Chapter 19

Shared Coins

19.1 Random Oracles and Bit Strings

Definition 19.1 (Random Oracle)**.** *A random oracle is a trusted (non-byzantine) random source that can generate random values.*

Algorithm 19.2 Algorithm 17.19 with a Magic Random Oracle

```
1: Replace Line 12 in Algorithm 17.19 by
2: return $c_i$, where $c_i$ is the $i$th random bit from the oracle
```

Remarks:

- Algorithm 19.2, as well as the upcoming Algorithm 19.5 modify Algorithm 17.19 by replacing Line 12. Instead of every node throwing a local coin (and hoping that they all result in the same bit), the nodes coordinate their random decision based on the proposed mechanisms.

Theorem 19.3. *Algorithm 19.2 solves asynchronous byzantine agreement in an expected constant number of rounds.*

Proof. If there is a large majority for one of the input values in the system, all nodes will decide within two rounds since Algorithm 17.19 satisfies all-same-validity; the coin is not even used.

If there is no significant majority for any of the input values at the beginning of Algorithm 17.19, all correct nodes will run Line 2 from Algorithm 19.2. Therefore, they will set their new value to the bit given by the random oracle and terminate in the following round.

If neither of the above cases holds, some of the nodes receive an input value more than $n/2+f$ times, while other nodes rely on the oracle. With probability 1/2, the value of the oracle will coincide with the deterministic majority value of the other nodes. Therefore, with probability 1/2, the nodes will terminate in the following round. The expected number of rounds for termination in this case is 3. □

Remarks:

- Unfortunately, random oracles are a bit like pink fluffy unicorns: they do not really exist in the real world. Can we fix that?

Definition 19.4 (Random Bitstring)**.** *A* ***random bit string*** *is a string of random binary values, known to all participating nodes when starting a protocol.*

Algorithm 19.5 Algorithm 17.19 with Random Bit String

```
1: Replace Line 12 in Algorithm 17.19 by
2: return $b_i$, where $b_i$ is the $i$th bit of the common random bit string
```

Remarks:

- But is such a precomputed bit string really random enough? We should be worried because of Theorem 16.27.

Theorem 19.6. *If the scheduling is worst-case, Algorithm 19.5 does not terminate.*

Proof. Let's assume that n is even. We start Algorithm 19.5 with the following input: $n/2+f+1$ nodes have input value 1, and $n/2-f-1$ nodes have input value 0. Assume w.l.o.g. that the first bit of the random bit string is 0.

If the second random bit in the bit string is also 0, then a worst-case scheduler will let $n/2+f+1$ nodes see all $n/2+f+1$ values 1. These will therefore deterministically choose the value 1 as their new value. Because of scheduling (or byzantine nodes), the remaining $n/2-f-1$ nodes receive strictly less than $n/2+f+1$ values 1 and therefore have to rely on the value of the shared coin, which is 0. The nodes will not come to a decision in this round. Moreover, we have created the very same distribution of values for the next round (which has also random bit 0).

If the second random bit in the bit string is 1, then a worst-case scheduler can let $n/2-f-1$ nodes see all $n/2+f+1$ values 1, and therefore deterministically choose the value 1 as their new value. Because of scheduling (or byzantine nodes), the remaining $n/2+f+1$ nodes receive strictly less than $n/2+f+1$ values 1 and therefore have to rely on the value of the shared coin, which is 0. The nodes will not decide in this round. And we have created the symmetric situation for input value 1 that is coming in the next round.

So if the current and the next random bit are known, worst-case scheduling will keep the system in one of two symmetric states and the nodes will never decide. □

Remarks:

- Theorem 19.6 shows that a worst-case scheduler cannot be allowed to know the random bits of the future.

- Note that in the proof of Theorem 19.6 we did not even use any byzantine nodes. Just bad scheduling was enough to prevent termination.

19.2 Simple Shared Coin

Definition 19.7 (Shared Coin). *A **shared coin** is a binary random variable shared among all nodes. It is* 0 *for all nodes with constant probability and* 1 *for all nodes with constant probability. The shared coin is allowed to fail (be* 0 *for some nodes and* 1 *for other nodes) with constant probability.*

Algorithm 19.8 Shared Coin (code for node u)

```
1: Choose local coin $c_u = 0$ with probability $1/n$, else $c_u = 1$
2: Broadcast myCoin($c_u$)
3: Wait for $n - f$ coins and store them in the local coin set $C_u$
4: Broadcast mySet($C_u$)
5: Wait for $n - f$ coin sets
6: if at least one coin is 0 among all coins in the coin sets then
7:     return 0
8: else
9:     return 1
10: end if
```

Remarks:

- Since at most f nodes crash, all nodes will always receive $n - f$ coins and coin sets in Lines 3 and 5, respectively. Therefore, all nodes make progress and termination is guaranteed.
- We show the correctness of the algorithm for $f < n/3$. To simplify the proof we assume that $n = 3f + 1$, i.e., we assume the worst case.

Lemma 19.9. *Let u be a node, and let W be the set of coins that u received in at least $f + 1$ different coin sets. It holds that $|W| \geq f + 1$.*

Proof. Let C be the multiset of coins received by u. Observe that u receives exactly $|C| = (n-f)^2$ many coins, as u waits for $n-f$ coin sets each containing $n - f$ coins.

Assume that the lemma does not hold. Coins in W occur in at most $n - f$ coin sets. Hence, at most $(n - f)|W|$ entries in C come from coins in W. Coins in $\overline{W}$ occur in at most f coin sets, and there are $n - |W|$ of them. Hence, at most $f(n - |W|)$ entries in C come from coins in $\overline{W}$. Hence, the total number of coins that u received is bounded by

$$|C| \leq (n - f)|W| + f(n - |W|) \leq (n - f)f + f(n - f) = 2f(n - f).$$

Our assumption was that $n > 3f$, i.e., $n - f > 2f$. Therefore $|C| \leq 2f(n-f) < (n - f)^2 = |C|$, which is a contradiction. □

Lemma 19.10. *All coins in W are seen by all correct nodes.*

Proof. Let $w \in W$ be such a coin. By definition of W we know that w is in at least $f + 1$ sets received by u. Since every other node also waits for $n - f$ sets before terminating, each node will receive at least one of these sets, and hence w must be seen by every node that terminates. □

Theorem 19.11. *If $f < n/3$ nodes crash, Algorithm 19.8 implements a shared coin.*

Proof. Let us first bound the probability that the algorithm returns 1 for all nodes. With probability $(1 - 1/n)^n \approx 1/e \approx 0.37$ all nodes choose their local coin equal to 1 (Line 1), and in that case 1 will be decided. This is only a lower bound on the probability that all nodes return 1, as there are also other scenarios based on message scheduling and crashes which lead to a global decision for 1. But a probability of 0.37 is good enough, so we do not need to consider these scenarios.

With probability $1 - (1 - 1/n)^{|W|}$ there is at least one 0 in W. Using Lemma 19.9 we know that $|W| \geq f + 1 \approx n/3$, hence the probability is about $1 - (1 - 1/n)^{n/3} \approx 1 - (1/e)^{1/3} \approx 0.28$. We know that this 0 is seen by all nodes (Lemma 19.10), and hence everybody will decide 0. Thus Algorithm 19.8 implements a shared coin. □

Remarks:

- We only proved the worst case. By choosing f fairly small, it is clear that $f + 1 \not\approx n/3$. However, Lemma 19.9 can be proved for $|W| \geq n - 2f$. To prove this claim you need to substitute the expressions in the contradictory statement: At most $n - 2f - 1$ coins can be in all $n - f$ coin sets, and $n - (n - 2f - 1) = 2f + 1$ coins can be in at most f coin sets. The remainder of the proof is analogous, the only difference is that the math is not as neat. Using the modified lemma we know that $|W| \geq n/3$, and therefore Theorem 19.11 also holds for *any* $f < n/3$.

Theorem 19.12. *Plugging Algorithm 19.8 into Algorithm 16.28 we get a randomized consensus algorithm that terminates in a constant expected number of rounds tolerating up to $f < n/3$ crash failures.*

Remarks:

- Worst-case scheduling is an issue that we have only briefly considered so far, in particular, to show that the random bit string does not help to speed up Algorithm 17.19.

- What if scheduling is worst-case in Algorithm 19.8?

Lemma 19.13. *Algorithm 19.8 has exponential expected running time under worst-case scheduling.*

Proof. In Algorithm 19.8, worst-case scheduling may hide up to f rare zero coin flips. In order to receive a zero as the outcome of the shared coin, the nodes need to generate at least $f + 1$ zeros. The probability for this to happen is $O((1/n)^{f+1})$, which is exponentially small for $f \in \Omega(n/\log(n))$. If the worst-case scheduler makes sure that some nodes will always deterministically go for 0, then the algorithm needs $\Omega(n^{f+1})$ rounds until it terminates. □

Remarks:

- In Chapter 17 we have developed a fast solution for synchronous byzantine agreement (Algorithm 17.13), yet our *asynchronous* byzantine agreement solution (Algorithm 17.19) is still awfully slow. Some simple methods to speed up the algorithms did not work, mostly due to unrealistic assumptions. Can we at least solve asynchronous (assuming worst-case scheduling) *consensus* if we have crash failures? Possibly based on some advanced communication methods?

19.3 Shared Coin on a Blackboard

Definition 19.14 (Blackboard Model). *The **blackboard** is a trusted authority that supports two operations. A node can **write** its message to the blackboard and a node can **read** all the values that have been written to the blackboard so far.*

Algorithm 19.15 Crash-Resilient Shared Coin with Blackboard (for node u)

```
1: while true do
2:   Choose new local coin $c_u = +1$ with probability $1/2$, else $c_u = -1$
3:   Write $c_u$ to the blackboard
4:   Set $C$ = Read all coin flips on the blackboard
5:   if $|C| \geq n^2$ then
6:     return sign(sum(C))
7:   end if
8: end while
```

Remarks:

- We assume that the nodes cannot reconstruct the order in which the messages are written to the blackboard since the system is asynchronous.
- In Algorithm 19.15 the outcome of a coin flips is -1 or $+1$ instead of 0 or 1 because it simplifies the analysis, i.e., "$-1 \approx 0$".
- The *sign* function is used for the decision values. The sign function returns $+1$ if the sum of all coin flips in C is greater than zero, and -1 otherwise.
- The algorithm is unusual compared to other asynchronous algorithms we have dealt with so far. So far we often waited for $n - f$ messages from other nodes. In Algorithm 19.15, a single node can single-handedly generate all n^2 coin flips, without waiting.
- If a node does not need to wait for other nodes, we call the algorithm *wait-free*.
- Many similar definitions beyond wait-free exist: lock-free, deadlock-free, starvation-free, and generally non-blocking algorithms.

Theorem 19.16 (Central Limit Theorem)**.** *Let $\{X_1, X_2, \ldots, X_N\}$ be a sequence of independent random variables with $Pr[X_i = -1] = Pr[X_i = 1] = 1/2$ for all $i = 1, \ldots, N$. Then for every positive real number z,*

$$\lim_{N\to\infty} \Pr\left[\sum_{i=1}^{N} X_i \geq z\sqrt{N}\right] = 1 - \Phi(z) > \frac{1}{\sqrt{2\pi}}\frac{z}{z^2+1}e^{-z^2/2},$$

where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution evaluated at z.

Theorem 19.17. *Algorithm 19.15 implements a shared coin.*

Proof. Each node in the algorithm terminates once at least n^2 coin flips are written to the blackboard. Before terminating, nodes may write one additional coin flips. Therefore, every node decides after reading at least n^2 and at most $n^2 + n - 1$ coin flips. The power of the adversary lies in the fact that it can prevent $n - 1$ nodes from writing their last coin flips to the blackboard by delaying their writes. Here, we will consider an even stronger adversary that can hide up to n coin flips written on the blackboard.

We need to show that both outcomes for the shared coin ($+1$ or -1 in Line 6) will occur with constant probability, as in Definition 19.7. Let X be the sum of all coin flips that are visible to every node. Since some of the nodes might read n more values from the blackboard than others, the nodes cannot be prevented from deciding if $|X| > n$. By applying Theorem 19.16 with $N = n^2$ and $z = 1$, we get:

$$Pr(X \leq -n) = Pr(X \geq n) = 1 - \Phi(1) > 0.15. \qquad \square$$

Lemma 19.18. *Algorithm 19.15 uses n^2 coin flips, which is optimal in this model.*

Proof. The proof for showing quadratic lower bound makes use of configurations that are indistinguishable to all nodes, similar to Theorem 16.27. It requires involved stochastic methods and we therefore will only sketch the idea of where the term n^2 comes from.

The basic idea follows from Theorem 19.16. The standard deviation of the sum of n^2 coin flips is n. The central limit theorem tells us that with constant probability the sum of the coin flips will be only a constant factor away from the standard deviation. As we showed in Theorem 19.17, this is large enough to disarm a worst-case scheduler. However, with much less than n^2 coin flips, a worst-case scheduler is still too powerful. If it sees a positive sum forming on the blackboard, it delays messages trying to write $+1$ in order to turn the sum temporarily negative, so the nodes finishing first see a negative sum, and the delayed nodes see a positive sum. $\square$

Remarks:

- Algorithm 19.15 cannot tolerate even one byzantine failure: assume the byzantine node generates all the n^2 coin flips in every round due to worst-case scheduling. Then this byzantine node can make sure that its coin flips always sum up to a value larger than n, thus making the outcome -1 impossible.

- In Algorithm 19.15, we assume that the blackboard is a trusted central authority. Like the random oracle of Definition 19.1, assuming a blackboard does not seem practical. However, fortunately, we can use advanced broadcast methods in order to implement something like a blackboard with just messages.

19.4 From the Blackboard to Message Passing

Algorithm 19.19 Crash-Resilient Shared Coin (code for node u)

```
 1: r = 1
 2: while true do
 3:    Choose local coin $c_u = +1$ with probability $1/2$, else $c_u = -1$
 4:    FIFO-broadcast coin($c_u, r$) to all nodes
 5:    Save all received coins coin($c_v, r$) in a set $C_u$
 6:    Wait until accepted own coin($c_u, r$)
 7:    Request $C_v$ from $n - f$ nodes $v$, and add newly seen coins to $C_u$
 8:    if $|C_u| \geq n^2$ then
 9:       return sign(sum($C_u$))
10:    end if
11:    r := r + 1
12: end while
```

Theorem 19.20. *Algorithm 19.19 implements a shared coin for $f < n/2$ crash failures.*

Proof. The upper bound for the number of crash failures results from the upper bound in Theorem 18.8. The idea of this algorithm is to simulate the read and write operations from Algorithm 19.15.

Line 4 simulates a write operation: by accepting its own coin flips, a node verifies that $n - f$ correct nodes have received its most recent generated coin flips $\texttt{coin}(c_u, r)$. At least $n - 2f \geq 1$ of these nodes will never crash and the value therefore can be considered as stored on the blackboard. While a value is not accepted and therefore not stored, node u will not generate new coin flips. Therefore, at any point of the algorithm, there are at most n additional generated coin flips next to the accepted coins.

Line 7 of the algorithm corresponds to a read operation. A node reads a value by requesting C_v from at least $n-f$ nodes v. Assume that for a coin flips $\texttt{coin}(c_u, r)$, f nodes that participated in the FIFO broadcast of this message have crashed. When requesting $n - f$ sets of coin flips, there will be at least $(n - 2f) + (n - f) - (n - f) = n - 2f \geq 1$ sets among the requested ones containing $\texttt{coin}(c_u, r)$. Therefore, a node will always read all values that were accepted so far.

This shows that the read and write operations are equivalent to the same operations in Algorithm 19.15. Assume now that some correct node has terminated after reading n^2 coin flips. Since each node reads the stored coin flips before generating a new one in the next round, there will be at most n additional coins accepted by any other node before termination. This setting is equivalent to Theorem 19.17 and the rest of the analysis is therefore analogous to the analysis in that theorem. □

Remarks:

- So finally we can deal with worst-case crash failures *and* worst-case scheduling.
- But what about byzantine agreement? We need even more powerful methods!

19.5 Shared Coin Using Cryptography

Definition 19.21 (Threshold Secret Sharing)**.** *Let $t, n \in \mathbb{N}$ with $1 \leq t \leq n$. An algorithm that distributes a secret among n participants such that t participants need to collaborate to recover the secret is called a (t, n)**-threshold secret sharing** scheme.*

Definition 19.22 (Signature)**.** *Every node can **sign** its messages in a way that no other node can forge them, thus nodes can reliably determine which node a signed message originated from. We denote a message x signed by node u with* $\texttt{msg}(x)_u$.

Algorithm 19.23 (t, n)-Threshold Secret Sharing

```
1: Input: A secret $s \in \{0, \dots, q\}$ for some prime number $q > n$.
   Secret distribution by dealer d
2: Generate $t-1$ uniformly random values $a_1, \dots, a_{t-1} \in \mathbb{F}_q$
3: Obtain a polynomial $p$ of degree $t-1$ with $p(x) = s + a_1 x + \dots + a_{t-1}x^{t-1}$
4: Distribute share $\texttt{msg}(p(1))_d$ to node $v_1$, ..., $\texttt{msg}(p(n))_d$ to node $v_n$
   Secret recovery
5: Collect $t$ shares $\texttt{msg}(p(u))_d$ from at least $t$ nodes
6: Use Lagrange's interpolation formula to obtain $p(0) = s$
```

Remarks:

- Algorithm 19.23 relies on a trusted dealer that broadcasts the secret shares to the nodes.
- We use $\mathbb{F}_q$ (the finite field of integers modulo q) and not the field of random numbers $\mathbb{R}$ because $\mathbb{R}$ is not discrete, and more importantly because there is no such thing as a "uniformly random real number." We require $q > n$ so that $\mathbb{F}_q$ contains the distinct elements $0, \dots, n$ for which we consider evaluations of the polynomial p.
- Note that the communication between the dealer and the nodes must be private, i.e., a byzantine node cannot see the shares sent to the correct nodes.

- Using an $(f+1, n)$-threshold secret sharing scheme, we can encrypt messages in such a way that byzantine nodes alone cannot decrypt them. The properties of polynomials ensure that the byzantine nodes cannot learn anything about $f(0)$ with just f shares.

Algorithm 19.24 Preprocessing Step for Algorithm 19.25 (code for dealer d)

```
1: for $i = 1, \dots, \lambda$ do
2:     Choose coin flip $c_i$, where $c_i = 0$ with probability 1/2, else $c_i = 1$
3:     Using Algorithm 19.23, generate $n$ shares $(p(1)), \dots, p(n))$ for $c_i$
4: end for
5: Send shares $\texttt{msg}(p(1))_d, \dots, \texttt{msg}(p(n))_d$ to node $u$
```

Algorithm 19.25 Shared Coin using Secret Sharing

```
1: Request shares for $c_i$ from at least $f+1$ nodes
2: Using Algorithm 19.23, let $c_i$ be the value reconstructed from the shares
3: return $c_i$
```

Theorem 19.26. *When Line 12 in Algorithm 17.19 is replaced with Algorithm 19.25, then with the setup generated by 19.24, Algorithm solves 17.19 byzantine agreement for $f < n/10$ in expected 3 rounds.*

Proof. In Line 1 of Algorithm 19.25, the nodes collect shares from $f+1$ nodes. Since a byzantine node cannot forge the signature of the dealer, it is restricted to either send its own share or decide to not send it at all. Therefore, each correct node will eventually be able to reconstruct secret c_i of round i correctly in Line 2 of the algorithm. The running time analysis for Algorithm 17.19 modified as described follows then from the analysis of Theorem 19.3. □

Remarks:

- The dealer generates λ coins for some parameter λ. This λ should be high enough so that with overwhelming probability no node ever requests shares for the non-existent coin $c_{\lambda+1}$. Algorithm 17.19 can be shown to require only λ shared coins except with probability $2^{-\lambda}$, so λ does not have to be very large for practical security.

- Nodes have to be careful when responding to requests for shares. If a node freely sends out shares for any request, the byzantine nodes can reconstruct all secret bits. In this case, the situation is the same as when using a pre-shared bit string, which does not work as we saw in Section 19.1.

- In Algorithm 19.24 we assume that the trusted dealer generates the random bit string before each execution of Algorithm 17.19 is run. Ideally, the dealer would only generate setup once, or better yet we would need no dealer at all. One can reduce or eliminate the need for the dealer's help with more advanced cryptography and/or more advanced algorithms. See for example [CKS00] and [FM88].

- Algorithm 17.19 can also be implemented in the synchronous setting.

19.6 Synchronous Byzantine Agreement Using Shared Coins

Remarks:

- A randomized version of a synchronous byzantine agreement algorithm can improve on the lower bound of $f + 1$ rounds for the deterministic algorithms.

Definition 19.27 (Cryptographic Hash Function)**.** *A hash function hash* : $U \to S$ *is called* ***cryptographically strong****, if for a given* $z \in S$ *it is computationally hard to find an element* $x \in U$ *with* $hash(x) = z$*.*

Remarks:

- Popular hash functions used in cryptography include the "**S**ecure **H**ash **A**lgorithms" SHA-2 and SHA-3.

Algorithm 19.28 Simple Synchronous Byzantine Shared Coin (for node u)

```
1: Each node has a public key that is known to all nodes.
2: Let $r$ be the current round of Algorithm 17.19
3: Broadcast $\texttt{msg}(r)_u$, i.e., round number $r$ signed by node $u$
4: Compute $h_v = \text{hash}(\text{msg}(r)_v)$ for all received messages $\texttt{msg}(r)_v$
5: Let $h_{min} = \min_v\ h_v$
6: return least significant bit of $h_{min}$
```

Remarks:

- In Algorithm 19.28, Line 3 each node can verify the correctness of the signed message using the public key.
- Just as in Algorithm 17.7, the decision value is the minimum of all received values. While the minimum value is received by all nodes after 2 rounds there, we can only guarantee to receive the minimum with constant probability in this algorithm because a byzantine node may have the minimum value.
- We will treat the hash function like it is a uniformly random function, chosen just before the algorithm runs. This idealized way of treating a hash function is called the Random Oracle Model.
- The signature scheme must have the property that it is impossible (or at least computationally challenging) to compute two different valid signatures on a message with the same signing key. Otherwise, the byzantine nodes could sign the round number multiple times to find a signature with a small hash value, and this would break the algorithm.

Theorem 19.29. *Algorithm 19.28 plugged into Algorithm 17.19 solves synchronous byzantine agreement in approximately 3 rounds in expectation for up to* $f < n/10$ *byzantine failures.*

Proof. With probability lower than 1/10 the minimum hash value is generated by a byzantine node. In such a case, we can assume that not all correct nodes will receive the byzantine value and thus, different nodes might compute different values for the shared coin.

With probability greater than 9/10, the shared coin will be from a correct node, and with probability 1/2 the value of the shared coin will correspond to the value which was deterministically chosen by some of the correct nodes. Therefore, with probability 9/20 the nodes will reach consensus in the next iteration of Algorithm 17.19. Thus, the expected number of rounds is around 3 (the expected number of rounds until nodes agree is 20/9 plus one more round to terminate). □

Remarks:

- Theorem 19.29 states that byzantine agreement can be achieved in the synchronous model in a small number of rounds using advanced tools such as shared coins and cryptographically strong hash functions. Can we achieve a small number of rounds as well for a larger share of Byzantine nodes? The answer is yes!

Algorithm 19.30 Fast Synchronous Byzantine Agreement

```
1: $x_u \in \{0, 1\}$.
2: while true do
3:   broadcast propose($x_u$)
4:   $x_u$ := most frequently received value
5:   if $\geq n - f$ propose messages contain the same value $x_u$ then
6:     decide on $x_u$
7:     broadcast propose($x_u$, decided)
8:     terminate
9:   else
10:    broadcast propose($x_u$)
11:    $x_u$ := most frequently received value
12:    if $< n - f$ propose messages contain the same value $x$ and coin_toss() = 0 then
13:      $x_u$ := 0
14:    end if
15:  end if
16: end while
```

Remarks:

- `coin_toss` is implemented using Algorithm 19.28.
- Algorithm 19.30 looks similar to Algorithm 17.19 but it uses two broadcasts (i.e., two rounds) per while-loop iteration. If at least $n - f$ nodes propose the same value after the first broadcast in every loop, the node accepts the value and terminates.
- The flag *decided* in Line 6 is used to indicate that the node will not broadcast its value again in future rounds. All nodes receiving such

a message implicitly assume that they receive the same message from this node in every round moving forward.

- Let the random variable C denote the outcome of the shared random coin toss with domain $\{0, 1, \perp\}$, where $C = 0$ and $C = 1$ mean that all correct nodes obtain 0 and 1, respectively, and $C = \perp$ indicates that the random coin toss failed.
- In the following, we assume that $f < n/4$.

Lemma 19.31 (Shared Coin Toss)**.** *If $f < n/4$, it holds that $p(C = 0) = p(C = 1) > 27/64$.*

Proof. According to Algorithm 19.28, every node computes the least significant bit of the minimum hash of all messages received of a specific format. If a correct node happens to produce the smallest hash, the shared coin toss will succeed, which happens with probability $1 - f/n$. If a byzantine node produces the smallest hash value, it is still possible that a correct node produces the second smallest hash value *and* the least significant bit is the same as the least significant bit of the smallest hash value. The probability of this event is $\frac{f}{n}\frac{n-f}{n-1}\frac{1}{2}$. Thus, we get that

$$\begin{aligned} p(C = 0) \vee p(C = 1) &> 1 - f/n + \frac{f}{n}\frac{n-f}{n-1}\frac{1}{2} \\ &> 1 - f/n + \frac{f}{n}\frac{n-f}{n}\frac{1}{2} \\ &> 1 - \frac{1}{4} + \frac{3}{32} = \frac{27}{32}. \end{aligned}$$

Since $p(C = 0) = p(C = 1)$, the claim follows. □

Lemma 19.32 (All-Same Validity)**.** *If all correct nodes start with the same input value, the correct nodes will decide on this value.*

Proof. If all correct nodes broadcast the same value x, all of them will receive it at least $n - f$ times and decide on x, according to Lines 3-6. □

Lemma 19.33 (Agreement)**.** *Let round r be the first round when a correct node v decides on x_v. Each correct node w will decide on $x_w = x_v$ in round r or $r + 2$.*

Proof. Assume that w decides on $x_w \neq x_v$ in round r. Both v and w must have received $n - 2f$ `propose` messages for their values from distinct correct nodes. However, this is a contradiction because $n - 2f + n - 2f > n$. Thus, no correct node decides on a different value in round r. Assume that w does not decide in round r. Since every correct node received x_v at least $n - 2f > n/2$ times, x_v is the most frequently received value. Thus, every correct node broadcasts this value in round $r + 1$. As this value is broadcast by every correct node, every correct node receives it at least $n - f$ times, which implies that they will not toss a coin and change the value to 0. All correct nodes broadcast x_v in round $r + 2$ and thus receive at least $n - f$ times, causing them to decide on x_v. □

Lemma 19.34 (Termination)**.** *Every correct node decides on a value in at most $5\frac{3}{4}$ rounds in expectation.*

Proof. We bound the number of times `coin_toss` is invoked.

Case 1: No correct node receives the same value at least $n - f$ times in Line 11. In this case, all correct nodes perform a shared coin toss. If the coin toss returns 0, all correct nodes will decide on 0 in the next round.

Case 2: Some correct node v receives the same value at least $n - f$ times in Line 11. In this case, every correct node u receives this value at least $n - 2f > n/2$ times and sets $x_u := x_v$ in Line 11. If the coin toss returns 1, all correct nodes will decide on $x_u = x_v$ in the next round.

Thus, in either case, the probability of termination in the next round is at least 27/64 according to Lemma 19.31, i.e., there are fewer than 64/27 coin tosses in expectation. Since each loop consists of 2 rounds, and there is one more round after the successful coin toss before deciding, the expected number of rounds is lower than $1 + 2 \cdot 64/27 < 5\frac{3}{4}$. □

Remarks:

- The bound can be improved to 5 rounds in expectation using a single-round shared coin routine that guarantees that $p(C = 0) = p(C = 1) = 1/2$, which can be built, e.g., using *threshold signatures*.

- What about non-binary input? The algorithm still works; however, if all nodes are correct and have different values, they will agree on 0, which is not great. This can be fixed by introducing a round at the beginning where a *leader* proposes a value that correct nodes accept, i.e., they set their local variable to the leader's value, if they receive one. If the leader is correct, all correct nodes decide in 2 rounds deterministically! If the leader is byzantine, the correct nodes decide in $6\frac{3}{4}$ (or 6) rounds in expectation.

- In practice, such algorithms are not run just once. On the contrary, they are executed continuously to agree on the latest set of state changes. As we just saw, whenever the leader is correct, which happens in at least three out of four runs in expectation (using, e.g., a simple round-robin strategy), only two rounds are needed, and the expected number of rounds is $6\frac{3}{4}$ (or 6 using a perfect shared coin) if the leader is byzantine. Thus, over many executions, the expected number of rounds is only $2 \cdot \frac{3}{4} + 6\frac{3}{4} \cdot \frac{1}{4} \approx 3$ (or exactly 3 using a perfect shared coin)!

Chapter Notes

The concept of a shared coin was introduced by Bracha [Bra87]. The shared coin algorithm in this chapter was proposed by [AW04]and it assumes randomized scheduling. A shared coin that can withstand worst-case scheduling has been developed by Alistarh et al. [AAKS14a]; this shared coin was inspired by earlier shared coin solutions in the shared memory model [Cha96].

Asynchronous byzantine agreement is usually considered in one out of two communication models – shared memory or message passing. The first polynomial algorithm for the shared memory model that uses a shared coin was proposed by Aspnes and Herlihy [AH90] and required exchanging $O(n^4)$ messages in total. Algorithm 19.15 is also an implementation of the shared coin in the shared memory model and it requires exchanging $O(n^3)$ messages. This variant is due to Saks, Shavit and Woll [SSW91]. Bracha and Rachman [BR92] later reduced the number of messages exchanged to $O(n^2 \log n)$. The tight lower bound of $\Omega(n^2)$ on the number of coin flips was proposed by Attiya and Censor [AC08] and improved the first non-trivial lower bound of $\Omega(n^2/\log^2 n)$ by Aspnes [Asp98].

In the message-passing model, the shared coin is usually implemented using reliable broadcast, which we covered in Chapter 18. A possible way to reduce message complexity is by simulating the read and write commands [ABND95] as in Algorithm 19.19. The message complexity of this method is $O(n^3)$. Alistarh et al. [AAKS14b] improved the number of exchanged messages to $O(n^2 \log^2 n)$ using a binary tree that restricts the number of communicating nodes according to the depth of the tree.

It remains an open question whether asynchronous byzantine agreement can be solved in the message passing model without cryptographic assumptions. If cryptographic assumptions are used, however, byzantine agreement can be solved in a constant number of rounds in expectation. Algorithm 19.24 presents the first implementation due to Rabin [Rab83] using threshold secret sharing. This algorithm relies on the fact that the dealer provides the random bit string. Chor et al. [CGMA85] proposed the first algorithm where the nodes use verifiable secret sharing in order to generate random bits. Later work focuses on improving resilience [CR93] and practicability [CKS00]. Algorithm 19.28 by Micali [Mic18] shows that cryptographic assumptions can also help to improve the running time in the synchronous model.

Algorithm 19.30 can handle $f < n/4$ byzantine failures in the synchronous communication model. This algorithm currently achieves the lowest number of rounds in expectation [Loc20].

This chapter was written in collaboration with Darya Melnyk and Thomas Locher.

Bibliography

[AAKS14a] Dan Alistarh, James Aspnes, Valerie King, and Jared Saia. Communication-efficient randomized consensus. In *28th International Symposium of Distributed Computing (DISC), Austin, TX, USA, October 12-15, 2014*, pages 61–75, 2014.

[AAKS14b] Dan Alistarh, James Aspnes, Valerie King, and Jared Saia. Communication-efficient randomized consensus. In Fabian Kuhn, editor, *Distributed Computing*, pages 61–75, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[ABND95] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, January 1995.

[AC08] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *J. ACM*, 55(5):20:1–20:26, November 2008.

[AH90] James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441 – 461, 1990.

[Asp98] James Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM*, 45(3):415–450, May 1998.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition).* John Wiley Interscience, March 2004.

[BR92] Gabriel Bracha and Ophir Rachman. Randomized consensus in expected $o(n^2 logn)$ operations. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, WDAG '91, pages 143–150, Berlin, Heidelberg, 1992. Springer-Verlag.

[Bra87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, Oct 1985.

[Cha96] Tushar Deepak Chandra. Polylog randomized wait-free consensus. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA*, pages 166–175, 1996.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18:219–246, 2000.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, New York, NY, USA, 1993. ACM.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. STOC '88, pages 148–161, New York, NY, USA, 1988. Association for Computing Machinery.

[Loc20] Thomas Locher. Fast byzantine agreement for permissioned distributed ledgers. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 371–382, 2020.

[Mic18] Silvio Micali. Byzantine agreement , made trivial. 2018.

[Rab83] M. O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409, Nov 1983.

[SSW91] Michael Saks, Nir Shavit, and Heather Woll. Optimal time randomized consensus – making resilient algorithms fast in practice. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 351–362, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.