Failure Detectors: definition, algorithms, and applications

Alain Cournier Stéphane Devismes

Université de Picardie Jules Verne

April 26, 2023

Roadmap

Roadmap

1. Introduction
2. Definition
3. Application: a Consensus Algorithm
   - The Model
   - The Algorithm
   - The Proof
4. Implementation of a Failure Detector
   - $\diamond\mathcal{P}$
   - The Model
   - The Algorithm
   - The Proof
5. References

Outline of the FLP's Impossibility [3]

The impossibility of the consensus comes from the fact that correct processes cannot differentiate slow processes from crashed ones.

Outline of the FLP's Impossibility [3]

The impossibility of the consensus comes from the fact that correct processes cannot differentiate slow processes from crashed ones.

Using synchrony assumptions, processes can obtain information about crashes.

Then, this information enables solving problems, including consensus.

The Failure Detector Approach [1]

In a software engineering spirit:

- separate the necessary knowledge on crashes to solve the problem (the definition of the failure detector)
- from the way it can be obtained[1] (the implementation of the failure detector)

[1] In particular, the necessary assumptions on the system.

[2] A lesson we be dedicated to this aspect.

The Failure Detector Approach [1]

In a software engineering spirit:

- separate the necessary knowledge on crashes to solve the problem (the definition of the failure detector)
- from the way it can be obtained[1] (the implementation of the failure detector)

Advantages

- Separation of concerns: modularity and simplicity
- Possibility to compare and to have a necessary and sufficient assumption (the minimum failure detector to solve a problem)[2]

[1] In particular, the necessary assumptions on the system.

[2] A lesson we be dedicated to this aspect.

Roadmap

(Distributed) Failure Detector: an Oracle

Each process p can access a local failure detector module (an oracle function) denoted by $\mathcal{D}_p$.

[3] *N.b.*, some failure detectors, such as Ω or Σ, do not return a list of suspected processes.

(Distributed) Failure Detector: an Oracle

Each process p can access a local failure detector module (an oracle function) denoted by $\mathcal{D}_p$.

Each module watches a subset of system processes (usually the whole set of processes), and returns information about crashed: usually a set of suspected processes.[3]

Precisely, the identifiers of processes that are suspected of being crashed.

[3] *N.b.*, some failure detectors, such as Ω or Σ, do not return a list of suspected processes.

(Distributed) Failure Detector: an Oracle

Each process p can access a local failure detector module (an oracle function) denoted by $\mathcal{D}_p$.

Each module watches a subset of system processes (usually the whole set of processes), and returns information about crashed: usually a set of suspected processes.[3]

Precisely, the identifiers of processes that are suspected of being crashed.

Unless otherwise mentioned, we will always assume that

each local failure detector module watches all processes and returns a list of suspected processes.

[3] *N.b.*, some failure detectors, such as Ω or Σ, do not return a list of suspected processes.

Unreliable Failure Detectors

Each local module can make mistakes:

- by wrongly suspecting correct processes
- by missing some crashed processes

Failure Detector Classes

The classes of failure detectors are distinguished by two important properties:

Completeness: restrict the ability of the failure detector module to detect crashes

Accuracy: qualify the possibility of the failure detector module to wrongly suspect correct processes

Completeness: Examples

Strong Completeness: Every faulty process is eventually permanently suspected by every correct process.

Completeness: Examples

Strong Completeness: Every faulty process is eventually permanently suspected by every correct process.

Weak Completeness: Every faulty process is eventually permanently suspected by some correct process.

Accuracy: Examples

Strong accuracy: no process is suspected (by any alive process[4]) before it crashes.

[4] An alive process is either a correct process or a faulty process that has not crashed yet.

[5] As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Accuracy: Examples

Strong accuracy: no process is suspected (by any alive process[4]) before it crashes.

Weak accuracy: some correct process is never suspected (by any alive process).

[4] An alive process is either a correct process or a faulty process that has not crashed yet.

[5] As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Accuracy: Examples

Strong accuracy: no process is suspected (by any alive process[4]) before it crashes.

Weak accuracy: some correct process is never suspected (by any alive process).

Eventual strong accuracy: there is a time after which no correct process is suspected by any correct process.[5]

[4] An alive process is either a correct process or a faulty process that has not crashed yet.

[5] As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Accuracy: Examples

Strong accuracy: no process is suspected (by any alive process[4]) before it crashes.

Weak accuracy: some correct process is never suspected (by any alive process).

Eventual strong accuracy: there is a time after which no correct process is suspected by any correct process.[5]

Eventual weak accuracy: there is a time after which some correct process is never suspected by any correct process.

[4] An alive process is either a correct process or a faulty process that has not crashed yet.

[5] As explained in [1], we can use correct instead of alive with loss of generality for eventual strong/weak accuracy.

Some Classes of Failure Detectors

Completeness	**Accuracy**			
	Strong	Weak	Eventually Strong	Eventually Weak
Strong	Perfect $\mathcal{P}$	Strong $\mathcal{S}$	Eventually Perfect $\diamond\mathcal{P}$	Eventually Strong $\diamond\mathcal{S}$
Weak	Quasi-perfect $\mathcal{Q}$	Weak $\mathcal{W}$	Eventually Quasi-perfect $\diamond\mathcal{Q}$	Eventually Weak $\diamond\mathcal{W}$

Roadmap

Failure Detector $\mathcal{S}$

We now study a consensus algorithm[6] that uses $\mathcal{S}$:

Strong Completeness: Every faulty process is eventually permanently suspected by every correct process.

Weak accuracy: some correct process is never suspected (by any alive process).

[6] This algorithm is inspired for one of Chandra and Toueg [2].

Assumptions

1. Failure detector $\mathcal{S}$: local module $\mathcal{S}_p$ for each process p
2. Asynchronous processes
3. Asynchronous reliable links (not necessarily FIFO)
4. Any process p can broadcast a message to all processes (p included!)
5. No assumption on the number of crashes!

Principles

3 Phases:

Phase 1: $n - 1$ asynchronous rounds where proposed values are broadcast and relayed.

Phase 2: 1 asynchronous round where all alive processes agree on the value of a vector V based on proposed values.

Phase 3: each alive process decides according to the vector V.

Constants & Variables

- Processes are identified: a process and its identifier are used equivalently
- V: set of processes
- n: number of processes
- v_p: a boolean (read-only) input, the value proposed by process p
- $d_p \in \{\perp, 0, 1\}$: the decision variable
- $r_p \in \mathbb{N}$: the round number of process p
- $\Delta_p[], V_p[]$: vectors indexed on the process IDs.

  $\rightarrow \forall q \in V,\ V_p[q] \in \{0, 1, \perp\}$ and $\Delta_p[q] \in \{0, 1, \perp\}$

Algorithm of Chandra and Toueg

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** /* *Phase 1* */
    broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
    wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
broadcast $\langle V_p \rangle$ to V (p included) /* *Phase 2* */
wait to receive $\langle V_q \rangle$ from every process $q \in V \setminus \mathcal{S}_p$
Let *lastmsgs*$_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
    If $\exists V_q \in \textit{lastmsgs}_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* *Phase 3* */

Validity

Remark 1

At the end of Round $r \geq 1$, $\Delta_p[q] \neq \perp$ IFF $p \neq q$, $\Delta_p[q]$ is the value proposed by q, and p has received this value for the first time during Round r.

From Remark 1 and owing the fact that $V_p[p] = v_p$ after the initialization (Lines 1-4), we can deduce the following lemma by definition of the algorithm:

Lemma 1

For every two processes p and q, after the initialization (Lines 1-4) and while p is not crashed, $V_p[q]$ is either v_q or $\perp$.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** /* Phase 1 */
6:     `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` /* Phase 2 */
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** /* Phase 1 */
6:     broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
7:     wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: broadcast $\langle V_p \rangle$ to V (p included) /* Phase 2 */
18: wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.
The only way to prevent a correct process p from reaching Line 23 is to block it forever on Line 7 or 18 in some round r.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** /* Phase 1 */
6: `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7: `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` /* Phase 2 */
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.
The only way to prevent a correct process p from reaching Line 23 is to block it forever on Line 7 or 18 in some round r.

Let $Faulty \subseteq V$ be the set of faulty processes. Let $Correct \subseteq V$ be the set of correct processes.

By definition, $V = Faulty \,\dot\cup\, Correct$.

By strong completeness, eventually $V \setminus \mathcal{S}_p \subseteq Correct$ forever

1: $d_p \leftarrow \perp$

2: $V_p \leftarrow [\perp, \ldots, \perp]$

3: $V_p[p] \leftarrow v_p$

4: $\Delta_p \leftarrow V_p$

5: **For all** r_p from 1 to $n-1$ **do** /* Phase 1 */

6:     `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`

7:     `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$

8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$

9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$

10:     **For all** $k \in V$ **do**

11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**

12:             $V_p[k] \leftarrow \Delta_q[k]$

13:             $\Delta_p[k] \leftarrow \Delta_q[k]$

14:         **End If**

15:     **Done**

16: **Done**

17: `broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` /* Phase 2 */

18: `wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$

19: Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$

20: **For all** $k \in V$ **do**

21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$

22: **Done**

23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Termination

Lemma 2

Every correct process eventually executes Line 23.

Proof.
The only way to prevent a correct process p from reaching Line 23 is to block it forever on Line 7 or 18 in some round r.

Let $Faulty \subseteq V$ be the set of faulty processes. Let $Correct \subseteq V$ be the set of correct processes.

By definition, $V = Faulty \,\dot{\cup}\, Correct$.

By strong completeness, eventually $V \setminus \mathcal{S}_p \subseteq Correct$ forever

We now consider the cases of Line 7 and 18 separately.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** — */* Phase 1 */*
6: `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7: `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` — */* Phase 2 */*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p — */* Phase 3 */*

Termination

Case 1: Line 7

By induction on r: $\forall p \in \mathit{Correct}$, p completes every Round $r \in [1..n-1]$.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *`/* Phase 1 */`*
6: `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`
7: `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: `broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` *`/* Phase 2 */`*
18: `wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $\mathit{lastmsgs}_p$ be the set of received $\langle V_q\rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in \mathit{lastmsgs}_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *`/* Phase 3 */`*

Termination

Case 1: Line 7

By induction on r: $\forall p \in Correct$, p completes every Round $r \in [1..n-1]$.

$r = 1$: $\forall q \in Correct$, q completes Line 6 during Round 1.

By link reliability, $\forall p \in Correct$, p eventually receives $\langle 1, \Delta_q, q \rangle$ from every $q \in Correct$ in Round 1.

Since eventually $V \setminus \mathcal{S}_p \subseteq Correct$ forever, p completes Round 1.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
6: `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7: `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` *(/* Phase 2 */)*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Termination

Case 1: Line 7

By induction on r: $\forall p \in Correct$, p completes every Round $r \in [1..n-1]$.

$r = 1$: $\forall q \in Correct$, q completes Line 6 during Round 1.

By link reliability, $\forall p \in Correct$, p eventually receives $\langle 1, \Delta_q, q \rangle$ from every $q \in Correct$ in Round 1.

Since eventually $V \setminus \mathcal{S}_p \subseteq Correct$ forever, p completes Round 1.

$r > 1$: By induction hypothesis, $\forall q \in Correct$, q completes Round $r-1$, so q completes Line 6 during Round r.

By link reliability, $\forall p \in Correct$, p eventually receives $\langle r, \Delta_q, q \rangle$ from every $q \in Correct$ in Round r.

Since eventually $V \setminus \mathcal{S}_p \subseteq Correct$ forever, $\forall p \in Correct$, p completes Round r.

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
6:     `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` *(/* Phase 2 */)*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Termination

Case 2: Line 18

From the previous induction,
$\forall p \in \mathit{Correct}$, p completes Phase 1.

So, $\forall p \in \mathit{Correct}$, p completes Line 17.

By link reliability, $\forall p \in \mathit{Correct}$, p eventually receives $\langle V_q \rangle$ from every $q \in \mathit{Correct}$.

Since eventually $V \setminus \mathcal{S}_p \subseteq \mathit{Correct}$ forever, $\forall p \in \mathit{Correct}$, p completes Line18.

□

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
6:     `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` *(/* Phase 2 */)*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $\mathit{lastmsgs}_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in \mathit{lastmsgs}_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Notations

- By weak accuracy: there is a correct process c which is never suspected.
- Let Π_1 be the set of processes that terminate the $n-1$ rounds of Phase 1.
- Let Π_2 be the set of processes that terminate Phase 2.

  By definition, $\Pi_2 \subseteq \Pi_1$.
- We note $V_p \leq V_q$ IFF $\forall k$, either $V_p[k] = V_q[k]$ or $V_p[k] = \perp$.

Messages from c

Lemma 3

During Round r, with $1 \leq r \leq n-1$, every process of Π_1 receives (r, Δ_c, c) from c, *i.e.*, $(r, \Delta_c, c) \in R_p[c]$.

Proof. Since $p \in \Pi_1$, p terminates all rounds of Phase 1.

At each round, p waits and receives a $\langle r, \Delta_c, c\rangle$ message from c since $c \notin \mathcal{S}_p$ forever. $\square$

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** /* *Phase 1* */
  broadcast $\langle r_p, \Delta_p, p\rangle$ to V (p included)
  wait to receive $\langle r_p, \Delta_q, q\rangle$ from $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
broadcast $\langle V_p\rangle$ to V (p included) /* *Phase 2* */
wait to receive $\langle V_q\rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* *Phase 3* */

Learning Values

Lemma 4

$\forall p \in \Pi_1$, $V_c \leq V_p$ at the end of Phase 1.

Proof.

Assume that $V_c[q] \neq \perp$ at the end of Phase 1 for some process q.

By Lemma 1, $V_c[q] = v_q$.

Let $p \in \Pi_1$. We now show that $V_p[q] = v_q$ at the end of Phase 1.

The case $p = c$ is trivial. So, we now assume that $p \neq c$.

Let r be the first round where c receives v_q (if $c = q$, we let $r = 0$ and assume the end of Round 0 is Line 4)

$\Delta_c[q] = v_q$ at the end of Round r.

Two cases: $r < n-1$ and $r = n-1$

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *‌/* Phase 1 */*
6:     `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` *‌/* Phase 2 */*
18: `wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *‌/* Phase 3 */*

Learning Values

Case 1: $r < n-1$

During Round $r+1 \leq n-1$, c relays v_q by broadcasting $\langle r+1, \Delta_c, c\rangle$ with $\Delta_c[q] = v_q$.

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *Phase 1*
  `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`
  `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
`broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` *Phase 2*
`wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *Phase 3*

Learning Values

Case 1: $r < n-1$

During Round $r+1 \leq n-1$, c relays v_q by broadcasting $\langle r+1, \Delta_c, c\rangle$ with $\Delta_c[q] = v_q$.

By Lemma 3, p receives $\langle r+1, \Delta_c, c\rangle$ during Round $r+1$.

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
    broadcast $\langle r_p, \Delta_p, p\rangle$ to V (p included)
    wait to receive $\langle r_p, \Delta_q, q\rangle$ from $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
broadcast $\langle V_p \rangle$ to V (p included) *(/* Phase 2 */)*
wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Learning Values

Case 1: $r < n-1$

During Round $r+1 \leq n-1$, c relays v_q by broadcasting $\langle r+1, \Delta_c, c\rangle$ with $\Delta_c[q] = v_q$.

By Lemma 3, p receives $\langle r+1, \Delta_c, c\rangle$ during Round $r+1$.

From the code of the algorithm, $V_p[q] = v_q$ at the end of round $r+1$.

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *`/*` Phase 1 `*/`*
  broadcast $\langle r_p, \Delta_p, p\rangle$ to V (p included)
  wait to receive $\langle r_p, \Delta_q, q\rangle$ from $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
broadcast $\langle V_p\rangle$ to V (p included) *`/*` Phase 2 `*/`*
wait to receive $\langle V_q\rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *`/*` Phase 3 `*/`*

Learning Values

Case 2: $r = n-1$

Since c receives v_q for the first time during Round $n-1$ and every process relays v_q at most once:

v_q has been relayed by $n-1$ distinct different from c before reaching c.

p necessarily belongs to this set of processes.

$\square$

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
  broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
  wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
broadcast $\langle V_p \rangle$ to V (p included) *(/* Phase 2 */)*
wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Learning Values

Case 2: $r = n - 1$

Since c receives v_q for the first time during Round $n-1$ and every process relays v_q at most once:

> v_q has been relayed by $n-1$ distinct different from c before reaching c.

p necessarily belongs to this set of processes.

Now, $V_p[q] = v_q$ right before relaying v_q, so $V_p[q] = v_q$ at the end of Phase 1. □

```
d_p ← ⊥
V_p ← [⊥,...,⊥]
V_p[p] ← v_p
Δ_p ← V_p
For all r_p from 1 to n−1 do                                   /* Phase 1 */
    broadcast ⟨r_p, Δ_p, p⟩ to V (p included)
    wait to receive ⟨r_p, Δ_q, q⟩ from q ∈ V \ S_p
    let R_p[r_p] be the set of received ⟨r_p, Δ_q, q⟩
    Δ_p ← [⊥,...,⊥]
    For all k ∈ V do
        If V_p[k] = ⊥ ∧ (∃(r_p, Δ_q, q) ∈ R_p[r_p], Δ_q[k] ≠ ⊥) then
            V_p[k] ← Δ_q[k]
            Δ_p[k] ← Δ_q[k]
        End If
    Done
Done
broadcast ⟨V_p⟩ to V (p included)                               /* Phase 2 */
wait to receive ⟨V_q⟩ from q ∈ V \ S_p
Let lastmsgs_p be the set of received ⟨V_q⟩
For all k ∈ V do
    If ∃V_q ∈ lastmsgs_p, V_q[k] = ⊥ then V_p[k] ← ⊥
Done
d_p ← x where x is the first non-⊥ value in V_p                 /* Phase 3 */
```

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $V_c = V_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $V_p[q] = V_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $V_c[q] = v_q$ at the end of Phase 1.
- $V_c[q] = \perp$ at the end of Phase 1.

□

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** /* Phase 1 */
    `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`
    `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
`broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` /* Phase 2 */
`wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $V_c = V_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $V_p[q] = V_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $V_c[q] = v_q$ at the end of Phase 1.

  By Lemma 4, $\forall p' \in \Pi_1$ (p and c included), $V_{p'}[q] = v_q$ at the end of Phase 1.

  Thus, for all vectors V sent during Phase 2, we have $V[q] = v_q$.

  Hence, $V_p[q]$ and $V_c[q]$ remain equal to v_q during Phase 2.

- $V_c[q] = \perp$ at the end of Phase 1.

$\square$

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
    broadcast $\langle r_p, \Delta_p, p\rangle$ to V (p included)
    wait to receive $\langle r_p, \Delta_q, q\rangle$ from $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
broadcast $\langle V_p\rangle$ to V (p included) *(/* Phase 2 */)*
wait to receive $\langle V_q\rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Decision (1/3)

Lemma 5

$\forall p \in \Pi_2$, $V_c = V_p$ at the end of Phase 2.

Proof. Let $p \in \Pi_2$. Let q be a process. We should show that $V_p[q] = V_c[q]$ at the end of Phase 2.

By Lemma 1, we have the following two cases:

- $V_c[q] = v_q$ at the end of Phase 1.

  By Lemma 4, $\forall p' \in \Pi_1$ (p and c included), $V_{p'}[q] = v_q$ at the end of Phase 1.

  Thus, for all vectors V sent during Phase 2, we have $V[q] = v_q$.

  Hence, $V_p[q]$ and $V_c[q]$ remain equal to v_q during Phase 2.

- $V_c[q] = \perp$ at the end of Phase 1.

  Since $c \notin \mathcal{S}_p$ forever, p waits and receives V_c during Phase 2.

  Since $V_c[q] = \perp$, p sets $V_p[q]$ to $\perp$ during Phase 2.

□

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** */* Phase 1 */*
    `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
    `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
`broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` */* Phase 2 */*
`wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
Let *lastmsgs*$_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p */* Phase 3 */*

Decision (2/3)

Lemma 6

$\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2.

Proof.

□

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *$/*$ Phase 1 $*/$*
    broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
    wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
broadcast $\langle V_p \rangle$ to V (p included) *$/*$ Phase 2 $*/$*
wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *$/*$ Phase 3 $*/$*

Decision (2/3)

Lemma 6

$\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $V_c[c] = v_c$ at the end of Phase 1. ☐

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *[/* Phase 1 */]*
6: `broadcast` $\langle r_p, \Delta_p, p\rangle$ `to` V `(`p `included)`
7: `wait to receive` $\langle r_p, \Delta_q, q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: `broadcast` $\langle V_p\rangle$ `to` V `(`p `included)` */* Phase 2 */*
18: `wait to receive` $\langle V_q\rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p */* Phase 3 */*

Decision (2/3)

Lemma 6

$\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $V_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1$, $V_q[c] = v_c$ at the end of Phase 1.

□

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** /* Phase 1 */
6: broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
7: wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
8: let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9: $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10: **For all** $k \in V$ **do**
11: **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12: $V_p[k] \leftarrow \Delta_q[k]$
13: $\Delta_p[k] \leftarrow \Delta_q[k]$
14: **End If**
15: **Done**
16: **Done**
17: broadcast $\langle V_p \rangle$ to V (p included) /* Phase 2 */
18: wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21: **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Decision (2/3)

Lemma 6

$\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $V_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1$, $V_q[c] = v_c$ at the end of Phase 1.

Thus, no process sends a vector V such that $V[c] = \perp$ during Phase 2.

□

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** — */* Phase 1 */*
6:     `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` — */* Phase 2 */*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p — */* Phase 3 */*

Decision (2/3)

Lemma 6

$\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2.

Proof. From the code of the algorithm (Line 3), we know that $V_c[c] = v_c$ at the end of Phase 1.

By Lemma 4, $\forall q \in \Pi_1$, $V_q[c] = v_c$ at the end of Phase 1.

Thus, no process sends a vector V such that $V[c] = \perp$ during Phase 2.

Hence, from the code of the algorithm, we can deduce that $\forall p \in \Pi_2$, $V_p[c] = v_c$ at the end of Phase 2. □

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *(/* Phase 1 */)*
6:     broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
7:     wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: broadcast $\langle V_p \rangle$ to V (p included) *(/* Phase 2 */)*
18: wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *(/* Phase 3 */)*

Decision (3/3)

Corollary 1

No two processes decide differently.

Proof. By Lemma 6, all processes that execute Phase 3 take a well-defined decision.

By Lemma 5, all processes that execute Phase 3 have the same vector.

So, they take the same decision. □

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** *`/* Phase 1 */`*
  broadcast $\langle r_p, \Delta_p, p \rangle$ to V (p included)
  wait to receive $\langle r_p, \Delta_q, q \rangle$ from $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
broadcast $\langle V_p \rangle$ to V (p included) *`/* Phase 2 */`*
wait to receive $\langle V_q \rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *`/* Phase 3 */`*

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector $\mathcal{S}$.

Proof.

$\square$

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do**   */* Phase 1 */*
    broadcast $\langle r_p, \Delta_p, p\rangle$ to V (p included)
    wait to receive $\langle r_p, \Delta_q, q\rangle$ from $q \in V \setminus \mathcal{S}_p$
    let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q\rangle$
    $\Delta_p \leftarrow [\perp, \ldots, \perp]$
    For all $k \in V$ **do**
        If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
            $V_p[k] \leftarrow \Delta_q[k]$
            $\Delta_p[k] \leftarrow \Delta_q[k]$
        End If
    Done
Done
broadcast $\langle V_p\rangle$ to V (p included)   */* Phase 2 */*
wait to receive $\langle V_q\rangle$ from $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q\rangle$
For all $k \in V$ **do**
    If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p */* Phase 3 */*

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector $\mathcal{S}$.

Proof.

- Integrity: from the code of the algorithm, Phase 3 is executed only once.

□

$d_p \leftarrow \perp$
$V_p \leftarrow [\perp, \ldots, \perp]$
$V_p[p] \leftarrow v_p$
$\Delta_p \leftarrow V_p$
For all r_p from 1 to $n-1$ **do** /* Phase 1 */
  `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
  `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
  let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
  $\Delta_p \leftarrow [\perp, \ldots, \perp]$
  For all $k \in V$ **do**
    If $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
      $V_p[k] \leftarrow \Delta_q[k]$
      $\Delta_p[k] \leftarrow \Delta_q[k]$
    End If
  Done
Done
`broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` /* Phase 2 */
`wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
For all $k \in V$ **do**
  If $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
Done
$d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p /* Phase 3 */

Result

Theorem 1

The algorithm of Chandra and Toueg solves the consensus in an asynchronous system enriched with a failure detector $\mathcal{S}$.

Proof.

- Integrity: from the code of the algorithm, Phase 3 is executed only once.
- Termination: Lemma 2.
- Agreement: Corollary 1.
- Validity: Lemma 1 and Line 23.

□

1: $d_p \leftarrow \perp$
2: $V_p \leftarrow [\perp, \ldots, \perp]$
3: $V_p[p] \leftarrow v_p$
4: $\Delta_p \leftarrow V_p$
5: **For all** r_p from 1 to $n-1$ **do** *`/* Phase 1 */`*
6:     `broadcast` $\langle r_p, \Delta_p, p \rangle$ `to` V `(`p `included)`
7:     `wait to receive` $\langle r_p, \Delta_q, q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
8:     let $R_p[r_p]$ be the set of received $\langle r_p, \Delta_q, q \rangle$
9:     $\Delta_p \leftarrow [\perp, \ldots, \perp]$
10:     **For all** $k \in V$ **do**
11:         **If** $V_p[k] = \perp \wedge (\exists (r_p, \Delta_q, q) \in R_p[r_p], \Delta_q[k] \neq \perp)$ **then**
12:             $V_p[k] \leftarrow \Delta_q[k]$
13:             $\Delta_p[k] \leftarrow \Delta_q[k]$
14:         **End If**
15:     **Done**
16: **Done**
17: `broadcast` $\langle V_p \rangle$ `to` V `(`p `included)` *`/* Phase 2 */`*
18: `wait to receive` $\langle V_q \rangle$ `from` $q \in V \setminus \mathcal{S}_p$
19: Let $lastmsgs_p$ be the set of received $\langle V_q \rangle$
20: **For all** $k \in V$ **do**
21:     **If** $\exists V_q \in lastmsgs_p, V_q[k] = \perp$ **then** $V_p[k] \leftarrow \perp$
22: **Done**
23: $d_p \leftarrow x$ where x is the first non-$\perp$ value in V_p *`/* Phase 3 */`*

Roadmap

A Simple Example: $\diamond\mathcal{P}$

Every failure detector of the class $\diamond\mathcal{P}$ satisfies both strong completeness and eventual strong accuracy:

Strong Completeness: Every faulty process is eventually permanently suspected by every correct process.

Eventual strong accuracy: there is a time after which no correct process is suspected by any correct process.

System Assumptions

Motivation

Given a failure detector of class $\diamond\mathcal{P}$, consensus can be solved in an asynchronous crash-prone system with reliable links where a majority of processes is correct.[7]

[7] In these settings, consensus can be even solved with a weaker failure detector (Ω). This fact will be established in the next lesson.

System Assumptions

Motivation

Given a failure detector of class $\diamond\mathcal{P}$, consensus can be solved in an asynchronous crash-prone system with reliable links where a majority of processes is correct.[7]

The FLP implies that partial synchrony assumptions are required to implement such a failure detector!

[7] In these settings, consensus can be even solved with a weaker failure detector (Ω). This fact will be established in the next lesson.

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology
- Process failures: only crashes!

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (1/2)

- Complete Network Topology
- Process failures: only crashes!
- Correct processes are eventually synchronous: for every correct process p, there exists a time t_p (**a priori unknown** by all processes) from which p executes each of its instructions in a time belonging to $[\alpha_p..\beta_p]$ with $0 < \alpha_p \leq \beta_p$.

  α_p and β_p are **a priori unknown**, for every process p.

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (2/2)

- There exists at least one eventual bi-source $\diamond b$ (**a priori unknown** by processes): all its outgoing and incoming links are eventually reliable and synchronous, *i.e.*,

  there exists a time $t_{\diamond b}$ from which every message sent to or from $\diamond b$ is delivered within at most $\delta_{\diamond b}$ time units.

  $\diamond b$, $t_{\diamond b}$, and $\delta_{\diamond b}$ are **a priori unknown** (even by $\diamond b$!)

System Assumptions

The Partially Synchronous System $S_{\diamond b}$ (2/2)

- There exists at least one eventual bi-source $\diamond b$ (**a priori unknown** by processes): all its outgoing and incoming links are eventually reliable and synchronous, *i.e.*,

  there exists a time $t_{\diamond b}$ from which every message sent to or from $\diamond b$ is delivered within at most $\delta_{\diamond b}$ time units.

  $\diamond b$, $t_{\diamond b}$, and $\delta_{\diamond b}$ are **a priori unknown** (even by $\diamond b$!)

Every other link is arbitrary slow and lossy.

Recall that every message is delivered or lost within finite time.

Principles

1. Each process regularly broadcasts ALIVE messages tagged with its ID

   Each message is relayed once.

   This way, $\diamond b$ acts as a hub: eventually messages tagged with IDs of correct processes are delivered within bounded time.

Principles

1. Each process regularly broadcasts ALIVE messages tagged with its ID

   Each message is relayed once.

   This way, $\diamond b$ acts as a hub: eventually messages tagged with IDs of correct processes are delivered within bounded time.

2. Each process maintains a timer for each other process.

   On Time Out: the watched process is suspected.

   If later, the process receives a message tagged with the ID of some suspected process, it stops suspecting it and increases the waiting time of the associated timer.

Constants & Variables

- Processes are identified: a process and its identifier are used equivalently
- *ALIVE*: message type
- V: set of processes
- $k \in \mathbb{N}^*$: constant
- $Timer_p[]$, $ElapseTime_p[]$: arrays of integer indexed on the process IDs, except p.
- $Alive_p$, $Suspected_p$: sets of identifiers

  ($Suspected_p$ is the algorithm output)

Algorithm *EP*

Algorithm *EP* for each process p, output: $Suspected_p$

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
broadcast $\langle ALIVE, p, p \rangle$ to $V \setminus \{p\}$
For all $q \in V \setminus \{p\}$ **do**
If receive $\langle ALIVE, r, q \rangle$ **then**
If $r \neq p$ **then**
If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
$ElapseTime_p[r] \leftarrow Timer_p[r]$
$Alive_p \leftarrow Alive_p \cup \{r\}$
If $r = q$ **then** broadcast $\langle ALIVE, r, p \rangle$ to $V \setminus \{p\}$
End If
End If
Done
For all $q \in V \setminus \{p\}$ **do**
If $ElapseTime_p[q] = 0$ **then**
$Alive_p \leftarrow Alive_p \setminus \{q\}$
else
$ElapseTime_p[q]--$
End If
Done
$Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard “_” designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, \_\rangle$ where q is a faulty process.

Proof.

□

```
Alive_p ← V; Suspected_p ← ∅
Timer_p ← [k, ..., k]; ElapseTime_p ← [k, ..., k]
While true do
    broadcast ⟨ALIVE, p, p⟩ to V \ {p}
    For all q ∈ V \ {p} do
        If receive ⟨ALIVE, r, q⟩ then
            If r ≠ p then
                If ElapseTime_p[r] ≤ 0 then Timer_p[r]++
                ElapseTime_p[r] ← Timer_p[r]
                Alive_p ← Alive_p ∪ {r}
                If r = q then broadcast⟨ALIVE, r, p⟩ to V \ {p}
            End If
        End If
    Done
    For all q ∈ V \ {p} do
        If ElapseTime_p[q] = 0 then
            Alive_p ← Alive_p \ {q}
        else
            ElapseTime_p[q] −−
        End If
    Done
    Suspected_p ← V \ Alive_p
Done
```

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard "_" designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, \_\rangle$ where q is a faulty process.

Proof. q broadcasts finitely many $\langle ALIVE, q, q\rangle$ messages before crashing. □

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \dots, k]$; $ElapseTime_p \leftarrow [k, \dots, k]$
While *true* **do**
  `broadcast` $\langle ALIVE, p, p\rangle$ `to` $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If `receive` $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** `broadcast`$\langle ALIVE, r, p\rangle$ `to` $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q] --$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard "_" designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, \_\rangle$ where q is a faulty process.

Proof. q broadcasts finitely many $\langle ALIVE, q, q\rangle$ messages before crashing.

Now, each $\langle ALIVE, q, q\rangle$ message is relayed at most once by every other processes. □

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
    broadcast $\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
    For all $q \in V \setminus \{p\}$ **do**
        If receive $\langle ALIVE, r, q\rangle$ **then**
            If $r \neq p$ **then**
                If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
                $ElapseTime_p[r] \leftarrow Timer_p[r]$
                $Alive_p \leftarrow Alive_p \cup \{r\}$
                If $r = q$ **then** broadcast $\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
            End If
        End If
    Done
    For all $q \in V \setminus \{p\}$ **do**
        If $ElapseTime_p[q] = 0$ **then**
            $Alive_p \leftarrow Alive_p \setminus \{q\}$
        else
            $ElapseTime_p[q]--$
        End If
    Done
    $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (1/3)

Recall that the wildcard "_" designates any value.

Lemma 1

Each correct process p eventually no more receives $\langle ALIVE, q, \_\rangle$ where q is a faulty process.

Proof. q broadcasts finitely many $\langle ALIVE, q, q\rangle$ messages before crashing.

Now, each $\langle ALIVE, q, q\rangle$ message is relayed at most once by every other processes.

As every sent message is eventually either received or lost, the lemma holds. □

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \dots, k]$; $ElapseTime_p \leftarrow [k, \dots, k]$
While *true* **do**
  `broadcast` $\langle ALIVE, p, p\rangle$ `to` $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If `receive` $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** `broadcast` $\langle ALIVE, r, p\rangle$ `to` $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]--$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p, each iteration of the "while" loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.
By definition, $\exists$ a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

$\square$

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p, each iteration of the "while" loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.

By definition, $\exists$ a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

There is a finite number of instructions before the "while" loop. So, in the worst case, the "while" loop begins within bounded time t_p after t.

□

Eventual Suspicion of Faulty Processes (2/3)

Lemma 2

Let p be a correct process. $\exists t_p, K_p, K'_p \in \mathbb{N}$ with $K_p \geq K'_p$ such that after time t_p, each iteration of the "while" loop by p lasts a time belonging to $[K'_p, K_p]$.

Proof.

By definition, $\exists$ a time t from which each instruction is executed by p within a time that is both lower and upper bounded since p is eventually synchronous.

There is a finite number of instructions before the "while" loop. So, in the worst case, the "while" loop begins within bounded time t_p after t.

Similarly, after t_p, $\exists K_p, K'_p \in \mathbb{N}$ with $K'_p \leq K_p$ such that each iteration of the "while" loop by p lasts a time belonging to $[K'_p, K_p]$ since the "while" loop contains a bounded number of instructions and the time to execute any instruction is both lower and upper bounded.

□

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by every correct process.

Proof. Let q be a faulty process and p be a correct process. □

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \dots, k]$; $ElapseTime_p \leftarrow [k, \dots, k]$
While *true* **do**
  `broadcast` $\langle ALIVE, p, p \rangle$ `to` $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If `receive` $\langle ALIVE, r, q \rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** `broadcast` $\langle ALIVE, r, p \rangle$ `to` $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]--$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by every correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a "while" loop iteration within at most K_p time units.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]{+}{+}$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]{-}{-}$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by every correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a "while" loop iteration within at most K_p time units.

So, from time t_p, p eventually satisfies $ElapseTime_p[q] = 0$ forever, by Lemma 1.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]{+}{+}$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]{-}{-}$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Eventual Suspicion of Faulty Processes (3/3)

Corollary 1

Every faulty process is eventually forever suspected by every correct process.

Proof. Let q be a faulty process and p be a correct process.

By Lemma 2, there exists a time t_p from which p executes a "while" loop iteration within at most K_p time units.

So, from time t_p, p eventually satisfies $ElapseTime_p[q] = 0$ forever, by Lemma 1.

Hence, q is eventually removed from $Alive_p$ forever and so eventually belongs forever to $Suspected_p$. □

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  `broadcast` $\langle ALIVE, p, p\rangle$ `to` $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If `receive` $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** `broadcast` $\langle ALIVE, r, p\rangle$ `to` $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]--$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Unsuspicion of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p, p receives $\langle ALIVE, q, \_\rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

Unsuspicion of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p, p receives $\langle ALIVE, q, \_ \rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

By Lemma 2 and by definition of eventual bi-source, there exists $t \in \mathbb{N}$ such that after time t,

- q, $\diamond b$, and p respectively execute each "while" loop iteration within at most K_q, $K_{\diamond b}$, and K_p time units.
- Moreover, every message sent from or to $\diamond b$ is delivered in at most $\delta_{\diamond b}$ time units.

Unsuspicion of Correct Processes (1/4)

Lemma 3

Let p and q be two correct processes such that $p \neq q$. There exists $t_p, \Delta_p \in \mathbb{N}$ such that after time t_p, p receives $\langle ALIVE, q, \_ \rangle$ at least every Δ_p time units.

Proof. Let $\diamond b$ be an eventual bi-source.

By Lemma 2 and by definition of eventual bi-source, there exists $t \in \mathbb{N}$ such that after time t,

- q, $\diamond b$, and p respectively execute each "while" loop iteration within at most K_q, $K_{\diamond b}$, and K_p time units.
- Moreover, every message sent from or to $\diamond b$ is delivered in at most $\delta_{\diamond b}$ time units.

Two cases: $q = \diamond b$ or $q \neq \diamond b$.

Unsuspicion of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t+K_q]$ such that q starts a "while" loop iteration at time t'. From t', q executes a (full) "while" loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

Unsuspicion of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t+K_q]$ such that q starts a “while” loop iteration at time t'. From t', q executes a (full) ”while” loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to p within at most $\delta_{\diamond b}$ time units after their sending.

Unsuspicion of Correct Processes (2/4)

Case $q = \diamond b$

$\exists t' \in [t..t + K_q]$ such that q starts a “while” loop iteration at time t'. From t', q executes a (full) ”while” loop iteration at least every K_q time units. So, q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to p within at most $\delta_{\diamond b}$ time units after their sending.

Now, p executes a full ”while” loop iteration at least every $2.K_p$ time units.

Hence, with $\Delta_p = K_q + \delta_{\diamond b} + 2.K_p$, the lemma holds in this case.

Unsuspicion of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q \rangle$ at least every K_q time units from t'.

$\square$

Unsuspicion of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t+K_q]$ such that q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\square$

Unsuspicion of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t + K_q]$ such that q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b\rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

□

Unsuspicion of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t+K_q]$ such that q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b\rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

$\langle ALIVE, q, \_\rangle$ messages are delivered to p at least $K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b}$ time units.

$\square$

Unsuspicion of Correct Processes (3/4)

Case $q \neq \diamond b$

Similarly to the previous case: $\exists t' \in [t..t+K_q]$ such that q broadcasts $\langle ALIVE, q, q\rangle$ at least every K_q time units from t'.

These messages are delivered to $\diamond b$ within at most $\delta_{\diamond b}$ time units after their sending; so at least every $K_q + \delta_{\diamond b}$ time units.

$\diamond b$ sends $\langle ALIVE, q, \diamond b\rangle$ to p at least every $K_q + \delta_{\diamond b} + 2.K_{\diamond b}$ time units.

$\langle ALIVE, q, \_\rangle$ messages are delivered to p at least $K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b}$ time units.

Since p executes a full "while" loop iteration at least every $2.K_p$ time units, by letting $\Delta_p = K_q + 2.\delta_{\diamond b} + 2.K_{\diamond b} + 2.K_p$, the lemma holds in this case. $\square$

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.
Let p and q be two correct processes.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]--$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.
Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.

□

```
Alive_p ← V; Suspected_p ← ∅
Timer_p ← [k,...,k]; ElapseTime_p ← [k,...,k]
While true do
   broadcast⟨ALIVE,p,p⟩ to V \ {p}
   For all q ∈ V \ {p} do
      If receive ⟨ALIVE,r,q⟩ then
         If r ≠ p then
            If ElapseTime_p[r] ≤ 0 then Timer_p[r]++
            ElapseTime_p[r] ← Timer_p[r]
            Alive_p ← Alive_p ∪ {r}
            If r = q then broadcast⟨ALIVE,r,p⟩ to V \ {p}
         End If
      End If
   Done
   For all q ∈ V \ {p} do
      If ElapseTime_p[q] = 0 then
         Alive_p ← Alive_p \ {q}
      else
         ElapseTime_p[q]−−
      End If
   Done
   Suspected_p ← V \ Alive_p
Done
```

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.
Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]++$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]--$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.
Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

  Assume, by contradiction, that q is suspected infinitely often by p.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]{+}{+}$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]{-}{-}$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.

Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

  Assume, by contradiction, that q is suspected infinitely often by p.

  Between every removal and insertion of q, $Timer_p[q]$ is incremented.

□

```
Alive_p ← V; Suspected_p ← ∅
Timer_p ← [k, ..., k]; ElapseTime_p ← [k, ..., k]
While true do
    broadcast⟨ALIVE, p, p⟩ to V \ {p}
    For all q ∈ V \ {p} do
        If receive ⟨ALIVE, r, q⟩ then
            If r ≠ p then
                If ElapseTime_p[r] ≤ 0 then Timer_p[r]++
                ElapseTime_p[r] ← Timer_p[r]
                Alive_p ← Alive_p ∪ {r}
                If r = q then broadcast⟨ALIVE, r, p⟩ to V \ {p}
            End If
        End If
    Done
    For all q ∈ V \ {p} do
        If ElapseTime_p[q] = 0 then
            Alive_p ← Alive_p \ {q}
        else
            ElapseTime_p[q]−−
        End If
    Done
    Suspected_p ← V \ Alive_p
Done
```

Unsuspicion of Correct Processes (4/4)

Corollary 2

There exists a time from which no correct process is suspected by any correct process.

Proof.
Let p and q be two correct processes.

- If $p = q$, q is never removed from $Alive_p$ and so never inserted into $Suspected_p$. Hence, the lemma holds in this case.
- Otherwise, q is regularly inserted into $Alive_p$, by Lemma 3.

  Assume, by contradiction, that q is suspected infinitely often by p.

  Between every removal and insertion of q, $Timer_p[q]$ is incremented.

  So, the time between two consecutive receptions of $\langle ALIVE, q, \_\rangle$ by p regularly increases by Lemma 2, a contradiction to Lemma 3.

□

$Alive_p \leftarrow V$; $Suspected_p \leftarrow \emptyset$
$Timer_p \leftarrow [k, \ldots, k]$; $ElapseTime_p \leftarrow [k, \ldots, k]$
While *true* **do**
  broadcast$\langle ALIVE, p, p\rangle$ to $V \setminus \{p\}$
  For all $q \in V \setminus \{p\}$ **do**
    If receive $\langle ALIVE, r, q\rangle$ **then**
      If $r \neq p$ **then**
        If $ElapseTime_p[r] \leq 0$ **then** $Timer_p[r]{+}{+}$
        $ElapseTime_p[r] \leftarrow Timer_p[r]$
        $Alive_p \leftarrow Alive_p \cup \{r\}$
        If $r = q$ **then** broadcast$\langle ALIVE, r, p\rangle$ to $V \setminus \{p\}$
      End If
    End If
  Done
  For all $q \in V \setminus \{p\}$ **do**
    If $ElapseTime_p[q] = 0$ **then**
      $Alive_p \leftarrow Alive_p \setminus \{q\}$
    else
      $ElapseTime_p[q]{-}{-}$
    End If
  Done
  $Suspected_p \leftarrow V \setminus Alive_p$
Done

Correctness of the Algorithm

By Corollaries 1 and 2, follows.

Theorem 1

Algorithm *EP* is a failure detector of type $\diamond\mathcal{P}$ in System $\mathcal{S}_{\diamond b}$.

Roadmap

References

[1] 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.

[2] T. D. Chandra and S. Toueg.
Unreliable failure detectors for reliable distributed systems.
J. ACM, 43(2):225–267, 1996.

[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.