Chapter 4

Consensus Protocols

Scribe: Yuncong Hu, Yan Ji, Siqiu Yao

4.1 Introduction to Consensus

Consensus protocols are at the core of distributed systems, an important area that has been investigated for 30 years. For more than a decade, companies such as Google and Facebook crucially rely on distributed consensus protocols to replicate their database and computing infrastructure. Distributed computing is also closely related to cryptography — in particular, multi-party computation, which is at the core of modern cryptography, relies on consensus techniques to achieve consistency.

Motivating example. Let us consider a cryptocurrency application as a motivating example. The goal of a digital currency is to keep track of a ledger that stores each user's remaining account balance. If Alice has more than $1000 in her account, she can then pay Bob $1000, e.g., to purchase an iPad Pro.

A simple way to implement such a cryptocurrency is to rely on a central server to keep track of the ledger. Any transaction $\mathsf{tx} := (\mathsf{sender}, \mathsf{receiver}, \mathsf{amount})$ can be submitted to this server: the server would then check whether the sender has sufficient balance, and if so, it will update both the sender and receipient's account balance.

Such a centralized solution has a major drawback: if the single server crashes, then service will not be available and the server can also lose all historical data. For example, in September 2016, Delta Airlines had an entire day of outage due to a failure in their IT infrastructure. On that day, all Delta flights were cancelled, all flight booking systems failed, and as a result Delta lost $100M in revenue.

The idea of distributed systems is to *replicate* critical services such as these, such that we can achieve *robustness* against faults and failures. Simple as this idea may seem, it turns out that implementing this idea correctly is surprisingly challenging — and this explains why it has given rise to an exciting scientific area that has thrived for more than 30 years.

Key properties of a distributed system: consistency and liveness. Distributed systems typically require two important properties, *consistency* and *liveness*. Roughly speaking, consistency requires that all servers have the same view of transaction log; and liveness requires that when a user submits a transaction, it will be processed quickly.

At first sight, the problem might even seem deceptively simple — indeed, there is a somewhat trivial solution if all servers always behave correctly. As it turns out, when some servers can be faulty, the consensus problem is highly non-trivial!

Of course, if all consensus nodes are faulty, then no guarantee can be attained. In consensus protocols, we typically require that consistency and liveness properties must be retained if at least a threshold number of nodes are behaving correctly (and in the case of permissionless, proof-of-work protocols, we require that a sufficient fraction of the computational power is behaving correctly).

Permissioned vs. permissionless distributed systems. The (30 years of) classical distributed systems literature focused on *permissioned* distributed systems, where nodes have a-priori knowledge of the consensus nodes participating in the protocol. For example, consider a typical deployment scenario: Google wishes to rely on consensus protocols to replicate their Google Wallet service. In this case, perhaps Google can deploy a dozen servers that are inter-connected through fast, local-area networks. These servers can run a classical consensus protocol such as Paxos [L$^+$01] or PBFT [CL02a] to reach agreement about the state of the system.

Recently, as decentralized cryptocurrencies such as Bitcoin [Nak08] and Ethereum [Woo14] gained popularity, we have pushed consensus protocols to newer heights. Cryptocurrencies such as Bitcoin employ *permissionless* consensus protocols, where anyone can join (and leave) the consensus protocol at any time, and there is no a-priori knowledge of the consensus nodes. Such Internet-scale consensus also raises new challenges: for example, nodes are heterogeneous and may only participate sporadically; networks can be unstable

and have short-term outages; and finally, the number of nodes participating can be much larger than the traditional, controlled deployment environments.

Modeling faults. In the distributed systems literature, we typically consider two types of faults:

- *Crash faults*, where a faulty node simply stops pariticipating and ceases to send protocol messages; and
- *Byzantine faults*, where a faulty node can behave arbitrarily and send arbitray protocol messages.

Clearly, the latter type of faults is more difficult to handle — this is what we will mainly focus on in the remainder of this chapter.

Roadmap of the remainder of the chapter. In the remainder of this chapter, we will first describe examples of classical, permissioned consensus protocols; and then we will describe modern, permissionless blockchain protocols. We will also explain various modeling choices and their implications with respect to what is feasible and what is not.

4.2 Dolev-Strong Algorithm

Byzantine General's Problem. Assume there are N generals, each commanding a portion of the Byzantine army, encircling a city. The generals want to formulate a plan to attack the city. Every general needs to decide if he is going to attack or retreat. There is a commanding general who will send orders to other generals. Note that if all the generals could meet in person and have an agreement on the plan before conducting the attack, this problem would be trivial.

Network model. We will make the problem more interesting by considering a modern version: suppose that the generals must reach an agreement over the Internet. Every day, every general can send an email to all other generals. The email is guaranteed to be received by the end of the day, such that when a general wakes up the next morning, he can check his inbox containing emails from other generals sent on the previous day, and then decide what new message to send today. Suppose that the generals are tech-savvy, and they have exchanged their public keys ahead of time. In other words, all generals'

public keys are a-priori known. In this way, the generals can sign their emails such that no one can forge an honest general's emails.

Technically, we have described a network model that assumes the following:

- A public-key infrustructure. Note that a public-key infrastructure implies pairwise *authenticated channels*, i.e., the adversary cannot tamper with or modify messages sent by honest nodes (in our example, the generals act as the nodes).

- A *synchronous* communication model, i.e., a message sent by an honest node to an honest recipient is guaranteed to be delivered at the beginning of the next round.

Recall that our goal is to make every general reach a consensus on attacking or retreating. To make the problem more interesting, let us assume that some generals may be *corrupt*. For example, they may have been bribed by the enemy and their goal is to cause the remaining *honest* generals to split in opinion.

Modeling faults. Henceforth, we assume that honest generals will always faithfully follow the prescribed protocol; but *corrupt* generals can deviate arbitrarily. More specifically, corrupt generals can send any malicious message or fail to send a message during a day. Since our problem is called the *Byzantine* generals problem, this is why we commonly refer to such faults as *Byzantine* faults.

Problem definition. Since corrupt generals can behave arbitrarily, we cannot expect them output the same bit as the remaining honest generals. We therefore seek a protocol such that all *honest* generals would reach consensus at the end. More formally, we would like to achieve the following properties:

- **Validity.** If the commanding general is honest, every honest general will decide on the command (i.e., attack or retreat) proposed by the commanding general.
- **Agreement.** Every honest general reaches the same decision of whether to attack or retreat.
- **Termination.** At the end of the protocol, all honest generals will output a decision.

Observe that if we only require *Agreement*, *Termination*, but not *Validity*, then there is a trivial solution where everyone simply outputs a canonical answer such as "attack".

In other words, the Byzantine Generals problem is trying to realize a "broadcast" abstraction from pairwise authenticated channels.

Attempts at the Solution. Before describing Dolev-String protocol [DS83] which is capable of arbitrary number of corruptions, let's think of an intuitive method called *Majority Voting System*.
Assume that everyone has a pair of public and private keys for signing messages. Upon receiving an order from the commanding general, everyone sends their signed vote to others. And eventually every general decides on the majority decision he receives.

Think about a possible attack for this scheme. For simplicity, assume that there are 4 generals among which there is one commanding general. The commanding general and 1 other general are corrupt. In this scenario, the commanding general can send different orders to the honest generals and the other corrupt general does exactly the same. In the end the two honest generals will make different decisions, which breaks the *Agreement* requirement. Clearly this protocol will not be sufficient to defend against malicious participants.

Dolev-Strong protocol. Abstract the order of attacking or retreating as a single bit $b \in \{0, 1\}$. And the final decision of each general is abstracted as an output bit.
Assume among n generals there are at most f traitors. This protocol terminates after $f+1$ rounds and guarantees *Validity* and *Agreement* requirements.

Round 0**:** Initially for each general i, $\mathsf{extracted}_i = \varnothing$. Assume w.l.o.g. the commanding general is general 0. He sends the order b attached with his signature as message $\{b\}_{sig_0}$ to everyone.

Round $r \in \{1, \ldots, f+1\}$**:** For each general i, consider every $b \in \{0, 1\}$. If i has received signatures of r generals, denoted as general $c_0, c_1, \ldots, c_{r-1}$, attached to b throughout the former $r-1$ rounds and $b \notin \mathsf{extracted}_i$, then $\mathsf{extracted}_i = \mathsf{extracted}_i \cup \{b\}$ and the general appends his own signature together with the r signatures he has received to the order bit b and sends the

message $\{b\}_{sig_{c_0},sig_{c_1},\ldots,sig_{c_{r-1}},sig_i}$ to others.

After Round $f+1$: For each general i,

- if $\mathsf{extracted}_i = \{b\}$, then output b.
- otherwise (i.e. $|\mathsf{extracted}_i| \in \{0, 2\}$), output 0.

Proof. Having described the protocol, we must now show that it indeed satisfies *Validity*, *Agreement* and *Termination* requirements.
Termination is trivial since all honest generals will make a decision and terminate after $f+1$ rounds, guaranteed by the assumption of a synchronous model.
Validity is obvious because if the commanding general is honest, he will follow the protocol and output the order he sent to others. Thus we only need to prove *Agreement*.

Claim 4.1. *For a honest general i, if $b \in \mathsf{extracted}_i$ by the end of round $r < f+1$, then for each honest general j, we have $b \in \mathsf{extracted}_j$ by the end of round $r+1$.*

Proof. This can be inferred directly from the protocol specification. ■

Lemma 4.2. *If for some honest general i, $b \in \mathsf{extracted}_i$ by the end of round $f+1$, then for each honest general j, we have $b \in \mathsf{extracted}_j$ by the end of round $f+1$.*

Proof. We prove this lemma by case analysis.
For a honest general i,

1. if b first appeared in $\mathsf{extracted}_i$ by the end of round $r < f+1$, then the conclusion holds according to the *Claim* above.
2. if b first appeared in $\mathsf{extracted}_i$ by the end of round $f+1$, then i must have received b with $f+1$ distinct signatures in round $f+1$. Since there are at most f traitors, according to Pigeon Hole Principle, at least one of the signatures is from some honest general j. Assuming that the signature scheme is secure, this implies that there exists a honest general j such that b first appeared in $\mathsf{extracted}_j$ by the end of round $r < f+1$. By applying the *Claim* above, the conclusion holds.

■

Therefore, all honest generals will have the same extracted set and will output the same bit, by which *Agreement* is achieved.

Observation: If we analyze the proof, we will note that it relies heavily on the fact that there are $f+1$ rounds. In fact, if only required f rounds, the following attack is possible.

- Assume that the commanding general is corrupt, and in round 0 he sends $\{1\}_{sig_0}$ to everyone.
- For round $r = 1, \ldots, f-1$, corrupt generals don't send any messages to honest ones.
- In round f corrupt generals send $\{0\}_{sig_{c_0}, sig_{c_1}, \ldots, sig_{c_{f-1}}}$ to half of the nodes and $\{1\}_{sig_{c_0}, sig_{c_1}, \ldots, sig_{c_{f-1}}}$ to the other half. Note that $c_0, c_1, \ldots, c_{f-1}$ refers to the f corrupt generals here.

In the end, we see that this attack will break *Agreement* requirement since half of the honest generals will output 1 while the other half will output 0.