

14 Consensus

Chapter Contents

14.1	Overview	186
14.2	The Power of Consensus	190
14.3	The Phase King Algorithm	197
14.4	Reducing Consensus to Binary Consensus	201
14.5	Impossibility of Consensus with one Third of Faulty Nodes	204
14.6	Running Time Lower Bound	207

Learning Goals

- The Consensus problem and its relevance.
- A simple Binary Consensus algorithm tolerating $f < \frac{n}{3}$ Byzantine faults and running in $O(f)$ synchronous rounds.
- Corresponding lower bounds showing that $f < \frac{n}{3}$ is necessary and at least $f + 1$ rounds are required.
- An algorithm for multivalued Consensus based on a Binary Consensus algorithm.

Consensus is a key component in a distributed setting. The Consensus problem encapsulates the intention to bring all nodes to agree on a common input value that is a function of their individual input values.

In a fault free synchronous setting solving this task is trivial: each individual node collects the inputs of all other nodes and applies a given function on the set of values. In the presence of faults, the problem becomes challenging, and the ratio of faulty to non-faulty nodes determines our ability to solve the problem.

Consensus is a fundamental and extremely well-studied fault-tolerance primitive. There is a large number of variants of the problem, varying in terms of the model and the requirements on the solution. The common theme is the following question: In a system with faults, how can the correct nodes agree on a decision that is consistent with the given inputs? We will obtain intuition on the possibility of achieving consensus, as well as the fundamental limits on our ability to solve the problem.

14.1 Overview

Overcoming faults often boils down to finding a way for correct nodes to reach agreement on (some aspect of) the state of the system, despite inconsistent communication by faulty nodes. In the case of crash faults, this happens due to the failing node sending some information to a subset of the correct nodes, but not all of them. Byzantine nodes may exhibit more complicated behavior, but the key issue remains the same. Many tasks become *much* simpler when assuming that faulty nodes are forced to send the same information to all nodes. We refer to this as communication by *Safe Broadcast*.

-
- E14.1** Recall the algorithm from Section 9.5. Can you simplify it under the assumption of Safe Broadcast, i.e., a “propose” signal is always sent to all nodes, even if the sender is faulty?
- E14.2** Recall the algorithms from Section 10.2 and Section 10.3. Can you simplify them under the assumption of Safe Broadcast?
-

As a more general example, suppose the nodes want to vote on two possible courses of action. Each node has an opinion which course of action is best, based on its local state. Clearly, if all correct nodes have the same opinion, this choice should be taken. We also need to ensure that if opinions are mixed, all correct nodes agree on a single strategy, although any strategy proposed by a correct node is a valid choice. We cannot simply follow a pre-determined leader’s opinion, as a Byzantine leader could “lie” about their perception of the system. Nonetheless, if each node is forced to send the same message in each round to all other nodes, there is a trivial solution: have everyone broadcast

their opinion, replace any missing or invalid value by a default choice, and output the majority value.

E14.3 Convince yourself that this simple strategy achieves the stated goals provided that $f < \frac{n}{2}$, i.e., the majority of nodes is correct, and communication is by Safe Broadcast.

E14.4 Think about how the situation changes when faulty nodes may send conflicting messages to different nodes. How does the above strategy fail?

The above task turns out to be a—if not *the*—fundamental fault-tolerance primitive, and goes by the name of (binary) *Consensus*.

Definition 14.1 (Consensus). *Each node $v \in V_g$ is given an input $x_v \in X$. To solve Consensus, an algorithm must compute output values $o_v \in X$ at all correct nodes $v \in V_g$ meeting the following conditions:*

- **Agreement:** *There is $o \in X$ so that $o_v = o$ for all $v \in V_g$. We refer to o as the output of the Consensus algorithm.*
- **Validity:** *If there is $x \in X$ so that for all $v \in V_g$ it holds that $x_v = x$, then $o = x$.*
- **Termination:** *There is $r \in \mathbb{N}$ satisfying that each $v \in V_g$ terminates and outputs o_v by the end of round r .*

The algorithm has round complexity $R \in \mathbb{N}$, if it terminates in R rounds in all executions.

If $X = \{0, 1\}$, we refer to the task as Binary Consensus, and denote the input of node v by b_v (to indicate that it is a bit).

In the special case of Binary Consensus, Definition 14.1 guarantees that the output o equals the input of at least one correct node. If $|X| > 2$, this is not true anymore, and one could consider stronger validity conditions. However, the stated condition is the minimal meaningful requirement and at the same time sufficient for all applications in this book.

In this chapter, we will study this task in SMP. Note that once we have a (fault-tolerant!) way of simulating lock-step execution, we can use any Consensus algorithm for the synchronous model in the simulating system as well. This renders Consensus a highly useful primitive even under the strong assumption of synchronous operation.

E14.5 Think of objectives you can achieve using a Consensus primitive.

Throughout this chapter, we make the *additional assumption* that nodes are *uniquely labeled* $1, \dots, n$, where node v knows for each of its incoming ports what the label of the node w sending messages to this port is. To simplify

notation, we will simply refer to a node by its label, i.e., $V = \{1, \dots, n\}$. While this assumption can be relaxed, it is realistic in the kind of systems we consider in this book, as the designer chooses the interconnection network. All lower bounds and impossibility results discussed in this chapter hold under this additional assumption.

Consensus can be seen as a universal task. Maybe the best way of driving this message home is the following observation: in a fully connected system, we can use Consensus to *simulate* communication by Safe Broadcast.

Theorem 14.4. *Suppose that G is a complete graph on n nodes, X is a set of feasible messages, and \mathcal{A} is a Consensus algorithm for input set $X \cup \{\perp\}$ on G of round complexity R . Then we can simulate communication by Safe Broadcast for messages in X on G , where $R + 1$ rounds are required for each simulated round. Denoting by M the maximum message size of \mathcal{A} , the maximum message size of the simulation is $n \cdot \max\{M, \lceil \log(|X| + 1) \rceil\}$.*

Note that this simulation is inefficient, as it is slow—messages are received R rounds later—and causes large overheads in message size and computations. Nonetheless, from the perspective of what *can* be computed in the presence of faults, it is extremely powerful. It completely strips away the ability of faulty nodes to present inconsistent information to the system, which enables joint simulation of a single state machine by all nodes, even if its input is distributed among the nodes. In other words, we can limit the effect of faults to uncertainty about the local inputs to the joint state machine perceived at faulty nodes, so long as we can perform Consensus. This is clearly the best we can do under the Byzantine fault model, as a faulty node could claim different inputs, but otherwise follow the algorithm to the letter. Moreover, because this works for every possible state machine, this simulation is considered to be *universal*.

The above results are for fully connected graphs. However, they can be generalized to any known network topology if and only if it is $(2f + 1)$ -node connected.

Definition 14.2 (Node Connectivity). *Graph $G = (V, E)$ is k -node connected if after deleting up to $k - 1$ arbitrary nodes the remaining graph is still connected, i.e., there is a path between any pair of remaining nodes.*

Theorem 14.5. *Suppose G is an n -node graph that is not $(2f + 1)$ -node connected. Then Consensus with f Byzantine faults cannot be solved on G .*

Theorem 14.11. *Suppose that \mathcal{A} is a Consensus algorithm for a complete graph on n nodes with up to f Byzantine faults. Fix an arbitrary $(2f + 1)$ -node connected n -node graph G . Then there is an algorithm simulating \mathcal{A} on G .*

Its round complexity is at most factor n larger and it uses messages of size $O(n^2(M + \log n))$, where M is the maximum message size of \mathcal{A} .

Again, this simulation can be very inefficient, but it demonstrates that Consensus is a fundamental fault-tolerance primitive. We remark that analogous statements hold for crash faults and $(f + 1)$ -node connectivity.

-
- E14.6** Show that any Consensus algorithm for a fully connected graph with n nodes and up to f crash faults can be simulated in any $(f + 1)$ -node connected n -node graph.
- E14.7** Show that Consensus with f crash faults is impossible in graphs that are not $(f + 1)$ -connected.
-

After showing these statements demonstrating the relevance and fundamental nature of Consensus, in Section 14.3 we turn towards practical solutions in fully connected networks. Section 14.3 presents the Phase King algorithm, which solves Binary Consensus.

Theorem 14.18. *Algorithm 18 solves Binary Consensus in the synchronous model. It runs for $R(f) = 3(f + 1) \in O(f)$ rounds and correct nodes communicate exclusively by 1-bit broadcasts.*

At first glance, it might seem that the Phase King algorithm is of limited utility, as it can handle only the case of binary inputs. This intuition turns out to be false. In Section 14.4, we efficiently reduce the general case to Binary Consensus.

Theorem 14.22. *Suppose we are given a fully connected network G of n nodes and a Binary Consensus algorithm \mathcal{A} for it that tolerates $f < \frac{n}{3}$ Byzantine faults. Then Algorithm 19 is a Consensus algorithm on G for inputs from X that tolerates f faults. In addition to calling \mathcal{A} once as a subroutine, it runs for $2 \left\lceil \frac{\log |X|}{B} \right\rceil$ rounds, during which nodes broadcast messages of size B ; here, $B \in \mathbb{N}_{>0}$ can be chosen freely.*

Plugging in the Phase King algorithm, we obtain a Consensus algorithm that uses 1-bit broadcast messages only and runs for $O(f + \log |X|)$ rounds.

Corollary 14.3. *Algorithm 19 with Algorithm 18 as Binary Consensus subroutine solves Consensus with input set X in a fully connected network with $f < \frac{n}{3}$ Byzantine faults. It runs for $3(f + 1) + 2\lceil \log |X| \rceil \in O(f + \log |X|)$ rounds and correct nodes send 1-bit broadcast messages only.*

This result leads to two questions. First, is it possible to tolerate more faults? As might not surprise in light of the corresponding impossibility result for pulse synchronization (see Theorem 9.14), the answer is no.

Theorem 14.29. *If $3 \leq n \leq 3f$, Consensus with Byzantine faults cannot be solved.*

Second, can we further reduce the running time of the algorithm? The answer is a bit more involved here, but is also mostly no. The $O(\log |X|)$ term can be reduced by simply using larger messages, see Theorem 14.22. The $O(f)$ term from Binary Consensus, on the other hand, can at best be reduced by a factor of 3.

Theorem 14.35. *Consensus with f faults cannot be solved in fewer than $f + 1$ rounds, even if faults are restricted to crashing nodes.*

There is a lot more to say about the Consensus problem. For instance, a round complexity of $f + 1$ can be achieved, but the respective algorithms use larger messages and are computationally much more involved. Recursive solutions can reduce the overall number of bits that need to be sent. Randomized algorithms can terminate much faster and/or send even fewer bits, but this requires additional model assumptions and more complex algorithms. Last but not least, there is a huge body of work on manifold variants of the problem. We focus here on simple statements that demonstrate the importance of the task or are of practical interest in the setting of this book. ?? very briefly discusses some further results and provides the reader with entry points to the topic at large.

14.2 The Power of Consensus

E14.8 Think of how to achieve the following in SMP:

- Self-stabilizing FSM replication;
 - Distributed storage and logs (where with Consensus consistency can be re-established after a partition);
 - Self-stabilizing pulse synchronization (obtaining regular beats).
-

Theorem 14.4. *Suppose that G is a complete graph on n nodes, X is a set of feasible messages, and \mathcal{A} is a Consensus algorithm for input set $X \cup \{\perp\}$ on G of round complexity R . Then we can simulate communication by Safe Broadcast for messages in X on G , where $R + 1$ rounds are required for each simulated round. Denoting by M the maximum message size of \mathcal{A} , the maximum message size of the simulation is $n \cdot \max\{M, \lceil \log(|X| + 1) \rceil\}$.*

Proof. For each simulated round $r \in \mathbb{N}_{>0}$, we run n instances of \mathcal{A} , one for each node $i \in V$. To run all of these instances in parallel, in each round of the actual system, each node sends a vector of n many $\max\{M, \lceil \log(|X| + 1) \rceil\}$ -bit

words, one for each instance. To correctly map the words to the instances, node i sends the word belonging to the instance of node j in position j . To encode the potential messages, we fix some enumeration of the elements of X and \perp and transmit the binary encoding of the index in this enumeration, i.e., we need $\log(|X| + 1)$ bits to encode all elements of X and the special symbol \perp . We use \perp to denote that no message is sent, i.e., w.l.o.g. we can assume that the simulated algorithm *always* sends a message from a set of size $|X| + 1$. Observe that this means that all messages satisfy the stated bound on message size.

Assume that we have completed simulation of $r \in \mathbb{N}$ rounds within $r(R + 1)$ rounds of the actual system. To simulate round $r + 1$, in round $r(R + 1) + 1$ each node $v \in V_g$ computes and broadcasts the (encoded) message $x \in X$ it wants to send in simulated round $r + 1$, or \perp if it does not send a message. In rounds $r(R + 1) + 2, \dots, (r + 1)(R + 1)$, we run for each $v \in V_g$ an instance of \mathcal{A} with input set $X \cup \{\perp\}$, where node $w \in V_g$ uses as input the (alleged) message it received from v in round $r(R + 1) + 1$. If no (valid encoding of a) message is received by w , it uses \perp as default input value for the instance. At the end of round $(r + 1)(R + 1)$, node $w \in V_g$ feeds back the output of the Consensus instance to the simulation, as the message received from v in simulated round $r + 1$.

It remains to argue that carrying out the above procedure inductively results in correct simulation of communication by Safe Broadcast. The base case of the induction is trivial: each node can initialize the simulated state machine locally, anchoring the induction at $r = 0$. For the induction step, assume that $i \in V_g$ knows the state of its FSM at the end of round r and, if $r > 0$, the messages it received in round r . Then it can compute the state the FSM attains in round $r + 1$, as well as the message $x \in X \cup \{\perp\}$ it broadcasts. As correct nodes follow the protocol, each node $j \in V_g$ thus receives x from i in round $r(R + 1) + 1$. By validity and termination of the Consensus protocol, each $j \in V_g$ thus computes output x for the Consensus instance by the end of round $(r + 1)(R + 1)$.

This guarantee does not hold for faulty nodes $i \in V \setminus V_g$. However, agreement and termination of the Consensus protocol ensure that by the end of round $(r + 1)(R + 1)$, each $j \in V_g$ has determined the *same* output $x \in X \cup \{\perp\}$ for the instance of \mathcal{A} corresponding to i , implying that the simulation proceeds as if i had broadcast x to all nodes in round $r(R + 1) + 1$. Thus, the induction step succeeds, completing the proof. \square

14.2.1 Consensus in General Networks

Our first goal in this section is to establish that sufficient connectivity is necessary for solving consensus.

Theorem 14.5. *Suppose G is an n -node graph that is not $(2f + 1)$ -node connected. Then Consensus with f Byzantine faults cannot be solved on G .*

To prove Theorem 14.5, we need to reason similarly to Section 9.4. We assume for contradiction that a Consensus algorithm \mathcal{A} for a graph G with insufficient connectivity exists. Then we construct a sequence of executions, where any pair of adjacent executions satisfies that some correct nodes cannot distinguish the two executions. Accordingly, they need to behave in the same way, and in particular will compute the same output; the termination property forces them to eventually take a decision. Agreement then forces *all* correct nodes to output the same value, even those that could observe that the two executions are different. By induction, this forces all correct nodes in all executions to output the same value. However, our construction also ensures that there is an execution in which all correct nodes have inputs of 0 and one where they all have inputs of 1, respectively (w.l.o.g., we assume that $0, 1 \in X$). Thus, validity forces different outputs for these executions, implying the contradiction we seek.

There are some differences to the argument in Section 9.4, however. As we are operating in the synchronous model, the different executions all run at the same speed. In addition, Consensus is an “exact” task requiring to agree on the same input value—unlike pulse synchronization (Definition 9.6) or approximate agreement (Definition 10.1), for example. This simplifies our reasoning, providing us with a very elegant way of “encoding” all considered executions on G .

The idea of the proof is to run the Consensus algorithm on a helper graph H without faulty nodes, which has three copies of each node (each with the identity of the original node). While \mathcal{A} is not designed for H —it assumes fewer nodes and unique identities of nodes—we can show for each node indistinguishability of the execution on H from an execution on G with at most f faults. This means that the nodes must behave as if they were running the algorithm on G . Byzantine nodes can simulate this “imaginary” execution on H to determine which messages to send in the derived executions on G . We remark that we will make use of the same technique to prove Theorem 14.29.

We now execute the above plan. Fix an n -node graph $G = (V, E)$ that is not $(2f + 1)$ -node connected. Thus, removing a set of at most $2f$ nodes results in two disjoint non-empty sets $A, B \subseteq V$ such that there are no edges between the two sets. We split the removed set arbitrarily into two disjoint sets C and D of at most f nodes each. Thus, we have partitioned $V = A \dot{\cup} B \dot{\cup} C \dot{\cup} D$ such that (i) there are no edges between A and B , and (ii) $|C| \leq f$ and $|D| \leq f$. As you might expect, C and D are going to be the potentially faulty sets, and for each

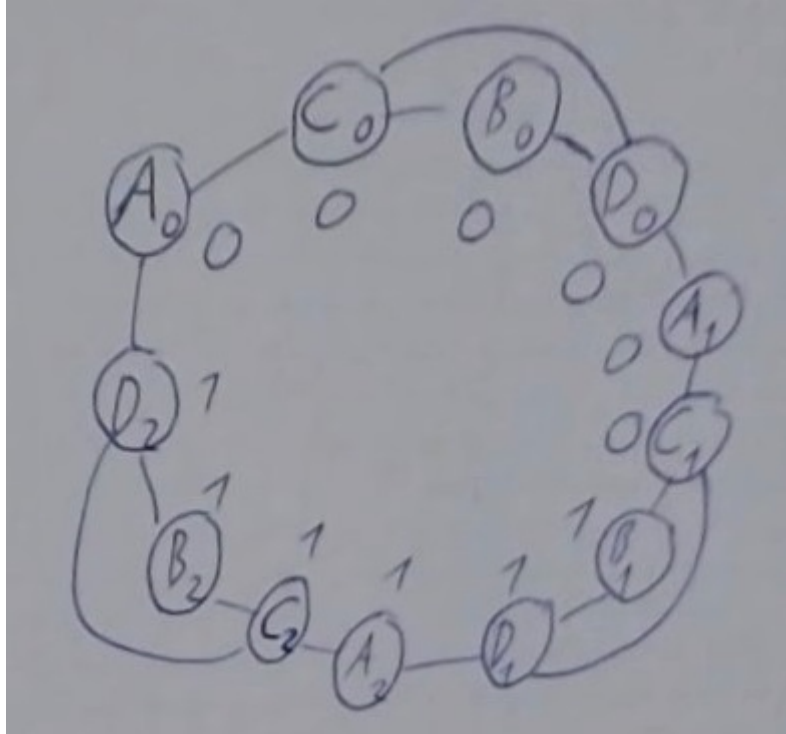


Figure 14.1

Abstract representation of the helper graph H . Nodes represent copies of sets, e.g. A_1 , A_2 , and A_3 are copies of A , and are labeled by the input all of their nodes receive. Edges indicate that nodes from these sets are connected by an edge if and only if the nodes they are copies of are connected by an edge in G .

adjacent pair of executions, one will have faulty node set C and one faulty node set D .

The construction of H is illustrated in Figure 14.1. We copy each set three times and arrange these sets as shown in the figure. Any two node copies are joined by an edge in H if and only if their originals in G and they belong to sets connected by an edge in Figure 14.1.

We first observe that we can execute \mathcal{A} on H , where we assume that the state machine of \mathcal{A} is allowed to halt whenever presented with a situation that cannot occur in an execution on G with at most f faults. By “halting” we mean that the FSM of the node may transition to an additional state \perp which it will not leave and in which it sends no messages. We remark that this is a mere

formality breaking down the proof into smaller units: we later show that this cannot happen.

Lemma 14.6. *Fix a graph G that is not $(2f + 1)$ -node connected and construct H as described above, where each node in H has the same label from $\{1, \dots, n\}$ as the node in G it is a copy of. Suppose \mathcal{A} is a Consensus algorithm on G that tolerates f Byzantine nodes. Then we can execute \mathcal{A} on H .*

Proof. We need to show that the state transition and output functions of a node v in H receive syntactically correct inputs. This means that (each!) node v with label $i \in \{1, \dots, n\}$ in H has the same number of neighbors as the node with label i in G , and that these neighbors have the same set of labels as the set of neighbors of i in G . As nodes in H have the same labels as their originals in H , this is equivalent to showing that if the node i in G has a neighbor with label j , then v has exactly one neighbor w that is a copy of j .

To see this, take a look at Figure 14.1. Observe that if S' is a copy of set S in G , for each neighboring set T of S in G , there is precisely one adjacent copy of the set in H . As we connect nodes in H by an edge if and only if the belong to adjacent copies of sets and their originals are joined by an edge in G , the claim follows. \square

Denote by \mathcal{E} the execution of \mathcal{A} on H in which each node receives as input the value its set is labeled by in Figure 14.1. By Lemma 14.6, \mathcal{E} is well-defined, although we cannot yet exclude that nodes may halt in \mathcal{E} without properly computing an output and terminating. With \mathcal{E} well-defined, we can proceed to show that there are several executions of \mathcal{A} on G that are indistinguishable from \mathcal{E} to a subset of the nodes.

Lemma 14.7. *For $k \in \{0, 1, 2\}$, consider the copies A' , B' , and C' of A , B , and C , respectively, that in Figure 14.1 appear after D_k in clockwise direction (e.g., for $k = 0$, $A' = A_1$, $B' = B_1$, and $C' = C_1$). Then there is an execution \mathcal{A}_{D_k} of \mathcal{A} on G with fault set D such that the unique node with label i in $A' \cup B' \cup C'$ cannot distinguish \mathcal{E} from \mathcal{E}_{D_k} at i .*

Proof. Denote for $i \in A \cup B \cup C$ by $v(i)$ the unique node with the same label in $A' \cup B' \cup C'$. For each $i \in A \cup B \cup C$, the indistinguishability relation we want to show defines its input (the same as of $v(i)$) and the message it receives in \mathcal{E}_{D_k} from a neighbor with label j in round $r \in \mathbb{N}$. Thus, our task is to show that the (faulty) nodes in D can send messages such that node $i \in A \cup B \cup C$ receives the same messages as $v(i)$.

We prove this by induction on the round number r , where the induction hypothesis is that for all $i \in A \cup B \cup C$, the FSMs of i and $v(i)$ are in the

same state at the end of round r and they receive the same messages in round r . The induction is anchored at $r = 0$, for which the claim is trivially satisfied because i and $v(i)$ have the same input in both executions (and no messages are sent in “round 0”). For the step from r to $r + 1$, observe that by the induction hypothesis, i and $v(i)$ transition to the same state in round $r + 1$ and thus send the same messages. Accordingly, i and $v(i)$ receive the same messages over incoming edges from neighbors with label $j \in A \cup B \cup C$. Consider a neighbor of $v(i)$ with label $j \in D$. There is a unique neighbor w of $v(i)$ in H with label j . In \mathcal{E}_{D_k} , the (faulty) node j simply sends the same message to i that $v(i)$ receives in \mathcal{E} . Hence, i and $v(i)$ receive the same messages in round r , completing the induction step. \square

Lemma 14.8. *For $k \in \{0, 1, 2\}$, consider the copies A' , B' , and D' of A , B , and D , respectively, that in Figure 14.1 appear after C_k in clockwise direction (e.g., for $k = 0$, $A' = A_1$, $B' = B_0$, and $D' = D_0$). Then there is an execution \mathcal{A}_{C_k} of \mathcal{A} on G with fault set C such that the unique node with label i in $A' \cup B' \cup D'$ cannot distinguish \mathcal{E} from \mathcal{E}_{C_k} at i .*

Proof. Analogous to the one of Lemma 14.7. \square

These indistinguishability results can be easily used to prove Theorem 14.5.

Corollary 14.9. *In \mathcal{E} , each node terminates and outputs the same value.*

Proof. Suppose first that v and w are nodes (i) from a set and the next set in clockwise direction in Figure 14.1, (ii) from A_k and B_k for some $k \in \{0, 1, 2\}$, or (iii) from B_k and $A_{k+1 \bmod 2}$ for some $k \in \{0, 1, 2\}$. A straightforward case distinction shows that we can always apply Lemma 14.8 or Lemma 14.7 for some $k \in \{0, 1, 2\}$ such that $v, w \in A' \cup B' \cup C'$. By Lemma 14.7, v and w cannot distinguish \mathcal{E} from an execution on G at the (non-faulty) nodes with the same labels. By agreement and termination of \mathcal{A} , they must terminate and output the same value.

Now consider the general case. Because A and B are non-empty, the same is true for A_k and B_k for all $k \in \{0, 1, 2\}$. Using cases (ii) and (iii) above inductively following the cycle in clockwise direction, we can show that all nodes in $\bigcup_{k=0}^2 A_k \cup B_k$ have the same output. All other nodes are in sets that can be reached from one these sets using case (i). \square

Corollary 14.10. *In \mathcal{E} , nodes in A_1 output 0, while nodes in A_2 output 1.*

Proof. If $v \in A_1$, we apply Lemma 14.8 with $k = 0$. By the lemma, v cannot distinguish \mathcal{E} from the execution \mathcal{E}_{C_0} at the node of the same label. In this execution, the correct nodes in $A \cup B \cup D$ have input 0, because nodes in

$A' \cup B' \cup D' = A_1 \cup B_1 \cup D_1$ have input 0 in \mathcal{E} . Therefore, by validity \mathcal{A} must output 0 at v . If $v \in A_2$, we apply Lemma 14.8 with $k = 1$ and reason analogously. \square

We now have all the pieces in place for showing Theorem 14.5.

Theorem 14.5. *Suppose G is an n -node graph that is not $(2f + 1)$ -node connected. Then Consensus with f Byzantine faults cannot be solved on G .*

Proof. Assume that there is a Consensus algorithm for a graph G that is not $(2f + 1)$ -node connected. Thus, we can construct H and an execution \mathcal{E} on H to which Corollaries 14.9 and 14.10 apply. By construction, A is non-empty, implying the same for A_1 and A_2 . Thus, by Corollary 14.10, there are two nodes in H that can only output different values in \mathcal{E} . This contradicts Corollary 14.9, which asserts that in \mathcal{E} all nodes terminate and output the same value. \square

We proceed to proving the matching positive result.

Theorem 14.11. *Suppose that \mathcal{A} is a Consensus algorithm for a complete graph on n nodes with up to f Byzantine faults. Fix an arbitrary $(2f + 1)$ -node connected n -node graph G . Then there is an algorithm simulating \mathcal{A} on G . Its round complexity is at most factor n larger and it uses messages of size $O(n^2(M + \log n))$, where M is the maximum message size of \mathcal{A} .*

Proof. We use communication in $G = (V, E)$ to simulate a fully connected network. We discuss how to simulate a single round; using this inductively, the general result follows.

Fix a pair of nodes $v, w \in V$. By Menger's theorem, there are $2f + 1$ (simple) node-disjoint paths from v to w . As G is fixed, these can be precomputed. We fix $2f + 1$ such paths and have all nodes on the paths store the next node on these paths together with the information that they are v - w paths (by marking down the labels of v and w) in the algorithm's code. When (correct) node v needs to send a message to node w , it will send it to each of the $2f + 1$ neighbors it has on these paths, indicating that the message is to be transmitted from v to w . These nodes then forward it to the next node on their respective path, and so on. Nodes will only forward messages from v to w if they are on one of the paths and receive them from their predecessor of their path.

After n rounds w evaluates whether it received from at least $f + 1$ of its predecessors on the $2f + 1$ fixed v - w paths the same message marked as communication from v to w . As $2(f + 1) > 2f + 1$, there can be at most one such message. If this is the case, w accepts this message as communication from v to w for this simulated round.

Hence, it remains to show that for $v, w \in V_g$, w accepts message m from v if and only if v decided to send m to w in this simulated round. To see this, note first that out of the $2f + 1$ paths from v to w , at most f can contain a faulty node. Hence, if v decides to send m , m will reach w via the at least $f + 1$ paths without a faulty node. As the paths are simple, they contain each node at most once. Hence, w will receive these at least $f + 1$ copies of m in time and accept m . On the other hand, if v does not send m , the above restrictions on which messages are forwarded imply that w will receive messages from its predecessors on at most f paths and accept no message.

We use the above strategy for all pairs of nodes concurrently. Nodes simply batch all messages they need to send to a neighbor in a given round together into a single message. This can be done as a list, where each entry contains the pair (v, w) the communication belongs to followed by the respective message. By using our knowledge of G and the simulated algorithm \mathcal{A} , each entry in the list can be made to have a fixed number of bits: $2\lceil \log n \rceil$ bits to encode the labels of v and w , followed by M many bits, where M is the maximum message size of \mathcal{A} . To ensure that the resulting message size bound holds even when faulty nodes send longer messages, correct nodes simply discard any messages that are inconsistent with this format. As each node can participate in at most $n(n - 1)$ paths as internal node, the total message size is thus bounded by $(n(n - 1) + 2f + 1)(2\lceil \log n \rceil + M) \in O(n^2(M + \log n))$. \square

We remark that the factor n slowdown and factor roughly n^2 increase in message size are pessimistic. There are many $(2f + 1)$ -connected graphs in which much better results are possible. However, there are examples where things get as bad as stated.

E14.9 For constant f and $M \geq \log n$, come up with a graph in which the speed and message size of the simulation are the best possible up to constant factors.

E14.10 How do the speed and message size of the simulation depend on f ?

14.3 The Phase King Algorithm

14.3.1 Uniform and Non-uniform Algorithms

In general, the round r by which all correct nodes have terminated may depend on the execution. However, we are interested in algorithms that guarantee termination within $R(f) \in \mathbb{N}$ rounds, regardless of the inputs and the behavior of faulty nodes. Analogously to Definition 14.1, we refer to $R(f)$ as the round complexity of “the” algorithm. This is a slight abuse of notation. A precise statement in accordance with Definition 14.1 might instead read “there

is a *family* of Consensus algorithms $\mathcal{A}(f)$ parametrized by $f \in \mathbb{N}$, such that $\mathcal{A}(f)$ has round complexity *at most* $R(f)$.” However, the first way of phrasing this is much more convenient, and there is no risk of misinterpretation: the dependency on f will be visible both in $R(f)$ and the pseudocode of the (family of) Consensus algorithm(s), and as each algorithm from the family is designed for a specific value of f , it “knows” f and, by extension, $R(f)$. The latter means that the algorithm can always be modified to delay termination at correct nodes until the end of round $R(f)$.

Similarly, one could argue that the state machine of each node depends on n as well, meaning that $\mathcal{A}(f)$ should actually be viewed as a family of algorithms $\mathcal{A}(f, n)$. Further, one could observe that the distinct identities of nodes, which the algorithm makes use of, mean that each node is running a *different* algorithm. This interpretation is the position that “algorithm” is a synonym for “state machine.” Instead, for the purpose of this book an algorithm is a template for all of the FSMs in the system.

Definition 14.12 (Uniform and Non-uniform Algorithms). *An algorithm is a template that, given all required parameters, can be translated into a collection of FSMs, one for each node of the system. If the template depends on a system parameter like, e.g., n , the algorithm is non-uniform in this parameter. Otherwise, it is uniform in this parameter.*

We now describe the *Phase King* algorithm, which solves Binary Consensus in fully connected systems. The algorithm lets correct nodes communicate by single-bit broadcasts. The algorithmic idea is as follows. Nodes maintain an opinion what the output should be, which initially is their input value. In each *phase* (i.e., iteration of the while loop in Algorithm 18), these opinions are updated, with the goal to reach agreement. This is facilitated by a leader – the phase king – which instructs all nodes to switch to opinion 0 or 1. A correct leader can help to overcome confusion spread by faulty nodes. However, this introduces the complication that the leader might be faulty.

A faulty leader may seek to sow discord, by instructing different nodes to switch to different values. Blindly following the instructions of the leader could hence fail to achieve agreement. It could also break validity. The strategy to overcome this has two components. First, the algorithm has $f + 1$ phases with different leaders, ensuring that there is at least one phase in which a correct leader ensures that all correct nodes adopt the same opinion. Second, in each phase nodes exchange their opinion, refusing to follow the advice of the leader if it *might* be the case that all correct nodes already agree. This ensures that existing agreement, which is achieved at the latest by the end of the first phase with a correct leader, will be maintained. By extension, this also guarantees

Algorithm 18 Phase King Algorithm at node $i \in V_g$. Note that for convenience the code assumes that i also receives its own broadcasts and all messages are consistent with the format required by the algorithm (i.e., invalid or missing messages by faulty nodes are replaced by valid default values).

```

1:  $op \leftarrow b_i$ 
2: for  $j = 1 \dots f + 1$  do
3:    $strong \leftarrow 0$ 
4:   broadcast  $op$  ▷ first broadcast
5:   if received at least  $n - f$  times  $op$  then
6:      $strong \leftarrow 1$ 
7:   end if
8:   if  $strong = 1$  then
9:     broadcast  $op$  ▷ second broadcast
10:  end if
11:  if received fewer than  $n - f$  times  $op$  then
12:     $strong \leftarrow 0$ 
13:  end if
14:  if  $i = j$  then ▷ king's broadcast
15:    if received at least  $f + 1$  times 0 then
16:      broadcast 0
17:    else
18:      broadcast 1
19:    end if
20:  end if
21:  if  $strong = 0$  and received  $b \in \{0, 1\}$  from node  $j$  then
22:     $op \leftarrow b$  ▷ if not sure, obey the king
23:  end if
24: end for
25: return  $op$ 

```

validity, as all correct nodes having the same input means that there is agreement right from the start.

The above strategy has to be slightly refined to deal with the complication that the leader must be able to give advice that ensures agreement. While it is straightforward to see that no two correct nodes with different opinions can refuse to change their opinion, the leader might not know the opinion from which correct nodes might refuse to deviate. This is addressed by a second broadcast, in which nodes with strong opinion repeat their support for this opinion. Only if it still looks like there might be existing agreement on this

opinion, nodes will remain stubborn—and in this case the leader will observe sufficient support for this opinion to correctly identify and support it.

Algorithm 18 shows the pseudocode of the algorithm. Recall that we refer to one iteration of the loop as a *phase*. We now formalize the intuition as individual statements and prove them, which will ultimately imply Theorem 14.18.

Lemma 14.13. *If, for some $b \in \{0, 1\}$ and all $i \in V_g$, $op_i = b$ at the beginning of a phase of Algorithm 18, then the same holds at the end of the phase.*

Proof. As $|V_g| \geq n - f$, each $i \in V_g$ will set strong to 1 after the first broadcast. Thus, in the second broadcast, $|V_g| \geq n - f$ nodes will broadcast b , and all correct nodes will maintain strong = 1. Thus, for no $i \in V_g$ op_i is changed by the king's broadcast. \square

Corollary 14.14. *Algorithm 18 satisfies validity.*

Proof. Suppose $b_i = b$ for some $b \in \{0, 1\}$ and all $i \in V_g$. Then each $i \in V_g$ initializes $op_i := b$, which by inductive use of Lemma 14.13 never changes. Thus each $i \in V_g$ outputs b . \square

Lemma 14.15. *Fix a phase $j \in \{1, \dots, f + 1\}$. There is a $b \in \{0, 1\}$ satisfying that each $i \in V_g$ holding strong = 1 after the first broadcast of phase j has $op_i = b$.*

Proof. Assuming towards a contradiction that the claim is false, for $b \in \{0, 1\}$ there are nodes $i_b \in V_g$, such that i_b receives at least $n - f$ messages b . As each correct node sends the same message to all nodes. Pick any $n - t$ votes for each of the two nodes. The union of these two sets contain at most $n + |V \setminus V_g|$ different votes, since only the faulty nodes may send different votes to both nodes. Thus,

$$2(n - f) \leq n + |V \setminus V_g| \leq n + f. \quad |V \setminus V_g| \leq f$$

This is equivalent to $n \leq 3f$, a contradiction. \square

Lemma 14.16. *Let phase $j \in \{1, \dots, f + 1\}$ satisfies that node $j \in V_g$. There is some $b \in \{0, 1\}$ so that $op_i = b$ for all $i \in V_g$ at the end of phase j .*

Proof. By Lemma 14.15, there is $b \in \{0, 1\}$ satisfying that each $i \in V_g$ with strong = 1 after the first broadcast in phase j has $op_i = b$. Accordingly, only faulty nodes may send a value different from b in the second broadcast of phase j . We distinguish two cases. The first is that there is no $i \in V_g$ with strong = 1 after the second broadcast. In this case, each $i \in V_g$ sets $op_i := b' \in \{0, 1\}$ after the king's broadcast of phase j , where b' is the value broadcasted by the king, i.e., node j .

The other case is that some node received $n - f$ times b in the second broadcast. At least $n - 2f$ of these messages must originate at correct nodes. Hence, the king (i.e., node j) received at least $n - 2f \geq f + 1$ times b , where we again use that $n > 3f$. On the other hand, there are at most f faulty nodes, so no node, including the king, did receive more than f times $1 - b$. It follows that the king broadcasts b in the king's broadcast. As $f < n - f$, each $i \in V_g$ with $op_i = 1 - b$ satisfies that $strong = 0$ after the second broadcast, and hence sets $op_i := b$ after the king's broadcast. \square

Corollary 14.17. *Algorithm 18 satisfies agreement.*

Proof. As there are at most f faults, $\{1, \dots, f + 1\} \cap V_g \neq \emptyset$. Let $j \in \{1, \dots, f + 1\} \cap V_g$. By Lemma 14.16, there is some $b \in \{0, 1\}$ so that $op_i = b$ for all $i \in V_g$ at the end of phase j . By inductive use of Lemma 14.13, these variables do not change any more. Hence all $i \in V_g$ output b . \square

Theorem 14.18. *Algorithm 18 solves Binary Consensus in the synchronous model. It runs for $R(f) = 3(f + 1) \in O(f)$ rounds and correct nodes communicate exclusively by 1-bit broadcasts.*

Proof. Agreement and validity hold by Corollary 14.17 and Corollary 14.14, respectively. The round complexity bound holds, because each of the $f + 1$ phases takes three rounds, one for each broadcast. That communication is by 1-bit broadcasts only is immediate from the pseudocode. \square

We remark that it might not be possible for nodes to *not* send a message, e.g. when communication is implemented by sampling whether the voltage on an incoming wire is 0 or 1 at some point during each synchronous round. In this case, either a 2-bit connection is required, where one wire indicates whether the sender has a strong opinion, or two rounds are used for encoding this information sequentially.

14.4 Reducing Consensus to Binary Consensus

We now describe the algorithm reducing Consensus with inputs from X to Binary Consensus. The algorithm assumes a fully connected network. Its pseudocode is given in Algorithm 19.

The main idea of the algorithm is to eliminate all but two possible choices and then apply the Binary Consensus algorithm to decide between these two. To do so, correct nodes will only support outputting a value different from the default choice of 0 (or any other fixed default value), if it might be the case that validity requires this. Accordingly, in the first broadcast nodes announce their

Algorithm 19 Consensus algorithm for input set X based on a Binary Consensus algorithm. The code is for node $i \in V_g$. For convenience, we assume that nodes also receive their own messages and that all received messages not adhering to the used format are replaced by valid default values.

```

1:  $c \leftarrow 0$                                  $\triangleright$  default output value, w.l.o.g.  $0 \in X$ 
2: broadcast  $x_i$                                  $\triangleright$  first broadcast
3: if received at least  $n - f$  times  $x_i$  then
4:    $c \leftarrow x_i$                              $\triangleright$  all correct nodes might have this input
5: end if
6:  $b \leftarrow 0$                                  $\triangleright$  input value for binary instance
7: broadcast  $c$                                  $\triangleright$  second broadcast
8: if received at least  $n - f$  times  $c' \in X \setminus \{0\}$  then
9:    $c \leftarrow c'$                              $\triangleright$  there can be at most one such  $c'$ 
10:   $b \leftarrow 1$                                  $\triangleright c'$  is known to all nodes
11: else if received at least  $f + 1$  times  $c' \in X \setminus \{0\}$  then
12:    $c \leftarrow c'$                              $\triangleright$  there can be at most one such  $c'$ 
13: end if
14: participate in binary consensus instance with input  $b$ 
15: if output is 1 then
16:   return  $c$                                  $\triangleright$  can only happen if everyone knows  $c$ 
17: else
18:   return 0
19: end if

```

inputs. They adopt them as candidate value c if and only if they see at least $n - f$ nodes claiming to have this input (counting themselves as well). As due to the condition that $n > 3f$ only one value $c \neq 0$ can have sufficient support to pass this simple check, this already eliminates all candidate values different from c and 0.

However, similarly to the Phase King algorithm, there is a catch. While only two candidate values remain, we cannot guarantee that all correct nodes can *determine* which value c passed the test at some correct node. Also very similarly to the Phase King algorithm, we resolve this issue by performing a second broadcast. As in the second broadcast the only candidate value different from 0 supported by correct nodes is c , no other value can be received more than f times. Thus, if *any* correct node sets b to 1, *each* correct node receives c at least $n - 2f > f$ times in the second broadcast, while not receiving more than f messages supporting any non-zero $c' \neq c$. Hence, if any correct node uses input 1 in the Binary Consensus instance, all nodes agree on c . Together with

these observations, the agreement property of the Binary Consensus routine implies agreement for Algorithm 19. At the same time, validity is maintained: if all correct nodes have the same input, it will be received at least $n - f$ times both in the first and second broadcast, all correct nodes will use input 1 for the Binary Consensus routine, and validity of the Binary Consensus routine implies validity of Algorithm 19.

We now formalize the above intuition and prove Theorem 14.22.

Lemma 14.19. *If the Binary Consensus algorithm called in Algorithm 19 satisfies validity, so does Algorithm 19.*

Proof. Assume first that all correct nodes have input $x \in X \setminus \{0\}$. Then each $v \in V_g$ receives x from at least $|V_g| \geq n - f$ nodes in the first broadcast. Consequently, the same applies in the second broadcast. Thus, each $v \in V_g$ uses input 1 for the Binary Consensus instance. By validity of the Binary Consensus algorithm, the output of this instance is 1. Therefore, each $v \in V_g$ outputs c_v , which by the above observations equals x .

Now consider the case that all correct nodes have input 0. Then in the second broadcast no correct node will receive more than f messages $c' \in X \setminus \{0\}$. Hence, each correct node uses input 0 for the Binary Consensus instance. By validity of the Binary Consensus algorithm, the output of this instance is 0. Thus, each $v \in V_g$ outputs 0. \square

The proof of the following lemma is analogous to that of Lemma 14.15.

Lemma 14.20. *If $n > 3f$, there is at most one value $c \in X \setminus \{0\}$ sent by correct nodes in the second broadcast.*

E14.11 Prove Lemma 14.20.

Lemma 14.21. *If $n > 3f$ and the Binary Consensus algorithm called in Algorithm 19 satisfies agreement and validity, Algorithm 19 satisfies agreement.*

Proof. By Lemma 14.20, there is a value $c \in X \setminus \{0\}$ such that no correct node broadcasts a value different from both c and 0 in the second broadcast. We distinguish two cases. First, assume that some $v \in V_g$ uses input 1 for the Binary Consensus instance. Thus, it received $n - f$ messages supporting some value $c' \neq 0$. Of these messages, $n - 2f > f$ must originate from correct nodes. Thus, $c' = c$ and each correct node receives c at least $f + 1$ times in the second broadcast. By agreement of the Binary Consensus algorithm, either all correct nodes output 1 or all output 0 for the Binary Consensus instance. We conclude that they either all output c or all output 0, satisfying agreement.

The second case is that no correct node uses input 1 for the Binary Consensus instance. By validity of the Binary Consensus algorithm, the instance thus outputs 0. We conclude that all correct nodes output 0, i.e., agreement holds in this case as well. \square

Theorem 14.22. *Suppose we are given a fully connected network G of n nodes and a Binary Consensus algorithm \mathcal{A} for it that tolerates $f < \frac{n}{3}$ Byzantine faults. Then Algorithm 19 is a Consensus algorithm on G for inputs from X that tolerates f faults. In addition to calling \mathcal{A} once as a subroutine, it runs for $2 \left\lceil \frac{\log |X|}{B} \right\rceil$ rounds, during which nodes broadcast messages of size B ; here, $B \in \mathbb{N}_{>0}$ can be chosen freely.*

Proof. Validity and agreement hold by Lemmas 14.19 and 14.21, respectively. Apart from calling \mathcal{A} , the algorithm has correct nodes perform two broadcasts of values from X . This requires to broadcast $\lceil \log |X| \rceil$ bits, which using B -bit broadcasts takes $\left\lceil \frac{\log |X|}{B} \right\rceil$ rounds each. \square

14.5 Impossibility of Consensus with one Third of Faulty Nodes

We now show that Consensus cannot be solved if $3 \leq n \leq 3f$. We use the same approach as for Theorem 14.5. Assume for contradiction that there is an algorithm \mathcal{A} solving Consensus on an n -node graph $G = (V, E)$ in the presence of up to f faults, such that $3 \leq n \leq 3f$. Thus, we can partition $V = A \dot{\cup} B \dot{\cup} C$ such that $1 \leq |V_k| \leq f$ for $k \in \{1, 2, 3\}$. We construct a new graph H by making two copies of each set, resulting in the partition $A_0 \dot{\cup} B_0 \dot{\cup} C_0 \dot{\cup} A_1 \dot{\cup} B_1 \dot{\cup} C_1$. Two copies of nodes are joined by an edge in H if and only if their was an edge between their originals in G and they belong to sets connected by an edge in Figure 14.2. Nodes in the sets indexed with 0 receive input 0, nodes in sets with index 1 receive input 1.

We first observe that we can execute \mathcal{A} on H , where we assume that \mathcal{A} is allowed to halt whenever presented with a situation that cannot occur in an execution on G with at most f faults. By “halting” we mean that the FSM of the node may transition to an additional state \perp which it will not leave and in which it sends no messages. The following statement is analogous to Lemma 14.23.

Lemma 14.23. *Assume that $3 \leq n \leq 3f$ and fix an n -node graph G . Construct H as described above, where each node in H has the same label from $\{1, \dots, n\}$ as the node in G it is a copy of. Suppose \mathcal{A} is a Consensus algorithm on G that tolerates f Byzantine nodes. Then we can execute \mathcal{A} on H .*

E14.12 Verify that the same arguments as used to prove Lemma 14.6 apply here.

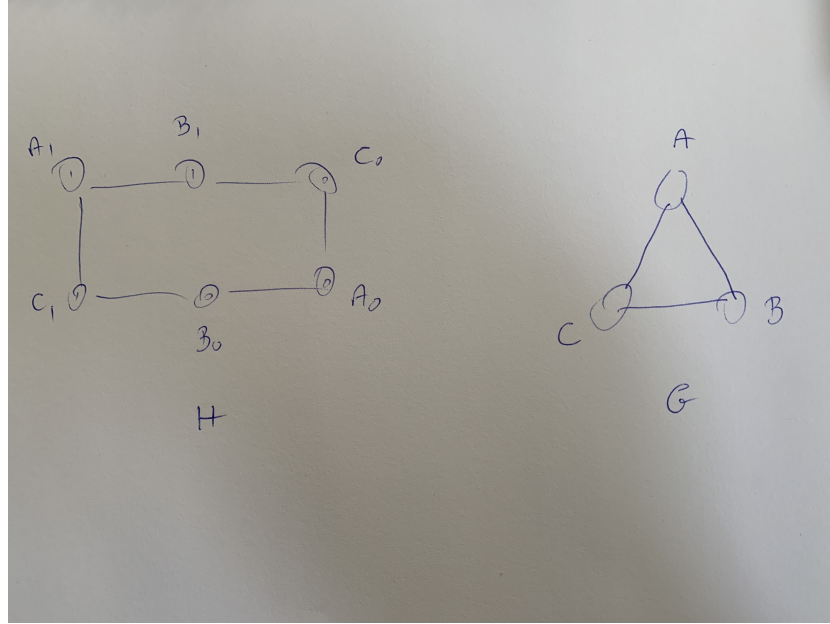
**Figure 14.2**

Illustration of 6 nodes confusion strategy

Denote by \mathcal{E} the execution of \mathcal{A} on H in which each node receives as input the value its set is labeled by in Figure 14.2. By Lemma 14.23, \mathcal{E} is well-defined, although we cannot yet exclude that nodes may halt in \mathcal{E} without properly computing an output and terminating. With \mathcal{E} well-defined, we can proceed to show that there are several executions of \mathcal{A} on G that are indistinguishable from \mathcal{E} to a subset of the nodes. This statement is analogous to Lemma 14.7.

Lemma 14.24. *For $k \in \{0, 1\}$, consider the copies B' and C' of B and C , respectively, that in Figure 14.2 appear after A_k in clockwise direction (e.g., for $k = 0$, $B' = B_0$, $C' = C_0$). Then there is an execution \mathcal{A}_{A_k} of \mathcal{A} on G with fault set A such that the unique node with label i in $B' \cup C'$ cannot distinguish \mathcal{E} from \mathcal{E}_{A_k} at i .*

E14.13 Verify that the same arguments as used to prove Lemma 14.7 apply here.

Symmetric statements hold for B and C .

Lemma 14.25. *For $k \in \{0, 1\}$, consider the copies C' and A' of C and A , respectively, that in Figure 14.2 appear after B_k in clockwise direction. Then*

there is an execution \mathcal{A}_{B_k} of \mathcal{A} on G with fault set B such that the unique node with label i in $C' \cup A'$ cannot distinguish \mathcal{E} from \mathcal{E}_{B_k} at i .

Proof. Analogous to the proof of Lemma 14.24. \square

Lemma 14.26. For $k \in \{0, 1\}$, consider the copies A' and B' of A and B , respectively, that in Figure 14.2 appear after C_k in clockwise direction. Then there is an execution \mathcal{A}_{C_k} of \mathcal{A} on G with fault set C such that the unique node with label i in $A' \cup B'$ cannot distinguish \mathcal{E} from \mathcal{E}_{C_k} at i .

Proof. Analogous to the proof of Lemma 14.24. \square

These indistinguishability results can be easily used to prove Theorem 14.5.

Corollary 14.27. In \mathcal{E} , each node terminates and outputs the same value.

Proof. Suppose first that v and w are nodes from a set and the next set in clockwise direction in Figure 14.2. A straightforward case distinction shows that we can always apply Lemma 14.24, Lemma 14.25, or Lemma 14.26 for some $k \in \{0, 1\}$ such that both v and w have corresponding nodes in G that are not faulty. As neither v nor w can distinguish \mathcal{E} from an execution on G at the (non-faulty) nodes with the same labels, by agreement of \mathcal{A} they must terminate and output the same value.

Now consider the general case. Because A , B , and C are each non-empty, the same is true for A_k , B_k , and C_k for $k \in \{0, 1, 2\}$. Using the above special case inductively following the cycle in clockwise direction, we can show that all nodes in $\bigcup_{k=0}^2 A_k \cup B_k \cup C_k$ have the same output. \square

Corollary 14.28. In \mathcal{E} , nodes in A_0 output 0, while nodes in A_1 output 1.

Proof. If $v \in A_0$, we apply Lemma 14.26 with $k = 0$. By the lemma, v cannot distinguish \mathcal{E} from the execution \mathcal{E}_{C_0} at the node of the same label. In this execution, the correct nodes in $A \cup B$ have input 0, because nodes in $B' \cup C' = B \cup C$ have input 0 in \mathcal{E} . Therefore, by validity \mathcal{A} must output 0 at v . For $v \in A_1$, we can reason analogously using $k = 1$. \square

Theorem 14.29. If $3 \leq n \leq 3f$, Consensus with Byzantine faults cannot be solved.

Proof. Assume that there is a Consensus algorithm for an n -node graph G that tolerates f faults, where $3 \leq n \leq 3f$. Thus, we can construct H and an execution \mathcal{E} on H to which Corollaries 14.27 and 14.28 apply. By construction, A is non-empty, implying the same for A_1 and A_2 . Thus, by Corollary 14.28,

there are two nodes in H that can only output different values in \mathcal{E} . This contradicts Corollary 14.27, which asserts that in \mathcal{E} all nodes terminate and output the same value. \square

14.6 Running Time Lower Bound

We now prove that any (deterministic) Consensus algorithm must run for at least $f + 1$ rounds in the worst case. In fact, we will show this for a much weaker fault model: crash faults.

Definition 14.30 (Crash Faults). *If node $v \in V$ crashes in round $r \in \mathbb{N}_{>0}$, it operates like a non-faulty node in rounds $1, \dots, r - 1$, does nothing at all in rounds $r + 1, r + 2, \dots$, and in round r sends an arbitrary subset of the messages it would send according to the algorithm.*

Crashing nodes fail in a well-organized fashion. They do not lie, we do not have to care about getting them up to speed again later, and by requiring that nodes always send messages to each other in each round, nodes will learn that a node failed from not receiving a message from the node. None of this affects the worst-case running time lower bound in any way—regardless of whether we consider Byzantine or crash faults, the bound of $f + 1$ rounds turns out to be tight.

We will show this lower bound now by a straightforward inductive argument. The key ingredient is the following definition.

Definition 14.31 (Pivotal Nodes). *Observe that an execution in the synchronous model with crash faults is fully determined by specifying the node inputs and, for each node, whether it crashes and, if so, in which round and which of its messages of this round get sent. Given an execution \mathcal{E} of a Consensus algorithm with at most $n - 2$ crash faults and a node $v \in V$ that does not crash in \mathcal{E} , we call v pivotal in round r (of \mathcal{E}) if changing \mathcal{E} by crashing v in round r of \mathcal{E} without v sending any messages results in an execution with a different output (the execution does have an output, because at least one node does not crash).*

In order to anchor the induction, we need to show that such nodes exist.

Lemma 14.32. *There is a fault-free execution with a node that is pivotal in round 1.*

Proof. Consider executions \mathcal{E}_i , $i \in [n + 1]$, which are fault-free with node $j \in V$ having input 0 if $j > i$ and input 1 otherwise. By validity, \mathcal{E}_0 has output 0 and \mathcal{E}_1 has output 1. Thus, there must be some $i \in [n]$ with the property that \mathcal{E}_i has output 0 and \mathcal{E}_{i+1} has output 1. Consider the execution \mathcal{E}' obtained by crashing node $i + 1$ in round 1, without $i + 1$ getting any messages out. If \mathcal{E}'

has output 0, $i + 1$ is pivotal in round 1 of execution \mathcal{E}_{i+1} ; if \mathcal{E}' has output 1, $i + 1$ is pivotal in round 1 of execution \mathcal{E}_i . \square

The induction step works the same way, except that the inputs are replaced by, for each node, the decision whether the pivotal node crashing in round r sends a message to the node or not.

Lemma 14.33. *Suppose $0 \leq f \leq n - 3$ and \mathcal{E} is an execution with f failing nodes, one in each round $1, \dots, f$, that has a pivotal node in round $f + 1$. Then there is an execution \mathcal{E}' which differs from \mathcal{E} only in that this pivotal node crashes in round $f + 1$ and satisfies that there is a pivotal node in round $f + 2$.*

Proof. For $i \in [n + 1]$, define \mathcal{E}_i by having the pivotal node of \mathcal{E} crash in round $f + 1$ and succeed in sending its message for that round to node $j \in \{1, \dots, n\}$ if and only if $j > i$. As we crashed a pivotal node, we know that \mathcal{E}_0 and \mathcal{E}_n have different outputs. Thus, there must be some i for which \mathcal{E}_i and \mathcal{E}_{i+1} have different outputs. Now consider the executions \mathcal{E}'_i and \mathcal{E}'_{i+1} obtained from \mathcal{E}_i and \mathcal{E}_{i+1} , respectively, in which node $i + 1$ crashes in round $f + 2$ without sending any messages. The only difference between these executions is whether $i + 1$ received the message from the crashing node in round $f + 1$ or not; as $i + 1$ does not get a message out telling anyone of this difference, the outputs of \mathcal{E}'_i and \mathcal{E}'_{i+1} are the same. Thus, either \mathcal{E}_i and \mathcal{E}'_i have different outputs or \mathcal{E}_{i+1} and \mathcal{E}'_{i+1} have different outputs, i.e., either $i + 1$ is pivotal in round $f + 2$ of \mathcal{E}_i or it is pivotal in round $f + 2$ of \mathcal{E}_{i+1} . \square

Corollary 14.34. *Any Consensus algorithm has an execution with a pivotal node in round $\min\{f, n - 2\}$.*

Theorem 14.35. *Consensus with f faults cannot be solved in fewer than $f + 1$ rounds, even if faults are restricted to crashing nodes.*

Proof. Consider the execution \mathcal{E} with a pivotal node in round $\min\{f, n - 2\}$ guaranteed to exist by Corollary 14.34, as well as the execution \mathcal{E}' obtained by crashing the pivotal node in round $\min\{f, n - 2\}$. The two executions have different output, but at all nodes but the pivotal one, the only difference to be observed before round $\min\{f + 1, n - 1\}$ is whether the respective message from the pivotal node in round $\min\{f, n - 2\}$ was received or not.

Assume for contradiction that, in both executions, the (at least two) non-crashed nodes terminate by the end of round $\min\{f, n - 2\}$. Let $i, j \in V$ be two such nodes crashing in neither \mathcal{E} nor \mathcal{E}' . These nodes must also terminate in the execution \mathcal{E}'' in which the pivotal node sends its message to i , but does not send its message to j : To i , this execution is indistinguishable from \mathcal{E} before

round $\min\{f + 1, n - 1\}$, and for j it is indistinguishable from \mathcal{E}' . However, this indistinguishability implies that they also output the same values as in \mathcal{E} and \mathcal{E}' , respectively. As these values differ, this violates agreement and hence is a contradiction. We conclude that our assumption must be wrong and there is some execution of the algorithm in which not all nodes terminate before round $\min\{f + 1, n - 1\}$. \square

Bibliography

- [1] Hopkins, A. L., T. B. Smith, and J. H. Lala. 1978. Ftmp – a highly reliable fault-tolerant multiprocess for aircraft. *Proceedings of the IEEE* 66 (10): 1221–1239. doi:10.1109/PROC.1978.11113.
- [2] Kopetz, H. 2003. Fault containment and error detection in the time-triggered architecture. In *The sixth international symposium on autonomous decentralized systems, 2003. isads 2003.*, 139–146. doi:10.1109/ISADS.2003.1193942.
- [3] Pease, M., R. Shostak, and L. Lamport. 1980. Reaching agreement in the presence of faults. *J. ACM* 27 (2): 228–234. doi:10.1145/322186.322188. <http://doi.acm.org/10.1145/322186.322188>.
- [4] Srikanth, T. K., and Sam Toueg. 1987. Optimal clock synchronization. *J. ACM* 34 (3): 626–645. doi:10.1145/28869.28876. <https://doi.org/10.1145/28869.28876>.