Chapter 5

Distributed Consensus with Link Failures

In this and the next two chapters, we study problems of *reaching consensus* in a distributed network. In such problems, each of the processes in the network begins with an initial value of a particular type and is supposed to eventually output a value of that same type. The outputs are required to be the same—the processes must *agree*—even though the inputs can be arbitrary. There is generally a *validity condition* describing the output values that are permitted for each pattern of inputs.

When there are no failures of system components, consensus problems are usually easy to solve, using a simple exchange of messages. To make matters more interesting, the problems are usually considered in settings that include failures. In this chapter, we consider basic consensus problems in the presence of communication failures, while in Chapter 6, we consider process failures. Chapter 7 contains some variations on the basic problems, also in the presence of process failures.

Consensus problems arise in many distributed computing applications. For example, processes may attempt to reach agreement on whether to commit or abort the results of a distributed database transaction. Or processes may try to agree on an estimate of an airplane's altitude based on the readings of multiple altimeters. Or they may attempt to agree on whether to classify a system component as faulty, given the results of separate diagnostic tests performed by separate processes.

The particular consensus problem that we present in this chapter is called the *coordinated attack problem*; it is a fundamental problem of reaching consensus in a setting where messages may be lost. We begin by presenting a basic

impossibility result for deterministic systems, and then explore the possibilities for randomized solution. We show that the problem can be solved by a randomized algorithm, with a certain (substantial) probability of error. Moreover, that probability of error turns out to be unavoidable.

5.1 The Coordinated Attack Problem—Deterministic Version

We begin with an informal (in fact, ambiguous) problem statement, in terms of a battlefield scenario.

> Several generals are planning a coordinated attack from different directions, against a common objective. They know that the only way the attack can succeed is if all the generals attack; if only some of the generals attack, their armies will be destroyed. Each general has an initial opinion about whether his army is ready to attack.
>
> The generals are located in different places. Nearby generals can communicate, but only via messengers that travel on foot. However, messengers can be lost or captured, and their messages may thus be lost. Using only this unreliable means of communication, the generals must manage to agree on whether or not to attack. Moreover, they should attack if possible.
>
> (We suppose that the "communication graph" of generals is undirected and connected, and that all of the generals know the graph. We also assume that there is a known upper bound on the time it takes for a successful messenger to deliver a message.)

If all the messengers are reliable, then all the generals can send messengers to all the other generals (possibly in several hops), saying whether or not they are willing to attack. After a number of "rounds" equal to the diameter of the "communication graph," all the generals will have all of this information. Then they can all apply a commonly agreed-upon rule to make the same decision about attacking: for example, they can decide to attack exactly if all the generals want to do so.

In a model in which messengers may be lost, this easy algorithm does not work. It turns out that this is not just a problem with this algorithm: we show that there is no algorithm that always solves this problem correctly.

The real computer science problem behind this description is the commit problem for distributed databases. This problem involves a collection of processes that have participated in the processing of a database transaction. After

this processing, each process arrives at an initial "opinion" about whether the transaction ought to be *committed* (i.e., its results made permanent and released for the use of other transactions) or *aborted* (i.e., its results discarded). A process will generally favor committing the transaction if all its local computation on behalf of that transaction has been successfully completed, and will favor aborting the transaction otherwise. The processes are supposed to communicate and eventually to agree on one of the outcomes, *commit* or *abort*. If possible, the outcome should be *commit*.

Before proving the impossibility result, we state the problem more formally and remove the ambiguities. We consider n processes indexed by $1, \ldots, n$, arranged in an arbitrary undirected graph network, where each process knows the entire graph, including the process indices. Each process starts with an input in $\{0, 1\}$ in a designated state component. We use 1 to denote "attack," or *commit*, and 0 to denote "don't attack," or *abort*. We use the same synchronous model that we have been working with so far, except that now we allow any number of messages to be lost during the course of an execution. (See Section 2.2 for the definition.) The goal is for all the processes to eventually output decisions in $\{0, 1\}$, by setting special *decision* state components to 0 or 1. There are three conditions imposed on the decisions made by the processes:

Agreement: No two processes decide on different values.

Validity:

1. If all processes start with 0, then 0 is the only possible decision value.
2. If all processes start with 1 and all messages are delivered, then 1 is the only possible decision value.

Termination: All processes eventually decide.

The agreement and termination requirements are the natural ones. The validity requirement is just one possibility—there are several useful alternatives. Validity conditions, in general, express the notion that the value decided upon should be "reasonable"; for instance, in this case, the trivial protocol that always decides 0 is ruled out by part 2 of the validity requirement. The particular validity condition we have stated above is quite weak: for example, if even one process starts with 1, the algorithm is allowed to decide 1, and if all processes start with 1 and even one message is lost, the algorithm is allowed to decide 0. The weak formulation is appropriate because our main focus in this chapter is on impossibility results. It turns out that even this weak version of the problem is impossible to solve in any graph with two or more nodes.

We prove the impossibility result for the special case of two nodes connected by one edge. We leave it as an exercise for you to show that impossibility for this case implies impossibility for any graph with two or more nodes. In this proof, we use the formal definitions of executions and indistinguishability ($\sim$) given in Chapter 2.

Theorem 5.1 *Let G be the graph consisting of nodes 1 and 2 connected by a single edge. Then there is no algorithm that solves the coordinated attack problem on G.*

Proof. By contradiction. Suppose a solution exists, say algorithm A. Without loss of generality, we may assume that, for each process, there is only one start state containing each input value; this implies that the system has exactly one execution for a fixed assignment of inputs and fixed pattern of successful messages. Also without loss of generality, we may assume that both processes send messages at every round in A, since we can always force them to send dummy messages.

Let α be the execution that results when both processes start with value 1 and all messages are delivered. By the termination requirement, both eventually decide, and by the validity condition, part 2, both decide on the value 1. Suppose that both decide within r rounds. Now let α_1 be the same as α, except that all messages after the first r rounds are lost. In α_1, both processes also decide on 1 within r rounds. The communication pattern in α_1 is represented in Figure 5.1. The edges represent messages that are delivered; messages sent but not delivered are simply not drawn.

Starting from α_1, we now construct a series of executions, each of them indistinguishable from its predecessor in the series with respect to one of the processes; it will follow that all of these executions must have the same decision value.

Let α_2 be the execution that is the same as α_1, except that the last (round r) message from process 1 to process 2 is not delivered (see Figure 5.2). Then, although process 2 may go to different states after round r in executions α_1 and α_2, this difference never gets communicated to process 1; therefore $\alpha_1 \overset{1}{\sim} \alpha_2$. Since process 1 decides 1 in α_1, it also decides 1 in α_2. By the termination and agreement properties, process 2 also (eventually) decides 1 in α_2.

Next, let α_3 be the same as α_2, except that the last message from process 2 to process 1 is lost. Since $\alpha_2 \overset{2}{\sim} \alpha_3$, process 2 decides 1 in α_3, and by termination and agreement, so does process 1.

Continuing in this way, by alternately removing the last message from process 1 and from process 2, we eventually reach an execution α' in which both processes

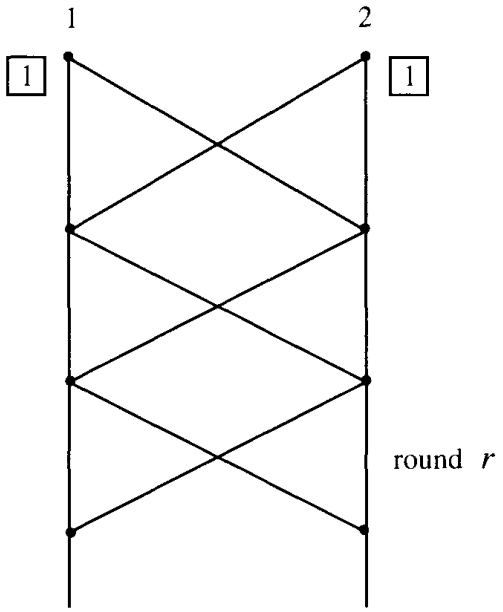

Figure 5.1: Pattern of message exchanges in execution α_1.

start with 1 and no messages are delivered. By the same reasoning as above, both processes are forced to decide 1 in this case.

But now consider the execution α'' in which process 1 starts with 1 but process 2 starts with 0, and no messages are delivered. Then $\alpha'' \overset{1}{\sim} \alpha'$, and hence process 1 still decides 1 in α'', and so does process 2, by termination and agreement. But $\alpha'' \overset{2}{\sim} \alpha'''$, where α''' is the execution in which both processes start with 0 and no messages are delivered. So process 2 decides 1 in α'''. But this yields a contradiction, because the validity condition, part 1, requires that both processes decide 0 in α'''. □

Theorem 5.1 describes a fundamental limitation on the capabilities of distributed networks. It suggests that there is little that can be done to solve basic consensus problems such as the distributed database commit problem in the face of unreliable communication. However, some versions of this problem must be solved in real systems. In order to cope with the limitation of Theorem 5.1, it is necessary to strengthen the model or relax the problem requirements.

One approach is to make some probabilistic assumptions about the loss of messages, while keeping the processes deterministic. Then we must allow for some possibility of violating the agreement and/or validity condition. We leave the development of an algorithm for this setting for an exercise. A second approach is to allow the processes to use randomization, again allowing some

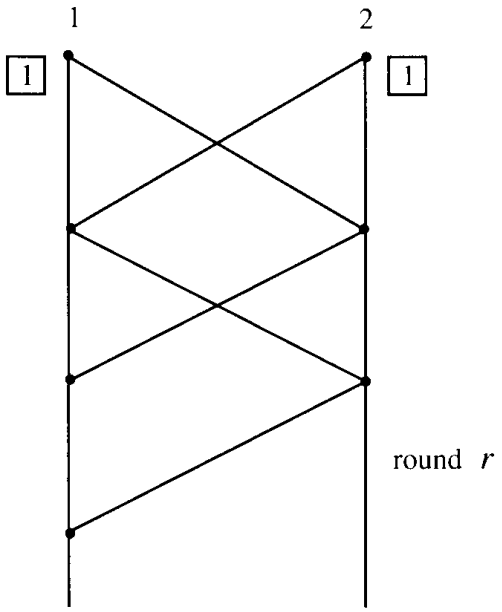

Figure 5.2: Pattern of message exchanges in execution α_2.

possibility of violating the agreement and/or validity condition; we discuss this approach in Section 5.2.

5.2 The Coordinated Attack Problem—Randomized Version

In this section, we consider the coordinated attack problem in the setting where processes can be randomized. As in the previous section, we consider n processes arranged in an arbitrary, known, undirected graph network. Each process starts with an input in $\{0, 1\}$ in a designated state component; we assume that for each process, there is exactly one start state containing each input value. For this section, we assume that the protocol terminates within a fixed number $r \geq 1$ of rounds; specifically, that by the end of round r, each process is required to output a decision in $\{0, 1\}$ by setting its *decision* variable to 0 or 1. We assume that a message is sent along each edge at each round k, $1 \leq k \leq r$, and that any number of these messages may be lost.

The goal is essentially the same as before, except that now we weaken the problem statement to allow for some probability of error. Namely, we use the same validity condition as before, but weaken the agreement condition to allow a small probability ϵ of disagreement. We obtain upper and lower bound results

for the achievable values of ϵ, in terms of the number r of rounds. As you will see, the achievable values of ϵ are not small.

5.2.1 Formal Modelling

In formalizing this problem, we must be clear about the meaning of the probabilistic statements—the situation is more complicated than it was for the MIS problem in Section 4.5. The complication is that the execution that is produced depends not only on the results of the random choices, but also on which messages are lost. We do not want to assume that message losses are determined randomly. Rather, we imagine that they are determined by some "adversary" that tries to make things as difficult as possible for the algorithm; we evaluate the algorithm by considering its worst-case behavior over the class of all possible adversaries.

Formally, we define a *communication pattern* to be any subset of the set

$$\{(i, j, k) : (i, j) \text{ is an edge in the graph, and } 1 \leq k\}.$$

A communication pattern γ is defined to be *good* if $k \leq r$ for every $(i, j, k) \in \gamma$ (for this chapter only—we will use another notion of "goodness" in Chapter 6). A good communication pattern represents the set of messages that are delivered in some execution: if (i, j, k) is in the communication pattern, then it means that a message sent by i to j at round k succeeds in getting delivered.

The notion of *adversary* that we use here is an arbitrary choice of

1. An assignment of input values to all the processes
2. A good communication pattern

For any particular adversary, any particular set of random choices made by the processes determines a unique execution. Thus, for any particular adversary, the random choices made by the processes induce a probability distribution on the set of executions. Using this probability distribution, we can express the probability of events such as the processes all agreeing. To emphasize the role of the adversary, we use the notation Pr^B for the probability function induced by a given adversary B.

We now restate the coordinated attack problem in this probabilistic setting. The statement uses the parameter ϵ, $0 \leq \epsilon \leq 1$.

Agreement: For every adversary B,

$$Pr^B[\text{some process decides 0 and some process decides 1}] \leq \epsilon.$$

Validity: Same as before.

We do not require a termination condition, because we have already assumed that all processes decide within r rounds. Our goals are to find an algorithm with the smallest possible value of ϵ and to prove that no smaller value of ϵ can be achieved.

5.2.2 An Algorithm

For simplicity, we restrict attention in this and the following subsection to the special case of an n-node complete graph. We leave the extensions to arbitrary graphs as exercises. For this special case, we present a simple algorithm that achieves $\epsilon = \frac{1}{r}$.

The algorithm is based on what processes know about each other's initial values and on what they know about each other's knowledge of the initial values, and so on. We need some definitions to capture such notions of knowledge.

First, for any communication pattern γ, we define a reflexive partial ordering $\leq_\gamma$ on pairs of the form (i, k), where i is a process index and k is a time, $0 \leq k$. (Recall from Chapter 2 that "time k" refers to the point in the execution just after k rounds have occurred.) This ordering represents information flow between the various processes at various times. We define the relation by

1. $(i, k) \leq_\gamma (i, k')$ for all i, $1 \leq i \leq n$, and all k, k', $0 \leq k \leq k'$.
2. If $(i, j, k) \in \gamma$, then $(i, k-1) \leq_\gamma (j, k)$.
3. If $(i, k) \leq_\gamma (i', k')$ and $(i', k') \leq_\gamma (i'', k'')$, then $(i, k) \leq_\gamma (i'', k'')$.

The first case describes information flow at the same process. The second case describes information flow from the sender to the receiver of a message. The third case just takes the transitive closure. Similar information-flow ideas will appear later in the book, for example, in Chapters 14, 16, 18, and 19.

Now for any good communication pattern γ, we define the *information level*, $level_\gamma(i, k)$ of any process i at any time k, $0 \leq k \leq r$, recursively. There are three cases:

1. $k = 0$:
   Then define $level_\gamma(i, k)$ to be 0.
2. $k > 0$ and there is some $j \neq i$ such that $(j, 0) \not\leq_\gamma (i, k)$:
   Then define $level_\gamma(i, k)$ to be 0.
3. $k > 0$ and $(j, 0) \leq_\gamma (i, k)$ for every $j \neq i$:
   Then for each $j \neq i$, let l_j denote $\max\{level_\gamma(j, k') : (j, k') \leq_\gamma (i, k)\}$.

(This is the largest level that i knows j has reached.) Note that $0 \leq l_j \leq k-1$ for all j. Then define $level_\gamma(i,k)$ to be $1 + \min\{l_j : j \neq i\}$.

In other words, each process starts out at level 0; when it hears from all the other processes, it advances to level 1. When it hears that all the other processes have reached level 1, it advances to level 2, and so on. If B is an adversary with communication pattern γ, we sometimes write $level_B(i,k)$ to mean $level_\gamma(i,k)$.

Example 5.2.1 Information level

Suppose that $n = 2$ and $r = 6$. Let γ be the good communication pattern consisting of exactly the following triples:

$$(1,2,1),(1,2,2),(2,1,2),(1,2,3),(2,1,4),(1,2,5),(2,1,5),(1,2,6)$$

Communication pattern γ is depicted in Figure 5.3. The information levels for processes 1 and 2, at times k, $0 \leq k \leq 6$, are as indicated by the labels.

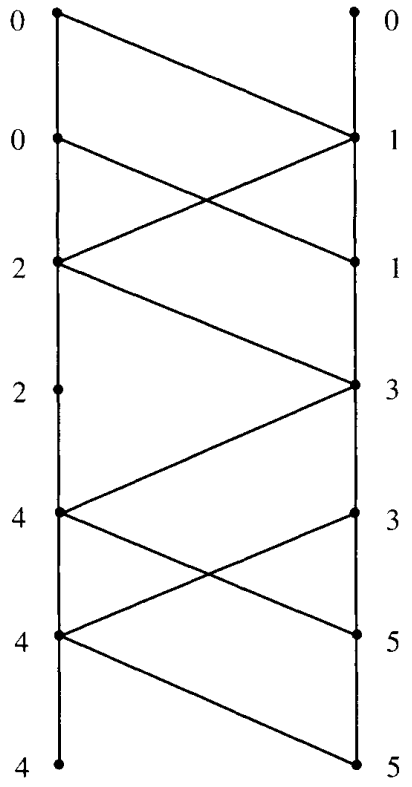

Figure 5.3: Good communication pattern γ.

The following lemma says that the information levels of different processes always remain within 1 of each other.

Lemma 5.2 *For any good communication pattern γ, any k, $0 \leq k \leq r$, and any i and j, $|level_\gamma(i,k) - level_\gamma(j,k)| \leq 1$.*

Proof. The proof is left as an exercise. □

The following lemma says that, in the case where all messages are delivered, the information level is equal to the number of rounds.

Lemma 5.3 *If γ is the "complete" communication pattern containing all triples (i,j,k), $1 \leq k \leq r$, then $level_\gamma(i,k) = k$ for all i and k.*

Proof. The proof is left as an exercise. □

The idea of the algorithm, which we call *RandomAttack*, is as follows:

***RandomAttack* algorithm (informal):**

Each process i keeps explicit track of its level, with respect to the communication pattern that occurs in the execution, in a variable *level*. Also, process 1 chooses a random *key* value, an integer in the range $[1,r]$; this value is piggybacked on all messages. In addition, the initial values of all processes are piggybacked on all messages.

After r rounds, each process decides 1 exactly, if its calculated *level* is at least as large as *key* and it knows that all processes' initial values are 1.

***RandomAttack* algorithm (formal):**

The message alphabet consists of triples of the form (L,V,k), where L is a vector assigning an integer in $[0,r]$ to each process index, V is a vector assigning a value in $\{0,1,undefined\}$ to each process index, and k is either an integer in $[1,r]$ or *undefined*.

***states*$_i$:**
rounds $\in \mathbb{N}$, initially 0
decision $\in \{unknown, 0, 1\}$, initially *unknown*
key $\in [1,r] \cup undefined$, initially *undefined*
for every j, $1 \leq j \leq n$:
  val$(j) \in \{0,1,undefined\}$; initially *val*$(i)$ is i's initial value and
    val$(j) =$ *undefined* for all $j \neq i$
  level$(j) \in [-1,r]$; initially *level*$(i) = 0$ and *level*$(j) = -1$ for all $j \neq i$

The variable *level*(j) is used to keep track of the largest level for process j that is known (through a chain of messages) to process i. For $j \neq i$, before i has heard anything from j, *level*(j) has the default value -1. In the random function $rand_i$, we use *random* to indicate a random choice of an integer in $[1,r]$, using the uniform distribution.

***rand*$_i$:**
if $i = 1$ and *rounds* $= 0$ then *key* $:=$ *random*

***msgs*$_i$:**
send (L, V, key) to all j, where L is the *level* vector and V is the *val* vector

***trans*$_i$:**
rounds $:=$ *rounds* $+ 1$
let (L_j, V_j, k_j) be the message from j, for each j from which a message arrives
if, for some j, $k_j \neq$ *undefined* then *key* $:= k_j$
for all $j \neq i$ do
    if, for some i', $V_{i'}(j) \neq$ *undefined* then *val*$(j) := V_{i'}(j)$
    if, for some i', $L_{i'}(j) >$ *level*(j) then *level*$(j) := \max_{i'}\{L_{i'}(j)\}$
level$(i) := 1 + \min\{$*level*$(j) : j \neq i\}$
if *rounds* $= r$ then
    if *key* $\neq$ *undefined* and *level*$(i) \geq$ *key* and *val*$(j) = 1$ for all j then
        decision $:= 1$
    else *decision* $:= 0$

In this code, the third line sets the *key* component; it does not matter if it is set more than once, since all values of *key* that get passed around are the same. The fifth line sets the *val* components for processes $j \neq i$, again with no danger of conflicting assignments. The sixth line updates the *level* components for processes $j \neq i$; these are intended to contain the largest levels that i knows about, for all the other processes. Next, i updates its own *level* component, setting it to be one more than the smallest level it knows about for any of the other processes. Finally, if this is the last round r, then i decides according to the rule described earlier.

Theorem 5.4 *RandomAttack solves the randomized version of the coordinated attack problem, for* $\epsilon = \frac{1}{r}$.

Proof. The key to the proof is just the claim that the algorithm correctly calculates the levels. That is, in any execution of *RandomAttack*, with any good communication pattern γ, for any k, $0 \leq k \leq r$, and for any i, after k rounds, the value of *level*$(i)_i$ is equal to *level*$_\gamma(i, k)$. Also, after k rounds, if *level*$(i)_i \geq 1$, then *key*$_i$ is defined and *val*$(j)_i$ is defined for all j; moreover, these values are equal to the actual *key* chosen by process 1 and the actual initial values, respectively.

Termination of the *RandomAttack* algorithm is obvious. For validity, if all processes have initial value 0, then obviously 0 is the only possible decision value. Now suppose that all processes start with 1 and all messages are delivered. Then Lemma 5.3 and the fact that the algorithm correctly calculates the levels imply that for each i, *level*$(i)_i = r$ at the point in round r where the decision is made.

Since $level(i)_i = r \geq 1$ at this point, it follows that key_i is defined and $val(j)_i$ is defined for all j. Since all possible key values are less than or equal to r, 1 is the only possible decision value.

Finally, we consider agreement. Let B be any adversary; we show that

$$Pr^B[\text{some process decides 0 and some process decides 1}] \leq \epsilon.$$

For each i, let l_i denote the value of $level(i)_i$ at the time process i makes its decision (in round r). Then Lemma 5.2 implies that all the values l_i are within one of each other. If the chosen value of key is strictly greater than $\max\{l_i\}$, or if there is some process with an initial value of 0, then all processes decide 0. On the other hand, if $key \leq \min\{l_i\}$ and all processes have initial value 1, then all processes decide 1. So the only case where disagreement is possible is where $key = \max\{l_i\}$. The probability of this event is $\frac{1}{r} = \epsilon$, since $\max\{l_i\}$ is determined by the adversary B and key is uniformly distributed in $[0, r]$. □

Example 5.2.2 Behavior of *RandomAttack*

Consider the case where $n = 2$ and $r = 6$. Consider the adversary B that supplies input 1 for both processes, together with the good communication pattern γ of Example 5.2.1. Let $\epsilon = \frac{1}{6}$. Theorem 5.4 says that the probability of disagreement for adversary B is at most ϵ. In fact, this probability is exactly ϵ: if the value of key chosen by process 1 is 5, then process 1 decides 0 and process 2 decides 1; if $key \leq 4$, then both decide 1; and if $key = 6$, then both decide 0.

On the other hand, if the adversary supplies any other combination of inputs together with communication pattern γ, then the probability of disagreement is 0, since both processes decide 0.

Using the ideas in the proof of Theorem 5.4, we can see that *RandomAttack* satisfies stronger validity conditions than we have so far claimed. Namely, we can show:

Validity:

1. If any process starts with 0, then 0 is the only possible decision value.
2. For any adversary B for which all the initial values are 1,

$$Pr^B[\text{all processes decide 1}] \geq l\epsilon,$$

where l is the minimum level of any process at time r in B.

The second of these properties might be useful in some applications, such as warfare or distributed database commit, where it is considered desirable to favor the positive outcome. If, for example, only a single message is lost, then the probability of coordinated attack is guaranteed to be high, at least $\frac{r-1}{r}$. The proof that *RandomAttack* satisfies the stronger validity conditions is left as a simple exercise.

5.2.3 A Lower Bound on Disagreement

Now we show that it is not possible to do much better than the bound described in Theorem 5.4. (Recall from the previous subsection that we are only considering n-node complete graphs.)

Theorem 5.5 *Any r-round algorithm for the randomized coordinated attack problem has probability of disagreement at least $\frac{1}{r+1}$.*

For the remainder of this section, we assume a particular r-round algorithm A that solves the coordinated attack problem with disagreement probability ϵ in an n-node complete graph; we show that $\epsilon \geq \frac{1}{r+1}$.

In order to prove the theorem, we need one more definition. If B is any adversary, γ its communication pattern, and i any process, then we define another adversary, $prune(B, i)$. Adversary $prune(B, i)$ simply "prunes out" information that i does not hear about in B. $B' = prune(B, i)$ is defined as follows:

1. If $(j, 0) \leq_\gamma (i, r)$, then j's input in B' is the same as it is in B, and otherwise it is 0.

2. A triple (j, j', k) is in the communication pattern of B' exactly if it is in the communication pattern of B and $(j', k) \leq_\gamma (i, r)$.

That is, adversary B' includes all the messages that i knows about in B, but no others, and B' specifies that all the inputs that i does not know about in B are 0. The following lemma says that the pruned version of an adversary is sufficient to determine the probability distribution of outputs.

Lemma 5.6 *If B and B' are two adversaries, i is a process, and $prune(B, i) = prune(B', i)$, then $Pr^B[i \text{ decides } 1] = Pr^{B'}[i \text{ decides } 1]$.*

Proof. The proof is left as an exercise. □

The proof of Theorem 5.5 is based on the following lemma.

Lemma 5.7 *Let B be any adversary for which all initial values are 1 and let i be any process. Then*

$$Pr^B[i \text{ decides } 1] \leq \epsilon(level_B(i,r)+1).$$

Proof. By induction on $level_B(i,r)$.

Basis: Suppose $level_B(i,r)=0$. Define $B' = prune(B,i)$. Then $prune(B',i) = B' = prune(B,i)$, so by Lemma 5.6,

$$Pr^B[i \text{ decides } 1] = Pr^{B'}[i \text{ decides } 1]. \tag{5.1}$$

Since $level_B(i,r)=0$, there must be some process $j \neq i$ such that $(j,0) \not\lesssim_\gamma (i,r)$, where γ is the communication pattern of B. Then adversary B' specifies an initial value of 0 for j and includes no messages with destination j in its communication pattern. It follows that $prune(B',j)$ is the trivial adversary for which all the initial values are 0 and there are no messages in the communication pattern. Let B'' denote this trivial adversary. Then $prune(B'',j) = B'' = prune(B',j)$, so by Lemma 5.6,

$$Pr^{B'}[j \text{ decides } 1] = Pr^{B''}[j \text{ decides } 1].$$

The validity condition implies that

$$Pr^{B''}[j \text{ decides } 1] = 0,$$

so therefore

$$Pr^{B'}[j \text{ decides } 1] = 0.$$

But since there is at most probability ϵ of disagreement, we have that

$$|Pr^{B'}[i \text{ decides } 1] - Pr^{B'}[j \text{ decides } 1]| \leq \epsilon.$$

Therefore,

$$Pr^{B'}[i \text{ decides } 1] \leq \epsilon,$$

which by Equation 5.1 implies that

$$Pr^{B}[i \text{ decides } 1] \leq \epsilon,$$

as needed.

Inductive step: Suppose $level_B(i,r) = l > 0$, and suppose that the lemma holds for all levels less than l. Define $B' = prune(B,i)$. Then Lemma 5.6 implies that

$$Pr^B[i \text{ decides } 1] = Pr^{B'}[i \text{ decides } 1]. \tag{5.2}$$

Since $level_B(i,r) = l$, the definition of *level* implies that there must be some process j such that $level_{B'}(j,r) \leq l-1$. By the inductive hypothesis,

$$\begin{aligned} Pr^{B'}[j \text{ decides } 1] &\leq \epsilon\,(level_{B'}(j,r)+1) \\ &\leq \epsilon l. \end{aligned}$$

But since there is at most probability ϵ of disagreement, we have that

$$|Pr^{B'}[i \text{ decides } 1] - Pr^{B'}[j \text{ decides } 1]| \leq \epsilon.$$

Therefore,

$$Pr^{B'}[i \text{ decides } 1] \leq \epsilon(l+1),$$

which by Equation 5.2 implies that

$$Pr^{B}[i \text{ decides } 1] \leq \epsilon(l+1),$$

as needed. □

We can now prove the theorem.

Proof (of Theorem 5.5). Let B be the adversary for which all inputs are 1 and no messages are lost. The probability that all processes decide 1 is at most the probability that any of them decides 1, which is, by Lemma 5.7, at most $\epsilon(level_B(i,r)+1) \leq \epsilon(r+1)$. But the validity condition says that all processes must decide 1 in all executions generated by this adversary B; hence the probability that all decide 1 must be exactly 1. This implies that $\epsilon(r+1) \geq 1$, that is, that $\epsilon \geq \frac{1}{r+1}$, as needed. □

5.3 Bibliographic Notes

The coordinated attack problem was originated by Gray [142] in order to model the problem of distributed database commit. The impossibility result for the deterministic version of the problem is also due to Gray [142]. The results on randomized coordinated attack are derived from work of Varghese and Lynch [281].

5.4 Exercises

5.1. Show that a solution to the (deterministic) coordinated attack problem for any nontrivial connected graph implies a solution for the simple graph consisting of two processes connected by one edge. (Therefore, this problem is unsolvable in any nontrivial graph.)

5.2. Consider the following variant of the (deterministic) coordinated attack problem. Assume that the network is a complete graph of $n > 2$ participants. The termination and validity requirements are the same as those in Section 5.1. However, the agreement requirement is weakened to say: "If any process decides 1, then there are at least two that decide 1." (That is, we want to rule out the case where one general attacks alone, but allow two or more generals to attack together.) Is this problem solvable or unsolvable? Prove.

5.3. Consider the coordinated attack problem with link failures for the simple case of two processes connected by an edge. Suppose that the processes are deterministic, but the message system is probabilistic, in the sense that each message has an independent probability p, $0 < p < 1$, of getting delivered successfully. (As usual, we allow each process to send only one message per round.)

For this setting, devise an algorithm that terminates in a fixed number r of rounds, has probability at most ϵ of disagreement, and likewise has probability at most ϵ of violating the validity condition. Obtain the smallest ϵ you can.

5.4. For the setting described in the previous exercise, prove a lower bound on the size of the bound ϵ that can be obtained.

5.5. Prove Lemma 5.2.

5.6. Prove Lemma 5.3.

5.7. Prove carefully the first claims in the proof of Theorem 5.4, that is, that the *RandomAttack* algorithm correctly computes the *level* values, and correctly conveys the initial values and key.

5.8. For the *RandomAttack* algorithm, prove the stronger validity properties given at the end of Section 5.2.2. That is, prove

(a) If any process starts with 0, then 0 is the only possible decision value.

(b) For any adversary B for which all the initial values are 1,

$$Pr^B[\text{all processes decide 1}] \geq l\epsilon,$$

where l is the minimum level of any process at time r in B.

5.9. Generalize the randomized version of the coordinated attack problem to allow for probability ϵ of violating the validity condition as well as of violating the agreement condition. Adjust the *RandomAttack* algorithm so that it achieves the smallest possible ϵ for this modified problem statement. Analyze.

5.10. Extend the *RandomAttack* algorithm and its analysis to arbitrary (not necessarily complete) undirected graphs.

5.11. Prove Lemma 5.6.

5.12. Extend the lower bound result in Theorem 5.5 to arbitrary (not necessarily complete) undirected graphs.

5.13. What happens to the results of this chapter for the randomized setting, if the communication pattern determined by the adversary is not fixed in advance as we have assumed, but is determined on-line? More precisely, suppose that the adversary is an entity that is able to examine the entire execution up to the beginning of any round k, before deciding which round k messages will be delivered.

(a) What bound ϵ on disagreement is guaranteed by the *RandomAttack* algorithm, when working against arbitrary on-line adversaries?

(b) Can you prove an interesting lower bound on attainable values of ϵ?