Chapter 12

Broadcast & Shared Coins

In Chapter 11 we have developed a fast solution for synchronous byzantine agreement (Algorithm 11.14), yet our *asynchronous* byzantine agreement solution (Algorithm 11.21) is still awfully slow. Is there a fast asynchronous algorithm, possibly based on some advanced communication methods?

12.1 Random Oracle and Bitstring

Definition 12.1 (Random Oracle)**.** *A random oracle is a trusted (non-byzantine) random source which can generate random values.*

Algorithm 12.2 Shared Coin with Magic Random Oracle

```
1: return $c_i$, where $c_i$ is $i$th random bit by oracle
```

Remarks:

- Algorithm 12.2 as well as the following shared coin algorithms will for instance be called in Line 12 of Algorithm 11.21. So instead of every node throwing a local coin (and hoping that they all show the same), the nodes throw a *shared* coin. In other words, the value x_u in Line 12 of Algorithm 11.21 will be set to the return value of the shared coin subroutine.

- We have already seen a shared coin in Algorithm 8.22. This concept deserves a proper definition.

Definition 12.3 (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.*

Theorem 12.4. *Algorithm 12.2 plugged into Algorithm 11.21 solves asynchronous byzantine agreement in 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 11.21 satisfies all-same-validity; the shared coin is not even used.

If there is no significant majority for any of the input values at the beginning of algorithm 11.21, all correct nodes will run Algorithm 12.2. 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 see an $n/2 + f + 1$ majority for one of the input values, 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 12.5 (Random Bitstring)**.** *A* ***random bitstring*** *is a string of random binary values, known to all participating nodes when starting a protocol.*

Algorithm 12.6 Naive Shared Coin with Random Bitstring

```
1: return $b_i$, where $b_i$ is $i$th bit in common random bitstring
```

Remarks:

- But is such a precomputed bitstring really random enough? We should be worried because of Theorem 8.14.

Theorem 12.7. *If the scheduling is worst-case, Algorithm 12.6 plugged into Algorithm 11.21 does not terminate.*

Proof. We start Algorithm 12.6 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 bitstring is 0.

If the second random bit in the bitstring 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 bitstring 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 that never decide.

□

Remarks:

- Theorem 12.7 shows that a worst-case scheduler cannot be allowed to know the random bits of the future.
- Note that in the proof of Theorem 12.7 we did not even use any byzantine nodes. Just bad scheduling was enough to prevent termination.
- Worst-case scheduling is an issue that we have not considered so far, in particular in Chapter 8 we implicitly assumed that message scheduling was random. What if scheduling is worst-case in Algorithm 8.22?

Lemma 12.8. *Algorithm 8.22 has exponential expected running time under worst-case scheduling.*

Proof. In Algorithm 8.22, worst-case scheduling may hide up to f rare zero coinflips. 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 $(1/n)^{f+1}$, which is exponentially small for $f \in \Omega(n)$. In other words, with worst-case scheduling, with probability $1-(1/n)^{f+1}$ the shared coin will be 1. The worst-case scheduler must make sure that some nodes will always deterministically go for 0, and the algorithm needs n^{f+1} rounds until it terminates. □

Remarks:

- With worst-case asynchrony, some of our previous results do not hold anymore. Can we at least solve asynchronous (assuming worst-case scheduling) *consensus* if we have crash failures?
- This is indeed possible, but we need to sharpen our tools first.

12.2 Shared Coin on a Blackboard

Definition 12.9 (Blackboard Model)**.** *The* ***blackboard*** *is a trusted authority which 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.*

Remarks:

- We assume that the nodes cannot reconstruct the order in which the messages are written to the blackboard, since the system is asynchronous.

Algorithm 12.10 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 coinflips on the blackboard
5:   if $|C| \geq n^2$ then
6:     return sign(sum(C))
7:   end if
8: end while
```

Remarks:

- In Algorithm 12.10 the outcome of a coinflip 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 coinflips in C is positive, and -1 if it is negative.
- 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 12.10, a single node can singlehandedly generate all n^2 coinflips, 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 12.11 (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 real number z,*

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

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

Theorem 12.12. *Algorithm 12.10 implements a polynomial shared coin.*

Proof. Each node in the algorithm terminates once at least n^2 coinflips are written to the blackboard. Before terminating, nodes may write one additional coinflip. Therefore, every node decides after reading at least n^2 and at most $n^2 + n$ coinflips. The power of the adversary lies in the fact that it can prevent $n - 1$ nodes from writing their coinflips to the blackboard by delaying their writes. Here, we will consider an even stronger adversary that can hide up to n coinflips which were 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 12.3. Let X be the sum of all coinflips 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 12.11 with $N = n^2$ and $z = 1$, we get:

$$Pr(X < -n) = Pr(X > n) = 1 - Pr(X \leq n) = 1 - \Phi(1) > 0.15.$$

□

Lemma 12.13. *Algorithm 12.10 uses n^2 coinflips, 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 8.14. It requires involved stochastic methods and we therefore will only sketch the idea of where the n^2 comes from.

The basic idea follows from Theorem 12.11. The standard deviation of the sum of n^2 coinflips is n. The central limit theorem tells us that with constant probability the sum of the coinflips will be only a constant factor away from the standard deviation. As we showed in Theorem 12.12, this is large enough to disarm a worst-case scheduler. However, with much less than n^2 coinflips, 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. □

Remarks:

- Algorithm 12.10 cannot tolerate even one byzantine failure: assume the byzantine node generates all the n^2 coinflips in every round due to worst-case scheduling. Then this byzantine node can make sure that its coinflips always sum up to a value larger than n, thus making the outcome -1 impossible.

- In Algorithm 12.10, we assume that the blackboard is a trusted central authority. Like the random oracle of Definition 12.1, 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.

12.3 Broadcast Abstractions

Definition 12.14 (Accept)**.** *A message received by a node v is called **accepted** if node v can consider this message for its computation.*

Definition 12.15 (Best-Effort Broadcast)**.** ***Best-effort broadcast** ensures that a message that is sent from a correct node u to another correct node v will eventually be received and accepted by v.*

Remarks:

- Note that best-effort broadcast is equivalent to the simple broadcast primitive that we have used so far.
- Reliable broadcast is a stronger paradigm which implies that byzantine nodes cannot send different values to different nodes. Such behavior will be detected.

Definition 12.16 (Reliable Broadcast)**.** ***Reliable broadcast*** *ensures that the nodes eventually agree on all accepted messages. That is, if a correct node v considers message m as accepted, then every other node will eventually consider message m as accepted.*

Algorithm 12.17 Asynchronous Reliable Broadcast (code for node u)

```
1: Broadcast own message msg($u$)
2: if received msg($v$) from node $v$ then
3:     Broadcast echo($u$, msg($v$))
4: end if
5: if received echo($w$, msg($v$)) from $n - 2f$ nodes $w$ but not msg($v$) then
6:     Broadcast echo($u$, msg($v$))
7: end if
8: if received echo($w$, msg($v$)) from $n - f$ nodes $w$ then
9:     Accept(msg($v$))
10: end if
```

Theorem 12.18. *Algorithm 12.17 satisfies the following properties:*

1. *If a correct node broadcasts a message reliably, it will eventually be accepted by every other correct node.*
2. *If a correct node has not broadcast a message, it will not be accepted by any other correct node.*
3. *If a correct node accepts a message, it will be eventually accepted by every correct node*

Proof. We start with the first property. Assume a correct node broadcasts a message msg(v), then every correct node will receive msg(v) eventually. In Line 3, every correct node (including the originator of the message) will echo the message and, eventually, every correct node will receive at least $n - f$ echoes, thus accepting msg(v).

The second property follows from byzantine nodes being unable to forge an incorrect sender address, see Definition 11.1.

The third property deals with a byzantine originator b. If a correct node accepted message msg(b), this node must have received at least $n - f$ echoes for this message in Line 8. Since at most f nodes are byzantine, at least $n - 2f$ correct nodes have broadcast an echo message for msg(b). Therefore, every correct node will receive these $n - 2f$ echoes eventually and will broadcast an echo itself. Thus, all $n - f$ correct nodes will have broadcast an echo for msg(b) and every correct node will accept msg(b).

□

Remarks:

- Algorithm 12.17 does not terminate. Only *eventually*, all messages by correct nodes will be accepted.
- The algorithm has a linear message overhead, since every node again broadcasts every message.
- Note that byzantine nodes can issue arbitrarily many messages. This may be a problem for protocols where each node is only allowed to send one message (per round). Can we fix this, for instance with sequence numbers?

Definition 12.19 (FIFO Reliable Broadcast)**.** *The* ***FIFO (reliable) broadcast*** *defines an order in which the messages are accepted in the system. If a node u broadcasts message m_1 before m_2, then any node v will accept message m_1 before m_2.*

Algorithm 12.20 FIFO Reliable Broadcast (code for node u)

```
Broadcast own round $r$ message msg$(u, r)$
if received first message msg$(v, r)$ from node $v$ for round $r$ then
  Broadcast echo$(u, \text{msg}(v, r))$
end if
if not echoed any msg'$(v, r)$ before then
  if received echo$(w, \text{msg}(v, r))$ from $f+1$ nodes $w$ but not msg$(v, r)$ then
    Broadcast echo$(u, \text{msg}(v, r))$
  end if
end if
if received echo$(w, \text{msg}(v, r))$ from $n - f$ nodes $w$ then
  if accepted msg$(v, r - 1)$ then
    Accept(msg$(v, r)$)
  end if
end if
```

Theorem 12.21. *Algorithm 12.20 satisfies the properties of Theorem 12.18. Additionally, Algorithm 12.20 makes sure that no two messages* `msg`(v, r) *and* `msg'`(v, r) *are accepted from the same node. It can tolerate $f < n/3$ Byzantine nodes or $f < n/2$ crash failures.*

Proof. Just as reliable broadcast, Algorithm 12.20 satisfies the first two properties of Theorem 12.18 by simply following the flow of messages of a correct node.

For the third property, assume again that some message originated from a byzantine node b. If a correct node accepted message `msg`(b), this node must have received at least $n - f$ echoes for this message in Line 10.

- Byzantine case: If at most f nodes are byzantine, at least $n - 2f > f + 1$ correct nodes have broadcast an echo message for `msg`(b).
- Crash-failure case: If at most f nodes can crash, at least $n - f > f + 1$ nodes have broadcast an echo message for `msg`(b).

In both cases, every correct node will receive these $f + 1$ echoes eventually and will broadcast an echo. Thus, all $n - f$ correct nodes will have broadcast an echo for msg(b) and every correct node will accept msg(b).

It remains to show that at most one message will be accepted from some node v in a round r.

- Byzantine case: Assume that some correct node u has accepted msg(v, r) in Line 12. Then, u has received $n-f$ echoes for this message, $n-2f$ of which were the first echoes of the correct nodes. Assume for contradiction that another correct node accepts msg'(v, r). This node must have collected $n-f$ messages echo(w, msg'(v, r)). Since at least $n-2f$ of these messages must be the first echo messages sent by correct nodes, we have $n - 2f + n - 2f = 2n - 4f > n - f$ (for $f < n/3$) echo messages sent by the correct nodes as their first echo. This is a contradiction.
- Crash-failure case: At least $n - 2f$ not crashed nodes must have echoed msg(v, r), while $n-f$ nodes have echoed msg'(v, r). In total $2n-3f > n-f$ (for $f < n/2$) correct nodes must have echoed either of the messages, which is a contradiction.

□

Definition 12.22 (Atomic Broadcast). ***Atomic broadcast*** *makes sure that all messages are received in the same order by every node. That is, for any pair of nodes u, v, and for any two messages m_1 and m_2, node u receives m_1 before m_2 if and only if node v receives m_1 before m_2.*

Remarks:

- Definition 12.22 is equivalent to Definition 7.8, i.e., atomic broadcast = state replication.
- Now we have all the tools to finally solve asynchronous consensus.

12.4 Blackboard with Message Passing

Algorithm 12.23 Crash-Resilient Shared Coin (code for node u)

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

Theorem 12.24. *Algorithm 12.23 solves asynchronous binary agreement for $f < n/2$ crash failures.*

Proof. The upper bound for the number of crash failures results from the upper bound in 12.21. The idea of this algorithm is to simulate the read and write operations from Algorithm 12.10.

Line 3 simulates a read operation: by accepting the own coinflip, a node verifies that $n - f$ correct nodes have received its most recent generated coinflip $\texttt{coin}(c_u, r)$. At least $n - 2f > 1$ 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 coinflips. Therefore, at any point of the algorithm, there is at most n additional generated coinflips next to the accepted coins.

Line 6 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 coinflip $\texttt{coin}(c_u, r)$, f nodes that participated in the FIFO broadcast of this message have crashed. When requesting $n - f$ sets of coinflips, there will be at least $(n - 2f) + (n - f) - (n - f) = n - 2f > 1$ 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 12.10. Assume now that some correct node has terminated after reading n^2 coinflips. Since each node reads the stored coinflips 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 12.12 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!

12.5 Using Cryptography

Definition 12.25 (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 12.26 (Signature)**.** *Every node can **sign** its messages in a way that no other node can forge, 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 12.27 (t, n)-Threshold Secret Sharing

```
1: Input: A secret $s$, represented as a real number.
   *Secret distribution by dealer $d$*
2: Generate $t-1$ random numbers $a_1, \dots, a_{t-1} \in \mathbb{R}$
3: Obtain a polynomial $p$ of degree $t-1$ with $p(x) = s + a_1 x + \dots + a_{t-1}x^{t-1}$
4: Generate $n$ distinct $x_1, \dots, x_n \in \mathbb{R} \setminus \{0\}$
5: Distribute share $\texttt{msg}(x_1, p(x_1))_d$ to node $v_1$, ..., $\texttt{msg}(x_n, p(x_n))_d$ to node $v_n$
   *Secret recovery*
6: Collect $t$ shares $\texttt{msg}(x_u, p(x_u))_d$ from at least $t$ nodes
7: Use Lagrange's interpolation formula to obtain $p(0) = s$
```

Remarks:

- Algorithm 12.27 relies on a trusted dealer, who broadcasts the secret shares to the 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.

Algorithm 12.28 Preprocessing Step for Algorithm 12.29 (code for dealer d)

```
1: According to Algorithm 12.27, choose polynomial $p$ of degree $f$
2: for $i = 1, \dots, n$ do
3:     Choose coinflip $c_i$, where $c_i = 0$ with probability $1/2$, else $c_i = 1$
4:     Using Algorithm 12.27, generate $n$ shares $(x_1^i, p(x_1^i)), \dots, (x_n^i, p(x_n^i))$ for $c_i$
5: end for
6: Send shares $\texttt{msg}(x_u^1, p(x_u^1))_d, \dots, \texttt{msg}(x_u^n, p(x_u^n))_d$ to node $u$
```

Algorithm 12.29 Shared Coin using Secret Sharing (ith iteration)

```
1: Request shares from at least $f+1$ nodes
2: Using Algorithm 12.27, let $c_i$ be the value reconstructed from the shares
3: return $c_i$
```

Theorem 12.30. *Algorithm 11.21 together with Algorithm 12.28 and Algorithm 12.29 solves asynchronous byzantine agreement for $f < n/3$ in expected* 3 *number of rounds.*

Proof. In Line 1 of Algorithm 12.29, 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 follows then from the analysis of Theorem 12.4. □

Remarks:

- In Algorithm 12.28 we assume that the dealer generates the random bitstring. This assumption is not necessary in general.
- We showed that cryptographic assumptions can speed up asynchronous byzantine agreement.
- Algorithm 11.21 can also be implemented in the synchronous setting.
- A randomized version of a synchronous byzantine agreement algorithm can improve on the lower bound of $t + 1$ rounds for the deterministic algorithms.

Definition 12.31 (Cryptographic Hash Function)**.** *A hash function hash* $: U \to S$ *is called* ***cryptographic****, 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 Secure Hash Algorithm (SHA) and the Message-Digest Algorithm (MD).

Algorithm 12.32 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 11.21
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 12.32, Line 3 each node can verify the correctness of the signed message using the public key.
- Just as in Algorithm 11.9, 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.
- Hashing helps to restrict byzantine power, since a byzantine node cannot compute the smallest hash.

Theorem 12.33. *Algorithm 12.32 plugged into Algorithm 11.21 solves synchronous byzantine agreement in expected* 5 *rounds for up to* $f < n/10$ *byzantine failures.*

Proof. With probability 1/3 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 2/3, 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 1/3 the nodes will reach consensus in the next iteration of Algorithm 11.21. The expected number of rounds is:

$$1 + \sum_{i=0}^{\infty} 2 \cdot \left(\frac{2}{3}\right)^i = 5$$

□

Chapter Notes

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 12.10 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 coinflips 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. Reliable broadcast was first proposed by Srikanth and Toueg [ST87] as a method to simulate authenticated broadcast. There is also another implementation which was proposed by Bracha [Bra87]. Today, a lot of variants of reliable broadcast exist, including FIFO broadcast [AAD05], which was considered in this chapter. A good overview over the broadcast routines is given by Cachin et al. [CGR14]. A possible way to reduce message complexity is by simulating the read and write commands [ABND95] as in Algorithm 12.23. The message complexity of this method is $O(n^3)$. Alistarh et al. [AAKS14] 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 however used, byzantine agreement can be solved in expected constant number of rounds. Algorithm 12.28 presents the first implementation due to Rabin [Rab83] using threshold secret sharing. This algorithm relies on the fact that the dealer provides the random bitstring. 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 12.32 by Micali [Mic18] shows that cryptographic assumptions can also help to improve the running time in the synchronous model.

This chapter was written in collaboration with Darya Melnyk.

Bibliography

[AAD05] Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, OPODIS'04, pages 229–239, Berlin, Heidelberg, 2005. Springer-Verlag.

[AAKS14] 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.

[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.

[CGR14] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. *Introduction to Reliable and Secure Distributed Programming.* Springer Publishing Company, Incorporated, 2nd edition, 2014.

[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.

[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.

[ST87] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34:626–645, 1987.