ConsenSys Academy

Distributed Coordination: Consensus Protocols

After learning about cryptography, we know how to create an identity, how to make sure no one has messed with a file, and how to capture a user's intent all in a decentralized peer-to-peer way. This is all well and good for current interactions or things we are doing *now* on a network, but how do we agree on things that have happened in the past? That is, how do actors in a network we coordinate and agree on the series of events that have led to the current state of the network? How do we know someone really *does* have digital money they can send to us and are not making it up? How do all the actors in the network then maintain that knowledge in a secure way? This is what we're going to learn in this module.

To understand how all the actors in a network can coordinate and agree about the historical state of a blockchain network we will learn about **consensus**. Distributed consensus is not only used in blockchain, to be clear. Any internet service that needs to coordinate their servers all over the globe (which is all the major internet services) uses distributed consensus mechanisms to make that happen. Blockchain takes traditional distributed consensus one step further. We'll see what that step is once we understand the historical and fundamental principles of distributed computing.

(Note: In this section, we will use the terms **Distributed Computing**, **Distributed Systems**, **Distributed Consensus**, and **Distributed Coordination** interchangeably. We'll also be speaking about consensus strictly in the computer scientific sense. Later in the course, we'll discuss consensus among people in a network, typically called **governance**.)

Development of Distributed Computing

Distributed Computing became an important field of study in the 1970s when airplanes started using electronic control systems. Airline manufacturers wanted to make sure that if a certain part of the electronics gave out while the plane was in the air, the whole plane wouldn't shut down. In the scenario of an airplane, the "bad actor" is not a hacker trying to purposefully disable the entire airplane, but rather a single part that is not behaving as it should. As a result, researchers began researching and developing **consensus protocols** for the airplane computer systems.

At the most basic level, "consensus protocols are used to allow computers to work together" and "let different servers agree on the state of a system." (Software Engineering Daily). For the airplane manufacturers, a good consensus protocol would continue to function with some errors. This way, if one or two things failed, the entire system wouldn't fail. The ability of a consensus protocol to adapt to failure is called resilience.

Crucially, early work around distributed computing and consensus protocols dealt with *non-adversarial systems*. This meant that any of the faults that were happening in a computer network, like an airplane, were the result of natural system errors (power failure, faulty parts, etc), not some sort of active meddling or hacking. This has now developed to encompass much more than aerospace technology. As we mentioned before, it now also covers many digital services, such as:

- Any multi-party real-time communication stream (like a social media feed)
- An online media streaming service which requires multiple regional servers holding and updating the exact same information on customers
- A search engine service that needs to maintain and update indexed information across many regions

Consensus protocols help these systems maintain historical information also called **state.** Broadly speaking, state can be defined as a set of variables describing a certain system at a specific time. Let’s describe that in a real-world situation. Take a look around at whatever environment you’re in—bus station, coffee shop, office—and pick out a few variables you could use to describe it. If you’re inside a room, you could describe any number of things:

- The number of walls
- The types of furniture
- The placement of furniture
- The number of people
- The kind of light in the room

Taken all together, these variables will paint a picture of the room. And if things change (say, you turn off a light), we'll update "The kind of light in the room" variable, which changes the state. If multiple people needed to maintain a record about the state of our room, we'd have to find a way to communicate this state change. Consensus protocols help us do exactly that: agree on a sequential series of system state which allows all network participants adhering to the consensus protocol to have a similar understanding of the historical changes adding up to the current network state.

For a distributed computer network, state typically involves technical information about critical actors in a system. For a social media site, the state includes when a user logged in, what they did, where they were, etc. For an airplane or spaceship, the state includes current status of different parts of the ship, fuel or energy levels, temperature or atmospheric data, etc. As each individual actor in the network uses the consensus protocol to propagate the changes they're doing locally and update their own state based on updates they're getting from others, a historical understanding of the system begins to *emerge* from these state changes. This coordination of state among multiple actors in a common system allows for many interesting systems, including many of the digital services we use today.

Please note that we're using terms like "actors" or "participants" to describe the active parties in a distributed system. Despite the name, these traditionally refer to machines or computers in a network more commonly called **nodes**. It can be confusing but just try to remember these are general models we're discussing. Once we get into application and practice, it may be easier to understand.

A distributed system where multiple actors are using a consensus protocol to maintain state can be called a state machine or finite state machine. This is fairly technical, but it simply requires us to expand our understanding of a machine, which we typically think of as a metal box which containing small electronics connected by circuits. "State machine" allows us to consider larger systems, such as a cellphone network or all the electronic parts comprising an airplane, as themselves being machines comprised of nodes consistently maintaining a global state among themselves without a central point of failure.

Along with state, distributed consensus relies on a few concepts, such as:

- **Nodes** In a strict technical sense, a node is defined as "an electronic device that is attached to a network, and is capable of creating, receiving, or transmitting information over a communication channel." Distributed systems are comprised of nodes. We also call nodes participants or actors. Nodes typically fall into three categories: Leaders (nodes responsible for proposing values), Acceptors (nodes that receive values from Leader and accept them), Processors (nodes that do some operations or processing on received values) (source). These roles are not exclusive, a single node may take on one, two, or all three roles.
- **Message Propagation** A node can update its state in a distributed networks exclusively through messages. How those messages pass or propagate through the network is a critical part of maintaining state. If a node cannot pass a message through a network, there cannot be a unified state that all network nodes agree on. How nodes in a network propagate their messages

is known as its topology. Centralized systems, as shown below, can quickly distribute messages. However, they aren't particularly resilient (if the single central node collapses, so does the network). As a result, distributed systems have developed their own peer-to-peer protocols. Below is a famous diagram showing centralized, decentralized and distributed network topologies:

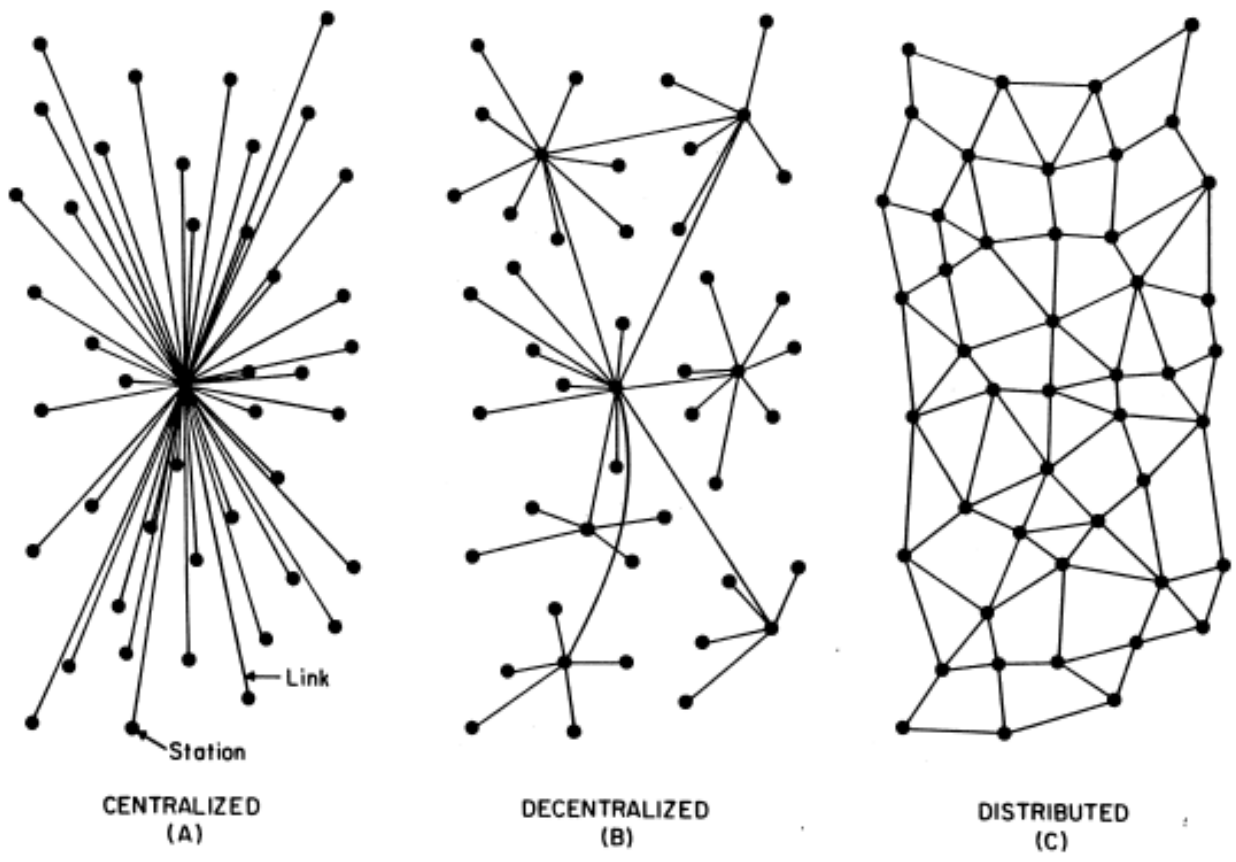

FIG. I — Centralized, Decentralized and Distributed Networks

- **Time** The notion of time is very important in a distributed system as it creates a sort of order for the larger system. Ordering events that occur in a system is particularly important. Think about making breakfast, for example. If you mix up the order of a series of actions, like eating your eggs before you cook them, it can create chaos and confusion. Here's an article from Dean Eigenmann discussing the concept of Time, Clocks, and Order in distributed systems.

- **Periods** Related to the idea of time, every consensus protocol requires discrete periods of activity. Perhaps a node is waiting to hear from a leader, perhaps a series of transactions are being prepared for a block, perhaps the nodes are passing around the latest agreed-upon state. These periods are critical to any consensus protocol. In blockchain systems, these periods typically revolve around the creation and propagation of transactions in a *block*.
- **Fault Tolerance** This is a formal description of resilience: How many mistakes can a system tolerate before it will collapse completely? Put another way, how many bad nodes can we have in a system before the system ceases to propagate state? Leslie Lamport proposed a subset of fault tolerance called Byzantine Fault Tolerance. We discussed it briefly in the video before. Essentially, the most amount of fault a distributed network can absorb is one-third. So, if 2/3rds of a system are still available and coordinating, the system can still run. Because of this famous thought experiment, you may see fault tolerance referred to as Byzantine Fault tolerance or Practical Byzantine Fault Tolerance. However, it's similar to how

a rectangle is not a square but a square is a rectangle: Not all fault tolerance is Byzantine fault tolerance. We'll discuss more of this later in the section on trustless consensus.

Please note: Distributed consensus outside of blockchain only deals with systems that are non-adversarial meaning all the nodes trust each other. This means the only errors that would show up would be from things like power failures or misbehaving parts, etc. You would not attribute malice to any misbehaving actor. Blockchain's big innovation, which we'll discuss later, was the creation of consensus protocols in adversarial networks in which you *must* assume everyone is out to get you. This is what we call **trustless consensus**.

Conclusion

In our search for the primitives underlying blockchain technology, consensus holds an important piece by allowing a network to have a memory of its own history, which we are calling *state*. We saw how cryptography allowed us to ensure peer-to-peer authenticity in the moment. Consensus protocols allow us to "save" that authenticity across time by facilitating the coordination of all network nodes around a global state. It also allows new participants (nodes) to enter the system and get "up to date" on what has happened previously in the system.

The next section is an excellent overview of a basic consensus protocol system called Raft. Raft is a simplified consensus algorithm which we feel makes it more approachable to understand. However, Raft is a production-ready consensus protocol used by such major projects as MongoDB. The website The Secret Lives of Data has created an extraordinary walkthrough of Raft, which we hope will illustrate consensus in a concrete way.

Additional Links

Basic

- Interactive: Raft: Understandable Distributed Consensus A really excellent, interactive walkthrough of the Raft consensus protocol, a basic consensus protocol. The simple and easy way in which the tutorial walks through the consensus mechanism will help you understand how consensus protocols work on a practical level.
- Article: Let's Take a Crack at Understanding Distributed Consensus (Preethi Kasireddy)
- Article Series: Distributed Ssytems Digest (Dean Eigenmann) This is an exellent series of articles discussing distributed systems, in an approachable way. If you click "Let me read it first," you can access the articles.

- Podcast: Distributed Systems with Ethan Buchman (Software Engineering Daily) An overview on distributed systems, including the history of their development
- Article: Want to Really Understand Blockchain? You Need to Understand State (ConsenSys)
- Article: A Brief Overview of Kademlia and Its Use In Various Decentralized Platforms The Kademlia protocol is a peer-to-peer file sharing system used by many decentralized systems, including Ethereum.
- Article: Nodes and Links (Explained from First Principles)
- Interactive Code: BitTorrent Simulator A very cool visualization of how files are distributed over BitTorrent, which uses peer-to-peer file sharing
- Wikipedia: Consensus Methods, Distributed Computing, Network Resilience, Fault Tolerance, Network State, Byzantine Fault Tolerance State Machines, Peer-to-Peer Protocols, Gossip Protocol

Advanced

- Course: Distributed Systems
- Textbook: Foundations of Distributed Consensus and Blockchains (Elaine Shi) An advanced and extremely technical but comprehensive view on distributed consensus as it pertains to blockchain development.

- Academic Article: Leslie Lamport's Byzantine Generals Problem
- Article: Times, Clocks and Ordering (Leslie Lamport)
- Github: Notes for Dean Eigenmann's Article "Times, Clocks and Ordering"
- Github: P2P Workshops This site has a series of exercises to learn about building P2P networks