Christian Cachin (`www.zurich.ibm.com/~cca`)

9 Asynchronous Byzantine Agreement

9.1 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 over pairwise reliable 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 (signatures and pseudorandom generators) is used to cope with potentially malicious failures. This usually includes a trusted dealer that sets up the cryptographic keys ahead of time. Since deterministic asynchronous consensus and agreement protocols have infinite runs, we use randomized protocols that achieve agreement with all but negligible probability.

9.2 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 (perhaps) *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 9.1 (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 9.2 (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 $(\texttt{send}, m)$ to all

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

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

upon *receiving* $(\texttt{final}, m, \Sigma)$ *from* P_s *with valid signatures in* Σ*:*
c-deliver(m)

Theorem 9.3. *Assuming perfectly unforgeable signatures, Algorithm 9.2 implements consistent broadcast with Byzantine faults for* $n > 3t$.

Proof. The message m in any `final` message with valid signatures in Σ is unique. □

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

Definition 9.4 (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 9.5 (Bracha Broadcast).

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

upon *receiving* $(\texttt{send}, m)$ *from* P_s:
send $(\texttt{echo}, m)$ to all

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

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

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

Theorem 9.6 ([Bra84]). *Algorithm 9.5 implements reliable broadcast with Byzantine faults for* $n > 3t$.

Proof. Consistency follows from the same argument as in Theorem 9.3, 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 $2t + 1$. □

9.3 Secret Sharing

Secret sharing is used in randomized Byzantine agreement and forms the basis of *threshold cryptography*. A secret is *shared* among n parties such that the cooperation of at least $t+1$ parties is needed to recover s.

Algorithm 9.7. To *share* $s \in \mathbb{F}_q$, a *dealer* $P_d \notin \{P_1, \ldots, P_n\}$ chooses uniformly at random a polynomial $f(x) \in F_q[x]$ of degree t subject to $f(0) = s$, generates *shares* $s_i = f(i)$, and sends s_i to P_i. To recover s, a group $\mathcal{S}$ of $t+1$ servers computes $s = f(0) = \sum_{i \in \mathcal{S}} \lambda_{0,i}^{\mathcal{S}} s_i$ for the appropriate Lagrange coefficients $\lambda_{0,i}^{\mathcal{S}} = \ldots$. The scheme has perfect security, i.e., the shares held by every group of t or fewer servers are statistically independent of s.

9.4 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 9.8 (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 9.8 in asynchronous systems. 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

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

Algorithm 9.9 ([Tou84]). Suppose a trusted dealer has *shared* a sequence $s_0, s_1, \ldots$ of random bits, or "coins", among the servers, which can be accessed using a *recover* operation (this will involven exchaning some messages). 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):
  $r \leftarrow 0$
  while not *decided* **do**
    send the signed message $(\texttt{1-vote}, r, v)$ to all
    receive properly signed $(\texttt{1-vote}, r, v')$ messages from $n - t$ distinct servers

$\Pi \leftarrow$ set of received `1-vote` messages
$v \leftarrow$ value v' that is contained most often in Π
r-broadcast the message $(\texttt{2-vote}, r, v, \Pi)$
wait for *r-delivery* of $(\texttt{2-vote}, r, v', \Pi)$ messages with valid proofs Π 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 $(\texttt{decide}, v)$ to all
  decide(v)
$r \leftarrow r + 1$

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

Lemma 9.10. *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 Π for a `2-vote` message with a vote $v \neq v_0$. □

Lemma 9.11. *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 9.12. *Assuming perfectly unforgeable signatures, Algorithm 9.9 implements binary Byzantine agreement for $n > 3t$, where termination holds with probability 1.*

Since Algorithm 9.9 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 $O(n^3)$.

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.

[Tou84] S. Toueg, *Randomized Byzantine agreements*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 163–178.