Modern Distributed Computing

Theory and Applications

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 4
Tuesday, March 26, 2013

Initial Assumptions

- Exercises correspond to problems studied during the course.
- We attempt to formulate a distributed computing solution.
- The formulation is done in an abstract way.
- A problem may be differentiated from its original version if the initial assumptions are modified:
    - either making it easier, or more difficult,
    - or totally different.
- Our first step is to identify and understand all the initial assumptions stated by the problem.
- We need to understand why an initial assumption is made (or not).

Network Topology

- The network is abstracted using a communication graph
    - Vertices correspond to processes,
    - Edges correspond to communication channels.
- What are the assumptions about the communication graph ?
- Is it a special category graph (e.g. symmetric) or is it a generic one ?
- Is it directed or undirected ?
- Is it fully connected ?
- Do we know the total number of vertices / total number of edges ?
- We need to understand the nature of the graph – our solution may be totally wrong.

Initial Input

- What is available to each process from system init ?
    - Topology
    - Diameter
    - Total number of processes
- Processes have (or not) unique identities
- A leader is available from system init (e.g., u_0)
- An input value is given to each process
    - integer i_u
    - a value from a given set S
- We need to understand why each peace of information is provided to us (or not).

Problem to solve

- Usually exercises are related to the design of a new algorithm
  - Other problems are related to proving an impossibility result
  - or identifying best-case/worst-case input scenaria
- What is the problem at hand ?
  - What is the system goal ?
  - Do we need all processes to acquire some specific knowledge ?
- Does it fit to one of the problems studied ?
  - How does the initial input differentiate the problem ?
  - Do we need to employ an additional initial step ?
  - Does a known solution require a different topology than the one we have at hand ?
- Does the algorithm need to terminate ?

Methodology: Understand the question

- Identify & Understand assumptions
- Communication Model, Failures and Topology
- Initial input
- Understand the problem statement
- Identify similar problems/solutions in the bibliography

Methodology: Initial solution

- Do we have a rough idea of a solution ?
- Do we have identified an approach to solving the problem ?
  - think again !
  - go through the assumptions – maybe we overlooked something ?
- Write down a solution sketch
  - check if it adheres to the initial assumptions
  - does it use all the available input ?
- Is the solution correct ? can we provide some arguments ?
- What is the complexity (time, communication) ?
- What is the achieved robustness ?
- Can we think of a more efficient solution ?

Write-down the solution

- Pedantic definition of process variables
  - state the purpose – scope of use
  - type of variable
  - initial value
- Pedantic definition of message exchanged between processes
  - state the purpose – scope of use
  - contents (list of variables: types + initial values)
- Initialization phase
  - state if a particular "virtual" topology is needed.
  - execute a sub-algorithm (for acquiring specific knowledge on the net)
  - initialization of variables
- Basic round of execution
- Special cases
- Termination

Final document

1. Short description
2. Description of each process
   - variables
   - message types
   - initialization
3. Required sub-algorithms
4. Basic algorithm – description of execution
   - "simple" / "typical" round of execution
   - special cases
5. Pseudocode (maybe for specific parts)
6. Correctness – Some arguments . . . full proof
7. Time Complexity – Some arguments . . . full proof
8. Message Complexity – Some arguments . . . full proof

Part 2: Failures

1. Link failures, Node failures, Impossibility Results
2. Agreement
3. Byzantine Failures
4. Failures in Asynchronous Systems

Consensus Problem

In a synchronous network G, each process begins with an arbitrary initial value of a particular type. We require all processes to reach consensus, that is, output the same value and terminate.

- There is a validity condition describing the output values that are permitted for each pattern of inputs.
- When there are no failures of system components,
  - consensus problems are usually easy to solve,
  - using a simple exchange of messages.
- Consensus problems arise in many distributed computing applications.

The Distributed Consensus Problem

We assume n processes, connected by a synchronous, undirected graph where each process has a unique ID.
Each process u receives an input value i_u from the set S, that is $i_u \in S$.
An algorithm solves the problem of distributed consensus if it adheres to the following specifications:

1. Agreement: No pair of processes agrees on different output values, that is, $\nexists u, v : o_u \neq o_v$
2. Validity: If all processes start with the same value $i \in S$, i.e., $\forall u \in [1, n] : i_u = i$, then value i is the only possible decision value, that is $\forall u \in [1, n] : o_u = i$
3. Termination: All processes eventually decide.

SimpleConsensus Algorithm

Each process $u \in [1, n]$ maintains a list l_u with pairs of IDs and input values. Initially the list contains only one set: the ID of u and the input value $i_u \in S$. In each round, all processes transmit the list l to their local neighborhood. When they receive list l_v from a neighbor v, they merge it with their internal list. After $\delta + 1$ rounds, all processes maintain a list containing a pair (u, i_u) for each other process of the system. Then they apply a predefined consensus rule and terminate by outputting the common value $o \in S$.

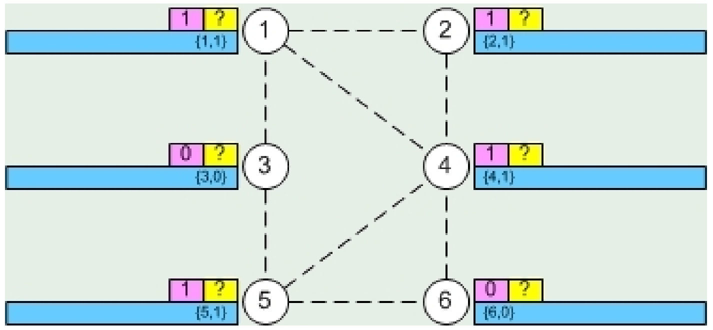
- Each process knows δ.
- The algorithm solves the consensus problem.
- The consensus rule can be: minimum value, average value, majority ...

Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

General Graph

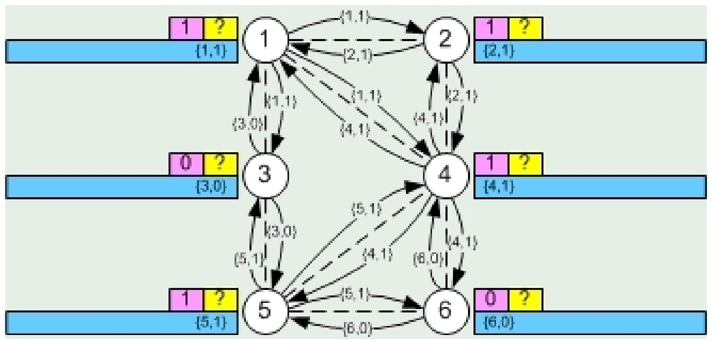


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

1st Round – message transmission

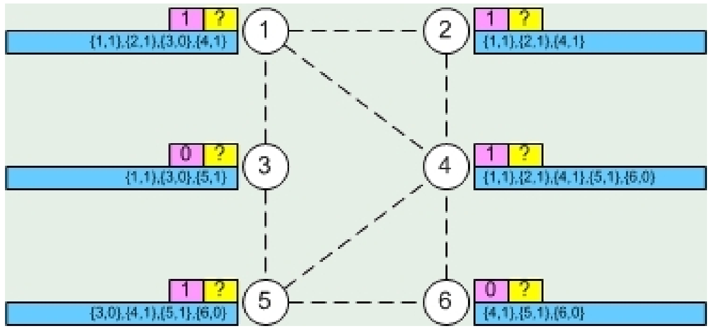


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

1st Round – processing

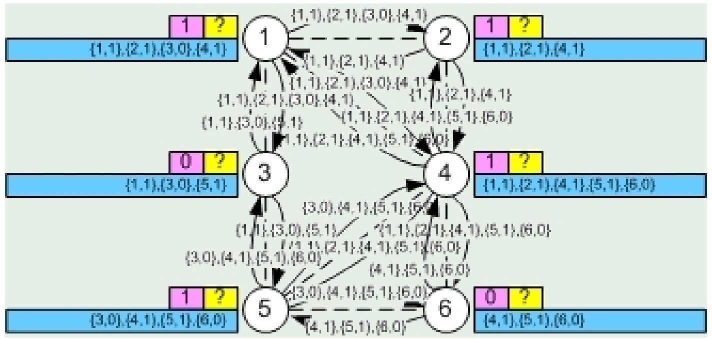


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

2nd Round – message transmission

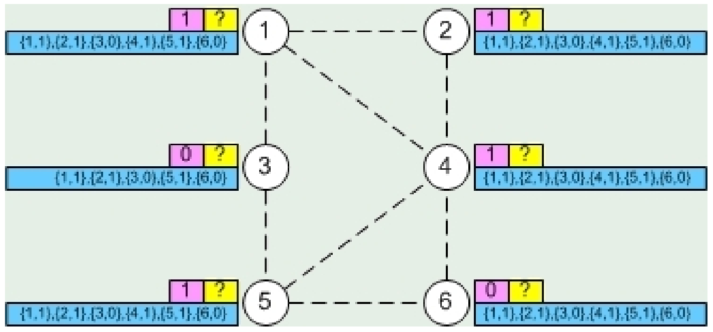


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

2nd Round – processing

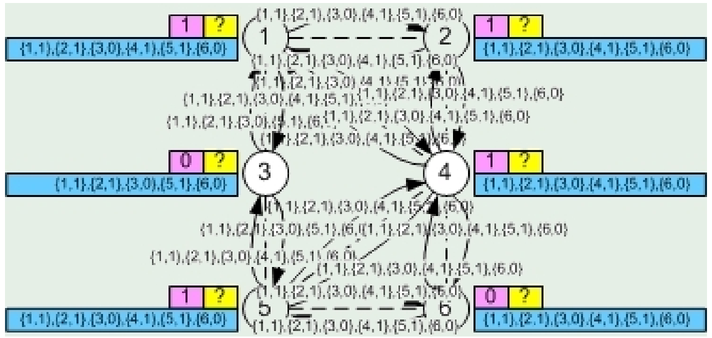


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round – message transmission

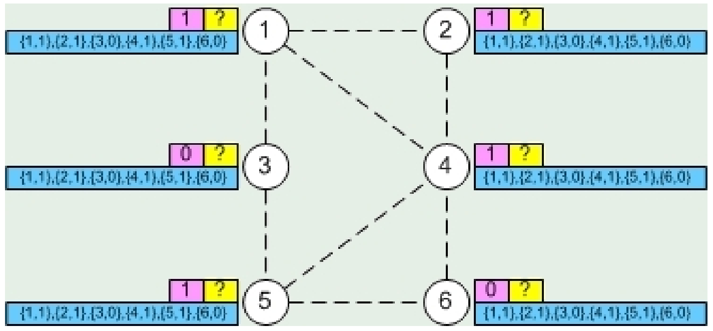


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round – processing

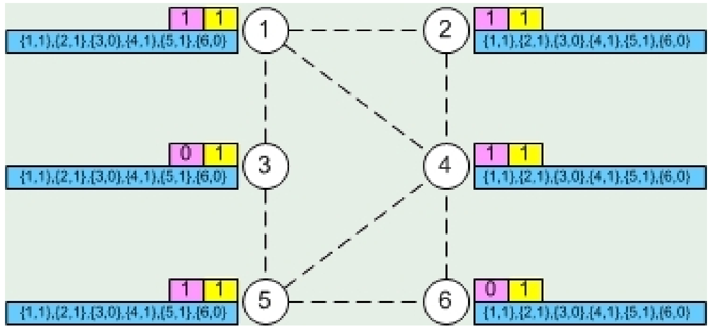


Example of execution of SimpleConsensus

Let a synchronous network of $n = 6$ processes and $\delta = 2$.

- Consensus rule: simple majority

3rd Round– decision



Properties of SimpleConsensus Algorithm

Let a synchronous network G with n processes and m channels

- At the end of round δ each process $u \in [1, n]$ will maintain a list $l_u = \{(1, i_1), (2, i_2), \ldots, (n, i_n)\}$
- The lists maintained by all processes are identical, i.e., $\forall u \in [1, n] : l_u = l$
- The time complexity is $\mathcal{O}(diam(G))$
- The message complexity is $\mathcal{O}(diam(G) \cdot m)$
- The message complexity in bits is $\mathcal{O}(diam(G) \cdot n \cdot m)$

Considerations

How will the execution evolve if failures occur during the transmission of messages ?

Given the presence of failures,

- can we guarantee the correctness of SimpleConsensus ?
- can we identify failure ?
- can we prevent/deal with failure ?

We look into the case when

- during the execution of a distributed algorithm,
- communication failures during the transmission of messages.

Link Failure

The communication network interconnecting the processing units of a distributed system may fail during the transmission of any message over a (faulty) channel. The delivery of messages is not guaranteed. We assume a number of the messages transmitted during the execution of the system will not be delivered successfully.

Coordinated Attack Problem

Two generals are planning a coordinated attack from different directions, against a common objective. They know that they only way the attack can be successful is if both generals attack; if only one attacks, their armies will be destroyed. Each general has an initial opinion about whether his army is ready to attack. Generals need to agree on a common decision by communicating via messengers that travel on foot. However, messengers can be lost or captures, and their messages may thus be lost.

- Distributed Consensus problem with a simple system of $n = 2$ processes.
- Possible input/output values are "yes" or "no" – i.e., $S = \{\text{"yes"}, \text{"no"}\}$

We assume that the two generals decide to attack when the following conditions are met:

1. Agreement: The generals u, v decided on a common opinion, that is $o_u = o_v$
2. Validity:
   - If the initial opinion of both generals is "no", then the only valid common decision is "no".
   - If the initial opinion of both generals is "yes", and all message are delivered, then the only valid common decision is "yes".
3. Termination: Both generals decide.

Validity Condition

- If the initial opinion of both generals is "no", then the only valid common decision is "no".
- If the initial opinion of both generals is "yes", and all message are delivered, then the only valid common decision is "yes".

We say that the validity condition is "weak"

- If even one general starts with "yes", the algorithm is allowed to decide "yes".
- If all generals start with "yes", and one message is lost, the algorithm is allowed to decide "no".

The weak formulation is appropriate to show the following impossibility result.

Impossibility Result

Let assume that we have no knowledge on the total number of link failures that occur.

It turns out that even this weak version of problem is impossible to solve.

Let an algorithm $\mathcal{A}$

- solves the problem of Coordinated Attack,
- while an unbounded number of link failures occur.
- it has 2 initial states, one for each input value, i.e. $\forall u \in [1, n], |start_u| = 2$
- for any given initial state assignment, and a successful exchange of messages, there is only one possible execution.
- each round, all processes send one message – they may send a `null` message.

Let ϵ be an execution of $\mathcal{A}$ in which

- both processes start with initial value "yes", that is $i_1 = i_2 =$ "yes"
- all messages are delivered successfully.

Based on the termination condition

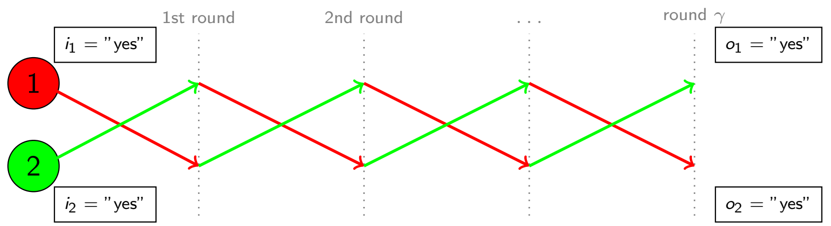
- the exists a round γ when both processes reach a decision.

According to the validity condition

- both processes decide "yes", that is $o_1 = o_2 =$ "yes"

Let ϵ_1 be an execution of $\mathcal{A}$ derived from ϵ during which all messages transmitted after round γ fail to deliver.

Execution ϵ_1 message transmission diagram

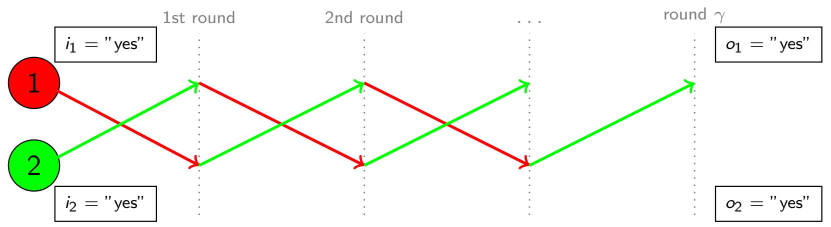


In the message transmission diagram

- arrows represent successful message transmissions.
- transmissions that encounter a link failure are not shown.

Let ϵ_2 be an execution of $\mathcal{A}$ derived from ϵ_1 during which at round γ the message sent by the red general is lost.

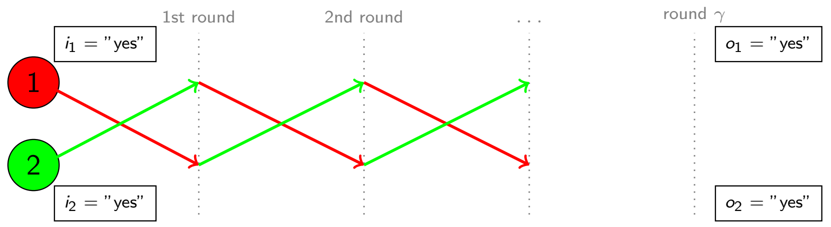
Execution ϵ_2 message transmission diagram



- The decision of the green general at the end of round γ may be different in ϵ_2 than that in ϵ_1.
- However, the red general is not aware of this change – the state of red in ϵ_2 is the same as in ϵ_1
- Due to the agreement condition: is forced to decide the same with red.

Let ϵ_3 be an execution of $\mathcal{A}$ derived from ϵ_2 during which at round γ the message sent by the green general is lost.

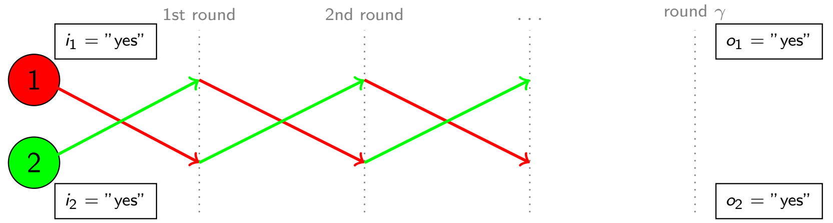
Execution ϵ_3 message transmission diagram



- The decision of the red general at the end of round γ may be different in ϵ_3 than in ϵ_2
- However, the green general not aware of this change – the state of green in ϵ_3 is the same as in ϵ_2
- Due to the agreement condition: red is forced to decide the same with green

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

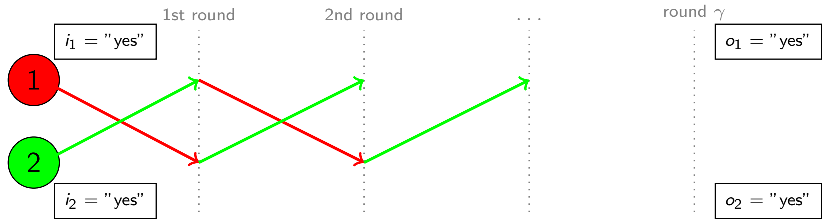
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

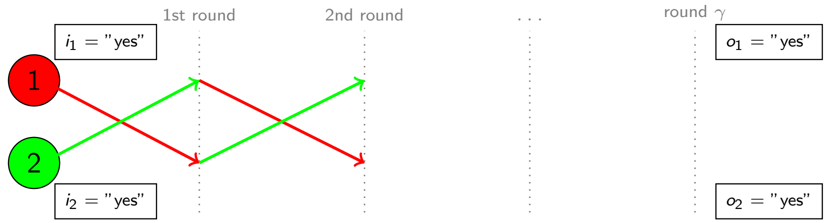
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

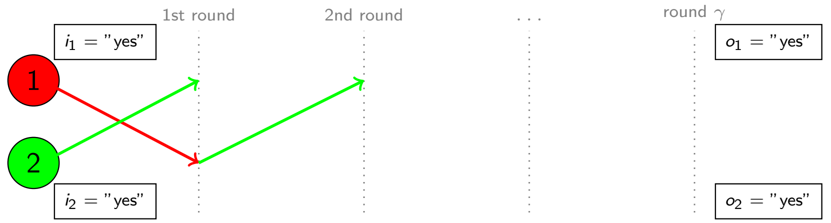
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

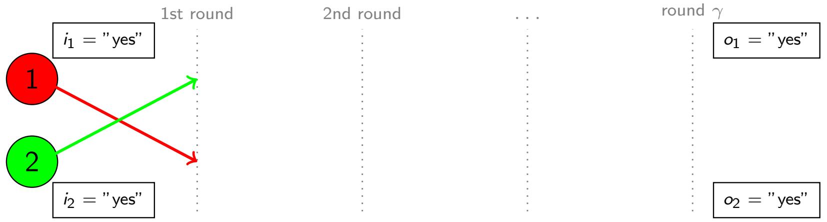
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

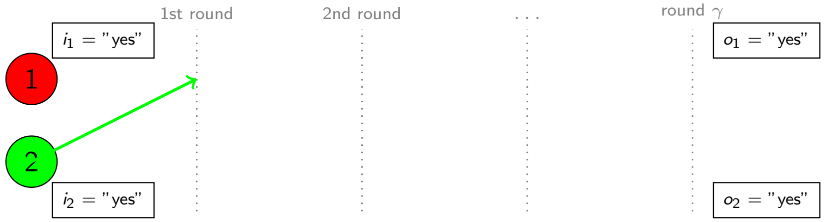
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

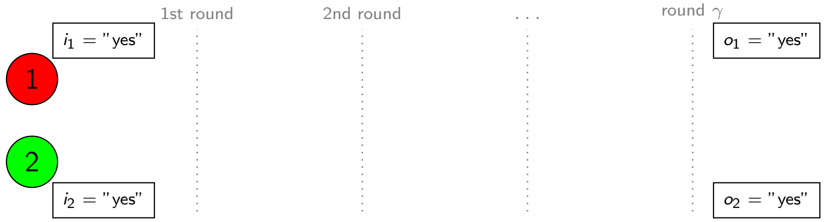
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Continuing in this way, by alternately removing the last message from the red general and from the green general, we eventually reach an execution ϵ' in which both processes start with “yes” and no messages are delivered.

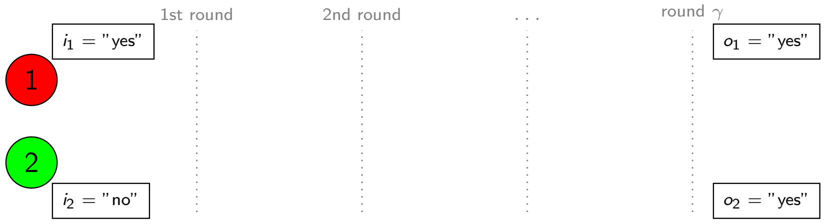
Execution ϵ' message transmission diagram



- Since γ is a finite number, in a finite number of steps we will reach construct execution ϵ'.
- Both generals start with “yes”.
- Both decide to attach with any message exchange.

Let execution ϵ'' be constructed from ϵ', where the green general has an initial opinion “no”.

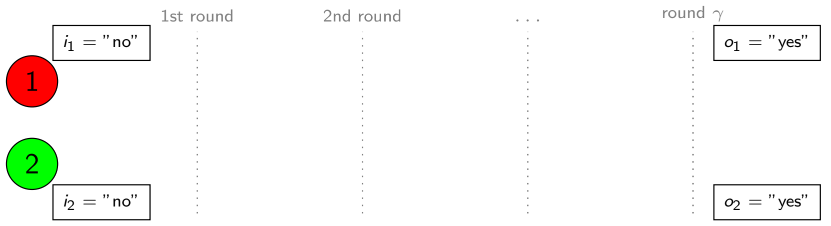
Execution ϵ'' message transmission diagram



- The decision of the green general at the end of round γ may be different in ϵ'' than in ϵ'.
- However, the red generalis not aware of this change – the state of red in ϵ'' is the same as in ϵ'.
- Due to the agreement condition: green is forced to decide the same with red.

Let execution ϵ''' be constructed from ϵ'', where the red general has an initial opinion "no".

Execution ϵ''' message transmission diagram

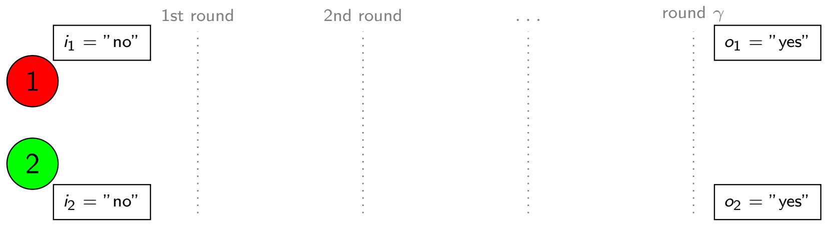


- The decision of the red general at the end of round γ may be different in ϵ''' than in ϵ''
- However, the green general not aware of this change – the state of green in ϵ''' is the same as in ϵ''
- Due to the agreement condition: red is forced to decide the same with green

This yields a contradiction.

- Both generals have the same decision "no".
- According to the validity condition the only acceptable value is "no".

Execution ϵ''' message transmission diagram



Thus algorithm $\mathcal{A}$ does not solve the Coordinated Attack problem.

Fundamental limitation

Theorem

Let G be the graph constituting of nodes 1 *and* 2 *connected by a single edge. Then, there is no algorithm that solves the coordinated attack problem on G given an unbounded number of link failures.*

- Impossible to solve basic consensus problems when dealing with totally unreliable network.
- To overcome, it is necessary to strengthen the model
  - Assume an upper bound on the number of link failures.
  - Assume that link failures occur with a probability p.
- or relax the problem requirements
  - Allow the possibility of violating the agreement condition.
  - Allow the possibility of violating the validity condition.
- Allow processes to use randomization.

Stopping Failures

Processes may simply stop arbitrarily without warning, at any point during a round of execution of a distributed algorithm. The process will halt immediately and terminate without further interaction with the other processes of the system.

- Stopping failures model unpredictable processor crashes.
- We assume an upper bound σ on the number of stopping failures
  - such an upper bound holds for the complete execution of the distributed system.
  - is equivalent to other measures, e.g., rate of stopping failure per round.

FloodSet Algorithm

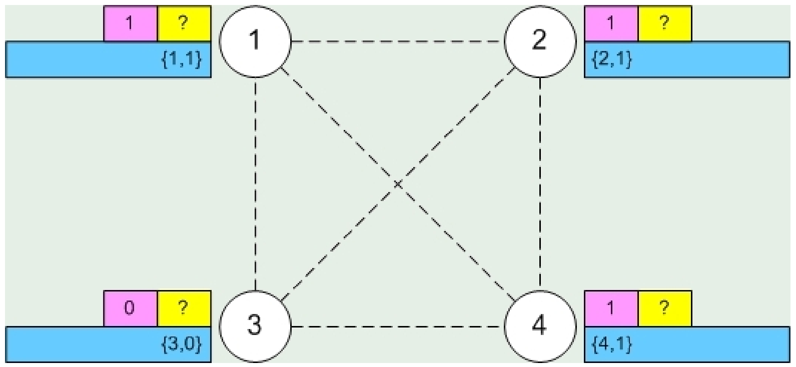
Each process $u \in [1, n]$ maintains a list I_u with input values, initially included only the input value $i_u \in S$ of u, $I_u = \{i_u\}$. In each round, each process broadcasts I, then adds all the elements of the received sets to I_u. After $\sigma + 1$ rounds, if I_u is a singleton set (i.e., $|I_u| = 1$), then u decides on the unique element of I_u; otherwise u decides on the default value $i_0 \in S$.

- We assume a complete graph G.
- We assume an upper bound on process failures σ
- Let $I_u(\gamma)$ be the values in I_u of u at round γ

Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

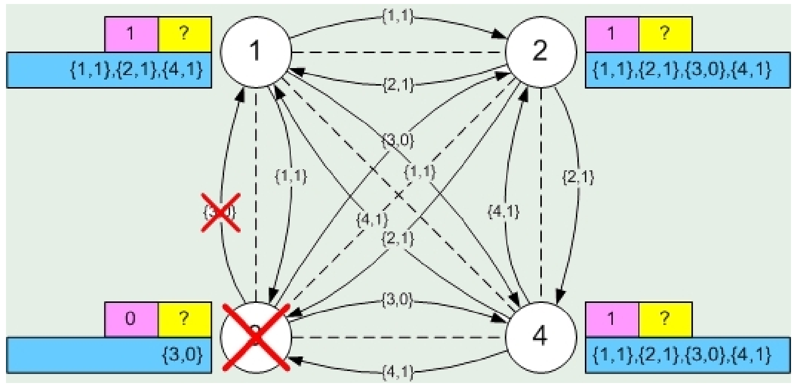
Complete Graph



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

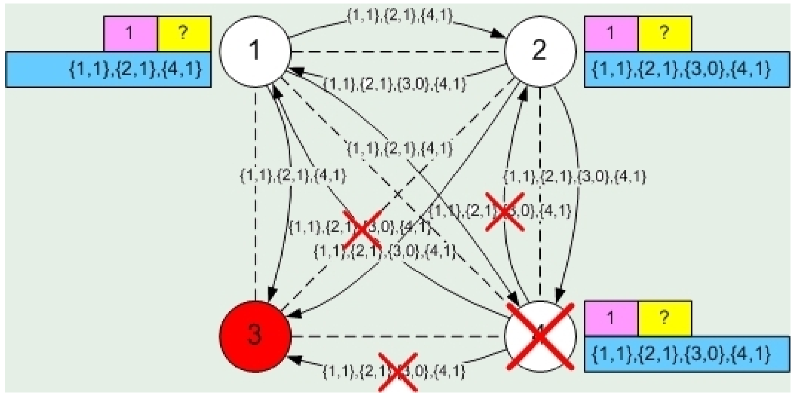
1st Round – process 3 fails



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

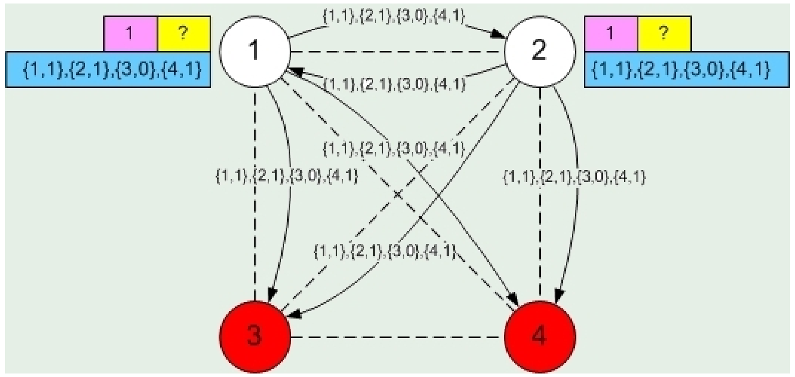
2nd Round – process 4 fails



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

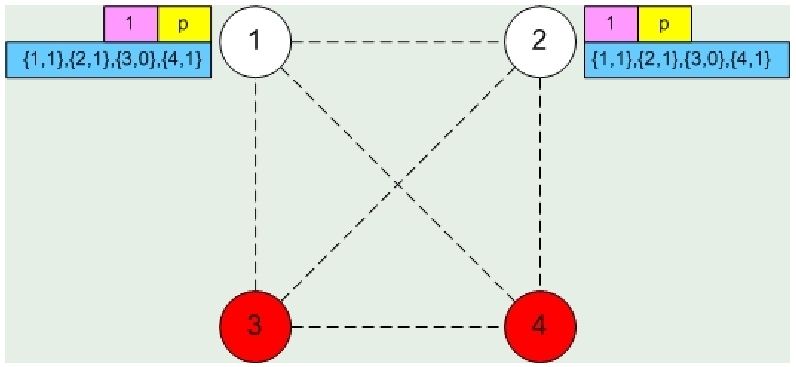
3rd Round – no failures



Example of execution of FloodSet algorithm

Let a synchronous complete graph $n = 4$ and $\sigma = 2$.

3rd Round – agreement



Properties of FloodSet

Lemma (FloodSet.1)

If no process failes during a particular round $\gamma, 1 \leq \gamma \leq \sigma + 1$, then $l_u(\gamma) = l_v(\gamma)$ for all u and v that are active after γ rounds.

Proof: Suppose that no process fails at round γ and let I be the set of processes that are active after $\gamma - 1$ rounds.
Then, $\forall u \in I$ will send its own $l_u(\gamma)$ to all other processes at the end of round $\gamma - 1$.
Thus at round γ,

$$\forall u \in I, l_u(\gamma) = \cup_{v \in I} l_v(\gamma - 1)$$

■

Properties of FloodSet

Lemma (FloodSet.2)

Suppose that $l_u(\gamma) = l_v(\gamma)$ for all u, v that are active after γ rounds. Then for any round $\gamma', \gamma \leq \gamma' \leq \sigma + 1$, the same holds , that is, $l_u(\gamma') = l_v(\gamma')$ for all u, v that are active after γ' rounds.

Proof: All processes that have not failed for γ rounds have identical lists.
The processes that have not failed after γ round still maintain identical lists.
Since no other active process exists, after round γ no new value is circulated in the network.
Therefore the value of $l_u, \forall u \in I$ will not change in any consecutive round.

■

Properties of FloodSet

Lemma (FloodSet.3)

If processes u, v are both active after $\sigma + 1$ rounds, then $l_u(\sigma + 1) = l_v(\sigma + 1)$ at the end of round $\sigma + 1$.

Proof: Since there are at most σ failures, there must be a round $\gamma, 1 \leq \gamma \leq \sigma + 1$ where no process fails.

- According to lemma FloodSet.1 $l_u(\gamma) = l_v(\gamma)$ for each u, v that are still active after round γ
- According to lemma FloodSet.2 $l_u(\sigma + 1) = l_v(\sigma + 1)$ for each u, v that are still active after round $\sigma + 1$

■

Properties of FloodSet

Theorem

Algorithm FloodSet solves the agreement problem for stopping failures.

Proof:

Termination condition holds – all processes that are active until the end of round $\sigma + 1$, terminate.

Validity condition holds –

- If all processes have initial value τ then the list transmitted is $\{\tau\}$
- The list l_u will not changed at the end of round $\sigma + 1$

Agreement condition holds –

- According to FloodSet.3

■

Properties of FloodSet

- Time complexity is $\sigma + 1$ rounds
- Message complexity is $\mathcal{O}\left((\sigma + 1) \cdot n^2\right)$
- Each message may be of size $\mathcal{O}(n)$ bits
- Communication complexity in bits is $\mathcal{O}\left((\sigma + 1) \cdot n^3\right)$

Alternative rules

- Instead of a predefined value $i_0 \in S$, choose $\min(S)$
- Processes send only messages when they detect a change in their list (OptFloodSet)

Open Problem # 7

Assume a synchronous distributed system consisting of a set of n processes that are connected by an unidirectional ring network. Each process has a unique identity and is not aware of the total number of processes. Each process u receives an integer input i_u. Design a distributed algorithm that detects the neighborhood of three processes $((v-1), v, (v+1))$ with the largest sum (i_{v-1}, i_v, i_{v+1}). Define algorithm's properties and verify it's correctness, as well as the time and message complexity. You should prove your claims.

Open Problem # 8

Assume a synchronous distributed system consisting of a set of n processes that are connected by a directed ring network, where each process has a unique identity but is not aware of the total number of processes neither of the network's topology. Design a distributed leader election algorithm that can tolerate σ number of communication failures. Define algorithm's properties and verify it's correctness, as well as the time and message complexity. You should prove your claims.