

6 Byzantine Agreement and Broadcasts

6.1 Beyond Crash Failures

Malicious activity can be interpreted as a failure that is more severe than crashing; this idea leads to a refined classification of faults.

A. random, non-adversarial faults

- 1) fail-stop [servers may crash, but crashes can be detected remotely; as with a perfect failure detector]
- 2) crash(-silent) [servers may crash, not detectable remotely]
- 3) crash-recovery [servers may crash and recover afterwards; using stable storage]
- 4) omission [servers may omit receiving, processing, or sending of messages]

B. adversarial and malicious faults

- 1) passive (or honest-but-curious or semi-honest model) [no deviation from specification, adversary with read-only privileges]
- 2) passive with crashes [a crash is the only deviation from specification]
- 3) active (or fully Byzantine) [arbitrary behavior]

This classification is intended for asynchronous distributed systems, but most concepts apply also in other contexts. So far, we have mostly used **A2** crash failures; in this section, we address **B1** and **B2**, with a passive adversary. Note that this classification is not one-dimensional, although there are some strict inclusion relations, like **A1-A2-B2-B3** and **A2-A4-B3**.

Additional, orthogonal assumptions for **B** adversarial and malicious faults:

a. setup

- i) trusted dealer [for generation and initial distribution of keys]
- ii) active (or fully Byzantine) [arbitrary behavior]

b. adversary power

- i) computationally bounded (running time bounded by a polynomial)
- ii) unbounded

In the context of adversarial faults, the correct, i.e., non-faulty, servers are sometimes also called *honest* and faulty servers are typically called *corrupted*, to emphasize that a malicious adversary may be coordinating the faulty parties.

6.2 Model

Problem. There are n servers, of which up to t may be *corrupted* by an *adversary* and exhibit arbitrary faults; the remaining servers are *correct*. The servers connected by pairwise reliable and authenticated links, and the system is asynchronous (no bounds on message delays, no local clocks).

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.

6.3 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 correct sender P_s *c-broadcasts* m , then P_s eventually *c-delivers* m .

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

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

Termination: If the sender is correct, then all correct 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 Σ is unique. □

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 correct server r -delivers a message, then all correct servers eventually r -deliver a message.

Totality ensures that all correct 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 correct server is unique. Totality is implied by the “amplification” of ready messages from $t + 1$ to $2t + 1$. □

6.4 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 correct servers *propose* *v*, then some correct server eventually *decides* *v*.

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

Termination: Every correct 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 correct servers, *termination with probability 1* means that

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

Algorithm 6.8 ([Tou84]). Suppose a trusted dealer has *shared*, using secret sharing as (see Chapter 7), 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 Π 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*):

$r \leftarrow 0$

loop

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 including the signatures

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

r-broadcast the message (2-vote, *r*, *v*, Π)

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

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

$c \leftarrow$ number of *r-delivered* 2-vote messages with $v' = v''$

recover(s_r)

if $c = n - t$ **then**

$v \leftarrow v''$

else

$v \leftarrow s_r$

if $v'' = s_r$ **then**

send the message (decide, *v*) to all

// note that $v = s_r = v''$

$r \leftarrow r + 1$

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

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

Proof. It is impossible to create a valid Π for a 2-vote message with a vote $v \neq v_0$ because v must be set to the majority value in $n - t$ received 1-vote messages and $n - t > 2t$. \square

Lemma 6.10. *In every round r , the following holds:*

- a) *If a correct server sends a `decide` message for v_0 at the end of round r , then all correct servers will terminate round r with vote v_0 .*
- b) *With probability at least $1/2$, all correct servers will terminate round r with the same vote.*

Proof. Consider the assignment of v'' and c in round r . If some correct server obtains $c = n - t$ and $v'' = v_0$, then no correct server obtains $c = n - t$ but $v'' \neq v_0$. All correct servers with $c = n - t$ set v to v_0 ; every other correct server sets v to s_r . Hence, if $s_r = v_0$, all correct servers terminate round r with vote v_0 .

Claim a) now follows upon noticing that a correct server only sends a `decide` message for v when $v'' = s_r$.

Claim b) follows because the first correct server to assign v'' and c does so *before* anything about s_r is known (to the adversary), s_r and v_0 are independent; hence, $s_r = v_0$ with probability $1/2$. \square

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 $O(n^3)$.

6.5 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 5.20): 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.
- [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.