Consensus (with failures) in synchronous systems

Etienne Renault

2 octobre 2020

https://www.lrde.epita.fr/~renault/teaching/algorep/

What is a consensus ?

Consensus

All process must agree on a value even iff inputs can be arbitrary.

There is generally a validity condition describing the outputs values that are permitted for each patterns of input.

What is a consensus ?

Consensus

All process must agree on a value even iff inputs can be arbitrary.

There is generally a validity condition describing the outputs values that are permitted for each patterns of input.

- Agreement on wether to commit or abort transaction in a database
- Agreement on a specific value reading multiples captors (altitude for instance)

What is a consensus ?

Consensus

All process must agree on a value even iff inputs can be arbitrary.

There is generally a validity condition describing the outputs values that are permitted for each patterns of input.

- Agreement on wether to commit or abort transaction in a database
- Agreement on a specific value reading multiples captors (altitude for instance)
- Classification of a system component as fautly

What is a consensus ?

Consensus

All process must agree on a value even iff inputs can be arbitrary.

There is generally a validity condition describing the outputs values that are permitted for each patterns of input.

- Agreement on wether to commit or abort transaction in a database
- Agreement on a specific value reading multiples captors (altitude for instance)
- Classification of a system component as fautly
- Ressource Allocation : who has the priority to obtain a resssource ?

Failures

- Communication failures
  - Omission, Timing, Response, Crash, Arbitrary
- Process failures
  - Fail-stop, Fail-safe (detectable), Fail-silent, Fail-arbitrary

1. Link Failures
2. Process Failures
   - Stopping Algorithm : FloodSet
   - Stopping Algorithm : EIG
   - Byzantine Algorithm
3. Conclusion

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed
- Each general has an initial opinion about to attack or not to attack

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed
- Each general has an initial opinion about to attack or not to attack
- Generals are located on different places

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed
- Each general has an initial opinion about to attack or not to attack
- Generals are located on different places
- They can communicate with messengers

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed
- Each general has an initial opinion about to attack or not to attack
- Generals are located on different places
- They can communicate with messengers
- Messengers can be lost (killed), or captured

The Coordinated Attack Problem

Informal Scenario :

- Several generals plan a coordinated attack
- The only way to succeed is if all generals attack
- If only one general attacks its army will be destroyed
- Each general has an initial opinion about to attack or not to attack
- Generals are located on different places
- They can communicate with messengers
- Messengers can be lost (killed), or captured

The generals must agree on wether to attack or not

Easy case : Communications are reliable

Messengers are reliables

1. All generals broadcast their intentions
2. After D rounds, all generals have the information of other generals
3. If all general agreed then attack, otherwise to not attack.

Hard case : Communication are **not** reliable

How to solve this problem ?

More Formally 1/2

- n processes indexed by $1 \dots n$
- Arbitrary arrange an undirected graph network
- Each process knows the entire graph, indexes included
- Processes start with 0 (don't attack) or 1 (attack) as initial value
- Synchronous model with communication loss

More Formally 2/2

Processes must eventually outputs the decision by setting a special decision component to 0 or 1.

Conditions :

1. **Agreement** : two processes decide on different values.
2. **Validity** :
   - If all processes starts with 0, 0 is the only decision possible
   - If all processes starts with 1 and **all messages are delivered**, 1 is the only possible decision
3. **Termination** : all processes eventually decide.

Impossibility Result 1/3

Let G be a graph with 2 nodes connected by a single edge.

Then, no algorithm solves the coordinated attack problem on G

Proof. (by contraction)

- Suppose a solution exists, given by an algorithm A

Impossibility Result 1/3

Let G be a graph with 2 nodes connected by a single edge.
Then, no algorithm solves the coordinated attack problem on G

Proof. (by contraction)

- Suppose a solution exists, given by an algorithm A
- Let α be the execution when both processes starts with 1 and eventually outputs 1 **with all messages delivered**.

Impossibility Result 1/3

Let G be a graph with 2 nodes connected by a single edge.
Then, no algorithm solves the coordinated attack problem on G

Proof. (by contraction)

- Suppose a solution exists, given by an algorithm A
- Let α be the execution when both processes starts with 1 and eventually outputs 1 **with all messages delivered**.
- Let α_1 be the same than α except that all messages are lost after *r* rounds. In α_1 both processes output 1.

Impossibility Result 1/3

Let G be a graph with 2 nodes connected by a single edge.
Then, no algorithm solves the coordinated attack problem on G

Proof. (by contraction)

- Suppose a solution exists, given by an algorithm A
- Let α be the execution when both processes starts with 1 and eventually outputs 1 **with all messages delivered**.
- Let α_1 be the same than α except that all messages are lost after r rounds. In α_1 both processes output 1.
- Let α_2 be the same than α_1 except that the last (round r) message from process 1 to process 2 is not delivered.

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \overset{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \overset{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.
- Since process 1 outputs 1 in α_1, then it outputs 1 in α_2.

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \overset{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.
- Since process 1 outputs 1 in α_1, then it outputs 1 in α_2.
- By termination and agreement, process 2 outputs 1

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \overset{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.
- Since process 1 outputs 1 in α_1, then it outputs 1 in α_2.
- By termination and agreement, process 2 outputs 1
- Let α_3 be the same than α_2 except that the last message from process 2 to process 1 is not delivered.

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \stackrel{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.
- Since process 1 outputs 1 in α_1, then it outputs 1 in α_2.
- By termination and agreement, process 2 outputs 1
- Let α_3 be the same than α_2 except that the last message from process 2 to process 1 is not delivered.
- $\alpha_2 \stackrel{2}{\sim} \alpha_3$: α_2 is indistinguishable from α_3 from process 2 point of view.

Impossibility Result 2/3

Proof. (contd.)

- $\alpha_1 \overset{1}{\sim} \alpha_2$: α_1 is indistinguishable from α_2 from process 1 point of view.
- Since process 1 outputs 1 in α_1, then it outputs 1 in α_2.
- By termination and agreement, process 2 outputs 1
- Let α_3 be the same than α_2 except that the last message from process 2 to process 1 is not delivered.
- $\alpha_2 \overset{2}{\sim} \alpha_3$: α_2 is indistinguishable from α_3 from process 2 point of view.
- Since process 2 outputs 1 in α_2, then it outputs 1 in α_3. The same for process 1 by termination and agreement.

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).
- Let α'' the execution where no messages are delivered and where process 1 starts with 1 and process 2 starts with 0

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).
- Let α'' the execution where no messages are delivered and where process 1 starts with 1 and process 2 starts with 0
- $\alpha' \overset{1}{\sim} \alpha''$: process 1 outputs 1 in α'' and so does process 2

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).
- Let α'' the execution where no messages are delivered and where process 1 starts with 1 and process 2 starts with 0
- $\alpha' \overset{1}{\sim} \alpha''$: process 1 outputs 1 in α'' and so does process 2
- Let α''' the execution where no messages are delivered and where both processes starts with 0

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).
- Let α'' the execution where no messages are delivered and where process 1 starts with 1 and process 2 starts with 0
- $\alpha' \stackrel{1}{\sim} \alpha''$: process 1 outputs 1 in α'' and so does process 2
- Let α''' the execution where no messages are delivered and where both processes starts with 0
- In α''' all processes output 0 by termination and agreement.

Impossibility Result 3/3

Proof. (contd.)

- Continue this way to obtain α' on which both processes starts with 1 and no messages are delivered.
- Both process are forced to output 1 (same reasoning as above).
- Let α'' the execution where no messages are delivered and where process 1 starts with 1 and process 2 starts with 0
- $\alpha' \overset{1}{\sim} \alpha''$: process 1 outputs 1 in α'' and so does process 2
- Let α''' the execution where no messages are delivered and where both processes starts with 0
- In α''' all processes output 0 by termination and agreement.
- $\alpha'' \overset{2}{\sim} \alpha'''$: process 2 outputs 0 in α'' and so does process 1

Consensus with link failures

IMPOSSIBLE !

Consensus with link failures

IMPOSSIBLE !

Some solutions exist using probabilities (not in this lecture)

Problem Statement

What if communications are reliable, but processes may fail ?

Two kind of failure models :

- Stopping failures : Processes may stop without warning
- Byzantine failures[1] : fautly processes may exhibit completely unconstrained behaviors.

1. The term comes from Lamport, Pease and Shostak in a paper about consensus between byzantine generals that may have traitorous behaviors

Agreement Problem

Consensus sub-problem

All processes start with a value v.

All **non-fautly** processes are required to output the same value with agreement and validity conditions.

Real world problem in airplane :

- multiple processors
- with access to different altimeters
- attempt to detect airplane altitude

Limitations

We consider that a process can only have a fixed number of failures.

In practice this assumption may be realistic since it may be unlikely that more than f failures occur.

Problem Statement

- n processes indexed by $1 \ldots n$
- Arbitrary arrange an undirected graph network
- Each process knows the entire graph, indexes included
- The graph is complete
- Processes start with a value $v \in V$
- Synchronous model with reliable communications
- A limited number f of processes might fail

Failure models 1/2

Stopping

The process can stop at any moment, even in the middle of *message sending*. We assume that any subset of the message are sent.

- **Agreement** : Not two processes decide different values.
- **Validity** : If all processes start with the same initial value $v \in V$, then v is the only possible value.
- **Termination** : All non-fautly processes eventually decide.

Failure models 2/2

Byzantine

The process can fail at any moment not only by stopping but by exhibiting arbitrary behavior. The only limitation is that the behavior can only affect component on which the process have control.

- **Agreement** : Not two processes decide different values.
- **Validity** : If all non-fautly processes start with $v \in V$, then v is the only decision for non fautly processes.
- **Termination** : All non-fautly processes eventually decide.

Remarks

Relationship between failure models

An algorithm solving the Byzantine agreement does not necessarily solves the stopping one !

Complexity

The complexity is determined in rounds until all the non-fautly processes decide.

For communication complexity, only messages from non-fautly processes are considered.

1. Link Failures

2. Process Failures
   - Stopping Algorithm : FloodSet
   - Stopping Algorithm : EIG
   - Byzantine Algorithm

3. Conclusion

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E,
for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v
- At each round, each process broadcasts its set W

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v
- At each round, each process broadcasts its set W
- When values are received, they are added to W

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v
- At each round, each process broadcasts its set W
- When values are received, they are added to W
- After $f + 1$ rounds

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E,
for instance the minimum of E

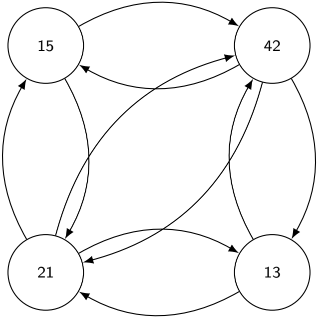
- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v
- At each round, each process broadcasts its set W
- When values are received, they are added to W
- After $f + 1$ rounds
  - If $|W| = 1$, then decide $v \in W$

Algorithm : Informal

We denote by $v_0 \in E$, a prespecified value of the set E, for instance the minimum of E

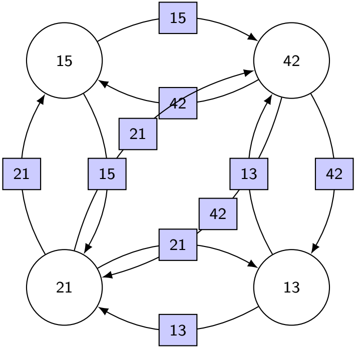
- Initially, each process p have its own initial value $v \in V$
- Each process maintains a set $W \subseteq V$.
- Initially, W only contains v
- At each round, each process broadcasts its set W
- When values are received, they are added to W
- After $f + 1$ rounds
  - If $|W| = 1$, then decide $v \in W$
  - Otherwise decide $v0$

Example



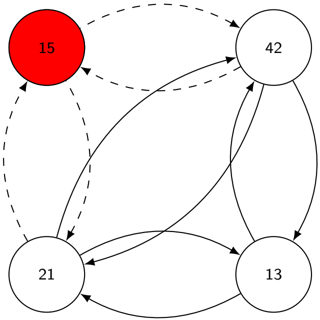
$f = 1$, Initial state

Example



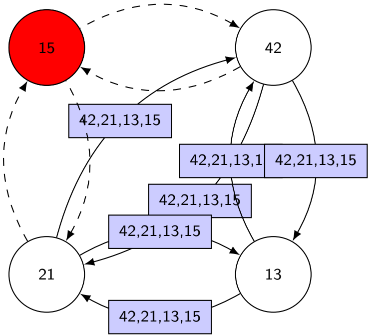
$f = 1$, round 1

Example



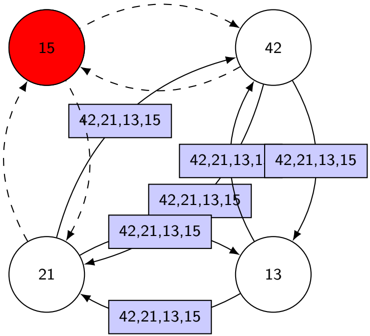
15 stops

Example



$f = 1$, round 2

Example



Decision is v_0

Complexity

- **Time complexity** : $f + 1$ rounds
- **Communication complexity** : $O((f + 1)n^2)$
- **Size for a single message** : considering b as an upper bound for $v \in V$, $O(nb)$

Complexity

- **Time complexity** : $f + 1$ rounds
- **Communication complexity** : $O((f + 1)n^2)$
- **Size for a single message** : considering b as an upper bound for $v \in V$, $O(nb)$

Reducing the communication

Fixing v_0 as a specified value help to reduce communication since, only two broadcasts are necessary.

Exponential Information Gathering (EIG)

Main Idea

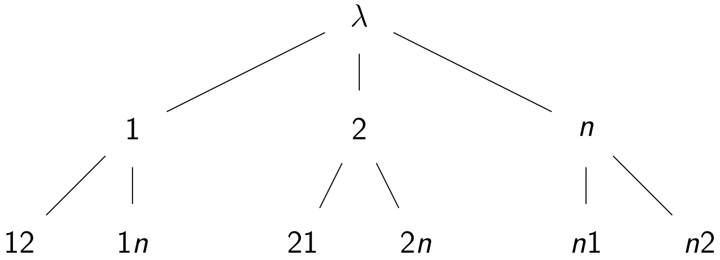
- Process send an relay initial values along paths in a structure called *EIG tree*
- Each process maintains an EIG tree
- At the end, they use a decision rule base on theur EIG tree

EIG algorithms are costly but can be partially reused
to cope with byzantine faults.

EIG Tree

- Paths from the root represent chains of processes along which initial values are propagated
- All chains represented consist in different processes
- The tree has $f + 2$ levels
- Each node at level k have n children
- Each node is labelled by a string :
  - The root is the empty-string
  - A node with label i has n children labelled i_1 to i_n

EIG Example



EIG Tree for $f = 1$

EIG for stopping failures : Informal

1. Each process maintains an EIG tree

EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value

EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value
3. Every process broadcast this value to all processes (even itself)

EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value
3. Every process broadcast this value to all processes (even itself)
4. Received information is inserted into the tree

EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value
3. Every process broadcast this value to all processes (even itself)
4. Received information is inserted into the tree
5. For all the other rounds, process i broadcasts all pair (x, v) where x is a $f - 1$ label that does not contains i

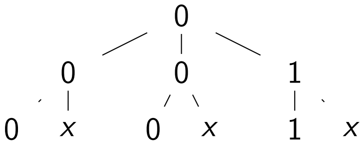
EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value
3. Every process broadcast this value to all processes (even itself)
4. Received information is inserted into the tree
5. For all the other rounds, process i broadcasts all pair (x, v) where x is a $f - 1$ label that does not contains i
6. Each nodes decorates level k with values in V or *null* at the end of round k.

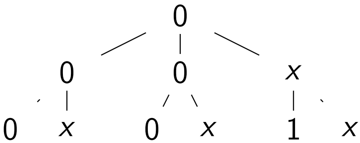
EIG for stopping failures : Informal

1. Each process maintains an EIG tree
2. Initially each process decorates the root with its own initial value
3. Every process broadcast this value to all processes (even itself)
4. Received information is inserted into the tree
5. For all the other rounds, process i broadcasts all pair (x, v) where x is a $f - 1$ label that does not contains i
6. Each nodes decorates level k with values in V or *null* at the end of round k.
7. At the end of the $f + 1$ roudns processes apply a decision rule

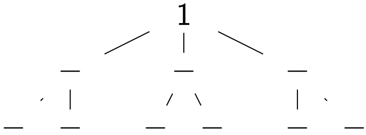
Example



Process 1



Process 2



Process 3

Process 3 fails at round 1 and
it's initial message has been sent to 1 but not to 2.

Complexity

- **Time complexity** : $f + 1$ rounds
- **Communication complexity** : $O((f+1)n^2)$

The number of bits communicated is exponential in the number of failures, i.e. with b an upper bound for V, we have $O(n^{f+1}b)$ bits exchanged

EIG algorithm for stopping

Can cope with a restriction of the Byzantine problem (Byzantine with authentication) :

- Correct processes can sign correctly their messages
- Incorrect processes can't sign correctly their messages

EIG stopping algorithm solves this problem.

Why byzantine is more complicated than stopping ?

Three processes cannot solve the agreement problem if one of them is fautly !

Why byzantine is more complicated than stopping ?

Three processes cannot solve the agreement problem if one of them is fautly !

Basic Idea :

- Let us consider 3 processes p_1, p_2 et p_3
- Let 0 be the initial value of p_1 and 1 the initial value of p_2

Why byzantine is more complicated than stopping ?

Three processes cannot solve the agreement problem if one of them is fautly !

Basic Idea :

- Let us consider 3 processes p_1, p_2 et p_3
- Let 0 be the initial value of p_1 and 1 the initial value of p_2
- Let us consider that every correct process broadcast its initial value

Why byzantine is more complicated than stopping ?

Three processes cannot solve the agreement problem if one of them is fautly !

Basic Idea :

- Let us consider 3 processes p_1, p_2 et p_3
- Let 0 be the initial value of p_1 and 1 the initial value of p_2
- Let us consider that every correct process broadcast its initial value
- If p_3 broadcast 0 to p_1 and 2 to p_2 no agreement can be done !

Byzantine agreement : informal

- For n processes and f failures, $n > 3.f$

Byzantine agreement : informal

- For n processes and f failures, $n > 3.f$
- Use EIG tree data structure

Byzantine agreement : informal

- For n processes and f failures, $n > 3.f$
- Use EIG tree data structure
- Same propagation strategy that in EIG algorithm for stopping

Byzantine agreement : informal

- For n processes and f failures, $n > 3.f$
- Use EIG tree data structure
- Same propagation strategy that in EIG algorithm for stopping
- Only difference : when a process receive an ill-formed message, it corrects the information to make it look *sensible*.

Byzantine agreement : informal

- For n processes and f failures, $n > 3.f$
- Use EIG tree data structure
- Same propagation strategy that in EIG algorithm for stopping
- Only difference : when a process receive an ill-formed message, it corrects the information to make it look *sensible*.
- The decision procedure is also modified to mask incorrect data.

Byzantine agreement algorithm

- The process propagate values for $f + 1$ rounds

Byzantine agreement algorithm

- The process propagate values for $f + 1$ rounds
- If a process i receive a message from j that is not of the specified form, the message is discarded

Byzantine agreement algorithm

- The process propagate values for $f + 1$ rounds
- If a process i receive a message from j that is not of the specified form, the message is discarded
- At the end of the $f + 1$ rounds i adjust null values to v_0

Byzantine agreement algorithm

- The process propagate values for $f + 1$ rounds
- If a process i receive a message from j that is not of the specified form, the message is discarded
- At the end of the $f + 1$ rounds i adjust null values to v_0
- To determine the wining value, the process walk its EIG tree from leaf to roots.

Byzantine agreement algorithm

- The process propagate values for $f+1$ rounds
- If a process i receive a message from j that is not of the specified form, the message is discarded
- At the end of the $f+1$ rounds i adjust null values to v_0
- To determine the wining value, the process walk its EIG tree from leaf to roots.
- If a majority exist then the new value is decided, otherwise the processes decide v_0

Example

Other Results

For general graphs

Agreement for n nodes and f faults in a graph G require

1. $n > 3f$
2. $conn(G) > 2f$

Stopping with failures

Cannot be solved in fewer than $f + 1$ rounds

Conclusion

- No algorithm for consensus with **link failures**
- Different kind of fault : stopping, Byzantine
- Algorithms for consensus with fault
- More synchronous problems
  - $k-$agreement problem
  - commit problem