State Machine Replication(SMR) under honest setting

Instructor: Nitin Awathare

September 7, 2024

So far, we've assumed that all nodes are honest and always follow the protocol, but this isn't always realistic. In most scenarios, some nodes may behave maliciously. In this chapter, we'll explore consensus protocols in the presence of such malicious nodes. We'll begin by discussing different types of malicious (faulty) behavior and then dive into the Byzantine broadcast protocol.

1 Type of faults

This section outlines three models of faulty nodes, ranging from the most to least benign. If you're mainly interested in the relevant context for permissionless blockchain protocols, you can skip ahead to the third model, which deals with "Byzantine" nodes and is the most pertinent to our lecture series.

Crash Fault: A crash fault occurs when a node suddenly stops working, like it was unplugged—ceasing all communication after a certain time, though it follows the protocol correctly until then. In the 1980s, researchers focused on crash faults, especially in scenarios like database replication (e.g., IBM running multiple machines with database copies). This focus made sense when hardware failures were the primary concern, rather than software bugs, network issues, or malicious attacks.

Omission Fault: An omission fault occurs when a node deviates from the protocol by withholding some or all messages it is supposed to send, without fabricating new ones. These faults can be caused by bad actors or innocent network delays. For example, in synchronous protocols, messages delayed beyond their expected time step are effectively omitted as they are ignored by recipients. A crash fault is a specific type of omission fault where, after a certain point, the node stops sending all future messages.

Byzantine Fault: In blockchain protocols securing vast amounts of value, assuming away potential deviations by dishonest nodes is not feasible. A Byzantine node can deviate arbitrarily from the protocol. Researchers defined Byzantine faults in the 1980s, not necessarily due to malicious actors, but due to unpredictable software errors. Unlike hardware failures (causing crashes) or network delays (causing omissions), modeling a node with buggy software is challenging. To achieve broad applicability, researchers examined the limits of what is possible in the presence of Byzantine nodes. Byzantine nodes might ignore the protocol entirely, often sending contradictory messages to different nodes, which can disrupt consistency in consensus protocols.

Given this, now we will relax one assumption (4th one) and consider the bounded number of Byzantine nodes. Considering this we will discuss various protocols and a few impossibility proofs.

A good protocol should enable honest nodes to achieve the desired functionality (e.g., consistent and live state machine replication) despite the coordinated efforts of Byzantine nodes. The more Byzantine nodes there are, the harder this becomes. A sensible relaxation of the "all-honest" assumption is to assume a maximum bound f on the number of Byzantine nodes, meaning at least $n - f$ of the n nodes follow the protocol correctly. The parameter f is known upfront, allowing the protocol to depend on its value, but the identities of the Byzantine nodes are unknown. To explain the scenario better we will introduce the Byzantine Broadcast problem, which we are going to discuss next.

2 The Byzantine Broadcast (BB) Problem

Our simple rotating leaders SMR protocol ensures consistency and liveness when $f = 0$ but fails when $f = 1$. To achieve fault tolerance, we need a more sophisticated protocol. Fortunately, we can retain the rotating leaders concept (e.g., round-robin). However, since there will be steps when the leader is Byzantine, honest nodes must cross-check rather than blindly trust the leader. This cross-checking leads us to the Byzantine broadcast problem, a single-shot consensus problem essential for preventing inconsistency.

In the Byzantine broadcast problem, one node is the sender, and the remaining $n-1$ nodes are non-senders. The sender's identity is known to all nodes (similar to the rotating leaders protocol), and the sender has a private input v^* from a set V. For example, v^* could be an ordered list of transactions. "Private" means that at the start of the protocol, only the sender knows v^*.

A solution to the Byzantine broadcast problem requires that honest senders can broadcast their input to all honest non-senders while preventing a Byzantine sender from causing inconsistency among honest nodes. The protocol that provides a solution to BB problem must satisfy following three key properties:

- Termination - Every honest node i eventually halts with an output $v_i \in V$, which represents node i's best guess of the sender's private input v^*.
- Agreement - All honest nodes halt with the same output.
- Validity - If the sender is honest, all honest nodes' output matches the sender's private input v^*.

Agreement ensures that no two nodes disagree on their outputs, even if the sender is Byzantine. Validity, combined with termination, ensures that when the sender is honest, the correct input is broadcasted. Byzantine nodes can deviate arbitrarily, so requirements like validity or termination don't apply to them. For Byzantine senders, only agreement is relevant, as they can misrepresent their input or act indefinitely. While termination and agreement, or termination and validity, are easily achieved separately (e.g., default values or straightforward broadcasts), the challenge is to design a protocol that satisfies termination, agreement, and validity simultaneously.

Next, we'll show that fault-tolerant state machine replication reduces to fault-tolerant Byzantine broadcast. The challenge, as with SMR, is designing a protocol that meets both safety and liveness requirements, where in SMR, liveness is analogous to validity in Byzantine broadcast, and consistency is analogous to agreement.

2.1 SMR Reduces to Byzantine Broadcast

We introduced the Byzantine broadcast problem because its solution can serve as a 'black box' for solving state machine replication under the same assumptions (e.g., with the same value of f). The approach is straightforward: use rotating leaders and, in each iteration, invoke a Byzantine broadcast subroutine with the current leader as the sender.

Let π be a protocol for the Byzantine broadcast problem that guarantees agreement, validity, and termination within T time steps when at most f nodes are Byzantine. Given this the detailed reduction steps are followed at each time step 0, T, $2T$, $3T$, . . . , i.e., a multiple of T:

- Define the current leader node in a round-robin order: node 1 is the leader at time step 0, node 2 at time step T, and so forth.
- The leader gathers all pending transactions it has received and compiles them into an ordered list L^*.
- Invoke the subroutine π for the Byzantine broadcast problem, with the leader node as the sender and L^* as its private input.
- Upon π's termination, each node i appends its output L^* to its local history.

This reduction defines a state machine replication (SMR) protocol using a Byzantine broadcast protocol π. Given a global clock and a known set of nodes, the leader is always identifiable. Since π terminates within T steps, each invocation completes before the next starts. This extends the rotating leaders protocol with a fault-tolerant Byzantine broadcast. The reduction preserves the safety and liveness guarantees of Byzantine broadcast, ensuring SMR consistency and liveness, the formal proof of which is detailed below (in 1).

Theorem 1. Under the given assumptions, the SMR protocol from the reduction satisfies consistency and liveness, with the same upper bound f on the number of Byzantine nodes.

Proof. For consistency, all honest nodes proceed in sync, appending the same ordered list of transactions in each iteration. Since the Byzantine broadcast subroutine satisfies agreement, every invocation ends within T steps with all honest nodes appending the same list T to their local histories. Note that They all start with the empty local history and, by induction, would then remain perfectly in sync forevermore.

For liveness, every transaction submitted to at least one honest node eventually reaches all honest nodes' local histories. Since every node leads infinitely often, any transaction tx received by an honest node will be included in a future leader's list of not-yet-executed transactions. Given the subroutine's validity, all honest nodes append L^*, including tx, to their local histories within T steps.

$\square$

Now our aim is to come up with a fault-tolerant Byzantine broadcast protocol. Let's start by discussing naive solutions and their success and failure cases in the following sections.

2.2 The f = 1 Case

the goal here is to build intuition about designing and analyzing consensus protocols. We'll first explore using 'cross-checking' to solve the Byzantine broadcast problem for $f = 1$, and in next subsection, we'll see why this approach fails for $f = 2$ and how additional rounds of cross-checking are needed.

We already know a simple protocol for $f = 0$ (sender broadcasts their input, non-senders output what they received) but it fails for $f = 1$ due to discrepancies caused by Byzantine senders. The idea is for honest non-senders to cross-check received messages for consistency. However, a Byzantine non-sender might frame an honest sender during this phase. Under the PKI assumption, we'll explore the simplest way to implement this 'cross-checking' approach below.

- In the first time step, the sender broadcasts its private value v^* signed with digital signature to all non-senders.
- In the second time step, each non-sender, say i, forwards the message m_i it received from the sender, adding its own signature.
- In the final time step, each non-sender selects the most frequently referenced value from the received messages, breaking ties consistently (e.g., lexicographically). The sender outputs its private input v^*.

Honest nodes can ignore messages from Byzantine nodes by checking timing and consistency. For instance, messages from the sender outside the first step or from non-senders outside the second step are disregarded. If an honest node misses or receives multiple messages, it treats this as receiving a default value (e.g., σ-the empty list of transactions).

This simple protocol is robust against the misbehavior of a single Byzantine node, as demonstrated below. Formally we have to prove that for $f = 1$ and $n \geq 4$, the simple cross-checking protocol (explained above) satisfies termination, agreement, and validity.

The protocol terminates after three time steps. Validity is assured if the sender is honest, as it outputs v^*. Each honest non-sender receives a vote for v^* from the sender and at least one from another honest non-sender. Since $n \geq 4$ and $f = 1$, an honest non-sender can receive at most one vote for a different value, ensuring that the majority vote in the third step will be v^*.

For agreement, we only need to address the case of a Byzantine sender (validity ensures agreement with an honest sender). Since $f = 1$, all non-senders must be honest and will echo the sender's message to all others in the second time step. By the third time step, all non-senders have the same pool of messages from the sender and will perform the same majority vote computation, ensuring a consistent final output (with tie-breaking handled uniformly).

However, the simple cross-checking protocol fails for $f = 2$. Let's brainstorm a solution before proceeding.

2.3 The f = 2 Case

A Byzantine-fault-tolerant protocol must handle any strategy or collusion by Byzantine nodes, including coordinated deviations. Essentially, these nodes might have secret, instantaneous communication channels. The following example illustrates the impact of such conspiracies.

We claim that considering the cross-checking protocol, explained above, with $n \geq 4$ nodes and $f = 2$ a coordinated strategy by the Byzantine nodes (a Byzantine sender and one Byzantine non-sender) can cause the protocol to fail in achieving agreement. The steps are detailed below and shown in Figure 1:

- In the first time step, the Byzantine sender sends a "0" with its signature to half of the honest non-senders (set A) and a "1" with its signature to the other half (set B).
- In the first time step, the Byzantine sender sends both a "0" and a "1" (with signatures) to the Byzantine non-sender, or alternatively, provides its private key for the non-sender to create these messages.
- In the second time step, the Byzantine non-sender echoes the "0" message to nodes in A and the "1" message to nodes in B.

Byzantine nodes use conflicting messages to different honest nodes, breaking the PKI assumption. In the third time step, nodes in A receive $n/2$ votes for 0 and output 0, while nodes in B receive n/2 votes for 1 and output 1, violating agreement.

Key takeaways from this example:

- Many protocols aren't robust to Byzantine faults due to the effective coordination of Byzantine nodes.
- Proving robustness is challenging because of the diverse Byzantine strategies.
- Multiple rounds of cross-checking are needed for robustness, as seen in the Dolev-Strong protocol, which we are gonna discuss next, where each additional round handles one more Byzantine node.

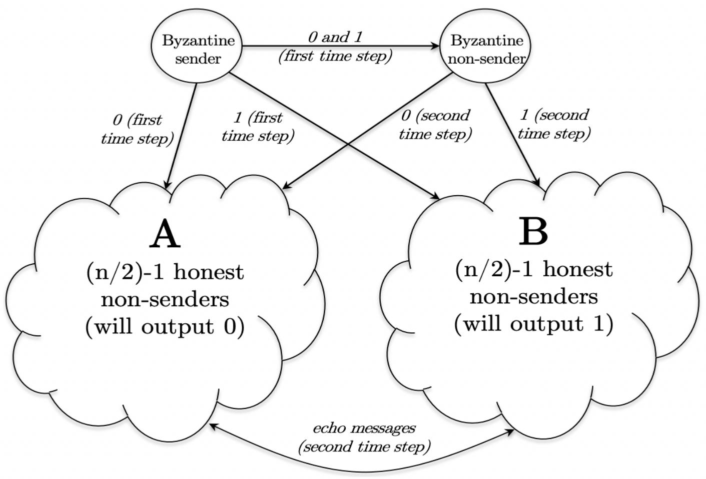
Byzantine sender
0 and 1 (first time step)
Byzantine non-sender
0 (first time step)
1 (first time step)
0 (second time step)
1 (second time step)
A
(n/2)-1 honest non-senders (will output 0)
B
(n/2)-1 honest non-senders (will output 1)
echo messages (second time step)

Figure 1: Violation of agreement when f=2 with simple cross-checking

3 The Dolev-Strong Protocol

This section covers the classic 1983 solution to the Byzantine broadcast problem by Dolev and Strong, applicable in permissioned, synchronous settings with PKI.

This protocol is rarely mentioned in blockchain discussions due to its reliance on the synchronous model and its requirement for a number of time steps linear in f, the maximum number of Byzantine nodes

Nevertheless, there are several reasons to closely examine the Dolev-Strong protocol. First, It's a landmark in distributed computing, akin to experiencing famous works of art in the world of algorithms and proofs. Second, it's a quick read: the protocol is brief, and the proofs of agreement and validity are both clever and concise. Because of its simplicity, it's pedagogically better to start with simpler, synchronous model protocols before tackling complex real-world blockchain consensus protocols. Gradually increasing the difficulty allows for a solid foundation before designing solutions for weaker communication assumptions.

Before diving into the protocol details, let's familiarize ourselves with the definition.

Definition 3.1 ((Convincing Messages)**.** A node i is convinced of value v at time step t if it received a message m_i referring to v before t such that:

- m_i is first signed by the sender;
- m_i must also signed by at least $t-1$ other distinct non-senders nodes, apart from i

For instance, if node 7 receives a message at time step 3 with a value '0' signed by the sender, say node 17, and nodes 23 and 29 (non-sender), node 7 is convinced of value 0 by this message. With the definition and example provided, we are now prepared to dive into the detailed steps of the protocol explained below.

- **At time $t = 0$:** The sender sends its private input v^* with its signature to all non-senders and outputs v^*.
- **At time $t = 1, 2, \cdots, f+1$:** if a non-sender i is convinced of a value v by a message m received prior to this time step and had not previously been convinced of v, the node adds its own signature to m and sends the resulting signed message (m, s) to all other non-senders.
- For each non-sender, if convinced of exactly one value v, it outputs v; otherwise, it come default value, say σ.

Here, f denotes the maximum number of Byzantine nodes, a known parameter crucial for the protocol. It relies on the PKI assumption for signature verification and the synchronous model for reliable communication. Time steps beyond the first involve multiple cross-checking rounds, where non-senders broadcast any newly convinced values to detect discrepancies from the sender. Despite its brevity, the protocol is sophisticated, as demonstrated by the forthcoming proofs of its validity and agreement in the synchronous model.

3.1 Proof of validity and agreement of the Dolev-Strong Protocol

This section proves that, under assumptions of PKI, Synchronous Network, and permissioned setup, the Dolev-Strong protocol solves the Byzantine broadcast problem by ensuring termination, validity, and agreement. While termination is straightforward, we will now demonstrate why validity and agreement also hold.

Validity: Assuming the sender is honest and sends signed copies of its private value v^* at time step 0, no other messages will bear the sender's signature. Given that only the sender knows its private key and signatures cannot be forged, no honest non-sender will be convinced of (Definition 3.1) any value other than v^*. The argument for agreement is slightly trickier, which we are gonna discuss next.

Agreement: Assuming a Byzantine sender, we aim to show that all honest non-senders end up convinced of the same set of values. If successful, this ensures agreement: all honest non-senders will either output the same single value v, the same set of values (outputting σ), or no values at all. Specifically, if an honest non-sender i is convinced of a value v by a message m before time step t, we must demonstrate that all other honest non-senders are also convinced of v by the end of the protocol. We will divide this proof in two cases.

Case1: (t<f) If an honest non-sender i is convinced of v and still has time before the timer goes off (protocol ends), i will add its signature to m and send the signed message (m, s) to all non-senders. In the synchronous model, every other honest non-sender j will receive (m, s) before time step $t + 1$. Since (m, s) is signed by at least t distinct nodes (including i), if j receives (m, s) with $j's$ signature, j must have been previously convinced of v; otherwise, j will be convinced of v by time step $t + 1$.

Case1: (t=f+1) If node i becomes convinced of v only at the clock's expiration (at $t = f + 1$), it may not have time to inform its honest colleagues. However, the Dolev-Strong protocol ensures that if i becomes convinced of v at time step $f + 1$, it must have received m with v and signatures from at least $f + 1$ different nodes, including at least one honest non-sender. This honest node j would have added its signature at a prior step $t' \leq f$. Since j broadcasts its signed messages to all non-senders, all honest non-senders will be convinced of v by time step $t' + 1 \leq f + 1$.

Try proving that the Dolev-Strong protocol fails if the protocol stops one round early, at time step f instead of $f + 1$. Consider the potential strategies for Byzantine nodes.

To wrap up, note a unique aspect of the Dolev-Strong protocol regarding the number of Byzantine nodes f. While the protocol's description and running time scale linearly with f, it remains correct regardless of the value of f. This is unusual in distributed computing, where protocols often fail or become unsolvable once f exceeds a certain threshold, such as $n/3$ or $n/2$. The above proofs hold true for any f, a rare property in the field.

Lastly, the reason we studied the Byzantine broadcast problem is that protocols solving it can be used as a subroutine, along with rotating leaders, to address state machine replication (SMR), which is our primary concern. Now we are convinced and shows that that combining this reduction with the Dolev-Strong protocol ensures that the resulting SMR protocol maintains consistency and liveness, regardless of f.

However, SMR applications generally assume an honest majority (i.e., $f < n/2$). I have stated this in during lecture multiple time, if you can recall. For instance, in database or blockchain scenarios, all honest nodes will agree on a query's result. If a strict majority is honest, a user can rely on a majority vote to determine the correct response. In a 50/50 split, with Byzantine nodes potentially fabricating answers, a client cannot reliably determine which responses are accurate.

This ends the discussion on Dolev-Strong protocol. Next, we'll discuss a 'hexagon proof' that demonstrates it's impossible to achieve the same level of fault tolerance without pre-distributing public keys. This shows the critical role of public-key cryptography and trusted setups.

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 :