of the nodes
- the sequence of messages received thus far by each of the nodes.

A protocol's run can be visualized as a walk through a large (potentially infinite) directed graph, where vertices represent configurations (C), and edges represent message deliveries, $C \xrightarrow{(r,m)} C'$. When a message $(r, m) \in M$ is delivered, the configuration changes in three ways: (i)(r, m) is removed from the pool M (ii) the recived message is appended to the end of r's sequence of received messages and (iii) new messages (sent by r, according to π or r's Byzantine strategy) are added to M. Showing a sequence of message deliveries where π runs indefinitely (contradicting termination) is equivalent to finding an infinite path in this graph.

Given this we will define three type of configurations:

- **a 0-configuration :** If every Byzantine strategy and message delivery sequence results in the all-zero outcome for all nodes;
- **a 1-configuration :** If every Byzantine strategy and message delivery sequence results in the all-one outcome for all nodes;
- **Ambiguous configuration :** An ambiguous configuration is one that can lead to either a 0- or 1-configuration, depending on the adversary's actions.

In a 0- or 1-configuration, the outcome is predetermined regardless of adversarial actions. In an ambiguous configuration, the adversary can steer the outcome as desired. These definitions are specific to a given protocol π; a configuration may be ambiguous for one protocol but not for another.

Given the definition of the configurations, the proof's aim is to find an infinite path of ambiguous configurations in the directed graph relative to the assumed correct protocol π. This would contradict π's termination property, completing the proof.

To find such a sequence, we use two lemmas: First lemma serves as the base case by ensuring the existence of an initial ambiguous configuration C_0 given a choice of private inputs. On the other hands, the second Lemma acts as the inductive step, showing how to derive a new ambiguous configuration C_{i+1} from C_i. Applying the first Lemma once, followed by repeated applications of second Lemma, produces an infinite sequence $C_0 - > C_1 - > C_2 - > \cdots$ of ambiguous configurations, which is our goal. Let's start with the first Lemma, as it's easier to prove and doesn't require the full power of the adversary in the asynchronous model.

$$\underbrace{1111.....110}_{i}\underbrace{0000.....00}_{n\text{-}i}$$

Figure 1: Initial configuration X_i, where nodes $1, 2, ..., i$ have a private input of 1 and nodes $i + 1, i + 2, ..., n$ have a private input of 0.

Lemma 1. For every deterministic protocol π that satisfies agreement and validity upon termination (with $f = 1$ and some $n \geq 2$), there exists a choice of nodes' private inputs such that the corresponding initial configuration is ambiguous.

Proof. Visualize nodes' private inputs as an n-bit string, where the ith bit represents node i's private input. Start with the all-0s string and sequentially flip 0s to 1s from left to right, resulting in the all-1s string. This generates $n+1$ configurations: $X_0, X_1, X_2, \cdots, X_n$, where in X_i, the first i nodes have a private input of 1, and the remaining $n - i$ nodes have a private input of 0, as depicted in Figure 1.

By the definition of configuration, the initial configuration X_0 must be a 0-configuration. Similarly, by invoking validity, the initial configuration X_n (where all nodes' inputs are 1) must be a 1-configuration.

We claim that one of the intermediate configurations $X_1, X_2, \ldots, X_{n-1}$ must be ambiguous. To see this, consider the smallest $i \geq 1$ such that X_i is not a 0-configuration (such an i exists because $X_n$is a 1-configuration). By the choice of i, X_{i-1}must be a 0-configuration. If $X_i$is ambiguous, we're done. The concern is that X_i might be a 1-configuration, directly transitioning from 0 to 1 when node i's input flips from 0 to 1. Intuitively, this pivotal behavior is implausible—if node i is Byzantine, it can conceal its input. Suppose node i is Byzantine with two strategies: A strategy forcing the all-one outcome (since X_i is not a 0-configuration).Pretending its input is 0 and otherwise following protocol π honestly. With strategy (ii), the protocol's trajectory mirrors that from $Xi - 1$, where all nodes follow π honestly. Since X_{i-1} is a 0-configuration, the protocol terminates in the all-zero outcome. Thus, the Byzantine node i can force either the all-zero or all-one outcome from X_i, making X_i the ambiguous configuration we seek. □

Definition 2.1 $(0^*, 1^*, and Ambiguous^*$ **Configurations).** Let C be a configuration reachable from C_i via the delivery of a sequence of messages different than (r, m). The configuration C is:

- a 0^*-configuration if delivering (r, m) at C leads to a 0-configuration.
- a 1^*-configuration if delivering (r, m) at C leads to a 1-configuration.

- an *Ambiguous**-configuration if delivering (r, m) at C leads to a Ambiguous-configuration

Lemma 2. Fix a deterministic protocol π that satisfies agreement and validity upon termination (with f = 1 and some $n \geq 2$). Let C_i denote an ambiguous configuration and (r, m) a message in C_i's message pool. Then, there exists a sequence of message deliveries such that:

- The last step of the sequence is the delivery of (r, m);
- The end of the sequence is an ambiguous configuration C_{i+1}.

Proof. For the proof, fix a deterministic protocol π that satisfies agreement and validity upon termination (with $f = 1$ and some $n \geq 2$), an configuration C_i (which is an ambiguous configuration) , and a message (r, m) belong to C_i's message pool (Note that, it's not the message pool of C_i itself, but rather the message pool of a node in the network when the system is in configuration C_i.). We need to show that it's possible to eventually deliver (r, m) while retaining ambiguity.

Next, we classify configurations into three categories with respect to the fixed protocol π and the configuration $C_i$along with message (r, m). If delivering (r, m) from C_i directly leads to another ambiguous configuration, the proof is complete. Otherwise, if delivering (r, m) results in a 0-configuration, forcing an all-zero outcome, we proceed accordingly. (The argument for leading to a 1-configuration follows similarly, with roles of 0 and 1 reversed.) These three configurations used in the proof ahead are defined in Definition 2.1.

Since every configuration is either a 0-, 1-, or ambiguous configuration, any configuration reachable from C_i without delivering message (r, m) must be a $0^*, 1^*$ or *ambiguous**configuration. A 0^*configuration can either be a 0-configuration (where delivering (r, m) does not affect the outcome) or an ambiguous configuration (where (r, m) blocks all paths to the all-one outcome). An *ambiguous** configuration is an ambiguous configuration that remains ambiguous after delivering (r, m).

Now, consider performing a breadth-first search starting from the initial ambiguous (and 0^*)-configuration C_i, but with a modification: ignore edges corresponding to the delivery of the message (r, m). This search must eventually find a configuration that is not a 0^*-configurations. If it only encountered 0^*-configurations, then delivering (r, m) would always lead to a 0-configuration, forcing the all-zero outcome. Since the adversary must eventually deliver (r, m), this would imply the all-zero outcome is already forced at C_i, contradicting the assumption that C_i is ambiguous.

This non-0^*-configuration could be either a 1^*-configuration or an *ambiguous**-configuration. If it's an *ambiguous**-configuration, then we've found the sequence of ambiguous configurations, completing the proof. However, we still need to rule out the possibility that the breadth-first search only encounters 1^* -configurations. Referring to Figure 2, let Y be the first non-0^*-configuration reachable from C_i by delivering the fewest messages (excluding (r, m)).

Let X denote Y's predecessor configuration on the $C_i -> Y$ path, and (r', m') the last message delivered along this path (triggering the $X -> Y$ transition). The message (r', m') must differ from (r, m), as the search only considers paths excluding (r, m). Since (r, m) is in C_i's message pool, it must also be in both X's and Y's message pools. X may or may not be C_i(depending on whether a message in C_i's pool leads directly to a non-0^*-configuration). Regardless, because Y is the nearest non-0^*-configuration to C_i and X precedes Y, X must be a 0^*-configuration, as depicted in Figure 2.

To complete the proof, we argue that Y cannot be a 1^*-configuration (and thus must be the *ambiguous** configuration we seek). For contradiction, assume Y is a 1^*-configuration and focus on configurations X and Y, with the delivery of (r', m') at X leading to Y. Since X and Y are 0^* and 1^*-configurations respectively, delivering (r, m) at X or Y would lead to a 0-configuration W or a 1-configuration Z, respectively. Additionally, delivering (r', m') at W (instead of X) would reach configuration V. Since W is a 0-configuration (forcing the all-zero outcome), V is also a 0-configuration. This scenario is depicted in Figure 3

The key takeaway from Figure 3 is that starting from configuration X (with message pool containing both (r, m) and (r', m') is: First, Delivering (r, m) followed by (r', m') results in a 0-configuration (V). Second, on the other hands, Delivering (r', m') followed by (r, m) results in a 1-configuration (Z).

Starting from configuration X, the order of receiving messages (r, m) and (r', m') is crucial, dictating whether the final outcome is all-zero or all-one (reversing the order flips the outcome). A contradiction looms, which we'll prove using two cases: the first is straightforward with no Byzantine nodes, and the second mirrors the argument in Lemma 1's proof.

Case 1 ($r \neq r'$): - If the two messages are meant for different recipients, no node can know the order in which they were received. A node only knows its private input and the sequence of messages it has received, and its actions are based solely on this information. Reversing the order of the messages does not change the sequence received by any node, so all nodes behave identically and produce the same output in both scenarios, contradicting with two scenarios of message ordering mentioned above.

Case 1 ($r = r'$): If the two messages are intended for the same recipient r, only node r knows their delivery order. However, if r is a Byzantine node, it can conceal the true order of (r, m) and (r', m') from the other

nodes. To clarify this, consider two strategies r might use as a Byzantine node.
(i) Follow the protocol π honestly;
(ii) Pretend that it received the messages (r, m) first and otherwise follow π honestly, although r has received (r', m')

Nodes other than r can't distinguish between these scenarios: r received (r, m) before (r', m') and is honest, or r received (r', m') first and is Byzantine. This creates a contradiction since in the first scenario where (r, m) before (r', m') forces an all-zero outcome, as depicted in Figure 3. On the other hand in other scenario, where (r', m') receives first leads to all-one outcome. Thus, our assumption that Y is a 1^* configuration is incorrect; Y is indeed the desired *ambiguous** configuration.

□

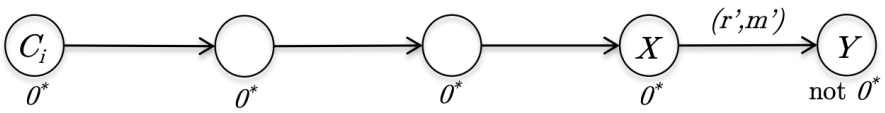

Figure 2: Search for the first non-0^*-configuration

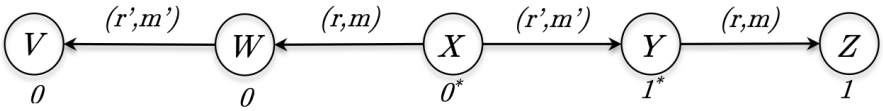

Figure 3: Conclude the contradiction by sending the messages in different orders from the first non-0^*-configuration

Theorem 1. For every $n \geq 2$ and $f = 1$, no deterministic protocol for Byzantine agreement satisfies termination, agreement, and validity in the asynchronous model.

Proof. Fix a deterministic protocol π satisfying termination, agreement and validity (with $f = 1$ and $n \geq 2$). Start by applying Lemma 1 to select an initial ambiguous configuration C_0.

We then want to invoke Lemma 2 over and over. To apply Lemma 2 repeatedly, we need to choose messages from the current pool. If our goal was merely to ensure all messages are eventually delivered, we could use a FIFO strategy: deliver the oldest message first, even if it shifts an ambiguous configuration to a 0- or 1-configuration. Each message will be delivered after a finite number of iterations, as $|M|$ denotes the number of messages in the pool, and M remains finite.

Lemma 2 allows us to balance between two extremes: the trivial solution of delivering only dummy messages (which maintains ambiguity but ensures no delivery) and the FIFO solution (which ensures delivery but not ambiguity). The strategy is to simulate FIFO delivery as closely as possible while preserving ambiguity. Here's how we'll define our sequence of ambiguous configurations:

- Define C_0 as the ambiguous configuration promised by Lemma 1
- For $i = 1, 2, \cdots$
  - Let (r_i, m_i) be the oldest message in C_i's pool. Define C_{i+1} as the ambiguous configuration given by Lemma 2 with respect to C_i and (r_i, m_i).

The above mentioned steps constructs a sequence of configurations, with potentially many intermediate steps between C_i and C_{i+1}, as depicted in Figure 4. The sequence effectively stalls, delivering messages while preserving ambiguity, until (r_i, m_i) can be delivered without resolving the ambiguity.

□

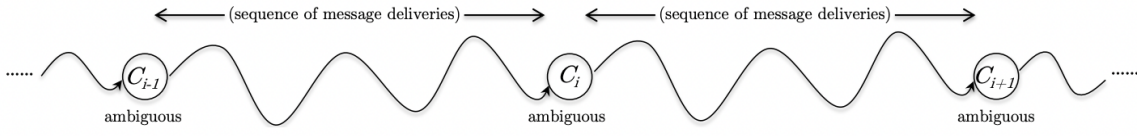

Figure 4: Repeated applications of Lemma 2 generate an arbitrarily long sequence of ambiguous configurations.

References:

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

Disclaimer :

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