

## 6 Broadcast and Agreement with Byzantine Faults

### 6.1 Introduction

**Problem.** There are  $n$  servers, of which up to  $t$  may be *corrupted* by an *adversary* and exhibit arbitrary faults; the remaining servers are *honest*. The servers connected by pairwise reliable and authenticated links, and the system is asynchronous (no bounds on message delays, no local clocks). Every server starts out with an initial value and the goal is to agree on a common value.

**Methods.** Cryptography, in particular, threshold cryptography (signatures and pseudorandom generators), is used to cope with potentially malicious failures. Keys and key shares (for threshold cryptography) are initially distributed by a trusted dealer. Randomized protocols are used to reach agreement, since deterministic asynchronous consensus and agreement protocols have infinite runs. Such randomized protocols achieve agreement in finite time with all but negligible probability.

### 6.2 Broadcast Primitives

Broadcasts are parameterized by a tag  $ID$ , which is contained (implicitly) in every message. In *consistent* and *reliable* broadcasts, a distinguished sender  $P_s$  *broadcasts* a message  $m$  and all servers (should) *deliver*  $m$ .

Consistent broadcast (“c-broadcast”) ensures only that the delivered message is consistent for all receivers. In particular, termination is not guaranteed with a faulty sender.

**Definition 6.1 (Consistent Broadcast).** A protocol for consistent broadcast satisfies:

*Validity:* If an honest sender  $P_s$  *c-broadcasts*  $m$ , then  $P_s$  eventually *c-delivers*  $m$ .

*Consistency:* If some honest server *c-delivers*  $m$  and a distinct honest server *c-delivers*  $m'$ , then  $m = m'$ .

*Integrity:* Every honest server *c-delivers* at most one  $m$ .

*Termination:* If the sender is honest, then all honest servers eventually *c-deliver* a message.

**Algorithm 6.2 (Echo Broadcast using Digital Signatures).** Assume every server can digitally sign messages, which can be verified by any server.

```
upon c-broadcast( $m$ ):           // sender  $P_s$  only
    send (send,  $m$ ) to all
```

**upon** receiving (send,  $m$ ) from  $P_s$  :  
 compute signature  $\sigma$  on (echo,  $s, m$ )  
 send (echo,  $m, \sigma$ ) to  $P_s$

**upon** receiving  $\lceil \frac{n+t+1}{2} \rceil$  messages (echo,  $m, \sigma_i$ ) with valid  $\sigma_i$  : // sender  $P_s$  only  
 let  $\Sigma$  be the list of all received signatures  $\sigma_i$   
 send (final,  $m, \Sigma$ ) to all

**upon** receiving (final,  $m, \Sigma$ ) from  $P_s$  with  $\lceil \frac{n+t+1}{2} \rceil$  valid signatures in  $\Sigma$ :  
*c-deliver*( $m$ )

The message complexity of Echo Broadcast is  $O(n)$  and its communication complexity is  $O(n^2(k + |m|))$ , where  $k$  denotes the length of a digital signature. Using a non-interactive threshold signature scheme, the communication complexity can be reduced to  $O(n(k + |m|))$ .

**Theorem 6.3.** *Assuming perfectly unforgeable signatures, Algorithm 6.2 implements consistent broadcast with Byzantine faults for  $n > 3t$ .*

*Proof.* The message  $m$  in any final message with enough valid signatures in  $\Sigma$  is unique.  $\square$

Reliable broadcast (“r-broadcast”) ensures additionally agreement on the delivery of a message.

**Definition 6.4 (Reliable Broadcast or the “Byzantine Generals Problem”).** A protocol for reliable broadcast is a consistent broadcast protocol that satisfies also:

*Totality:* If some honest server *r-delivers* a message, then all honest servers eventually *r-deliver* a message.

Totality ensures that all honest servers either deliver a message or don’t. In the literature *consistency* and *totality* are often combined into a single condition called *agreement*.

**Algorithm 6.5 (Bracha Broadcast).**

**upon** *r-broadcast*( $m$ ): // sender  $P_s$  only  
 send (send,  $m$ ) to all

**upon** receiving (send,  $m$ ) from  $P_s$ :  
 send (echo,  $m$ ) to all

**upon** receiving  $\lceil \frac{n+t+1}{2} \rceil$  messages (echo,  $m$ ) and not having sent (ready,  $m$ ):  
 send (ready,  $m$ ) to all

**upon** receiving  $t + 1$  messages (ready,  $m$ ) and not having sent (ready,  $m$ ):  
 send (ready,  $m$ ) to all

**upon** receiving  $2t + 1$  messages (ready,  $m$ ):  
*r-deliver*( $m$ )

**Theorem 6.6 ([Bra84]).** *Algorithm 6.5 implements reliable broadcast with Byzantine faults for  $n > 3t$ .*

*Proof.* Consistency follows from the same argument as in Theorem 6.3, since the message  $m$  in any ready message of an honest server is unique. Totality is implied by the “amplification” of ready messages from  $t + 1$  to  $2t + 1$ .  $\square$

### 6.3 Randomized [Binary] Byzantine Agreement

Binary Byzantine agreement is characterized by two events *propose* and *decide*; every server executes *propose*( $b$ ) to start the protocol and *decide*( $b$ ) to terminate it, for a bit  $b$ .

**Definition 6.7 (Binary Byzantine Agreement).** A protocol for binary Byzantine Agreement satisfies:

*Validity:* If all honest servers *propose*  $v$ , then some honest server eventually *decides*  $v$ .

*Agreement:* If some honest server *decides*  $v$  and a distinct honest server *decides*  $v'$ , then  $v = v'$ .

*Termination:* Every honest server eventually *decides*.

It is not possible to implement Definition 6.7 in asynchronous systems [FLP85]. But one can relax either *termination* or *agreement* to hold with high probability, and there are protocols that satisfy them with probability 1 after infinite running time. More precisely, given a logical time measure  $T$ , such as the number of steps performed by all honest servers, *termination with probability 1* means that

$$\lim_{T \rightarrow \infty} \Pr[\text{some honest server has not decided after time } T] = 0.$$

**Algorithm 6.8 ([Tou84]).** Suppose a trusted dealer has *shared*, using secret sharing as in Section 5.3, a sequence  $s_0, s_1, \dots$  of random bits, or “coins”, among the servers, which can be accessed using a *recover* operation (this will involve exchanging some messages) [Rab83]. The two *upon* clauses of the algorithm below are executed in parallel threads.

The value  $v$  is called the “vote”; the value  $\Pi$  is a “proof” that justifies the choice of  $v$  in the 2-vote message; a “round” of the algorithm consists of two rounds of message exchanges.

**upon** *propose*( $v$ ):

$r \leftarrow 0$

**while** not *decided* **do**

send the signed message (1-vote,  $r, v$ ) to all

receive properly signed (1-vote,  $r, v'$ ) messages from  $n - t$  distinct servers

$\Pi \leftarrow$  set of received 1-vote messages

$v \leftarrow$  value  $v'$  that is contained most often in  $\Pi$

*r-broadcast* the message (2-vote,  $r, v, \Pi$ )

wait for *r-delivery* of (2-vote,  $r, v', \Pi$ ) messages with valid proofs  $\Pi$  from  $n - t$  senders

$m_2 \leftarrow$  value  $v'$  that is contained most often among the *r-delivered* 2-vote messages

```

 $c_2 \leftarrow$  number of r-delivered 2-vote messages with  $v' = m_2$ 
 $recover(s_r)$ 
if  $c_2 = n - t$  then
     $v \leftarrow m_2$ 
else
     $v \leftarrow s_r$ 
if  $c_2 \geq t + 1$  and  $m_2 = s_r$  then
    send the message (decide,  $v$ ) to all
     $decide(v)$ 
 $r \leftarrow r + 1$ 

```

**upon receiving  $t + 1$  messages (decide,  $b$ ):**  
 send the message (decide,  $b$ ) to all  
 $decide(b)$

**Lemma 6.9.** *If all honest servers start some round  $r$  with vote  $v_0$ , then all honest servers will also terminate round  $r$  with vote  $v_0$ .*

*Proof.* It is impossible to create a valid  $\Pi$  for a 2-vote message with a vote  $v \neq v_0$ . □

**Lemma 6.10.** *If two distinct honest servers start some round  $r$  with different votes, then with probability at least  $1/2$ , all honest servers will terminate round  $r$  with the same vote.*

*Proof.* Consider the assignment of  $m_2$  and  $c_2$  in some round  $r$ . If some honest server obtains  $c_2 = n - t$  and  $m_2 = v_0$ , then no honest server obtains  $c_2 = n - t$  but  $m_2 \neq v_0$ . All honest servers with  $c_2 = n - t$  set  $v$  to  $v_0$ ; every other honest server sets  $v$  to  $s_r$ . Since the first honest server to assign  $m_2$  and  $c_2$  does so *before* anything about  $s_r$  is known (to the adversary),  $s_r$  and  $v_0$  are independent and  $s_r = v_0$  with probability  $1/2$ . □

**Theorem 6.11.** *Assuming perfectly unforgeable signatures, Algorithm 6.8 implements binary Byzantine agreement for  $n > 3t$ , where termination holds with probability 1.*

Since Algorithm 6.8 reaches agreement with probability at least  $1/2$  in every round, the expected number of rounds is 2, and the expected number of messages sent is  $O(n^3)$ .

## 6.4 Byzantine Agreement using Cryptographic Randomness

Algorithm 6.8 assumes a sequence of unpredictable, shared random coins  $s_0, s_1, \dots$ . When implemented using secret sharing, every instance of agreement needs a fresh sequence of shared bits, which is infeasible in practice. More practical implementations can be obtained using threshold cryptography as follows.

**Algorithm 6.12 ([CKS00]).** In Algorithm 6.8, implement the sequence  $s_0, s_1, \dots$  of coins by the Diffie-Hellman-based threshold pseudorandom function from Exercise 5.1, using a polynomial of degree  $n - t - 1$  to share the secret key.

Recall that the pseudorandom function  $F_x(s)$  uses a secret seed  $x$ , takes an arbitrary input string  $s$ , and maps the input to a pseudorandom value in  $\{0, 1\}^k$ . We set  $k = 1$  and let the input

string for coin  $s_j$  consist of  $ID||j$ , i.e., the tag  $ID$  of the protocol instance concatenated with the coin counter  $j$ .

The threshold pseudorandom function is initialized by the trusted dealer who chooses the seed  $x$  and shares it among the servers. Note that it can be accessed and used by a practically unlimited number of concurrent protocol instances.

Since the threshold pseudorandom function is non-interactive, the *recover* operation for  $c_r$  can be implemented simply by having every server send its share of  $c_r$  to all others, collecting  $n - t$  shares, and combining them to the coin value  $F_x(ID||r)$ .

The randomized Byzantine agreement protocol of Algorithm 6.12 is based on threshold cryptography and has expected message complexity  $O(n^3)$ . It can be reduced to  $O(n^2)$  expected messages as shown in [CKS00] by replacing the reliable broadcasts with a two-phase voting structure that uses *justifications* for each vote obtained through threshold signatures on earlier votes.

## 6.5 Atomic Broadcast with Byzantine Faults

Given a Byzantine agreement protocol, an asynchronous atomic broadcast protocol that tolerates Byzantine faults can be realized using the same approach as with crash failures (Algorithm 4.21): proceed in global asynchronous rounds and for each round, agree on a batch of messages to be delivered at the end of the round. Such an algorithm is described in [CKPS01].

## References

- [Bra84] G. Bracha, *An asynchronous  $[(n - 1)/3]$ -resilient consensus protocol*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 154–162.
- [CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, *Secure and efficient asynchronous broadcast protocols (extended abstract)*, Advances in Cryptology: CRYPTO 2001 (J. Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, 2001, pp. 524–541.
- [CKS00] C. Cachin, K. Kursawe, and V. Shoup, *Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography*, Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC), 2000, Revised version to appear in *Journal of Cryptology*, 2005.
- [FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, Journal of the ACM **32** (1985), no. 2, 374–382.
- [Rab83] M. O. Rabin, *Randomized Byzantine generals*, Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS), 1983, pp. 403–409.
- [Tou84] S. Toueg, *Randomized Byzantine agreements*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 163–178.