Chapter 8

Coordinated attack

(See also [Lyn96, §5.1].)

The **Two Generals** problem was the first widely-known distributed consensus problem, described in 1978 by Jim Gray [Gra78, §5.8.3.3.1], although the same problem previously appeared under a different name [AEH75].

The setup of the problem is that we have two generals on opposite sides of an enemy army, who must choose whether to attack the army or retreat. If only one general attacks, his troops will be slaughtered. So the generals need to reach agreement on their strategy.

To complicate matters, the generals can only communicate by sending messages by (unreliable) carrier pigeon. We also suppose that at some point each general must make an irrevocable decision to attack or retreat. The interesting property of the problem is that if carrier pigeons can become lost, there is no protocol that guarantees agreement in all cases unless the outcome is predetermined (e.g., the generals always attack no matter what happens). The essential idea of the proof is that any protocol that does guarantee agreement can be shortened by deleting the last message; iterating this process eventually leaves a protocol with no messages.

Adding more generals turns this into the **coordinated attack** problem, a variant of **consensus**; but it doesn't make things any easier.

8.1 Formal description

To formalize this intuition, suppose that we have $n \geq 2$ generals in a synchronous system with unreliable channels—the set of messages received in round $i + 1$ is always a subset of the set sent in round i, but it may be a proper subset (even the empty set). Each general starts with an input 0 (retreat) or 1 (attack) and must output 0 or 1 after some bounded number of rounds. The requirements for the protocol are that, in all executions:

Agreement All processes output the same decision (0 or 1).

Validity If all processes have the same input x, and no messages are lost, all processes produce output x. (If processes start with different inputs or one or more messages are lost, processes can output 0 or 1 as long as they all agree.)

Termination All processes terminate in a bounded number of rounds.[1]

Sadly, there is not protocol that satisfies all three conditions. We show this in the next section.

8.2 Impossibility proof

To show coordinated attack is impossible,[2] we use an **indistinguishability proof**.

The basic idea of an indistinguishability proof is this:

- Execution A is **indistinguishable** from execution B for some process p if p sees the same things (messages or operation results) in both executions.
- If A is indistinguishable from B for p, then because p can't tell which of these two possible worlds it is in, it returns the same output in both.

So far, pretty dull. But now let's consider a chain of hypothetical executions $A = A_0 A_1 \dots A_k = B$, where each A_i is indistinguishable from A_{i+1} for some process p_i. Suppose also that we are trying to solve an agreement task, where every process must output the same value. Then since p_i outputs the same value in A_i and A_{i+1}, every process outputs the same

[1]**Bounded** means that there is a fixed upper bound on the length of any execution. We could also demand merely that all processes terminate in a *finite* number of rounds. In general, finite is a weaker requirement than bounded, but if the number of possible outcomes at each step is finite (as they are in this case), they're equivalent. The reason is that if we build a tree of all configurations, each configuration has only finitely many successors, and the length of each path is finite, then **König's lemma** (see `http://en.wikipedia.org/wiki/Konig's_lemma`) says that there are only finitely many paths. So we can take the length of the longest of these paths as our fixed bound. [BG97, Lemma 3.1]

[2]Without making additional assumptions, always a caveat when discussing impossibility.

value in A_i and A_{i+1}. By induction on k, every process outputs the same value in A and B, even though A and B may be very different executions.

This gives us a tool for proving impossibility results for agreement: show that there is a path of indistinguishable executions between two executions that are supposed to produce different output. Another way to picture this: consider a graph whose nodes are all possible executions with an edge between any two indistinguishable executions; then the set of output-0 executions can't be adjacent to the set of output-1 executions. If we prove the graph is connected, we prove the output is the same for all executions.

For coordinated attack, we will show that no protocol satisfies all of agreement, validity, and termination using an indistinguishability argument. The key idea is to construct a path between the all-0-input and all-1-input executions with no message loss via intermediate executions that are indistinguishable to at least one process.

Let's start with $A = A_0$ being an execution in which all inputs are 1 and all messages are delivered. We'll build executions A_1, A_2, etc., by pruning messages. Consider A_i and let m be some message that is delivered in the last round in which any message is delivered. Construct A_{i+1} by not delivering m. Observe that while A_i is distinguishable from A_{i+1} by the recipient of m, on the assumption that $n \geq 2$ there is some other process that can't tell whether m was delivered or not (the recipient can't let that other process know, because no subsequent message it sends are delivered in either execution). Continue until we reach an execution A_k in which all inputs are 1 and no messages are sent. Next, let A_{k+1} through A_{k+n} be obtained by changing one input at a time from 1 to 0; each such execution is indistinguishable from its predecessor by any process whose input didn't change. Finally, construct A_{k+n} through $A_{k+n+k'}$ by adding back messages in the reverse process used for A_0 through A_k; note that this might not result in exactly k new messages, because the number of messages might depend on the inputs. This gets us to an execution $A_{k+n+k'}$ in which all processes have input 0 and no messages are lost. If agreement holds, then the indistinguishability of adjacent executions to some process means that the common output in A_0 is the same as in $A_{k+n+k'}$. But validity requires that A_0 outputs 1 and $A_{k+n+k'}$ outputs 0: so either agreement or validity is violated in some execution.

8.3 Randomized coordinated attack

So we now know that we can't solve the coordinated attack problem. But maybe we want to solve it anyway. The solution is to change the problem.

Randomized coordinated attack is like standard coordinated attack, but with less coordination. Specifically, we'll allow the processes to flip coins to decide what to do, and assume that the communication pattern (which messages get delivered in each round) is fixed and independent of the coin-flips. This corresponds to assuming an **oblivious adversary** that can't see what is going on at all or perhaps a **content-oblivious adversary** that can only see where messages are being sent but not the contents of the messages. We'll also relax the agreement property to only hold with some high probability:

Randomized agreement For any adversary A, the probability that some process decides 0 and some other process decides 1 given A is at most ϵ.

Validity and termination are as before.

8.3.1 An algorithm

Here's an algorithm that gives $\epsilon = 1/r$. (See [Lyn96, §5.2.2] for details or [VL92] for the original version.) A simplifying assumption is that network is complete, although a strongly-connected network with r greater than or equal to the diameter also works.

- First part: tracking information levels
  - Each process tracks its "information level," initially 0. The state of a process consists of a vector of (input, information-level) pairs for all processes in the system. Initially this is (my-input, 0) for itself and $(\perp, -1)$ for everybody else.
  - Every process sends its entire state to every other process in every round.
  - Upon receiving a message m, process i stores any inputs carried in m and, for each process j, sets $\mathsf{level}_i[j]$ to $\max(\mathsf{level}_i[j], \mathsf{level}_m[j])$. It then sets its own information level to $\min_j(\mathsf{level}_i[j]) + 1$.
- Second part: deciding the output
  - Process 1 chooses a random key value uniformly in the range $[1, r]$.

- This key is distributed along with $\mathsf{level}_i[1]$, so that every process with $\mathsf{level}_i[1] \geq 0$ knows the key.
- A process decides 1 at round r if and only if it knows the key, its information level is greater than or equal to the key, and all inputs are 1.

8.3.2 Why it works

Termination Immediate from the algorithm.

Validity

- If all inputs are 0, no process sees all 1 inputs (technically requires an invariant that processes' non-null views are consistent with the inputs, but that's not hard to prove.)
- If all inputs are 1 and no messages are lost, then the information level of each process after k rounds is k (prove by induction) and all processes learn the key and all inputs (immediate from first round). So all processes decide 1.

Randomized Agreement

- First prove a lemma: Define $\mathsf{level}_i^t[k]$ to be the value of $\mathsf{level}_i[k]$ after t rounds. Then for all i, j, k, t, (1) $\mathsf{level}_i[j]^t \leq \mathsf{level}_j[j]^{t-1}$ and (2) $\left|\mathsf{level}_i[k]^t - \mathsf{level}_j[k]^t\right| \leq 1$. As always, the proof is by induction on rounds. Part (1) is easy and boring so we'll skip it. For part (2), we have:
  - After 0 rounds, $\mathsf{level}_i^0[k] = \mathsf{level}_j^0[k] = -1$ if neither i nor j equals k; if one of them is k, we have $\mathsf{level}_k^0[k] = 0$, which is still close enough.
  - After t rounds, consider $\mathsf{level}_i^t[k] - \mathsf{level}_i^{t-1}[k]$ and similarly $\mathsf{level}_j^t[k] - \mathsf{level}_j^{t-1}[k]$. It's not hard to show that each can jump by at most 1. If both deltas are $+1$ or both are 0, there's no change in the difference in views and we win from the induction hypothesis. So the interesting case is when $\mathsf{level}_i[k]$ stays the same and $\mathsf{level}_j[k]$ increases or vice versa.
  - There are two ways for $\mathsf{level}_j[k]$ to increase:
    * If $j \neq k$, then j received a message from some j' with $\mathsf{level}_{j'}^{t-1}[k] > \mathsf{level}_j^{t-1}[k]$. From the induction hypothesis, $\mathsf{level}_{j'}^{t-1}[k] \leq \mathsf{level}_i^{t-1}[k] + 1 = \mathsf{level}_i^t[k]$. So we are happy.
    * If $j = k$, then j has $\mathsf{level}_j^t[j] = 1 + \min_{k \neq j} \mathsf{level}_j^t[k] \leq 1 + \mathsf{level}_j^t[i] \leq 1 + \mathsf{level}_i^t[i]$. Again we are happy.

- Note that in the preceding, the key value didn't figure in; so everybody's level at round r is independent of the key.
- So now we have that $\mathsf{level}_i^r[i]$ is in $\{\ell, \ell+1\}$, where ℓ is some fixed value uncorrelated with the key. The only way to get some process to decide 1 while others decide 0 is if $\ell + 1 \geq \mathsf{key}$ but $\ell < \mathsf{key}$. (If $\ell = 0$, a process at this level doesn't know key, but it can still reason that $0 < \mathsf{key}$ since key is in $[1, r]$.) This can only occur if $\mathsf{key} = \ell + 1$, which occurs with probability at most $1/r$ since key was chosen uniformly.

8.3.3 Almost-matching lower bound

The bound on the probability of disagreement in the previous algorithm is almost tight. Varghese and Lynch [VL92] show that no synchronous algorithm can get a probability of disagreement less than $\frac{1}{r+1}$, using a stronger validity condition that requires that the processes output 0 if any input is 0. This is a natural assumption for database commit, where we don't want to commit if any process wants to abort. We restate their result below:

Theorem 8.3.1. *For any synchronous algorithm for randomized coordinated attack that runs in r rounds that satisfies the additional condition that all non-faulty processes decide 0 if any input is 0,* $\Pr[\textit{disagreement}] \geq 1/(r+1)$.

Proof. Let ϵ be the bound on the probability of disagreement. Define $\mathsf{level}_i^t[k]$ as in the previous algorithm (whatever the real algorithm is doing). We'll show $\Pr[i \text{ decides } 1] \leq \epsilon \cdot (\mathsf{level}_i^r[i] + 1)$, by induction on $\mathsf{level}_i^r[i]$.

- If $\mathsf{level}_i^r[i] = 0$, the real execution is indistinguishable (to i) from an execution in which some other process j starts with 0 and receives no messages at all. In that execution, j must decide 0 or risk violating the strong validity assumption. So i decides 1 with probability at most ϵ (from the disagreement bound).

- If $\mathsf{level}_i^r[i] = k > 0$, the real execution is indistinguishable (to i) from an execution in which some other process j only reaches level $k-1$ and thereafter receives no messages. From the induction hypothesis, $\Pr[j \text{ decides } 1] \leq \epsilon k$ in that pruned execution, and so $\Pr[i \text{ decides } 1] \leq \epsilon(k+1)$ in the pruned execution. But by indistinguishability, we also have $\Pr[i \text{ decides } 1] \leq \epsilon(k+1)$ in the original execution.

Now observe that in the all-1 input execution with no messages lost, $\mathsf{level}_i^r[i] = r$ and $\Pr[i \text{ decides } 1] = 1$ (by validity). So $1 \leq \epsilon(r+1)$, which implies $\epsilon \geq 1/(r+1)$. $\square$