Chapter 18

Broadcast

A fundamental building block of distributed systems is the functionality to broadcast a message, i.e., to send it to all other nodes.

18.1 Best-Effort Broadcast

Definition 18.1 (Accept)**.** *A message received by a node v is called **accepted** if node v can consider this message for its computation.*

Definition 18.2 (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.
- In the following, we study more powerful broadcast primitives.

18.2 Broadcast with Totality

Definition 18.3 (Broadcast with Totality)**.** ***Broadcast with totality** 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 correct node will eventually consider message m as accepted.*

Algorithm 18.4 Broadcast with totality (code for node u)

```
1: Sender only: Broadcast message msg(u)
2: upon receiving msg(v) from v and not broadcast msg(v) before:
3:     Broadcast msg(v)
4:     Accept msg(v)
5: end upon
```

Theorem 18.5. *Algorithm 18.4 satisfies the following properties:*

1. *Validity: If a correct node broadcasts a message reliably, it will eventually be accepted by every other correct node.*

2. *Weak integrity: If a correct node has not broadcast a message, it will not be accepted by any other correct node.*

3. *Totality: If a correct node accepts a message, it will be eventually accepted by every correct node.*

This algorithm can tolerate $f < n$ failures.

Proof. We start with the validity property. If a correct node broadcasts a message $\texttt{msg}(v)$, then every correct node will receive $\texttt{msg}(v)$ eventually and accept it.

The weak integrity property holds for the case of crash failures where nodes only accept and broadcast messages that they received via broadcast.

The totality property holds because every correct node re-broadcasts any message that it receives. Thus, all correct nodes eventually accept the same messages. □

Remarks:

- Algorithm 18.4 does not solve consensus according to Definition 16.1. It only makes sure that all messages of correct nodes will be accepted *eventually*. For correct nodes, this corresponds to sending and receiving messages in the asynchronous model (Model 16.15).

- The algorithm has a message overhead linear in the number of nodes 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 18.6 (FIFO Broadcast)**.** ***FIFO 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 18.7 FIFO Broadcast (code for node u)

1: Broadcast own round r message $\texttt{msg}(u, r)$
2: **upon** receiving first message $\texttt{msg}(v, r)$ from node v for round r or $n - 2f$ $\texttt{echo}(w, \text{msg}(v, r))$ messages:
3: Broadcast $\texttt{echo}(u, \text{msg}(v, r))$
4: **end upon**
5: **upon** receiving $\texttt{echo}(w, \text{msg}(v, r))$ from $n - f$ nodes and node u accepted $\texttt{msg}(v, r - 1)$:
6: Accept $\texttt{msg}(v, r)$
7: **end upon**

Theorem 18.8. *Algorithm 18.7 satisfies the properties of Theorem 18.5. Additionally, Algorithm 18.7 makes sure that no two messages* `msg`(v, r) *and* `msg'`(v, r) *are accepted from the same node. It can tolerate* $f < n/5$ *byzantine nodes or* $f < n/2$ *crash failures.*

Proof. We prove each property separately.

Validity: Assume that a correct node u has already FIFO broadcast a value for round $r - 1$. Node u now broadcasts `msg`(v, r). Every correct node w broadcasts `echo`$(w, \text{msg}(v, r))$ upon receiving u's message. Thus, all correct eventually receive $n - f$ such `echo` messages and accept `msg`(v, r). In the crash-failure model, nodes do not send bogus messages. In the byzantine-failure model, note that no correct node w broadcasts `echo`$(w, \text{msg}(v', r))$, for $v' \neq v$ because that requires $n - 2f > f$ `echo` messages.

Weak integrity: It is clear that weak integrity holds in the crash-failure model. As far as the byzantine-failure model is concerned, assume for the sake of contradiction that the correct node u never broadcast (v, r) but some correct nodes accept this message. Let w be the first correct node that sent `echo`$(w, \text{msg}(v, r))$. Since u never sent `msg`(v, r), w must have received $n - 2f$ `echo` messages. However, $n - 2f > f$, which implies that there must be another correct node that sent an `echo` message to w, which contradicts the assumption that w is the first such node.

Totality: Assume that a correct node u accepts `msg`(v, r). Since u must have received $n - f$ `echo` messages and thus at least $n - 2f$ `echo` messages from correct nodes, it follows that all correct nodes eventually receive at least $n - 2f$ such `echo` messages and broadcast an `echo` message themselves. Thus, all correct nodes eventually receive at least $n - f$ `echo` messages and accept `msg`(v, r).

It remains to show that at most one message will be accepted from some node v in round r. In the crash-failure case, this property holds because all nodes follow the algorithm and therefore send at most one message in a round. For the byzantine-failure case, assume some correct node u has accepted `msg`(v, r) in Line 6. This node must have received $n - f$ `echo` messages for this message, $n - 2f$ of which were sent from the correct nodes. At least $n - 2f - f = n - 3f$ of those messages are sent for the first time by correct nodes. Now, assume for contradiction that another correct node accepts `msg'`(v, r). Similarly, $n - 3f$ of those messages are sent for the first time by correct nodes. So, we have $n - 3f + n - 3f > n - f$ (for $f < n/5$) correct nodes sent `echo` that for the first time, a contradiction. □

18.3 Reliable Broadcast

Algorithm 18.4 has the valuable *totality* property: Regardless of whether or not the sender is correct, if a node v accepts a value, all other nodes accept the same value eventually. Algorithm 18.4, however, allows nodes to accept multiple values. A broadcast algorithm is called a reliable broadcast algorithm if it also has the *(strong) integrity* property:

Definition 18.9 (integrity)**.** *A broadcast algorithm has the integrity property if every correct node delivers at most one message. The delivered message must have been broadcast by a node.*

Remarks:

- Since only one message is delivered, we also insist that this message is the same across all nodes.

Definition 18.10 (Agreement)**.** *If correct nodes v and v' accept messages m and m', respectively, then it must hold that $m = m'$.*

Algorithm 18.11 Reliable Broadcast: Iteration i, Sender v_S

Code for sender v_S with input x_S:
Send $\texttt{msg}_{<i,v_S>}(x_S)$ to everyone.

Code for node v:
// Ignore any messages tagged with different values i, v_S.
upon receiving $\texttt{msg}_{<i,v_S>}(x)$ from v_S:
  If no $\texttt{echo}_{<i,v_S>}$ message was sent in this instance:
    Send $\texttt{echo}_{<i,v_S>}(x)$ to everyone.
end upon

upon receiving $\texttt{echo}_{<i,v_S>}(x)$ from $n - f$ distinct nodes or
  $\texttt{ready}_{<i,v_S>}(x)$ from $f + 1$ distinct nodes:
  Send $\texttt{ready}_{<i,v_S>}(x)$ to everyone.
end upon

upon receiving $\texttt{ready}_{<i,v_S>}(x)$ from $2f + 1$ distinct nodes:
  Accept $\texttt{msg}_{<i,v_S>}(x)$.
end upon

Theorem 18.12. *Algorithm 18.11 achieves the following properties, even when $f < n/3$ of the nodes involved are byzantine:*

- *Validity: If the sender v_S is correct, every correct node accepts $\texttt{msg}_{<i,v_S>}(x_S)$.*
- *Totality: If a correct node accepts $\texttt{msg}_{<i,v_S>}(x)$, then every correct node accepts $\texttt{msg}_{<i,v_S>}(x)$ eventually*
- *Integrity: At most one message is accepted, and this message must have been broadcast.*
- *Agreement: If a message is accepted, all correct nodes accept the same message.*

Remarks:

- The tag $< i, v_S >$ represents the instance's identifier. This enables nodes to distinguish which messages belong to which algorithm execution, which is helpful when running multiple instances of the algorithm. Most often, for simplicity of presentation, the identifiers are implicit.

- Messages with differents tag $< i', v'_S >$ are ignored in an instance i with sender v_S. You can think of them as being stored in some queue until instance i with sender v_S is running.

Lemma 18.13. *Assume that the sender v_S is correct and has input x_S. Then, every correct node accepts $\texttt{msg}_{<i,v_S>}(x_S)$.*

Proof. First, since v_S is correct, no correct node sends $\texttt{echo}_{<i,v_S>}(y)$ for any value $y \neq x_S$, and hence no correct node sends $\texttt{ready}_{<i,v_S>}(y)$ for any $y \neq x_S$. Therefore, no correct node accepts $y \neq x_S$.

Every node eventually receives $\texttt{msg}_{<i,v_S>}(x_S)$ from the correct sender, and therefore every correct node eventually sends $\texttt{echo}_{<i,v_S>}(x_S)$. It follows that every correct node eventually receives $n-f$ messages $\texttt{echo}_{<i,v_S>}(x_S)$ and hence sends $\texttt{ready}_{<i,v_S>}(x_S)$. Finally, every correct node eventually receives $n-f$ messages $\texttt{ready}_{<i,v_S>}(x_S)$ and accepts $\texttt{msg}_{<i,v_S>}(x_S)$. □

Lemma 18.14. *If a correct node v sends $\texttt{ready}_{<i,v_S>}(x)$, then no correct node sends $\texttt{ready}_{<i,v_S>}(y)$ for $y \neq x$.*

Proof. Without loss of generality, assume that v and u are the first correct nodes that send $\texttt{ready}_{<i,v_S>}(x)$ and $\texttt{ready}_{<i,v_S>}(y)$ respectively. Then, v has received $\texttt{echo}_{<i,v_S>}(x)$ from $n-f$ nodes, while u has received $\texttt{echo}_{<i,v_S>}(y)$ from $n-f$ nodes. Then, there are $(n-f)+(n-f)-n > f$ nodes, hence at least one correct node, that sent multiple `echo` messages for different values, which contradicts the algorithm. □

Lemma 18.15. *If correct nodes v and u accept $\texttt{msg}_{<i,v_S>}(x)$ and $\texttt{msg}_{<i,v_S>}(y)$ respectively, then $x = y$.*

Proof. Since v has accepted $\texttt{msg}_{<i,v_S>}(x)$, at least one correct node has sent $\texttt{ready}_{<i,v_S>}(x)$. Then, no correct node has sent $\texttt{ready}_{<i,v_S>}(y)$ for $y \neq x$, according to Lemma 18.14. Therefore, if u accepts $\texttt{msg}_{<i,v_S>}(y)$, $x = y$. □

Lemma 18.16. *If a correct node v accepts $\texttt{msg}_{<i,v_S>}(x)$, then every correct node accepts $\texttt{msg}_{<i,v_S>}(x)$ eventually.*

Proof. By Lemma 18.15, no correct node accepts $\texttt{msg}_{<i,v_S>}(y)$ with $y \neq x$.

Node v has received $2f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$, hence at least $f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$ coming from correct nodes. All correct nodes eventually receive these $f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$. As Lemma 18.14 guarantees that no correct node sends $\texttt{ready}_{<i,v_S>}(y)$ for $y \neq x$, it follows that every correct node sends $\texttt{ready}_{<i,v_S>}(x)$ eventually. These messages are delivered eventually, and therefore all correct nodes accept the same $\texttt{msg}_{<i,v_S>}(x)$. □

18.4 Efficient Reliable Broadcast

How "expensive" is Algorithm 18.11 in terms of the size $|m|$ of the message m that is broadcast?

Definition 18.17 (Communication complexity)**.** *The communication complexity of an algorithm is the number of bits that all correct nodes together send in the worst case.*

Theorem 18.18. *Algorithm 18.11 has a communication complexity of* $O(n^2|m|)$.

Proof. Every correct node other than the source node broadcasts the message m of size $|m|$ exactly once in an *echo* and a *ready* message. The source node further broadcasts m in the first step. The total communication complexity is therefore $n|m| + 2n^2|m| \in O(n^2|m|)$. □

Remarks:

- Each node needs to receive the full message m. Moreover, it can be shown that reliably broadcasting a single bit incurs a communication complexity of $\Omega(n^2)$, so the best possible bound is $n|m| + \Omega(n^2)$.
- In real-world systems, it often holds that $|m| \gg n$. Can we construct a more efficient reliable broadcast algorithm for this case? Yes, using a *coding-based* mechanism!

Definition 18.19 (Erasure code)**.** *A* (n, k)***-erasure code*** *is a code that transforms a message of* k ***symbols*** *(of some given alphabet) into a message with* $n > k$ *symbols such that the original message can be recovered from a subset of the* n *symbols.*

Remarks:

- We consider *optimal* erasure codes, which have the property that any subset of k symbols is sufficient to reconstruct the original message.
- When using an optimal erasure code, it is possible to split a message of size $L \geq k$ into n *fragments* of approximate size L/k each (consisting of one or more symbols). A fragment may be slightly larger, e.g., padding the original message to a size that is evenly divisible by k.
- The basic idea is to set $k := f+1$ and encode n fragments of size $L/(f+1)$, n being the number of nodes in the subnet. The sender can send each node a different fragment, which the receiving nodes broadcast. Each node can then reconstruct the message m when receiving $f + 1$ different and *valid* fragments.
- Note that at least $f + 1$ fragments must be required, otherwise the f malicious nodes could fabricate any message themselves.
- Each fragment must be a *valid* encoding. How can we ensure that malicious nodes do not send random data instead of valid fragments?

Definition 18.20 (Merkle tree)**.** *A* ***Merkle tree*** *is a tree in which every leaf is labeled with the hash of a data block, and every inner node is labeled with the hash of the labels of its children.*

Remarks:

- The n fragments are placed at the leaves. For example, if there are 4 fragments $f_1, \ldots, f_4$, we would get the Merkle tree depicted in Figure 18.21, where $h_1 := \mathcal{H}(\mathcal{H}(f_1)|\mathcal{H}(f_2))$, $h_2 := \mathcal{H}(\mathcal{H}(f_3)|\mathcal{H}(f_4))$, and $h_0 := \mathcal{H}(h_1|h_2)$. The operator $|$ denotes the concatenation of the two given hash values.
- A proof consists of all the hashes required to recompute h_0 given a fragment. For example, given f_2 in Figure 18.21, the hashes $h(f_1)$ and h_2 are needed. Note that a hash is required per level of the tree, i.e., the number of hashes is logarithmic in the number of fragments.

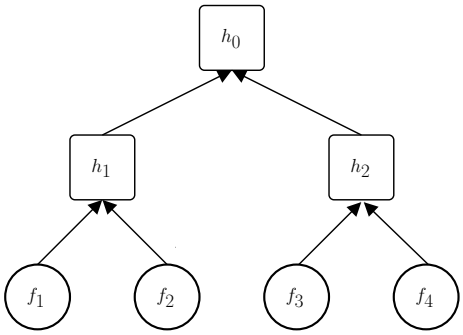

Figure 18.21: A Merkle tree for 4 fragments $f_1, \ldots, f_4$ is shown. It holds that $h_i := \mathcal{H}(\mathcal{H}(f_{2i-1})|\mathcal{H}(f_{2i}))$ for $i = 1, 2$ and $h_0 := \mathcal{H}(h_1|h_2)$.

Remarks:

- Algorithm 18.22 uses Algorithm 18.11 to first agree on the Merkle root hash h_0 of the message. The validity of received fragment is verified against the reliably broadcast root hash h_0.
- If the recovered message is not consistent with the broadcast root hash, it is safe not to deliver anything because all correct nodes that recover the message must reach the same conclusion.

Theorem 18.23. *Algorithm 18.22 implements reliable broadcast tolerating $f < n/3$ byzantine nodes in the asynchronous communication model.*

Proof. Let v be a correct node that delivers m. Since $n - f > n - 2f = f + 1$, $m \neq \perp$, and $root\_hash \neq \perp$, v must have successfully reconstructed the message and all fragments beforehand and broadcast its fragment. Since $|F| = n - f$, v must have received fragments from at least $n - 2f$ correct nodes. These $n - 2f$ correct nodes have broadcast their fragments, which implies that all correct nodes eventually receive at least $n - 2f = f + 1$ fragments. According to the algorithm, all correct nodes then reconstruct the message m successfully (because v reconstructed it successfully) and broadcast their fragments as well if they have not already done so earlier. Hence it follows that all correct nodes eventually broadcast their fragments and, since there are at least $n - f$ correct

Algorithm 18.22 Executed at node v_i. The algorithm uses a $(n, f+1)$-erasure code. Initially, $root\_hash = \perp$, $F = \{\}$, and $m = \perp$.

```
1:  (f_1, ..., f_n) := get_fragments(m)
2:  h_0 := get_merkle_root_hash((f_1, ..., f_n))
3:  Execute Algorithm 18.11 for message root_hash := h_0
4:  for v_j ∈ V do
5:     P_j := get_merkle_proof((f_1, ..., f_n), j)
6:     Send (f_j, P_j) to v_j
7:  end for
8:
9:  upon receiving (f_j, P_j) and root_hash ≠ ⊥:
10:    if valid(f_j, P_j, root_hash) then
11:       F := F ∪ {f_j}
12:       if i = j and not broadcast (f_i, P_i) before then
13:          Broadcast (f_i, P_i)
14:       end if
15:    end if
16: end upon
17:
18: if |F| = f + 1 and m = ⊥ then
19:    m := recover_message(F)
20:    (f_1, ..., f_n) := get_fragments(m)
21:    h_0 := get_merkle_root_hash((f_1, ..., f_n))
22:    if h_0 = root_hash then
23:       if not broadcast (f_i, P_i) before then
24:          P_i := get_merkle_proof((f_1, ..., f_n), i)
25:          Broadcast (f_i, P_i)
26:       end if
27:    else
28:       root_hash := ⊥
29:    end if
30: end if
31:
32: if |F| = n − f and m ≠ ⊥ and root_hash ≠ ⊥ and not delivered then
33:    deliver(m)
34: end if
```

nodes, $|F| = n - f$, $root\_hash \neq \perp$, and $m \neq \perp$ holds eventually, causing all correct nodes to deliver the message m. □

Remarks:

- The following theorem shows that the communication complexity is indeed much better for $|m| \gg n$! Let H denote the size of a hash in the Merkle tree.

Theorem 18.24. *Algorithm 18.22 has a communication complexity of* $3|m|n + O(n^2 \log(n)H)$ *in the asynchronous communication model.*

Proof. The initial reliable broadcast has a communication complexity of $O(n^2H)$. For a message of size $|m|$, the fragment size is $v/(f+1) \leq 3|m|/n$. A Merkle proof consists of approximately $\log(n)$ hashes of size H each. The n messages from the sender thus require $n \cdot (3|m|/n + \log(n)H) \in 3|m| + O(n \log(n)H)$ bits to be sent. Subsequently, every node broadcasts its fragment, together with the Merkle proof, at most once, which requires at most $n(n-1)\cdot(3|m|/n+\log(n)H) \in 3|m|(n-1)+O(n^2 \log(n)H)$ bits (not counting the fragments that the nodes send to themselves. Thus, the total communication complexity is upper bounded by $3|m|n + O(n^2 \log(n)H)$. □

Remarks:

- The communication complexity is asymptotically optimal for $|m| \in \Omega(n \log(n)H)$.
- If the sender is correct and messages arrive within constant time, the algorithm terminates in constant time.
- However, the overhead factor for large messages is 3. Can we do better?

Theorem 18.26. *Algorithm 18.25 implements reliable broadcast tolerating* $f < n/3$ *byzantine nodes in the asynchronous communication model.*

Proof. Let v be a correct node that delivers m. Exactly as for Algorithm 18.22, v must have successfully reconstructed the message and there can only be one such message because the root hash is reliably broadcast first.

Node v adds all nodes from which it received a fragment to R. For any node $w \in R$ it holds that w either sent its own fragment or v's fragment. However, the latter case also implies that w must have its own fragment. Since v sends fragments to every node *not* in R, every correct node will eventually get its fragment, which it will then broadcast. Thus, every correct node will eventually get at least $n - f$ fragments, reconstruct m, and deliver it. □

Theorem 18.27. *Algorithm 18.25 has a communication complexity of* $2|m|n + O(n^2 \log(n)H)$ *in the asynchronous communication model.*

Proof. As before, the initial reliable broadcast has a communication complexity of $O(n^2H)$, whereas the transmission of all Merkle proofs requires $O(n^2 \log(n)H)$ bits to be sent. For a message of size $|m|$, the fragment size is $|m|/(2f+1) \leq \frac{3}{2}|m|/n$. The initial sender first sends a fragment, including the corresponding

Algorithm 18.25 Executed at node v_i. The algorithm uses a $(n, 2f+1)$-erasure code. Initially, $root\_hash = \bot$, $F = R = \{\}$, and $m = \bot$.

```
1:  (f_1, ..., f_n) := get_fragments(m)
2:  h_0 := get_merkle_root_hash((f_1, ..., f_n))
3:  Execute Algorithm 18.11 for message root_hash := h_0
4:  for v_j ∈ V do
5:    P_j := get_merkle_proof((f_1, ..., f_n), j)
6:    Send (f_j, P_j) to v_j
7:  end for
8:
9:  upon receiving (f_j, P_j) and root_hash ≠ ⊥:
10:   if valid(f_j, P_j, root_hash) then
11:     F := F ∪ {f_j}; R := R ∪ {v_j}
12:     if i = j and not broadcast (f_i, P_i) before then
13:       Broadcast (f_i, P_i)
14:     end if
15:   end if
16: end upon
17:
18: if |F| = 2f + 1 and m = ⊥ then
19:   m := recover_message(F)
20:   (f_1, ..., f_n) := get_fragments(m)
21:   h_0 := get_merkle_root_hash((f_1, ..., f_n))
22:   if h_0 = root_hash then
23:     for v_j ∈ V \ R do
24:       P_j := get_merkle_proof((f_1, ..., f_n), j)
25:       Send (f_j, P_j) to v_j
26:     end for
27:   else
28:     root_hash := ⊥
29:   end if
30: end if
31:
32: if |F| = n − f and m ≠ ⊥ and root_hash ≠ ⊥ and not delivered then
33:   deliver(m)
34: end if
```

Merkle proof, to all other nodes, which requires $(n-1)|m| + O(Hn\log(n))$ bits. Every node broadcasts its fragment plus Merkle proof to all other nodes. Moreover, every node broadcasts w's fragment plus Merkle proof for every $w \in V \backslash R$. Since $|R| = 2f+1$, it holds that $|V \backslash R| < n/3$ and thus the communication complexity is upper bounded by

$$\begin{aligned}
&(n-1)|m| + n \cdot (n-1)|m| + n \cdot \frac{n}{3}|m| + O(n^2 \log(n)H) \\
&< \frac{4}{3}n^2|m| + O(n^2 \log(n)H) \\
&= \frac{4}{3}n^2 \frac{3}{2}\frac{|m|}{n} + O(n^2 \log(n)H) \\
&= 2|m|n + O(n^2 \log(n)H)
\end{aligned}$$

□

Remarks:

- Can we do better? Yes! There is a more complex algorithm that has a communication complexity of $\frac{3}{2}|m|n + O(n^2 \log(n)H)$. This algorithm encodes fragments into *mini-fragments* (i.e., it uses two layers of erasure coding!) to reduce the overhead further.
- It can be shown that any reliable broadcast algorithm that terminates in an optimal two rounds if the initial sender is correct and that is somewhat balanced (in that the initial sender does not send a disproportionate amount of data) has a communication complexity of at least $\frac{3}{2}|m|n$. Proving tight bounds that hold more generally is an open problem!

Definition 18.28 (Atomic Broadcast)**.** ***Atomic broadcast*** *makes sure that all messages are accepted 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 accepts m_1 before m_2 if and only if node v accepts m_1 before m_2.*

Remarks:

- Definition 18.28 is equivalent to Definition 15.8, i.e., atomic broadcast can be used to implement state replication. It is therefore harder to implement atomic broadcast than reliable broadcast.

Chapter Notes

Broadcast has been studied since the mid 1980s [BT85]. Bracha published the first reliable broadcast protocol, Algorithm 18.11, in 1987 [Bra87]. In the same year, it was shown how to replace signed communication with a broadcast primitive to obtain an equivalent non-authenticated algorithm [ST87]. In 2001, modular definitions for several broadcast problems, including reliable, atomic, and secure causal broadcast, were presented including protocols that implement these broadcast variants [CKPS01].

After the publication of Bracha's algorithm, it took 18 years until Cachin and Tessaro presented a reliable broadcast algorithm based on erasure coding [CT05], which brought the overhead down from a factor of n to 3. It took almost exactly as long until this bound was improved, first to an overhead factor of 2 [Loc24], and then to 1.5 [LS24]. While it has been shown that any "weakly balanced" reliable broadcast algorithm (where the sender sender sends $o(n|m|)$ bits) and the algorithm terminates in 2 or 3 rounds if the sender is honest, has a communication complexity of at least $1.5n|m|$ [Loc24], it is an open problem whether a better bound is possible in general.

This chapter was written in collaboration with Thomas Locher.

Bibliography

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

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[CT05] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005.

[Loc24] Thomas Locher. Byzantine reliable broadcast with low communication and time complexity. *arXiv preprint arXiv:2404.08070*, 2024.

[LS24] Thomas Locher and Victor Shoup. Minicast: Minimizing the communication complexity of reliable broadcast. *Cryptology ePrint Archive*, 2024.

[ST87] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, Jun 1987.