Circumvent the impossibility of FLP'85: Algorithms

Alain Cournier   Stéphane Devismes

Université de Picardie Jules Verne

April 29, 2026

Roadmap

Roadmap

Expressiveness *vs.* Type of Faults

Message Loss: **Every** distributed algorithm for fault-free environment can be made tolerant to message losses using the alternating bit protocol, provided that communication links are fair lossy.

Expressiveness *vs.* Type of Faults

Message Loss: **Every** distributed algorithm for fault-free environment can be made tolerant to message losses using the alternating bit protocol, provided that communication links are fair lossy.

Process Crash: The deterministic (binary) **consensus** is impossible in an asynchronous system where **at most one process may crash**. Fischer, Lynch et Paterson (1985) [3]

Even if

- the communication network is complete, and
- links are reliable.

Expressiveness *vs.* Type of Faults

Message Loss: **Every** distributed algorithm for fault-free environment can be made tolerant to message losses using the alternating bit protocol, provided that communication links are fair lossy.

Process Crash: The deterministic (binary) **consensus** is impossible in an asynchronous system where **at most one process may crash**. Fischer, Lynch et Paterson (1985) [3]

Even if

- the communication network is complete, and
- links are reliable.

Now, the (binary) **consensus** is the simplest agreement problem . . .

Tightness of FLP'85

The deterministic (binary) consensus is impossible in an asynchronous system where at most one process may crash.

However,

1. Consensus is solvable in partially synchronous crash-prone systems: FloodSet Algorithm in (fully) synchronous systems [4].

[1] Two lessons will be dedicated to the theory of failure detectors.

Tightness of FLP'85

The deterministic (binary) consensus is impossible in an asynchronous system where at most one process may crash.

However,

1. Consensus is solvable in partially synchronous crash-prone systems: FloodSet Algorithm in (fully) synchronous systems [4].
2. Probabilistic consensus is solvable in asynchronous crash-prone systems: Ben-Or Algorithm [1].

[1] Two lessons will be dedicated to the theory of failure detectors.

Tightness of FLP'85

The deterministic (binary) consensus is impossible in an asynchronous system where at most one process may crash.

However,

1. Consensus is solvable in partially synchronous crash-prone systems: FloodSet Algorithm in (fully) synchronous systems [4].
2. Probabilistic consensus is solvable in asynchronous crash-prone systems: Ben-Or Algorithm [1].
3. Consensus is solvable in asynchronous systems prone to restrictive crash patterns: FLP Algorithm (Initially Dead Crashes) [3].

[1] Two lessons will be dedicated to the theory of failure detectors.

Tightness of FLP'85

The deterministic (binary) consensus is impossible in an asynchronous system where at most one process may crash.

However,

1. Consensus is solvable in partially synchronous crash-prone systems: FloodSet Algorithm in (fully) synchronous systems [4].
2. Probabilistic consensus is solvable in asynchronous crash-prone systems: Ben-Or Algorithm [1].
3. Consensus is solvable in asynchronous systems prone to restrictive crash patterns: FLP Algorithm (Initially Dead Crashes) [3].
4. Consensus may be solvable if information about crashes are available: Failure Detectors [2].[1]

[1] Two lessons will be dedicated to the theory of failure detectors.

Roadmap

Definition

Synchronous systems = all processes & all links are synchronous

Partially synchronous systems = some processes & some links have synchrony properties

Definition

Synchronous systems = all processes & all links are synchronous

Partially synchronous systems = some processes & some links have synchrony properties

The (fully) synchronous system is a partially synchronous system

Synchronous links

If a message sent in the link is not lost, then it is delivered to its destination within bound time:

A link is synchronous if $\exists c \in \mathbb{N}, \forall t \in \mathbb{N}$, if m is sent in the link at time t, then m is delivered before $t + c$ or lost.

(the bound may be known or unknown by processes)

Eventually Synchronous and Asynchronous links

Eventually Synchronous Link: A link is eventually synchronous if $\exists c, t_0 \in \mathbb{N}, \forall t \geq t_0$, if m is sent in the link at time t, then m is delivered before $t + c$ or lost.

(the bound may be known or unknown by processes)

Asynchronous Link: No timing guarantee, *i.e.*, each sent message is either delivered or lost within finite time.

Synchronous Processes

Start: All (non-initially-dead) synchronous processes starts within bounded time.

Steps: While not crashed, a synchronous process executes steps within a (positive) bounded time.

Clock: The clock drifts of synchronous processes are bounded.

(those bounds may be known or unknown by processes)

We can define eventually synchronous processes similarly to eventually synchronous links: there is an *a priori* unknown time from which we have bounded time guarantees.

Other Examples of Partially Synchronous Systems

- A system where all processes are synchronous and where there is at least one source.

  A source is a (synchronous) correct[2] process with reliable and synchronous outgoing links.

- A system where all processes are eventually synchronous and all links are eventually reliable and synchronous.

[2] A process is correct if it never crashes; otherwise, it is faulty.

Other Examples of Partially Synchronous Systems

- A system where all processes are synchronous and where there is at least one source.

  A source is a (synchronous) correct[2] process with reliable and synchronous outgoing links.

- A system where all processes are eventually synchronous and all links are eventually reliable and synchronous.

The expressive power of those two systems is difficult to compare.

[2] A process is correct if it never crashes; otherwise, it is faulty.

The Round Model

The simplest synchronous model!

- The communication network is complete.
- All (non-initially-dead) processes start simultaneously.
- A process may crash at any moment in a round.
- After an initialization phase, the execution proceeds in synchronous rounds where the following three phases are synchonously executed, in order:

  Send Phase: Each non-crashed process broadcasts a message to all other processes[3]

  Receive Phase: All messages sent during the current round are received by non-crashed processes[4]

  Compute Phase: Non-crashed processes make a local computation.

[3] The communication network is complete. However, a process may crash during the round. In this case, the message may be sent to a part of processes only.

[4] Communications are synchronous and reliable.

Constants & Variables

- Processes are identified: a process and its identifier are used equivalently (V is the set of processes)
- n: number of processes
- f: maximum number of crashes (we may have $f = n$)
- $r \in \mathbb{N}$: the round number
- v_p: a boolean (read-only) input, the value proposed by process p
- $d_p \in \{\perp, 0, 1\}$: the decision variable of process p
- $Val_p[]$: array indexed on process IDs. $\forall q \in V,\ Val_p[q] \in \{0, 1, \perp\}$
- New_p: set of pairs $(v, q) \in \{0, 1\} \times V$

The Code

$Val_p \leftarrow [\perp, \ldots, \perp]$ /* *Beginning of the initialization* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \perp$ /* *End of the initialization* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
  Round Start
    `broadcast(`New_p`)` to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \perp$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\perp$ value in Val_p
  Round End
Done

Result

Theorem 1

FloodSet solves the consensus in the (synchronous) round model if at most $f < n$ processes crash.

Consensus problem: for every process p

Input : $v_p \in \{0,1\}$

Output : $d_p \in \{\perp,0,1\}$ initialized to $\perp$

Requirements:

Integrity Every process decides, *i.e.*, assigns its d-variable to a non-$\perp$ value, at most once

Termination : Every correct process eventually decides

Validity : Every decided value is an initially proposed value, *i.e.*, $\forall p \in V$, $d_p \neq \perp \Rightarrow d_p \in \{v_q : q \in V\}$

(Uniform) Agreement : If two processes p and q decide, then they decide the same value, *i.e.*, $d_p = d_q$

Integrity

Every process decides, *i.e.*, assigns its d-variable to a non-$\perp$ value, at most once

Trivial: a process can only decide at Line 18 and stops right after Line 18

```
1:  Val_p ← [⊥,...,⊥]        /* Initialization Beginning */
2:  Val_p[p] ← v_p
3:  New_p ← {(v_p, p)}
4:  d_p ← ⊥        /* Initialization End */

5:  For all r from 1 to f+1 do        /* Rounds */
6:      Round Start
7:          broadcast(New_p) to all other processes
8:          Let R_p[q] be the set received from q during r
    (∅ if no message received from q)
9:          New_p ← ∅
10:         For all process q ≠ p do
11:             For all (v,k) ∈ R_p[q] do
12:                 If Val_p[k] = ⊥ then
13:                     Val_p[k] ← v
14:                     New_p ← New_p ∪ {(v,k)}
15:                 End If
16:             Done
17:         Done
18:         If r = f+1 then  d_p ← x where x is the first
    non-⊥ value in Val_p
19:     Round End
20: Done
```

Termination

Every correct process eventually decides

Trivial:

- After the initialization, there exists at least one non-$\perp$ value in Val_p ($Val_p[p]$): Line 18 is a decision.
- Each process executes a bounded number of rounds, so every correct process eventually executes Line 18.

1: $Val_p \leftarrow [\perp, \ldots, \perp]$ /* *Initialization Beginning* */
2: $Val_p[p] \leftarrow v_p$
3: $New_p \leftarrow \{(v_p, p)\}$
4: $d_p \leftarrow \perp$ /* *Initialization End* */

5: **For all** r from 1 to $f+1$ **do** /* *Rounds* */
6: **Round Start**
7: `broadcast(`New_p`)` to all other processes
8: Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
9: $New_p \leftarrow \emptyset$
10: **For all** process $q \neq p$ **do**
11: **For all** $(v, k) \in R_p[q]$ **do**
12: **If** $Val_p[k] = \perp$ **then**
13: $Val_p[k] \leftarrow v$
14: $New_p \leftarrow New_p \cup \{(v, k)\}$
15: **End If**
16: **Done**
17: **Done**
18: **If** $r = f + 1$ **then** $d_p \leftarrow x$ where x is the first non-$\perp$ value in Val_p
19: **Round End**
20: **Done**

Validity

Every decided value is an initially proposed value, *i.e.*, $\forall p \in V$, $d_p \neq \perp \Rightarrow d_p \in \{v_q : q \in V\}$

Proof.

Every non-$\perp$ value in Val_p is an initially proposed value[a] □

[a] By induction on r; we let the round $r = 0$ correspond to the initialization. The property holds at the end of each round.

$Val_p \leftarrow [\perp, \ldots, \perp]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \perp$ /* *Initialization End* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
  Round Start
    broadcast(New_p) to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \perp$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\perp$ value in Val_p
  Round End
Done

Agreement

If two processes p and q decide, then they decide the same value, *i.e.*, $d_p = d_q$

Proof Outline:

Right before the decision (Line 18), we have $Val_p = Val_q$ for every pair of processes (p, q) that will decide

This property is trivial for $p = q$.

So, assume now that $p \neq q$

1: $Val_p \leftarrow [\bot, \ldots, \bot]$ */* Initialization Beginning */*
2: $Val_p[p] \leftarrow v_p$
3: $New_p \leftarrow \{(v_p, p)\}$
4: $d_p \leftarrow \bot$ */* Initialization End */*

5: **For all** r from 1 to $f+1$ **do** */* Rounds */*
6: **Round Start**
7: `broadcast(`New_p`)` to all other processes
8: Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
9: $New_p \leftarrow \emptyset$
10: **For all** process $q \neq p$ **do**
11: **For all** $(v, k) \in R_p[q]$ **do**
12: **If** $Val_p[k] = \bot$ **then**
13: $Val_p[k] \leftarrow v$
14: $New_p \leftarrow New_p \cup \{(v, k)\}$
15: **End If**
16: **Done**
17: **Done**
18: **If** $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
19: **Round End**
20: **Done**

Agreement

Assume that $\exists k, Val_p[k] = v_k \neq \perp$ at the end of the last round

(due to Line 2, such a value exists)

Let r be the round where p has received (v_k, k) for the first time

(we let $r = 0$ if $p = k$)

2 cases: $r < f+1$ or $r = f+1$

1: $Val_p \leftarrow [\perp, \ldots, \perp]$ /* *Initialization Beginning* */
2: $Val_p[p] \leftarrow v_p$
3: $New_p \leftarrow \{(v_p, p)\}$
4: $d_p \leftarrow \perp$ /* *Initialization End* */

5: **For all** r from 1 to $f+1$ **do** /* *Rounds* */
6: **Round Start**
7: `broadcast`(New_p) to all other processes
8: Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
9: $New_p \leftarrow \emptyset$
10: **For all** process $q \neq p$ **do**
11: **For all** $(v, k) \in R_p[q]$ **do**
12: **If** $Val_p[k] = \perp$ **then**
13: $Val_p[k] \leftarrow v$
14: $New_p \leftarrow New_p \cup \{(v, k)\}$
15: **End If**
16: **Done**
17: **Done**
18: **If** $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\perp$ value in Val_p
19: **Round End**
20: **Done**

Agreement

Case $r < f+1$

$Val_p \leftarrow [\bot, \ldots, \bot]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \bot$ /* *Initialization End* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
  Round Start
    `broadcast`(New_p) to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \bot$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
  Round End
Done

Agreement

Case $r < f+1$

p inserts (v_k, k) in New_p during the round r

$Val_p \leftarrow [\bot, \dots, \bot]$ */* Initialization Beginning */*
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \bot$ */* Initialization End */*

For all r from 1 to $f+1$ **do** */* Rounds */*
  Round Start
    `broadcast(`New_p`)` to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v,k) \in R_p[q]$ **do**
        If $Val_p[k] = \bot$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v,k)\}$
        End If
      Done
    Done
    If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
  Round End
Done

Agreement

Case $r < f + 1$

p inserts (v_k, k) in New_p during the round r

p sends it to q during the round $r + 1 \leq f + 1$.

n.b., since p is assumed to eventually decide, it completes all rounds!

$Val_p \leftarrow [\bot, \ldots, \bot]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \bot$ /* *Initialization End* */

For all r from 1 to $f + 1$ **do** /* *Rounds* */
  Round Start
    `broadcast`(New_p) to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \bot$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f + 1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
  Round End
Done

Agreement

Case $r < f+1$

p inserts (v_k, k) in New_p during the round r

p sends it to q during the round $r+1 \leq f+1$.

n.b., since p is assumed to eventually decide, it completes all rounds!

So, q receives (v_k, k) at the latest during the round $r+1 \leq f+1$.

$Val_p \leftarrow [\bot, \ldots, \bot]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \bot$ /* *Initialization End* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
  Round Start
    `broadcast(`New_p`)` to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \bot$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
  Round End
Done

Agreement

Case $r = f + 1$

(v_k, k) has been relayed along a path of processes from k to the process from which p receives (v_k, k) during Round $f + 1$

$Val_p \leftarrow [\bot, \ldots, \bot]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \bot$ /* *Initialization End* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
  Round Start
    `broadcast` (New_p) to all other processes
    Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
    $New_p \leftarrow \emptyset$
    For all process $q \neq p$ **do**
      For all $(v, k) \in R_p[q]$ **do**
        If $Val_p[k] = \bot$ **then**
          $Val_p[k] \leftarrow v$
          $New_p \leftarrow New_p \cup \{(v, k)\}$
        End If
      Done
    Done
    If $r = f + 1$ **then** $d_p \leftarrow x$ where x is the first non-$\bot$ value in Val_p
  Round End
Done

Agreement

Case $r = f + 1$

(v_k, k) has been relayed along a path of processes from k to the process from which p receives (v_k, k) during Round $f+1$

This path contains $f+1$ distinct processes since each process relays each pair (value,ID) at most once

Init	Round 1	Round 2		Round f	Round $f+1$

$$k_1 = k \underset{(v_k,k)}{\rightarrow} k_2 \underset{(v_k,k)}{\rightarrow} k_3 \ldots \qquad \ldots \underset{(v_k,k)}{\rightarrow} k_{f+1} \underset{(v_k,k)}{\rightarrow} p$$

```
Val_p ← [⊥,...,⊥]        /* Initialization Beginning */
Val_p[p] ← v_p
New_p ← {(v_p,p)}
d_p ← ⊥        /* Initialization End */

For all r from 1 to f+1 do        /* Rounds */
    Round Start
        broadcast(New_p) to all other processes
        Let R_p[q] be the set received from q during r
(∅ if no message received from q)
        New_p ← ∅
        For all process q ≠ p do
            For all (v,k) ∈ R_p[q] do
                If Val_p[k] = ⊥ then
                    Val_p[k] ← v
                    New_p ← New_p ∪ {(v,k)}
                End If
            Done
        Done
        If r = f+1 then  d_p ← x where x is the first
non-⊥ value in Val_p
    Round End
Done
```

Agreement

Case $r = f + 1$

(v_k, k) has been relayed along a path of processes from k to the process from which p receives (v_k, k) during Round $f + 1$

This path contains $f + 1$ distinct processes since each process relays each pair (value,ID) at most once

$$k_1 = k \underset{(v_k,k)}{\overset{\text{Init} \quad \text{Round 1}}{\rightarrow}} k_2 \underset{(v_k,k)}{\overset{\text{Round 2}}{\rightarrow}} k_3 \ldots \underset{(v_k,k)}{\overset{\text{Round } r'}{\rightarrow}} c \ldots \underset{(v_k,k)}{\overset{\text{Round } f}{\rightarrow}} k_{f+1} \underset{(v_k,k)}{\overset{\text{Round } f+1}{\rightarrow}} p$$

Since at most f processes eventually crash, this path contains at least one correct process c

c has received (v_k, k) during a round $r' < f + 1$

So, c sent (v_k, k) to q during Round $r' + 1 \leq f + 1$

Hence, q has received (v_k, k) at the latest during Round $r' + 1 \leq f + 1$

```
Val_p ← [⊥,...,⊥]        /* Initialization Beginning */
Val_p[p] ← v_p
New_p ← {(v_p, p)}
d_p ← ⊥        /* Initialization End */

For all r from 1 to f+1 do        /* Rounds */
    Round Start
        broadcast(New_p) to all other processes
        Let R_p[q] be the set received from q during r (∅ if no message received from q)
        New_p ← ∅
        For all process q ≠ p do
            For all (v, k) ∈ R_p[q] do
                If Val_p[k] = ⊥ then
                    Val_p[k] ← v
                    New_p ← New_p ∪ {(v, k)}
                End If
            Done
        Done
        If r = f+1 then d_p ← x where x is the first non-⊥ value in Val_p
    Round End
Done
```

Agreement

Hence, in both cases, $Val_q[k] \leftarrow v_k$ at the latest during Round $f+1$: if $Val_p[k] = v_k \neq \perp$ at the end of the last round, then we also have $Val_q[k] = v_k$ at the end of that round. □

```
Val_p ← [⊥, ..., ⊥]      /* Initialization Beginning */
Val_p[p] ← v_p
New_p ← {(v_p, p)}
d_p ← ⊥      /* Initialization End */

For all r from 1 to f+1 do      /* Rounds */
    Round Start
        broadcast(New_p) to all other processes
        Let R_p[q] be the set received from q during r (∅ if no message received from q)
        New_p ← ∅
        For all process q ≠ p do
            For all (v, k) ∈ R_p[q] do
                If Val_p[k] = ⊥ then
                    Val_p[k] ← v
                    New_p ← New_p ∪ {(v, k)}
                End If
            Done
        Done
        If r = f+1 then  d_p ← x where x is the first non-⊥ value in Val_p
    Round End
Done
```

Agreement

Hence, in both cases, $Val_q[k] \leftarrow v_k$ at the latest during Round $f+1$: if $Val_p[k] = v_k \neq \perp$ at the end of the last round, then we also have $Val_q[k] = v_k$ at the end of that round.

Similarly, if $Val_q[k] = v_k \neq \perp$ at the end of the last round, then we also have $Val_p[k] = v_k$ at the end of that round

Hence, $Val_p = Val_q$ when p and q decide, and the agreement property follows. □

$Val_p \leftarrow [\perp, \ldots, \perp]$ /* *Initialization Beginning* */
$Val_p[p] \leftarrow v_p$
$New_p \leftarrow \{(v_p, p)\}$
$d_p \leftarrow \perp$ /* *Initialization End* */

For all r from 1 to $f+1$ **do** /* *Rounds* */
    Round Start
        `broadcast`(New_p) to all other processes
        Let $R_p[q]$ be the set received from q during r ($\emptyset$ if no message received from q)
        $New_p \leftarrow \emptyset$
        For all process $q \neq p$ **do**
            For all $(v,k) \in R_p[q]$ **do**
                If $Val_p[k] = \perp$ **then**
                    $Val_p[k] \leftarrow v$
                    $New_p \leftarrow New_p \cup \{(v,k)\}$
                End If
            Done
        Done
        If $r = f+1$ **then** $d_p \leftarrow x$ where x is the first non-$\perp$ value in Val_p
    Round End
Done

Decision

Any boolean function on Val_p fulfilling validity can be used

Examples:

- Decide 0 if 0 appears more often than 1, decide 1 otherwise
- Decide $Val_p[q]$ such that q is the minimum identifier satisfying $Val_p[q] \neq \perp$

Remarks

FloodSet can be emulated in the general synchronous model if

1. the network is complete and
2. bounds are known by processes.

Remarks

FloodSet can be emulated in the general synchronous model if

1. the network is complete and
2. bounds are known by processes.

It can be even emulated in a system where

1. all processes are synchronous and there is at least one bi-source, *i.e.*, a (synchronous) correct process whose all (incoming and outgoing) links are reliable and synchronous
2. the network is complete, and
3. bounds are known by processes.

Consequently only $n-1$ reliable and synchronous bidirectional links are sufficient instead of $\frac{n(n-1)}{2}$.

Roadmap

Initially Dead

A process is initially dead if it never participated to the processing of the algorithm

Assumption on crashes: every faulty process is initially dead

Equivalently: every process is either correct or initially dead

Other System Assumptions

1. A majority of processes (*i.e.*, at least $L = \lceil \frac{n+1}{2} \rceil$) is correct
2. Asynchronous processes
3. Asynchronous reliable links (not necessarily FIFO)

Principles

2 Phases:

Phase 1: (distributedly) compute a digraph $G = (V, E)$ where nodes represented correct processes and have an in-degree $L - 1$[5]

Phase 2: (distributedly) compute the transitive closure G^+ of G, *i.e.*, (i, j) is an arc of G^+ IFF i is an ancestor of j in G.

Precisely, at the end of the phase, each correct process "knows"

- its predecessors in G^+, *i.e.*, its ancestors in G,
- their incoming arcs,
- as well as the value they propose.

[5] Recall that L is the majority value

Phase 1

1. Each process broadcasts to all other processes its identifier
2. Each process collects the IDs in the $L-1$ first received messages

Phase 1

1. Each process broadcasts to all other processes its identifier
2. Each process collects the IDs in the $L-1$ first received messages

Each correct process receives at least $L-1$ messages since there is at least $L-1$ other correct processes

In $G=(V,E)$, $(i,j) \in E$ IFF j has received a Phase 1 message from i.

After Phase 1, each correct process knows its predecessors in G

Phase 2

GOAL: compute the transitive closure G^+ of G

Each process initiates Phase 2 by broadcasting to all other processes a message containing

1. its ID,
2. the value it proposes, and
3. the IDs of its predecessors in G.

Phase 2 terminates at p when p has received a Phase 2 message from all ancestors it hears about.

At the beginning, p only knows its predecessors. It then waits for Phase 2 messages from them. After receiving such messages, p maybe discovers new ancestors (*i.e.*, predecessors of predecessors). So, it waits messages from them, and so on so forth.

Phase 2

GOAL: compute the transitive closure G^+ of G

Each process initiates Phase 2 by broadcasting to all other processes a message containing

1. its ID,
2. the value it proposes, and
3. the IDs of its predecessors in G.

Phase 2 terminates at p when p has received a Phase 2 message from all ancestors it hears about.

At the beginning, p only knows its predecessors. It then waits for Phase 2 messages from them. After receiving such messages, p maybe discovers new ancestors (*i.e.*, predecessors of predecessors). So, it waits messages from them, and so on so forth.

At the end of Phase 2, each correct process "knows"

- its ancestors in G
- their incoming arcs
- the value they propose

Decision

At the end of Phase 2, each correct process "knows"

- its ancestors in G
- their incoming arcs
- the value they propose

Each (correct) process computes the arc of G^+ going to its ancestors.

Decision

At the end of Phase 2, each correct process “knows”

- its ancestors in G
- their incoming arcs
- the value they propose

Each (correct) process computes the arc of G^+ going to its ancestors.

Each (correct) process then determines which of its ancestors belong to the initial clique of G^+.

An initial clique of G^+ is a clique without incoming arcs, *i.e.* its a subset of nodes V' satisfying:

- V' is a clique of G^+
- There is no arc (i,j) in G^+ such that $i \notin V'$ and $j \in V'$

Result

Theorem 2

The FLP Algorithm solves the consensus in an asynchronous system where at most f processes are initially dead with $n > 2f$.

Proof Outline

Claim 1: There exists an initial clique in G^+

Claim 2: G^+ has a unique initial clique

Claim 3: The initial clique of G^+ can be computed polynomially in n

1. Each correct process has all members of the initial clique of G^+ among its ancestors in G
2. A process k is in an initial clique of G^+ IFF k is itself an ancestor in G of every process j that is an ancestor of k in G

Proof Outline

Claim 1: There exists an initial clique in G^+

Claim 2: G^+ has a unique initial clique

Claim 3: The initial clique of G^+ can be computed polynomially in n

1. Each correct process has all members of the initial clique of G^+ among its ancestors in G
2. A process k is in an initial clique of G^+ IFF k is itself an ancestor in G of every process j that is an ancestor of k in G

Hence, all correct processes agree on the initial clique of G^+ and know values proposed by members of this clique: they decide the same valid value according to this common knowledge.

Basic Property

Let p and q be two distinct correct processes.

- p is a predecessor of q in G,
- q is a predecessor of p in G, or
- p and q have a common predecessor in G.

Proof. By contradiction.

Basic Property

Let p and q be two distinct correct processes.

- p is a predecessor of q in G,
- q is a predecessor of p in G, or
- p and q have a common predecessor in G.

Proof. By contradiction.

Let $Pred(p)$ and $Pred(q)$ be the set of p's and q's predecessors in G.

Basic Property

Let p and q be two distinct correct processes.

- p is a predecessor of q in G,
- q is a predecessor of p in G, or
- p and q have a common predecessor in G.

Proof. By contradiction.

Let $Pred(p)$ and $Pred(q)$ be the set of p's and q's predecessors in G.

Then, $(Pred(p) \cup \{p\}) \cap (Pred(q) \cup \{q\}) = \emptyset$.

Basic Property

Let p and q be two distinct correct processes.

- p is a predecessor of q in G,
- q is a predecessor of p in G, or
- p and q have a common predecessor in G.

Proof. By contradiction.

Let $Pred(p)$ and $Pred(q)$ be the set of p's and q's predecessors in G.

Then, $(Pred(p) \cup \{p\}) \cap (Pred(q) \cup \{q\}) = \emptyset$.

Now, $|Pred(p)| = |Pred(p)| = L - 1$.

Basic Property

Let p and q be two distinct correct processes.

- p is a predecessor of q in G,
- q is a predecessor of p in G, or
- p and q have a common predecessor in G.

Proof. By contradiction.

Let $Pred(p)$ and $Pred(q)$ be the set of p's and q's predecessors in G.

Then, $(Pred(p) \cup \{p\}) \cap (Pred(q) \cup \{q\}) = \emptyset$.

Now, $|Pred(p)| = |Pred(p)| = L - 1$.

So, $|Pred(p) \cup \{p\} \cup Pred(q) \cup \{q\}| = 2(L-1) + 2 = 2L > n$, a contradiction. □

Claim 1

There exists an initial clique in G^{+}

Proof. In any digraph, there is at least one strongly connected source component S, *i.e.*, a strongly connected component in which no node has a predecessor out of the component.[6]

[6] Otherwise, every node has at least one ancestor which is not one of its descendents: with a finite number of nodes, it is impossible!

Claim 1

There exists an initial clique in G^+

Proof. In any digraph, there is at least one strongly connected source component S, *i.e.*, a strongly connected component in which no node has a predecessor out of the component.[6]

In S,

(1) every node is an ancestor of each other

(2) no node has an ancestor out of the component

Hence, in the transitive closure of the digraph, nodes of S form a clique (by (1)) and this clique is initial (by (2)).

□

[6] Otherwise, every node has at least one ancestor which is not one of its descendents: with a finite number of nodes, it is impossible!

Claim 3.1

Each correct process has all members of an initial clique of G^+ among its ancestors in G

Claim 3.1

Each correct process has all members of an initial clique of G^+ among its ancestors in G

Proof. Assume, by contradiction, that a process p of an initial clique of G^+ is not an ancestor in G of the process q in G.

Claim 3.1

Each correct process has all members of an initial clique of G^+ among its ancestors in G

Proof. Assume, by contradiction, that a process p of an initial clique of G^+ is not an ancestor in G of the process q in G.

Then, p is neither an ancestor nor a descendent of q in G.

Claim 3.1

Each correct process has all members of an initial clique of G^+ among its ancestors in G

Proof. Assume, by contradiction, that a process p of an initial clique of G^+ is not an ancestor in G of the process q in G.

Then, p is neither an ancestor nor a descendent of q in G.

Now, by definition, p and q are correct. So, from the basic property, we know that p and q have some common predecessor r in G.

Claim 3.1

Each correct process has all members of an initial clique of G^+ among its ancestors in G

Proof. Assume, by contradiction, that a process p of an initial clique of G^+ is not an ancestor in G of the process q in G.

Then, p is neither an ancestor nor a descendent of q in G.

Now, by definition, p and q are correct. So, from the basic property, we know that p and q have some common predecessor r in G.

- If r is not in the clique of p in G^+, then this clique is not initial, a contradiction.
- If r is in the clique of p in G^+, then p is an ancestor of r and so an ancestor of q in G, a contradiction.

□

Claim 2

G^+ has a unique initial clique

Proof. Assume, by contradiction, that two processes, p and q, are in two different initial cliques of G^+.

Claim 2

G^+ has a unique initial clique

Proof. Assume, by contradiction, that two processes, p and q, are in two different initial cliques of G^+.

By definition, p and q are not ancestor of each other: a contradiction to Claim 3.1. □

Claim 3.2

A process k is in an initial clique of G^+ IFF k is itself an ancestor in G of every process j that is an ancestor of k in G

Proof.

- By Claim 3.1, if k is in the initial clique of G^+, then k is itself an ancestor in G of every process j that is an ancestor of k in G

Claim 3.2

A process k is in an initial clique of G^+ IFF k is itself an ancestor in G of every process j that is an ancestor of k in G

Proof.

- By Claim 3.1, if k is in the initial clique of G^+, then k is itself an ancestor in G of every process j that is an ancestor of k in G
- By definition, if k is itself an ancestor in G of every process j that is an ancestor of k in G, then
  - k and its ancestors in G form a clique C in G^+.

Claim 3.2

A process k is in an initial clique of G^+ IFF k is itself an ancestor in G of every process j that is an ancestor of k in G

Proof.

- By Claim 3.1, if k is in the initial clique of G^+, then k is itself an ancestor in G of every process j that is an ancestor of k in G
- By definition, if k is itself an ancestor in G of every process j that is an ancestor of k in G, then
  - k and its ancestors in G form a clique C in G^+.
  - Moreover, k has no predecessor out of C in G^+.

    Assume the contrary. Then, k has a predecessor in G^+, *i.e.*, an ancestor in G, that has not k as predecessor in G^+, *i.e.*, as ancestor in G; a contradiction.

  Hence, k is necessarily in the initial clique of G^+

□

Roadmap

Randomization Approaches

Las Vegas: randomized algorithm that always gives correct results

but, the termination is not deterministically guaranteed: it is guaranteed with a positive probability

$\rightarrow$ Only the expected runtime should be finite

Monte Carlo: termination is deterministically guaranteed

but, the output may be incorrect with a certain (typically small) probability

Randomization Approaches

Las Vegas: randomized algorithm that always gives correct results

but, the termination is not deterministically guaranteed: it is guaranteed with a positive probability

$\rightarrow$ Only the expected runtime should be finite

Monte Carlo: termination is deterministically guaranteed

but, the output may be incorrect with a certain (typically small) probability

We now study the Ben-Or Algorithm (Las Vegas approach)

Assumptions

1. A majority of processes is correct: the maximal number of crashes f satisfies $n > 2f$.
2. Asynchronous processes
3. Asynchronous reliable links (not necessarily FIFO)
4. Any process p can broadcast a message to all processes (p included!)

Constants & Variables

- n: number of processes
- f: maximum number of crashes
- v_p: a boolean input, the value proposed by process p — v_p may be modified
- $d_p \in \{\bot, 0, 1\}$: the decision variable of process p
- $r \in \mathbb{N}$: the round number

Randomization & Messages

Each process can use `Random(0,1)` which returns a random value 0 or 1 with uniform probability $\frac{1}{2}$.

Two types of message:

- *R*: a report
- *P*: a proposition

The Code

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

Rounds

Each process executes an infinite loop[a]

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

[a] We will see later how a process can halt this loop without compromising the consensus specification.

Rounds

Each process executes an infinite loop[a]

Loop iteration = (asynchronous) round

r: current round number

$d_p \leftarrow \bot$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    `broadcast` (R, r, v_p) `to all processes` (p `included`)
    `wait to receive` $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        `broadcast` (P, r, x) `to all processes` (p `included`)
    else
        `broadcast` $(P, r, ?)$ `to all processes` (p `included`)
    End If
    `wait to receive` $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \bot$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \texttt{Random}(0, 1)$
    End If
Done

[a] We will see later how a process can halt this loop without compromising the consensus specification.

Rounds

Each process executes an infinite loop[a]
Loop iteration = (asynchronous) round
r: current round number

Round = 2 phases:

- Report Phase: Every (non-crashed) process reports a value to all processes (R, r, x) with $x \in \{0,1\}$: p reports value x during Round r
- Proposition Phase: Every (non-crashed) process proposes a value to all processes (P, r, x) with $x \in \{0,1,?\}$: p proposes value x during Round r

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast $(P, r, ?)$ to all processes (p included)
    End If
    wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \mathtt{Random}(0,1)$
    End If
Done

[a] We will see later how a process can halt this loop without compromising the consensus specification.

Rounds

Each process executes an infinite loop[a]

Loop iteration = (asynchronous) round

r: current round number

Round = 2 phases:

- Report Phase: Every (non-crashed) process reports a value to all processes (R, r, x) with $x \in \{0,1\}$: p reports value x during Round r
- Proposition Phase: Every (non-crashed) process proposes a value to all processes (P, r, x) with $x \in \{0,1,?\}$: p proposes value x during Round r

N.B., each phase terminates at each correct process since it waits for $n - f$ messages and the maximum number of crashes is f

[a] We will see later how a process can halt this loop without compromising the consensus specification.

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast $(P, r, ?)$ to all processes (p included)
    End If
    wait to receive $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \texttt{Random}(0,1)$
    End If
Done

Integrity

From the code, we can deduce

Remark 1

Every process decides at most one.

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r++$
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast $(P, r, ?)$ to all processes (p included)
    End If
    wait to receive $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \text{Random}(0, 1)$
    End If
Done

Propositions

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

Proof.

□

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r++$
  broadcast (R, r, v_p) to all processes (p included)
  wait to receive $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    broadcast (P, r, x) to all processes (p included)
  else
    broadcast $(P, r, ?)$ to all processes (p included)
  End If
  wait to receive $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \text{Random}(0, 1)$
  End If
Done

Propositions

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

Proof. During Round r, at most n report messages are broadcast. □

```
d_p ← ⊥
r ← 0
While true do
   r++
   broadcast (R, r, v_p) to all processes (p included)
   wait to receive n − f messages (R, r, _) where "_" is 0 or 1
   If more than n/2 received messages (R, r, x) with the same value x
then
      broadcast (P, r, x) to all processes (p included)
   else
      broadcast (P, r, ?) to all processes (p included)
   End If
   wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
   If at least f + 1 received messages (P, r, x) with x ≠ ? then
      If d_p = ⊥ then d_p ← x
   End If
   If at least 1 received message (P, r, x) with x ≠ ? then
      v_p ← x
   else
      v_p ← Random(0,1)
   End If
Done
```

Propositions

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

Proof. During Round r, at most n report messages are broadcast.

So, if a process receives more than $\frac{n}{2}$ R messages with the same value x during a round, no other process can receive more than $\frac{n}{2}$ R messages with value $(x+1)$ mod 2 during the same round. □

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  broadcast (R, r, v_p) to all processes (p included)
  wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    broadcast (P, r, x) to all processes (p included)
  else
    broadcast $(P, r, ?)$ to all processes (p included)
  End If
  wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \mathrm{Random}(0,1)$
  End If
Done

Propositions

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

Proof. During Round r, at most n report messages are broadcast.

So, if a process receives more than $\frac{n}{2}$ R messages with the same value x during a round, no other process can receive more than $\frac{n}{2}$ R messages with value $(x+1) \bmod 2$ during the same round.

Hence, only x and ? can be proposed during Round r. □

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

Decide soon

Below, the value of v_p at Round 0 is the initial value of v_p

Lemma 2

Let $x \in \{0,1\}$. Let $r > 0$. Let q be any process that still has not decided at the end of Round $r-1$ and that will complete Round r.

If $v_p = x$ at the end of Round $r-1$ for every process p that will send a report during Round r, then q will decide x during Round r.

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
$r{+}{+}$
broadcast (R, r, v_p) to all processes (p included)
wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
broadcast (P, r, x) to all processes (p included)
else
broadcast $(P, r, ?)$ to all processes (p included)
End If
wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
If $d_p = \perp$ **then** $d_p \leftarrow x$
End If
If at least 1 received message (P, r, x) with $x \neq ?$ **then**
$v_p \leftarrow x$
else
$v_p \leftarrow \mathtt{Random}(0,1)$
End If
Done

Decide soon

Proof of Lemma 2

Proof. By hypothesis, every report received during Round r reports value x. □

```
d_p ← ⊥
r ← 0
While true do
   r++
   broadcast (R, r, v_p) to all processes (p included)
   wait to receive n − f messages (R, r, _) where "_" is 0 or 1
   If more than n/2 received messages (R, r, x) with the same value x then
      broadcast (P, r, x) to all processes (p included)
   else
      broadcast (P, r, ?) to all processes (p included)
   End If
   wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
   If at least f + 1 received messages (P, r, x) with x ≠ ? then
      If d_p = ⊥ then d_p ← x
   End If
   If at least 1 received message (P, r, x) with x ≠ ? then
      v_p ← x
   else
      v_p ← Random(0,1)
   End If
Done
```

Decide soon

Proof of Lemma 2

Proof. By hypothesis, every report received during Round r reports value x.

Since every (non-crashed) process receives at least $n-f$ reports during Round r and $n-f > \frac{n}{2}$, every process that completes Round r proposes the same value $x \neq ?$ during Round r.

□

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  broadcast (R, r, v_p) to all processes (p included)
  wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    broadcast (P, r, x) to all processes (p included)
  else
    broadcast $(P, r, ?)$ to all processes (p included)
  End If
  wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \mathrm{Random}(0,1)$
  End If
Done

Decide soon

Proof of Lemma 2

Proof. By hypothesis, every report received during Round r reports value x.

Since every (non-crashed) process receives at least $n-f$ reports during Round r and $n-f > \frac{n}{2}$, every process that completes Round r proposes the same value $x \neq ?$ during Round r.

Thus, every proposition sent during Round r will be only for x.

□

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

Decide soon

Proof of Lemma 2

Proof. By hypothesis, every report received during Round r reports value x.

Since every (non-crashed) process receives at least $n-f$ reports during Round r and $n-f > \frac{n}{2}$, every process that completes Round r proposes the same value $x \neq ?$ during Round r.

Thus, every proposition sent during Round r will be only for x.

Since all correct processes (at least $n-f$) will broadcast a proposition (for x) during Round r and $n-f > f$, each process that will terminate Round r will receive at least $f+1$ propositions for x (and only for x) during the round and so will decide x during the round. □

$d_p \leftarrow \bot$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  `broadcast` (R, r, v_p) `to all processes` $(p$ `included)`
  `wait to receive` $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    `broadcast` (P, r, x) `to all processes` $(p$ `included)`
  else
    `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included)`
  End If
  `wait to receive` $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \bot$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \texttt{Random}(0,1)$
  End If
Done

Decision & Termination

Lemma 3

If a process decides x during Round r, then all processes that still has not decided at the end of Round r and that will terminate Round $r+1$ will decide x at the end of Round $r+1$.

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  broadcast (R, r, v_p) to all processes (p included)
  wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    broadcast (P, r, x) to all processes (p included)
  else
    broadcast $(P, r, ?)$ to all processes (p included)
  End If
  wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \mathrm{Random}(0,1)$
  End If
Done

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1. □

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
    $r{+}{+}$
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
    If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast $(P, r, ?)$ to all processes (p included)
    End If
    wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
    If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
        If $d_p = \perp$ **then** $d_p \leftarrow x$
    End If
    If at least 1 received message (P, r, x) with $x \neq ?$ **then**
        $v_p \leftarrow x$
    else
        $v_p \leftarrow \mathrm{Random}(0,1)$
    End If
Done

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1.

Let q be a process that sends a report at Round $r+1$.

□

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  broadcast (R, r, v_p) to all processes (p included)
  wait to receive $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    broadcast (P, r, x) to all processes (p included)
  else
    broadcast $(P, r, ?)$ to all processes (p included)
  End If
  wait to receive $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \mathtt{Random}(0,1)$
  End If
Done

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1.

Let q be a process that sends a report at Round $r+1$.

q received at least $n-f$ propositions during Round r.

□

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1.

Let q be a process that sends a report at Round $r+1$.

q received at least $n-f$ propositions during Round r.

q received at least one proposition for x since there are at most n propositions during Round r and at least $f+1$ of them are for x, so at most $n-f-1$ are not for x.

□

```
d_p ← ⊥
r ← 0
While true do
   r++
   broadcast (R, r, v_p) to all processes (p included)
   wait to receive n − f messages (R, r, _) where "_" is 0 or 1
   If more than n/2 received messages (R, r, x) with the same value x then
      broadcast (P, r, x) to all processes (p included)
   else
      broadcast (P, r, ?) to all processes (p included)
   End If
   wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
   If at least f + 1 received messages (P, r, x) with x ≠ ? then
      If d_p = ⊥ then d_p ← x
   End If
   If at least 1 received message (P, r, x) with x ≠ ? then
      v_p ← x
   else
      v_p ← Random(0,1)
   End If
Done
```

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1.

Let q be a process that sends a report at Round $r+1$.

q received at least $n-f$ propositions during Round r.

q received at least one proposition for x since there are at most n propositions during Round r and at least $f+1$ of them are for x, so at most $n-f-1$ are not for x.

By Lemma 1, q did not receive any proposition for $(x+1) \bmod 2$ during Round r. Hence, $v_q \leftarrow x$ during Round r.

□

```
d_p ← ⊥
r ← 0
While true do
   r++
   broadcast (R, r, v_p) to all processes (p included)
   wait to receive n − f messages (R, r, _) where "_" is 0 or 1
   If more than n/2 received messages (R, r, x) with the same value x then
      broadcast (P, r, x) to all processes (p included)
   else
      broadcast (P, r, ?) to all processes (p included)
   End If
   wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
   If at least f + 1 received messages (P, r, x) with x ≠ ? then
      If d_p = ⊥ then d_p ← x
   End If
   If at least 1 received message (P, r, x) with x ≠ ? then
      v_p ← x
   else
      v_p ← Random(0,1)
   End If
Done
```

Decision & Termination

Proof of Lemma 3

Proof. If a process p decides x during Round r, then p has received at least $f+1$ propositions with $x \neq ?$ during Round r and these values are identical, by Lemma 1.

Let q be a process that sends a report at Round $r+1$.

q received at least $n-f$ propositions during Round r.

q received at least one proposition for x since there are at most n propositions during Round r and at least $f+1$ of them are for x, so at most $n-f-1$ are not for x.

By Lemma 1, q did not receive any proposition for $(x+1) \bmod 2$ during Round r. Hence, $v_q \leftarrow x$ during Round r.

So, every process q that sends a report during Round $r+1$ satisfies $v_q = x$ at the end of Round r.

By Lemma 2, all processes that still has not decided at the end of Round r and that will terminate Round $r+1$ will decide x during Round $r+1$. □

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  `broadcast` (R, r, v_p) `to all processes` $(p$ `included)`
  `wait to receive` $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    `broadcast` (P, r, x) `to all processes` $(p$ `included)`
  else
    `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included)`
  End If
  `wait to receive` $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \mathtt{Random}(0,1)$
  End If
Done

Result

Theorem 3

The Ben-Or Algorithm solves the probabilistic consensus , *i.e.*, it satisfies:

- Integrity
- Validity,
- Agreement, and
- Termination with probability 1,

in an asynchronous system where at most f processes crash with $n > 2f$

Proof of Theorem 3

Integrity:

Remark 1

Every process decides at most one.

Proof of Theorem 3

Integrity:

Remark 1

Every process decides at most one.

Validity:

Lemma 2

Let $x \in \{0, 1\}$. Let $r > 0$. Let q be any process that still has not decided at the end of Round $r - 1$ and that will complete Round r.

If $v_p = x$ at the end of Round $r - 1$ for every process p that will send a report during Round r, then q will decide x during Round r.

Recall that the value of v_p at Round 0 is the initial value of v_p. So, with $r = 1$, we obtain the validity.

Proof of Theorem 3

Agreement

Consider **the first round** r where at least one process decides.

```
d_p ← ⊥
r ← 0
While true do
   r++
   broadcast (R, r, v_p) to all processes (p included)
   wait to receive n − f messages (R, r, _) where "_" is 0 or 1
   If more than n/2 received messages (R, r, x) with the same value x then
      broadcast (P, r, x) to all processes (p included)
   else
      broadcast (P, r, ?) to all processes (p included)
   End If
   wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
   If at least f + 1 received messages (P, r, x) with x ≠ ? then
      If d_p = ⊥ then d_p ← x
   End If
   If at least 1 received message (P, r, x) with x ≠ ? then
      v_p ← x
   else
      v_p ← Random(0,1)
   End If
Done
```

Proof of Theorem 3

Agreement

Consider **the first round** r where at least one process decides.

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

All processes that decide during Round r, decide the same value x.

```
d_p ← ⊥
r ← 0
While true do
    r++
    broadcast (R, r, v_p) to all processes (p included)
    wait to receive n − f messages (R, r, _) where "_" is 0 or 1
    If more than n/2 received messages (R, r, x) with the same value x then
        broadcast (P, r, x) to all processes (p included)
    else
        broadcast (P, r, ?) to all processes (p included)
    End If
    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
    If at least f + 1 received messages (P, r, x) with x ≠ ? then
        If d_p = ⊥ then d_p ← x
    End If
    If at least 1 received message (P, r, x) with x ≠ ? then
        v_p ← x
    else
        v_p ← Random(0,1)
    End If
Done
```

Proof of Theorem 3

Agreement

Consider **the first round** r where at least one process decides.

Lemma 1

No two processes respectively propose 0 and 1 during the same round r.

All processes that decide during Round r, decide the same value x.

Lemma 3

If a process decides x during Round r, then all processes that still has not decided at the end of Round r and that will terminate Round $r+1$ will decide x at the end of Round $r+1$.

All processes that do not decide in Round r and that will complete Round $r+1$ will also decide x in Round $r+1$.

$d_p \leftarrow \perp$
$r \leftarrow 0$
While *true* **do**
  $r{+}{+}$
  `broadcast` (R, r, v_p) `to all processes` $(p$ `included)`
  `wait to receive` $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
  If more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
    `broadcast` (P, r, x) `to all processes` $(p$ `included)`
  else
    `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included)`
  End If
  `wait to receive` $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
  If at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
    If $d_p = \perp$ **then** $d_p \leftarrow x$
  End If
  If at least 1 received message (P, r, x) with $x \neq ?$ **then**
    $v_p \leftarrow x$
  else
    $v_p \leftarrow \texttt{Random}(0,1)$
  End If
Done

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

1: $d_p \leftarrow \perp$
2: $r \leftarrow 0$
3: **While** *true* **do**
4: $r{+}{+}$
5: `broadcast` (R, r, v_p) `to all processes` $(p$ `included)`
6: `wait to receive` $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
7: **If** more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
8: `broadcast` (P, r, x) `to all processes` $(p$ `included)`
9: **else**
10: `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included)`
11: **End If**
12: `wait to receive` $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
13: **If** at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
14: **If** $d_p = \perp$ **then** $d_p \leftarrow x$
15: **End If**
16: **If** at least 1 received message (P, r, x) with $x \neq ?$ **then**
17: $v_p \leftarrow x$
18: **else**
19: $v_p \leftarrow \mathtt{Random}(0, 1)$
20: **End If**
21: **Done**

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

By Lemma 1, all processes in S_d set their v variable to the same value x.

1: $d_p \leftarrow \perp$
2: $r \leftarrow 0$
3: **While** *true* **do**
4: $r{+}{+}$
5: broadcast (R, r, v_p) to all processes (p included)
6: wait to receive $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
7: **If** more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
8: broadcast (P, r, x) to all processes (p included)
9: **else**
10: broadcast $(P, r, ?)$ to all processes (p included)
11: **End If**
12: wait to receive $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
13: **If** at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
14: **If** $d_p = \perp$ **then** $d_p \leftarrow x$
15: **End If**
16: **If** at least 1 received message (P, r, x) with $x \neq ?$ **then**
17: $v_p \leftarrow x$
18: **else**
19: $v_p \leftarrow \mathrm{Random}(0, 1)$
20: **End If**
21: **Done**

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

By Lemma 1, all processes in S_d set their v variable to the same value x.

$\forall p \in S_r$, p chooses x with probability $\frac{1}{2}$.

1: $d_p \leftarrow \perp$
2: $r \leftarrow 0$
3: **While** *true* **do**
4: $r{+}{+}$
5: `broadcast` (R, r, v_p) `to all processes` $($p$ `included`$)$
6: `wait to receive` $n - f$ messages $(R, r, \_)$ where "_" is 0 or 1
7: **If** more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
8: `broadcast` (P, r, x) `to all processes` $(p$ `included`$)$
9: **else**
10: `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included`$)$
11: **End If**
12: `wait to receive` $n - f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
13: **If** at least $f + 1$ received messages (P, r, x) with $x \neq ?$ **then**
14: **If** $d_p = \perp$ **then** $d_p \leftarrow x$
15: **End If**
16: **If** at least 1 received message (P, r, x) with $x \neq ?$ **then**
17: $v_p \leftarrow x$
18: **else**
19: $v_p \leftarrow \texttt{Random}(0,1)$
20: **End If**
21: **Done**

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

By Lemma 1, all processes in S_d set their v variable to the same value x.

$\forall p \in S_r$, p chooses x with probability $\frac{1}{2}$.

Thus, with a probability $\geq \frac{1}{2^n}$, all process in S satisfy $v = x$ at the end of Round r.

1: $d_p \leftarrow \perp$
2: $r \leftarrow 0$
3: **While** *true* **do**
4: $\quad r{+}{+}$
5: $\quad$ `broadcast` (R, r, v_p) `to all processes` (p `included`)
6: $\quad$ `wait to receive` $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
7: $\quad$ **If** more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
8: $\quad\quad$ `broadcast` (P, r, x) `to all processes` (p `included`)
9: $\quad$ **else**
10: $\quad\quad$ `broadcast` $(P, r, ?)$ `to all processes` (p `included`)
11: $\quad$ **End If**
12: $\quad$ `wait to receive` $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
13: $\quad$ **If** at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
14: $\quad\quad$ **If** $d_p = \perp$ **then** $d_p \leftarrow x$
15: $\quad$ **End If**
16: $\quad$ **If** at least 1 received message (P, r, x) with $x \neq ?$ **then**
17: $\quad\quad$ $v_p \leftarrow x$
18: $\quad$ **else**
19: $\quad\quad$ $v_p \leftarrow \mathtt{Random}(0,1)$
20: $\quad$ **End If**
21: **Done**

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

By Lemma 1, all processes in S_d set their v variable to the same value x.

$\forall p \in S_r$, p chooses x with probability $\frac{1}{2}$.

Thus, with a probability $\geq \frac{1}{2^n}$, all process in S satisfy $v = x$ at the end of Round r.

By Lemma 2, all correct processes decides at Round $r+1$ with a probability $\geq \frac{1}{2^n}$.

1: $d_p \leftarrow \perp$
2: $r \leftarrow 0$
3: **While** *true* **do**
4: $r{+}{+}$
5: `broadcast` (R, r, v_p) `to all processes` $(p$ `included)`
6: `wait to receive` $n-f$ messages $(R, r, \_)$ where "_" is 0 or 1
7: **If** more than $\frac{n}{2}$ received messages (R, r, x) with the same value x **then**
8: `broadcast` (P, r, x) `to all processes` $(p$ `included)`
9: **else**
10: `broadcast` $(P, r, ?)$ `to all processes` $(p$ `included)`
11: **End If**
12: `wait to receive` $n-f$ messages $(P, r, \_)$ where "_" is 0, 1, or ?
13: **If** at least $f+1$ received messages (P, r, x) with $x \neq ?$ **then**
14: **If** $d_p = \perp$ **then** $d_p \leftarrow x$
15: **End If**
16: **If** at least 1 received message (P, r, x) with $x \neq ?$ **then**
17: $v_p \leftarrow x$
18: **else**
19: $v_p \leftarrow \texttt{Random}(0,1)$
20: **End If**
21: **Done**

Proof of Theorem 3

Termination with Probability 1

Let $S = S_d \cup S_r$ be the set of processes that modify their variable v at the end of Round r as follows.

- S_d: processes that execute Line 17
- S_r: processes that execute Line 19

By Lemma 1, all processes in S_d set their v variable to the same value x.

$\forall p \in S_r$, p chooses x with probability $\frac{1}{2}$.

Thus, with a probability $\geq \frac{1}{2^n}$, all process in S satisfy $v = x$ at the end of Round r.

By Lemma 2, all correct processes decides at Round $r+1$ with a probability $\geq \frac{1}{2^n}$.

The probability P that every correct process decides at Round $r > 1$ is $\geq \frac{1}{2^n}$:

Termination with probability $\lim_{r \to \infty} 1 - (1-P)^{r-1} = 1$

($O(2^n)$) **rounds at expectation)**

```
1:  d_p ← ⊥
2:  r ← 0
3:  While true do
4:     r++
5:     broadcast (R, r, v_p) to all processes (p included)
6:     wait to receive n − f messages (R, r, _) where "_" is 0 or 1
7:     If more than n/2 received messages (R, r, x) with the same value x then
8:        broadcast (P, r, x) to all processes (p included)
9:     else
10:       broadcast (P, r, ?) to all processes (p included)
11:    End If
12:    wait to receive n − f messages (P, r, _) where "_" is 0, 1, or ?
13:    If at least f + 1 received messages (P, r, x) with x ≠ ? then
14:       If d_p = ⊥ then d_p ← x
15:    End If
16:    If at least 1 received message (P, r, x) with x ≠ ? then
17:       v_p ← x
18:    else
19:       v_p ← Random(0,1)
20:    End If
21: Done
```

Leave the infinite loop

If p decides in Round r, all other correct processes decide at last during Round $r+1$.

Leave the infinite loop

If p decides in Round r, all other correct processes decide at last during Round $r+1$.

So, after deciding

- p can broadcast the messages R and P for the Round $r+1$ with value v_p without waiting anything
- and then leave the loop without compromising the specification.

Roadmap

References

[1] M. Ben-Or.
Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract).
In R. L. Probert, N. A. Lynch, and N. Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983.

[2] T. D. Chandra and S. Toueg.
Unreliable failure detectors for asynchronous systems (preliminary version).
In L. Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 325–340. ACM, 1991.

[3] M. J. Fischer, N. A. Lynch, and M. Paterson.
Impossibility of distributed consensus with one faulty process.
J. ACM, 32(2):374–382, 1985.

[4] N. A. Lynch.
Distributed Algorithms.
Morgan Kaufmann, 1st edition, 1996.