Chapter 3

Byzantine Broadcast and the Dolev-Strong Protocol

3.1 Introduction

3.1.1 The Byzantine Generals' Problem

In a seminal paper by Lamport et al. [LSP82], the problem of consensus is illustrated with the following example.

"Imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement."

Suppose that there are n generals, and one of them is called the *commanding general.* The commanding general would like to propose an order that is either ATTACK or RETREAT to all generals, such that

1. All loyal generals reach the same decision; and
2. If the commanding general is loyal, then all loyal generals will obey the commanding general's order.

Lamport et al. [LSP82] named this problem the Byzantine Generals problem; and it is also commonly referred to as *Byzantine Broadcast* (BB). Note that if the commanding general is guaranteed to be loyal, then the problem is trivial: the commanding general could send its order to all other generals, and all other generals could simply obey (assuming that they can verify that the order indeed came from the commanding general). Of course, even the commanding general can be a traitor, and in this case, it can propose different orders to different generals; and thus the aforementioned naïve solution would result in inconsistent decisions.

So is it still possible for the loyal generals to agree upon an attack plan by communicating with each other despite the influence of corrupted generals?

3.1.2 A Modern Variant

Here is a more modern variant. Suppose that during the Covid-19 pandemic, the program committee of the Blockchain'20 conference try to decide whether next year's Blockchain conference will be held virtually online or in person. The chair of the program committee is called the *program chair*. The program chair would like to convey a suggestion to the entire program committee, that is either "virtual" or "physical". Since the program committee are all in quarantine, they decide to reach agreement over the Internet by sending emails back and forth to each other. We may assume that an email sent sometime today will be received and read by the recipient at the beginning of tomorrow.

Now, there is a problem: a subset of the program committee are unhappy with the Blockchain'20 conference since their papers had got rejected earlier from the conference. These unhappy committee members may be secretly plotting to disrupt agreement and prevent the next Blockchain'20 conference from happening. Even the program chair herself may be secretly unhappy with the Blockchain conference. While the happy committee members will faithfully follow the protocol rules, the unhappy members can misbehave arbitrarily, and send arbitrary messages.

Can we devise a protocol such that

- all the happy committee members can reach a common agreement; and moreover,
- if the program chair is happy, then every happy committee member should output the chair's original suggestion?

3.1.3 Analogy to Reliable Distributed Systems

The above imaginary situations serve as an analogy with a computer system in which one or more components can fail. More precisely, the problem of distributed consensus in a distributed system is that while some of the nodes in the system might act in an arbitrary manner, the correctly functioning nodes still need to agree on a common value among themselves. Real-life consensus protocols (e.g., Bitcoin) typically need to repeatedly reach consensus over time, such that nodes jointly maintain an ever-growing, linearly-ordered log of transactions, sometimes called a *blockchain*. In our course, we shall start with Byzantine Broadcast which allows nodes to reach agreement once. This is important for understanding the foundations of distributed consensus. Later in our course, we will indeed cover how to define and construct "blockchains", i.e., a repeated consensus abstraction.

3.2 Problem Definition

Let us now formalize the problem more precisely. We will henceforth refer to the generals (or committee members) as *nodes*, and refer to the commanding general (or the program chair) as the *designated sender*, or simply *sender* for short.

Consider a distributed network of n nodes numbered $1, 2, \ldots, n$ respectively. We often use the notation $[n] := \{1, 2, \ldots, n\}$ to denote the set of all nodes. Without loss of generality, we may assume that *the sender is named node* 1.

We refer to the nodes that follow the prescribed protocol throughout as *honest* nodes. A subset of the nodes, however, can be *corrupt*. The corrupt nodes need not follow the prescribed protocol, and they may send/transmit arbitrary messages at arbitrary times, omit sending messages, stop or take an incorrect step. All the corrupt nodes can form a coalition and share information with each other, and perform a coordinated attack. For this reason, it is often instructive to imagine that all the corrupt nodes are controlled by a single *adversary*. We stress that a-priori, we do not know which nodes are corrupt — had we known, the problem would also be trivialized (see Exercise 4). In other words, our Byzantine Broadcast protocol must work no matter which subset of nodes are corrupt, as long as the total number of corruptions is upper bounded by $f < n$.

We assume that every pair of nodes can communicate with each other, and moreover every node has a public and secret key pair corresponding to a digital signature scheme, and all nodes' public keys are common knowledge. In the Dolev-Strong protocol below, every message will be signed by its creator. We use the notation $\langle m \rangle_i$ a pair (m, σ) where σ is a valid signature on the message m that can be verified under node i's public key.

Remark 2. For simplicity, unless otherwise noted, we assume that the set of corrupt nodes is chosen at the start of the protocol (but the protocol is unaware of which nodes are corrupt). This model is often referred to as the *static* corruption model. In the distributed systems and cryptography literature, an alternative model, called *adaptive* corruption, is also extensively studied. In the adaptive model, an adversary can decide which nodes to corrupt in the middle of the protocol's execution, after having observed messages sent by honest nodes. The Dolev-Strong protocol is in fact also secure against adaptive corruptions although we do not explicitly discuss it in this chapter.

3.2.1 Synchronous Network

We assume that the protocol takes place in a *synchronous* network, i.e., when honest nodes send messages, the honest recipients are guaranteed to receive them within a bounded amount of time, say, one minute. No matter how long the message delay is, we can define it as one *round*. Therefore, we shall assume that the protocol execution proceeds in rounds. At the beginning of each round, all nodes receive incoming messages from the network. They then perform some local computation, and send out new messages. The following is guaranteed by the network:

Synchrony assumption: *If an honest node sends a message in round r to an honest recipient, then the recipient will receive the message at the beginning of round $r + 1$.*

3.2.2 Definition of Byzantine Broadcast

At the beginning of a protocol, the designated sender receives an input bit $b \in \{0, 1\}$. The nodes then run some protocol; at the end of the protocol, every honest node outputs a bit. A Byzantine Broadcast (BB) protocol is supposed to satisfy the following two requirements (no matter how corrupt nodes behave):

- **Consistency** : If two honest nodes output b and b' respectively, then $b = b'$.
- **Validity** : If the sender is honest and receives the input bit b, then all honest nodes should output b.

Note that consistency alone would be trivial to achieve without the validity requirement: the protocol can simply require that every node output a canonical bit 0. Therefore, the problem definition is only non-trivial/meaningful if we required both properties simultaneously.

Remark 3. In this chapter, we shall assume that the signature scheme is ideal, i.e., no signature forgery can happen. Moreover, our Dolev-Strong protocol described below will be deterministic. Therefore, we may assume that we want the above requirements to be satisfied deterministically. In later chapters, we shall consider randomized protocols, in which case it may be sufficient for the above properties to hold with all but negligible probability.

3.3 A Naïve (Flawed) Protocol

We first look at a natural but naïve approach. As mentioned earlier, it would not work if everyone simply followed the bit heard from the designated sender: if the designated sender is corrupt and sends different bits to different nodes, then consistency can be violated.

What is the next most natural idea? Perhaps it would be a voting-based approach. For convenience, we shall assume that all nodes sign all messages before sending them, and only messages with valid signatures from purported senders are viewed as valid. All invalid messages are discarded immediately without being processed.

Now, imagine that the sender signs and sends its input bit to everyone in the first round. In the second round, everyone votes on the bit they heard from the sender. If no bit is heard or both bits are heard, they vote on a canonical bit 0. Now, if a node hears majority nodes vote for b, then it outputs b. We describe this simple voting-based protocol more formally below where we use the notation $\langle m \rangle_i$ to denote the message m along with a valid signature from node $i \in [n]$:

A naïve majority voting protocol

- Sender (i.e. node 1) receives the bit b as input.
- **Round** 1: Node 1 sends $\langle b \rangle_1$ to every node (including itself).
- **Round** 2: Every node $i \in [n]$ does the following: if a single bit $\langle b' \rangle_1$ is received, send the vote $\langle b' \rangle_i$. Else send the vote $\langle 0 \rangle_i$.
- **Round** 3: If no bit or both bits received more than $n/2$ votes from distinct nodes, then output 0. Else output the bit that received more than $n/2$ votes from distinct nodes.

Does this protocol work? Keep in mind that corrupt nodes can behave arbitrarily, including sending different votes to different nodes. It turns out that this naïve protocol is not secure under even a single corrupt node. Below we describe an attack in which the designated sender is the only corrupt node, and everyone else is honest, and we show that consistency can be violated.

The attack. Assume that $n = 2k+1$ is odd and only the sender, i.e., node 1, is corrupt. Let us divide the $2k$ honest nodes into two disjoint sets denoted S_0 and S_1, each of size k. In the first round, the sender sends $\langle 0 \rangle_1$ to the set S_0, and it sends $\langle 1 \rangle_1$ to the set S_1. In the second round, nodes in S_0 votes for 0, and nodes in S_1 votes for 1. Now, the corrupt sender skillfully votes for 0 to the set S_0 and it votes for 1 instead to the set S_1. Observe that nodes in S_0 receive majority votes for 0 whereas nodes in S_1 receive majority votes for 1, and thus they will output inconsistently.

So how can one design a *secure* Byzantine Broadcast protocol?

Exercise 1. Here is another simple idea. Round 1 and round 2 are the same as the naïve majority voting protocol. In other words, in round 1, the sender signs its input bit and sends it to everyone. In round 2, everyone votes for the bit it has heard from the sender. If no bit or both bits were heard, vote for the canonical bit 0. In round 3, if some bit b has gained the votes of very node, then output b; otherwise, output 0.

Does this protocol achieve Byzantine Broadcast? If so, please explain why. If not, please describe an explicit attack that either breaks consistency or validity. In your attack, use as few corrupt nodes as possible.

3.4 The Dolev-Strong Protocol

The celebrated Dolev-Strong protocol [DS83] solves the Byzantine Broadcast problem.

Recall that the nodes are numbered $1, 2, \ldots, n$, and we will assume that the designated sender is numbered 1. We denote the sender's input as b. We also assume that each node i maintains a set extr_i, also referred to as the node's "extracted set," of the distinct valid bits that have been chosen so far. For brevity, we will use the notation $\langle b \rangle_S$ to denote the message b attached with a valid signature on b verifiable under the public keys of nodes in $S \subseteq [n]$. In our protocol below, f denotes an upper bound on the number of corrupt nodes.

The Dolev-Strong protocol

Initially, every node i's extracted set $\mathsf{extr}_i = \emptyset$.

- **Round** 0: Sender sends $\langle b \rangle_1$ to every node.
- **For each round** $r = 1$ **to** $f+1$:

  For every message $\langle \tilde{b} \rangle_{1,j_1,j_2,\ldots,j_{r-1}}$ node i receives with r signatures from distinct nodes including the sender:

  – If $\tilde{b} \notin \mathsf{extr}_i$: add $\tilde{b}$ to extr_i and send $\langle \tilde{b} \rangle_{1,j_1,\ldots,j_{r-1},i}$ to everyone — note that here node i added its own signature to the set of r signatures it received.
- **At the end of round** $f+1$**:** If $|\mathsf{extr}_i| = 1$: node i outputs the bit in extr_i; else node i outputs 0.

3.4.1 Intuition

So why is $f+1$ rounds necessary for the Dolev-Strong protocol?

Suppose that all nodes have to output at the end of round f instead of $f+1$. We construct an attack as follows. In round 0, the corrupt sender sends $\langle 1 \rangle_1$ to all honest nodes. Thus, all honest nodes will add 1 to their extracted sets in round 1. Now, recall that there can be f corrupt nodes, including the sender. At the beginning of round f, the corrupt nodes make a single honest node v receive the bit 0 along with all f signatures, but does not deliver this message to every other honest node. In this case, v will end up with 2 bits in its extracted set whereas all other honest nodes have only 1 bit in their extracted sets in round f; and thus v will be inconsistent with all other honest nodes.

The above attack is not possible if the algorithm is run for one more round, i.e., if the nodes output at the end of round $f+1$ instead. Intuitively, this is because a bit b tagged with $f+1$ signatures must have been signed by at least 1 honest node, say, node i. However, when the honest node i signed the bit b earlier, say, in round $r < f+1$, it must have propagated a batch of $r+1$ signatures on b (including its own) to all other honest nodes, and therefore all other honest nodes will have received b along with $r+1$ signatures by the beginning of round $r+1 \le f+1$, and will have added the bit b to their extracted sets (if not earlier).

3.4.2 Analysis

Equipped with the above intuition, we now formally prove that the Dolev-Strong protocol satisfies both consistency and validity.

Lemma 1. *Let $r \leq f$. If by the end of round r, some honest node i has $\tilde{b}$ in extr_i, then by the end of round $r+1$ every honest node has $\tilde{b}$ in its extracted set.*

Proof. We know that node i has the bit $\tilde{b}$ in extr_i by the end of round r. This bit $\tilde{b}$ must have been added to extr_i in some earlier round. Suppose $t \leq r \leq f$ is the round in which the bit $\tilde{b}$ first got added to extr_i. According to the protocol, it must be that in round t, node i received $\langle\tilde{b}\rangle_{1,j_1,\ldots,j_{t-1}}$ with t distinct signatures including one from the sender. Moreover, none of these signatures must come from node i itself because if i had signed $\tilde{b}$ earlier, it would have added $\tilde{b}$ to extr_i earlier. Therefore, node i then must have sent $\langle\tilde{b}\rangle_{1,j_1,\ldots,j_{t-1},i}$ to every other node in round t. Now, by our synchrony assumption, all other honest nodes will receive this message with $t+1$ distinct signatures at the beginning of round $t+1 \leq f+1$ and will, therefore, add $\tilde{b}$ to their extracted sets in round $t+1$ (if it has not already been added). □

The above lemma says that if some honest node has included some bit $\tilde{b}$ in round $r < f+1$, then all honest nodes will have included the same bit in the *immediate next* round. To prove consistency, we need to show that if some honest node has included a bit $\tilde{b}$ in the final round $f+1$, then all honest nodes must have included it in the *same* round. The last round, i.e., round $f+1$, is where the magic happens such that *common knowledge* is reached.

Lemma 2. *If some honest node i has $\tilde{b}$ in extr_i by the end of round $f+1$, then every honest node has $\tilde{b}$ in its extracted set by the end of round $f+1$.*

Proof. We consider the following two cases:

1. **Case 1 (Node i first added $\tilde{b}$ to its extracted set in round $r < f+1$):** By Lemma 1, once $\tilde{b}$ is in extr_i at the end of round r, then every honest node will have $\tilde{b}$ in its extracted set by round $r+1 \leq f+1$.

2. **Case 2 (Node i first added $\tilde{b}$ to its extracted set in round $f+1$):** For this case to happen, node i must have received $f+1$

distinct other nodes' signatures on $\tilde{b}$ at the beginning of round $f+1$. Since at most f nodes are corrupt, at least one honest node must have signed $\tilde{b}$ in an earlier round $r < f+1$. Thus, by Lemma 1, every honest node, including node i, would have added $\tilde{b}$ to its extracted set by the end of round $r+1 \leq f+1$.

□

Using the two lemmas above, we now state the theorems that establish the desired properties for the Dolev-Strong protocol.

Exercise 2. Prove that the Dolev-Strong protocol satisfies validity, that is, if the sender is honest, then every honest node would output the sender's input bit.

Theorem 2 (The Dolev-Strong protocol [DS83])**.** *The Dolev-Strong protocol achieves Byzantine Broadcast in the presence of up to $f \leq n$ corrupt nodes.*

Proof. By Lemma 2, all honest nodes must have the same extracted set at the end of round $f+1$, and thus consistency is achieved. Proving validity is left as a homework exercise (see Exercise 2). □

3.4.3 Further Discussions

Dolev and Strong [DS83] also proved that any *deterministic* protocol solving Byzantine Broadcast (allowing ideal signatures) must incur at least $f+1$ rounds. In practice, $f+1$ rounds may be too expensive for large-scale applications. Fortunately, this $f+1$ round complexity lower bound can be circumvented by using randomness in the protocol design. We will explore randomized protocols later in the course.

3.5 The Muddy Children Puzzle

There is a cute puzzle called the "muddy children puzzle" that bears a remote resemblance to the Dolev-Strong protocol.

There are n children playing in the playground, and $k \leq n$ of them acquire mud on their forehead. After playing, the teacher gathers the children, and declares, "one or more of you have mud on your forehead". Every one can see if others have mud on their forehead, but they cannot tell for themselves.

The teacher says, “at this moment, if you know you have mud on your forehead, please step forward”. The teacher waits for a minute and no one steps forward. The teacher says again, “second call: at this momement, if you know that you have mud on your forehead, please step forward.”. This goes on for multiple rounds until some children step forward. In each round, the teacher calls for those who know that they have mud on their forehead to step forward.

Question: in which round will some children step forward? Note that the children do not communicate with each other. They know that at least one of them has mud on their forehead, and they know the current round number.

The puzzle can be solved by induction. The case $k = 1$ is easy. If Alice is the only kid with mud, then she would step forward in the first round: she sees that no one else has mud, so it must be herself. Now, consider the case $k = 2$. Say, Alice and Bob are the two kids with mud. In this case, no one steps forward in the first round, because Alice sees that Bob has mud, and she cannot be sure if she has mud too; and the same reasoning applies to Bob. However, knowing that no one stepped forward in the first round, Alice and Bob now know that at least two kids have mud (otherwise the only kid with mud would have stepped forward in the first round). Now, in the second round, Alice sees only one other kid with mud, so she knows that she must be the other. The same reasoning applies to Bob. Therefore, in the second round, both Alice and Bob step forward.

This argument can be carried out inductively, and one can show that if k kids have mud, then all k muddy kids will step forward in round exactly k.

In this puzzle, common knowledge is reached in round k. Therefore, it is somewhat reminicient of the Dolev-Strong protocol in which common knowledge is reached in round $f + 1$.

3.6 Additional Exercises

Exercise 3. In our lecture, we defined a one-bit version of Byzantine Broadcast, where the sender wants to distribute a single bit. We may considered a multi-valued variant (called Multi-Valued Byzantine Broadcast) in which the sender receives an ℓ-bit value $m \in \{0, 1\}^\ell$, and it wants to propagate this value to every one else. Consistency requires that all honest nodes output the same value; and validity requires that if the sender is honest, all honest nodes output the sender’s input value m.

Please design a protocol for achieving Multi-Valued Byzantine Broadcast,

Chapter 6

Blockchain and State Machine Replication

So far in our lectures, we have considered *single-shot* consensus. In practice, however, more often than not, it is not enough to reach consensus just once. Practical applications of consensus often require reaching consensus repeatedly over time. For example, in modern cryptocurrency systems such as Bitcoin and Ethereum, the underlying core abstraction is for a distributed set of nodes to maintain an *ever-growing public ledger* which records the sequence of all transactions that have taken place so far.

In this section, we will define a repeated consensus abstraction called a *blockchain*. The notion of a blockchain was classically called *state machine replication* in the long line of work in the distributed systems literature. In fact, before Bitcoin and Ethereum, state machine replication (or blockchain) protocols have been deployed by companies like Google and Facebook for more than a decade to replicate their computing infrastructue. The modern name "blockchain" was born together with Bitcoin and became popularized soon after.

6.1 Modeling Network Delay More Generally

So far in our textbook we have considered a "strongly" synchronous network model, where honest nodes' messages are delivered to honest recipients in the immediate next round. Starting in this section, we often adopt a more relaxed (i.e., general) model, where we assume that honest nodes' messages can take at most Δ rounds to be delivered to an honest recipient. The parameter Δ is often called the (maximum) *network delay*. More precisely, if an honest node sends a message m to an honest recipient in round r, then the recipient will have received the message by the beginning of round $r + \Delta$ if not earlier[1].

Defining this more general network model will lend to our discussions of various network timing assumptions in subsequent chapters. Note that the protocols we learned so far in this course, such as Dolev-Strong [DS83], are strongly synchronous protocols: essentially every Δ delay is renamed to be one round, and nodes only perform actions every Δ amount of time. Of course, not all consensus protocol must abide by this strongly synchronous restriction, even in the case when Δ is a-priori known.

6.2 Defining a Blockchain Protocol

In a blockchain protocol, a set of distributed nodes aim to agree on an ever-growing, linearly-ordered log of transactions. Rather than agreeing on one transaction at a time, often times the protocol would want to use *batching* to improve throughput — a block is exactly a batch of transactions (possibly attached with protocol metadata) and therefore a chain of blocks would be called a *blockchain*.

Roughly speaking, a blockchain protocol must satisfy two important security properties, *consistency* and *liveness*. Consistency requires that all nodes have the same view of the linearly ordered log — but since their network speeds may differ, we shall allow some nodes' logs to potentially grow a little faster than others. More specifically, in our formal definition below, we shall require that honest nodes' logs be prefixes of each other; but we do not require that all honest nodes' logs are exactly the same length in the same round. Liveness requires that if some honest node receives some transaction tx in some round r, then tx will appear in every honest node's "finalized log" by the end of round $r + T_{\text{conf}}$ where T_{conf} is often called the "confirmation time"[2]. Whenever a transaction appears in a node's finalized log, it is treated as having been confirmed, i.e., the transaction cannot be undone later. Below we define a blockchain abstraction more formally.

We assume that there are in total n nodes. The nodes receive transactions from an external environment, where transactions are represented as bit-strings that possibly need to abide by certain validity rules (e.g., with valid

[1]Alternatively, one can also model time as continuous rather than consisting of discrete rounds, this modeling choice is non-essential in understanding the results we shall present.

[2]$\mathsf{tx} \in \{0,1\}^*$ is a payload string. In some applications, there may be some validity or well-formedness rule on tx.

signatures from the coin's owner). The nodes each maintain a growing linearly ordered log of transactions. Henceforth let LOG_i^r denote node i's log in round r — LOG_i^r is also called a *finalized log*, i.e., every transaction or event contained in LOG_i^r cannot be undone later. A blockchain protocol must satisfy the following two requirements:

- *Consistency:* for any honest nodes i and j, and for any round numbers t and r, it must be that $\mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^r$ or $\mathsf{LOG}_i^t \succeq \mathsf{LOG}_j^r$. Here $\mathsf{LOG} \preceq \mathsf{LOG}'$ means that the former log is a prefix of the latter or they are the same.

- T_{conf}*-liveness:* If an honest node receives some transaction tx as input in some round r, then by the end of round $r + T_{\text{conf}}$, all honest nodes' local logs must include tx.

Clarifications on the definition. We make some further clarifications regarding the above definition:

- In the above consistency definition, the two honest nodes i and j are allowed to be the same, or different; and consistency must hold regardless.

- As we stipulated earlier, each (honest) node's local log is growing over time and *can never shrink* — this requirement is baked into the syntax and not reflected in the above consistency or liveness notions.

- The confirmation time T_{conf} can be a function of the number of nodes n, the maximum network delay Δ, and possibly other parameters. It is straightforward why T_{conf} might be a function of Δ since this is the maximum delay it takes for honest nodes to deliver messages to each other. Why can T_{conf} also depend on n? For example, jumping slightly ahead, we shall see how to construct a blockchain protocol through sequential composition of one-shot Byzantine Broadcast (BB) — in this case, if we instantiate the BB protocol with Dolev-Strong, then, as we have learned, the number of rounds will depend on n.

- If the blockchain protocol is deterministic (possibly in the ideal signature model), we would require that the above consistency and liveness properties hold deterministically. However, we will encounter randomized blockchain constructions later. If the protocol is randomized, we often require that regardless of corrupt nodes' strategy, the consistency and liveness properties must hold with probability $1 - \delta$ over the choice of the randomized execution. The term $\delta \in (0,1)$ is often referred to as the failure probability, and we typically want δ to be tiny.

- Note that the blockchain definition itself does not specify how the application-layer should process the messages included in the blockchain. The application layer can specify application-dependent validity rules for the format of the message to be included in each block: for example, a typical rule is that the message needs to be a set of transactions with valid signatures.

  Rules for dealing with double-spending are also application-specific decisions and therefore we do not include them in the blockchain abstraction. For example, if two or more transactions spending the same coin both appear in a node's finalized log, the application level can say, only the first one of them will be treated as valid, and the later ones will be discarded by the application semantics. In this case, when is it safe for a merchant to ship the goods to a buyer? The merchant should make sure that the corresponding transaction tx^* appears in the finalized log; and morever, there is no transaction before tx^* in the finalized log that spends the same coin.

6.3 Construction of a Blockchain Protocol from Byzantine Broadcast

In the remainder, we will show that assuming the existence of a PKI and digital signatures, we can construct a blockchain protocol by sequential composition of Byzantine Broadcast (BB). For convenience, we shall assume that the n nodes are numbered $0, 1, \ldots, n-1$. Here we will rely on the Multi-Valued variant of BB (see Exercise 3 of Chapter 3), and we assume that each BB instance runs in R number of rounds.

The blockchain construction is described below, where a new instance of BB is run every r rounds. Each instance of BB agrees on a block, and all blocks are sequentially concatenated to form a log.

Blockchain from sequential composition of BB

- In every round kR that is a multiple of R, i.e., where $k = 0, 1, 2, \ldots$, spawn a new BB protocol whose designated sender is defined to be $L_k := (k \mod n)$. Henceforth, the BB protocol spawned in round kR is denoted BB_k.

The designated sender $L_k := (k \mod n)$ of BB_k collects every transaction tx it has received as input, but that have not been included in its current log, i.e., $\mathsf{tx} \notin \mathsf{LOG}_{L_k}^{kR}$, and inputs the concatenation of all such transactions into BB_k.

- At any time, suppose $\mathsf{BB}_0, \mathsf{BB}_1, \ldots, \mathsf{BB}_{k'}$ have finished and their outputs are $m_1, m_2, \ldots, \mathsf{m}_{k'}$ respectively. The node's current output log is defined as the concatenation $m_1||m_2||\ldots||m_{k'}$ where "$||$" denotes concatenation.

Theorem 6. *Suppose that the BB protocol adopted realizes Multi-Valued Byzantine Broadcast for a network of n nodes and tolerating up to f corruptions, then the above blockchain construction satisfies consistency and $O(Rn)$-liveness also for the same n and f, where R denotes the round complexity of BB.*

Proof. Consistency of the blockchain is guaranteed due to consistency of the BB protocol. For liveness, observe that if a transaction tx is input to some honest node i in round r, then, it takes at most $(n+1)$ additional BB instances till the i becomes the designated sender again (note that the extra $+1$ is because in the worse case, the current BB instance has already started and i is the sender of the current BB instance). The next time i becomes the designated sender again in a BB instance, either tx is already included in i's log, or i will input tx (along with other transactions) into the BB, and by validity of the BB, every honest node's output in this BB will include tx.

What is the confirmation time of the above blockchain protocol?

Answer: in the above, since it takes at most $O(n)$ instances till the honest node i becomes the designated sender again in a BB instance, and every honest node's log includes tx after that BB instance, it is easy to see that the confirmation time is $O(Rn)$. □

Obviously, this is not the best approach to construct a blockchain protocol. However, it helps us to understand the relationship between BB and blockchains from a feasibility (but not necessarily efficiency) perspective.

Remark 8. The above construction of a blockchain protocol from BB does not require introducing any additional assumptions beyond those already used by the BB. That is, if the underlying BB requires a PKI and signatures, then the blockchain would require the same. If the underlying BB does not rely on setup assumptions, the blockchain protocol would need no setup too. However, in the next section, our construction of BB from blockchain requires the existence of a PKI and digital signatures.

6.4 Discussions

If blockchain is implied by BB from a feasibility perspective in the PKI setting, do we really need blockchain as a separate abstraction?

The answer is yes, and there are many reasons we care about blockchain as a separate abstraction. First, in practice, it is rarely a good idea to implement blockchains by sequentially composing BB. Although there are indeed better ways to sequentially compose BB than what's described above, the sequential composition approach makes it difficult to perform cross-instance pipelining, thereby making it difficult for performance optimizations. Not surprisingly, almost all blockchain protocols that have been deployed in practice (e.g., variants of PBFT [CL99] and Paxos [Lam98], and Bitcoin [Nak08, GKL15]) are "direct blockchain constructions" where there is no clear boundary that separates the blockchain into independent one-shot instances.

Besides performance concerns, the blockchain abstraction also seems useful for defining additional properties such as fairness and incentive compatibility [PS17a].