Notes on Consensus Protocols

Mahdi Zamani
Visa Research, Palo Alto, CA
mzamani@visa.com

Consensus. A set of n distributed nodes want to agree on a single value proposed by one of them. In the context of blockchains, consensus often means agreement on a sequence of values similar to an atomic broadcast [CV17]. A consensus protocol generally provides the following properties:

- *Agreement:* All honest nodes agree on the *same* message.
- *Validity:* All honest nodes decide on the the input of some honest node.
- *Integrity:* Every honest node decides on at most one value which is the input of some node.
- *Termination:* All honest nodes decide on some value.

Byzantine Faults. A node that can deviate arbitrarily from the protocol. Byzantine nodes can vote for both a statement and its contradiction, i.e., make different statements to different nodes to put non-faulty nodes into inconsistent states. When consensus is performed in such a setting, it is often called Byzantine consensus/agreement.

Byzantine Agreement (BA). One of the participants, called the *leader* or *source*, has an initial value. A Byzantine agreement protocol satisfies the following properties:

- *Agreement*: All non-faulty processes must agree on the same value.
- *Validity*: If the source is non-faulty, then the agreed upon value by all the non-faulty nodes must be the same as the initial value of the leader.
- *Termination*: Each non-faulty process must eventually decide on a value.

Interactive Consistency (IC) [PSL80]. Each participant has an initial value. All non-faulty participants must agree on a vector of values, one for each participant. More formally:

- *Agreement*: All honest parties must agree on the same array of values $A[v_1, ..., v_n]$.
- *Validity*: If party i is honest and its input is v_i, then all honest parties agree on v_i for $A[i]$. If party i is faulty, then honest parties can agree on any value for $A[i]$.
- *Termination*: Each honest party must eventually decide on A.

Equivalence of Consensus Variants. If any of the consensus variants (Byzantine agreement, consensus, and interactive consistency) has a solution, then all of them have a solution. For example, IC can be achieved from BA by run BA n times, one with each participant as the leader. Also, consensus can be achieved from IC by running IC, then outputting the first element of the vector.

Safety and Liveness Guarantees. *Safety* means that all non-faulty nodes agree on the same outcome. *Liveness* means that the non-faulty nodes will make progress during the course of the protocol. Safety can be trivially achieved by making no forward progress at all from an initial state. Liveness can be trivially achieved by making progress unsafely by just letting each node decide on arbitrary outcomes. Neither of these guarantees are useful by themselves.

Consensus Termination. A consensus protocol terminates when all non-faulty participants have terminated. If all non-faulty participants terminate in the same round, then we call they have achieved an *immediate agreement*. Otherwise, we call they have achieved an *eventual agreement* [Fis83]. Pease *et al.* [PSL80] show that at most $t < n/2$ nodes can be faulty for the Byzantine agreement to be solvable. Also, they show that if messages are not authenticated using a digital signature scheme, then at most $t < n/3$ nodes can be faulty.

CAP Theorem. It is impossible for a distributed data store to simultaneously provide more than two out of the following three properties:

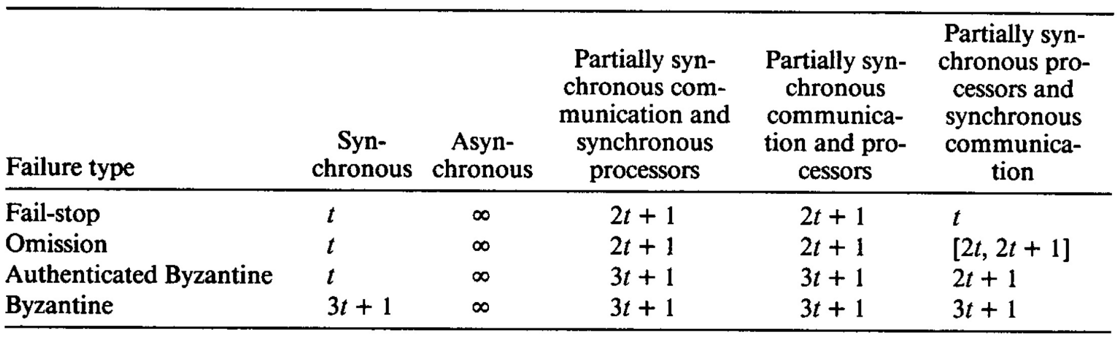
- **Consistency:** Every read receives the most recent write or an error.
- **Availability:** Every request receives a (non-error) response.
- **Partition Tolerance:** The system works even if an arbitrary number of messages being dropped (or delayed).

In other words, in the presence of a network partition, the system can only guarantee either consistency or availability. Existing decentralized systems (such as cryptocurrencies) work by compromising between availability and consistency.

Authenticated Byzantine Agreement. When messages exchanged between the participants are signed using a digital signature scheme, the protocol is called authenticated Byzantine agreement. This assumption requires that the author of a signed message be determined by anyone holding the message, regardless of where it came from (this is often called the *non-repudiation* property). Therefore, message authentication codes (MACs) cannot be solely used to provide authenticated BA. Such an assumption makes synchronous Byzantine agreement solvable when $f < n/2$ [RNAD17]. The intuition behind this is that corrupted nodes can no longer simulate the actions of honest nodes because the use of digital signatures prevents impersonations.

Dolev and Strong [DS83] show that authenticated BA can be achieved in $t+1$ rounds and $O(nt)$ messages using the following algorithm:

1. In each round, each node signs the value and sends it to only the participants who have not signed it yet.
2. After t rounds, each message has t signatures on it. Hence, each correct value has been seen by all the correct participants after $t + 1$ rounds.



Failure type	Synchronous	Asynchronous	Partially synchronous communication and synchronous processors	Partially synchronous communication and processors	Partially synchronous processors and synchronous communication
Fail-stop	t	∞	$2t + 1$	$2t + 1$	t
Omission	t	∞	$2t + 1$	$2t + 1$	$[2t, 2t + 1]$
Authenticated Byzantine	t	∞	$3t + 1$	$3t + 1$	$2t + 1$
Byzantine	$3t + 1$	∞	$3t + 1$	$3t + 1$	$3t + 1$

Figure 1: Smallest number of nodes for which t-resilient consensus protocol exists [DLS88].

Message and Round Impossibilities of BA. For deterministic protocols, it is shown that $t + 1$ rounds are needed for both unauthenticated and authenticated BA [FL81, DS82]. Also, any (deterministic) authenticated BA requires exchanging $\Omega(nt)$ messages [DR82].

FLP Impossibility [FLP85]. Deterministic consensus in the asynchronous setting is impossible even with one crash failure. This is because it is impossible to distinguish a failed process from a very slow one, so processes remain ambivalent when deciding. To avoid this impossibility and/or to achieve practicality, the protocol may be allowed to provide probabilistic safety or liveness guarantees. In the latter case, the protocol is said to provide *eventual liveness* (or *eventual consistency*) since nodes can decide on a value with probability one but with no time bound.

Bracha-Toueg Impossibility [BT83]. There is no (probabilistic) asynchronous consensus protocol resilient to $t \geq n/3$ malicious nodes. See Figure 1.

Broadcast Channel. If a broadcast channel is available, then the Byzantine agreement problem is trivial: each player simply broadcasts their value and then honest players decide on the majority value.

Atomic Broadcast. A set of distributed nodes agree on a sequence of values [HT93]. Consensus and atomic broadcast are equivalent because a sequence of consensus instances provides atomic broadcast. An atomic broadcast protocol generally provides the following properties:

- *Agreement:* All honest nodes eventually receive the *same* message.
- *Validity:* If an honest node broadcasts a message, then all honest nodes eventually receive it.
- *Integrity:* All honest nodes receive the same message exactly once.
- *Total Order:* An honest node receives m_1 before m_2 if and only if the sender sends m_1 before m_2.

Reliable Broadcast.

State-Machine Replication. A protocol that runs among a set of nodes to agree on a sequence of requests based on a deterministic state machine that implements the logic of a service to be replicated among the nodes.

Eventual Synchrony. Introduced by Dwork [DLS88], it models an asynchronous network that may delay messages among honest nodes arbitrarily, but eventually behaves synchronously delivering all messages within a *fixed* but *unknown* time bound. It is widely accepted as a realistic model for designing robust distributed systems [CV17].

Partial Synchrony. Similar to eventual synchrony but with probabilistic network behavior over time [DLS88].

View-Change Protocols. Some consensus protocols such as Paxos [Lam98], Raft [OO14], and PBFT [CL99] proceed in *epochs* (a.k.a., *views*), where a *leader* (a.k.a., the *primary*) is responsible for the progress of the protocol. When a node (a.k.a., a *replicate* or *backup*) suspects that the leader is faulty, the node broadcasts a View-Change message to replace the leader. The new leader has to gather at least $2f + 1$ signed View-Change messages, to broadcast a New-View message. A Byzantine leader cannot violate safety (*i.e.*, cause inconsistencies) but can violate liveness (*i.e.*, prevent progress) by not proposing. In PBFT [CL99], every honest participant monitors the current leader, and if a new message is not proposed after some time, the honest participant requests a new leader.

Byzantine Generals Problem. A commanding general orders its lieutenant generals to attack or to surrender. Some of the generals are traitors. The goal is:

- All loyal lieutenants obey the same order, or
- If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Why is consensus among generals important? Because, halfhearted attack by a few generals would be worse than a coordinated attack or a coordinated retreat. Pease et al. [PSL80] prove $3f + 1$ parties are necessary to achieve agreement in this problem.

References

[BT83] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 12–26, New York, NY, USA, 1983. ACM.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999.

[CV17] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[DR82] Danny Dolev and Ruediger Reischuk. Bounds on information exchange for Byzantine agreement. In *PODC '82: Proceedings of the first ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 132–140, New York, NY, USA, 1982. ACM.

[DS82] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 401–407, New York, NY, USA, 1982. ACM.

[DS83] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[Fis83] Michael Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Fundamentals of Computation Theory*, pages 127–140, 1983.

[FL81] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1981.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[HT93] Vassos Hadzilacos and Sam Toueg. Distributed systems. chapter Fault-tolerant Broadcasts and Related Problems, pages 97–145. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[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, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[RNAD17] Ling Ren, Kartik Nayak, Ittai Abraham, and Srinivas Devadas. Practical synchronous byzantine consensus. *CoRR*, abs/1704.02397, 2017.