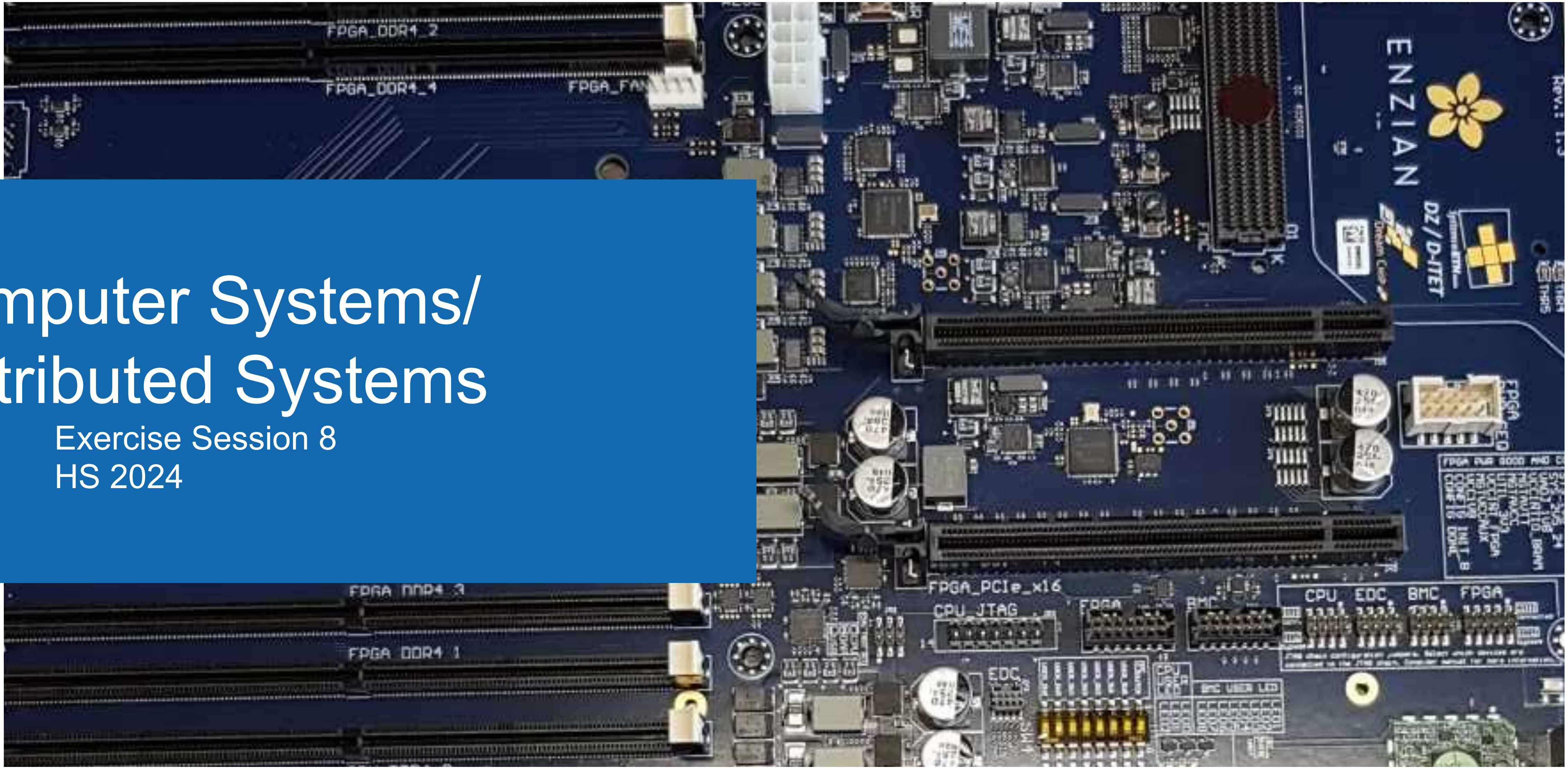
ETH zürich

Systems@ETH zürich

Distributed Computing

Slides last updated:
14.11.2023



Computer Systems/ Distributed Systems

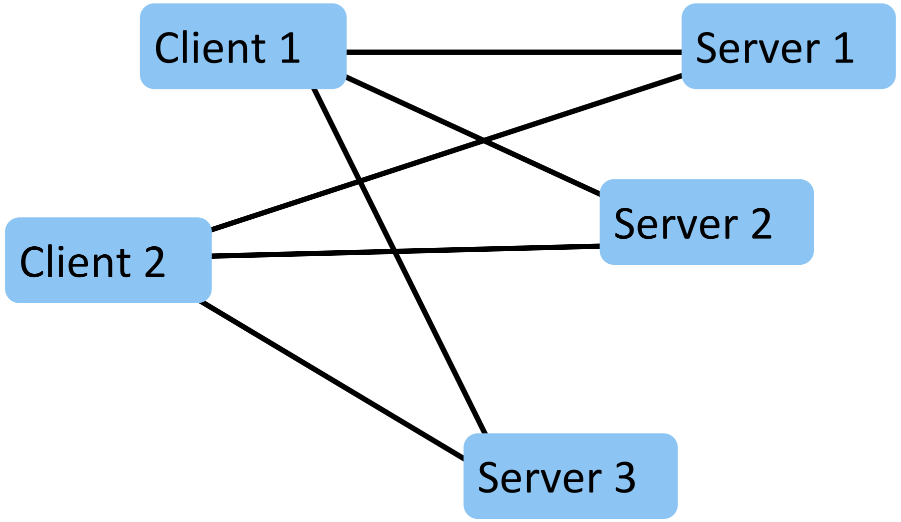
Exercise Session 8
HS 2024

ETH zürich Systems@ETH zürich Distributed Computing

Program

1. Lecture Recap
   a) Introduction: Distributed Systems
   b) Fault Tolerance and Paxos
   c) Consensus
2. Quiz
3. Assignment Preview

Set-Up



Node: single actor in a distributed system

Can be both client or server

Challenges

- Messages can get lost
- Nodes may crash
- Messages can have varying delays

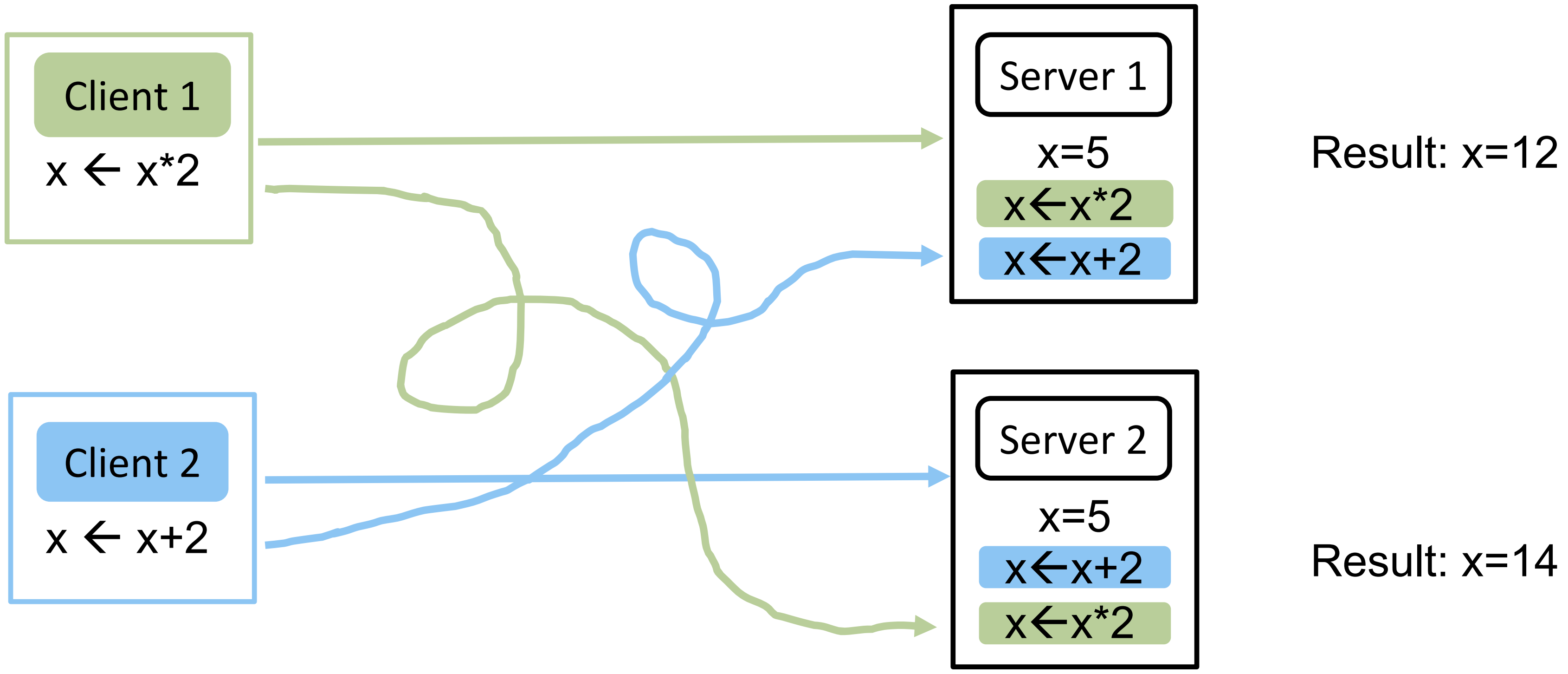
Challenges

- Messages can get lost
- Nodes may crash
- Messages can have varying delays

First Goal: State Replication

- All servers execute the *same commands* in the *same order*.

Why do we want State Replication?



ETH zürich — Distributed Computing

First Approaches

Server sends acknowledgment message

- Reasonable with one client
- Inconsistent state with multiple clients and servers

First Approaches

Server sends acknowledgment message

- Reasonable with one client
- Inconsistent state with multiple clients and servers

Serializer – all commands go through one node which orders them

- Single point of failure

First Approaches

Server sends acknowledgment message

- Reasonable with one client
- Inconsistent state with multiple clients and servers

Serializer – all commands go through one node which orders them

- Single point of failure

Two-Phase Protocol – ask for locks, execute once acquired all locks

- Breaks down if we even have just one node failure
- How to avoid deadlocks?

Paxos – Main Ideas

1. Tickets
   - “Weak lock”
   - Can be overwritten by later tickets
   - Reissuable
   - Expiration

Paxos – Main Ideas

1. Tickets
    - "Weak lock"
    - Can be overwritten by later tickets
    - Reissuable
    - Expiration
2. Require majority
    - Ensures only single command gets accepted

Paxos – Main Ideas

1. Tickets
   - “Weak lock”
   - Can be overwritten by later tickets
   - Reissuable
   - Expiration
2. Require majority
   - Ensures only single command gets accepted
3. Servers inform clients about their stored command
   - Client can switch to supporting this command

Paxos – Main Ideas

1. Tickets
   - "Weak lock"
   - Can be overwritten by later tickets
   - Reissuable
   - Expiration
2. Require majority
   - Ensures only single command gets accepted
3. Servers inform clients about their stored command
   - Client can switch to supporting this command

Good video with slightly different terminology:
https://www.youtube.com/watch?v=d7nAGI_NZPk

ETH zürich

Distributed Computing

Algorithm 7.13 Paxos

Client (Proposer) | **Server (Acceptor)**

Initialization ....

Client:

c ◁ *command to execute*

$t = 0$ ◁ *ticket number to try*

Server:

$T_{\max} = 0$ ◁ *largest issued ticket*

$C = \bot$ ◁ *stored command*

$T_{\text{store}} = 0$ ◁ *ticket used to store C*

Phase 1 ....

Client:

$t = t + 1$
Ask all servers for ticket t

Server:

if $t > T_{\max}$ **then**
  $T_{\max} = t$
  Answer with ok(T_{store}, C)
end if

Phase 2 ....

Client:

if a majority answers ok **then**
  Pick (T_{store}, C) with largest T_{store}
  if $T_{\text{store}} > 0$ **then**
    $c = C$
  end if
  Send propose(t, c) to same majority
end if

Server:

if $t = T_{\max}$ **then**
  $C = c$
  $T_{\text{store}} = t$
  Answer success
end if

Phase 3 ....

Client:

if a majority answers success **then**
  Send execute(c) to every server
end if

Clients can restart Phase 1 at any time.

Clients asks for a specific ticket *t*.

Server only issues ticket *t* if *t* is the highest ticket requested so far.

If client receives majority of tickets, it proposes a command.

When a server receives a proposal, and the ticket of the client is still valid, the server stores the command and notifies the client.

If a majority of servers store the command, the client notifies all servers to execute the command.

ETH zürich Systems@ETH Zürich Distributed Computing

Consensus

We want...

1. Agreement:

   All (correct) nodes decide on the same value.

Consensus

We want...

1. Agreement:
   All (correct) nodes decide on the same value.
2. Termination:
   All (correct) nodes terminate (violated by Paxos).

Consensus

We want...

1. Agreement:
   All (correct) nodes decide on the same value.
2. Termination:
   All (correct) nodes terminate (violated by Paxos).
3. Validity:
   The decision value is the input value of at least one node.

Consensus

We want...

1. Agreement:
   All (correct) nodes decide on the same value.
2. Termination:
   All (correct) nodes terminate (violated by Paxos).
3. Validity:
   The decision value is the input value of at least one node.

Impossibility

Consensus cannot be solved **deterministically** in the asynchronous model!

Synchronous Model

- Nodes operate in synchronous rounds
- In each round a given node can
  - send a message
  - receive messages
  - do some local computation
- Runtime is the number of rounds from start to finish of execution *in the worst case*

Synchronous Consensus

Algorithm 16.3 Synchronous Consensus with $f < n$ crash failures

$v_i \in \mathbb{R}$ ◁ input
$min := v_i$
for $i = 1, \ldots, f+1$ **do**
    Broadcast min
    Collect broadcast messages in set M
    $min := \min(M)$
end for
Decide on min

- Correctness?
  - Termination? yes! Runs for exactly f + 1 rounds
  - Validity? yes! Selection from set of broadcast input values
  - Agreement? yes! At least one round without failures guarantees finding local min of remaining nodes. (imagine only f rounds where in the last round the node with the global min crashes while boradcasting, reaching only a subset of all nodes – this would prohibit agreement)
- Runtime? Always f + 1 rounds

Synchronous Lower Bound

Any deterministic consensus algorithm in the synchronous model has a runtime of at least f + 1 rounds for any $f \leq n - 2$ even under a relaxed validity constraint

Definition 16.6 (Validity). *If all non-faulty nodes start with the same value x, the output must be x.*

instead of

- **Validity** *The decision value must be the input value of a* node.

→Detailed proof in the script!

Randomized Consensus

Easy cases:

- All inputs are equal (all 0 or 1)
- Almost all input values equal

Randomized Consensus

Easy cases:

- All inputs are equal (all 0 or 1)
- Almost all input values equal

Otherwise:

- Choose a ***random*** value locally → Expected time $O(2^n)$ until all agree (once)

8.4 Randomized Consensus

Algorithm 8.15 Randomized Consensus (Ben-Or)

```
v_i ∈ {0,1}                ◁ input bit
round = 1
decided = false

Broadcast myValue(v_i, round)

while true do

  Propose

  Wait until a majority of myValue messages of current round arrived
  if all messages contain the same value v then
    Broadcast propose(v, round)
  else
    Broadcast propose(⊥, round)
  end if

  if decided then
    Broadcast myValue(v_i, round+1)
    Decide for v_i and terminate
  end if

  Adapt

  Wait until a majority of propose messages of current round arrived
  if all messages propose the same value v then
    v_i = v
    decided = true
  else if there is at least one proposal for v then
    v_i = v
  else
    Choose v_i randomly, with Pr[v_i = 0] = Pr[v_i = 1] = 1/2
  end if
  round = round + 1
  Broadcast myValue(v_i, round)
end while
```

Majority has seen a majority → (if all messages propose the same value v then)

At least someone has seen majority → (else if there is at least one proposal for v then)

No majority seen → (else)

Ben-Or: Consensus Proof

Validity:

If all nodes start with the same value, then all proposals are for the same value. Thus, the algorithm terminated within one round, deciding on the common value.

If some nodes start with 0 and some start with 1, then both outcomes are legal.

Ben-Or: Consensus Proof

Agreement: (need to show: if one node decides → all nodes decide on the same value)

In a single round *r*:

- Nodes only decide after having received a proposal.
- Note, that a proposal required a majority, therefore a proposal in round *r* can only occur for one value.

→ In any round *r*, all nodes decide on at most one identical value.

Ben-Or: Consensus Proof

Agreement: (need to show: if one node decides → all nodes decide on the same value)

In any round *r*:

- Nodes only decide after having received a proposal.
- Note, that a proposal required a majority, therefore a proposal in round *r* can only occur for one value.

→ In any round *r*, all nodes decide on at most one identical value.

If any node decided in round *r*:

- Deciding node received > *n*/2 proposals for *v*. → All nodes received ≥ 1 proposal for *v*.
- They adapt their own value to *v* in round *r*, and broadcast it in round *r*+1.
- As all nodes broadcast *v*, they will also all propose *v* in the same round.
- All nodes receive > *n*/2 proposals for *v* in round *r*+1 and decide on *v*.

Ben-Or: Consensus Proof

Termination:

Trivial case: all nodes start with the same value
→ Termination after one round.

In the worst case: no node receives all0identical majorities, and all repeatedly choose a random value. The probability of all nodes getting the same value is 2^{-n}, thus we expect all nodes to send the same “my value” after 2^n runs.

Randomized Consensus

Easy cases:

- All inputs are equal (all 0 / all 1)
- Almost all input values equal

Otherwise:

- Choose a ***random*** value locally → Expected time $O(2^n)$ until all agree (once)
- Wouldn't it be useful if the nodes could all toss the ***same*** coin? → Shared Coin

Quiz Paxos

1. How does a node in Paxos know if a majority answered with ok?

2. Does the Paxos algorithm in the script achieve state replication?

3. How many nodes could crash so the Paxos still works?

4. Does Paxos solve consensus?

Quiz Paxos

1. How does a node in Paxos know if a majority answered with ok?
   Each node needs to know the number *n* of servers in the system.

2. Does the Paxos algorithm in the script achieve state replication?

3. How many nodes could crash so the Paxos still works?

4. Does Paxos solve consensus?

Quiz Paxos

1. How does a node in Paxos know if a majority answered with ok?
   Each node needs to know the number *n* of servers in the system.

2. Does the Paxos algorithm in the script achieve state replication?
   No, it only shows agreement on a single command, for several commands we would need to restart the system.

3. How many nodes could crash so the Paxos still works?

4. Does Paxos solve consensus?

Quiz Paxos

1. How does a node in Paxos know if a majority answered with ok?
   Each node needs to know the number *n* of servers in the system.

2. Does the Paxos algorithm in the script achieve state replication?
   No, it only shows agreement on a single command, for several commands we would need to restart the system.

3. How many nodes could crash so the Paxos still works?
   Less than $n/2$

4. Does Paxos solve consensus?

Quiz Paxos

1. How does a node in Paxos know if a majority answered with ok?
   Each node needs to know the number *n* of servers in the system.

2. Does the Paxos algorithm in the script achieve state replication?
   No, it only shows agreement on a single command, for several commands we would need to restart the system.

3. How many nodes could crash so the Paxos still works?
   Less than *n*/2

4. Does Paxos solve consensus?
   No, termination is not guaranteed.

More quiz questions (choose the right answer)

1. State replication is trivial for fewer than 3 nodes? T/F
2. In Paxos, a new ticket can only be issued if all previous tickets have been returned. T/F
3. Which is not a property of consenus? Agreement Termination Tolerance Validity
4. A configuration includes all received messages but not the messages in transit. T/F
5. In a synchronous system, a message has a delay of ___ time units. 1/n/f-n/ potentially infinite

More quiz questions (choose the right answer)

1. State replication is trivial for fewer than 3 nodes? T/**F**
2. In Paxos, a new ticket can only be issued if all previous tickets have been returned. T/**F**
3. Which is not a property of consenus? Agreement Termination **Tolerance** Validity
4. A configuration includes all received messages but not the messages in transit. T/**F**
5. In a synchronous system, a message has a delay of ___ time units. **1**/n/f-n/ potentially infinite

Assignment Preview

1.1 An Asynchronous Riddle

A hangman summons his 100 prisoners, announcing that they may meet to plan a strategy, but will then be put in isolated cells, with no communication. He explains that he has set up a switch room that contains a single switch. Also, the switch is not connected to anything, but a prisoner entering the room may see whether the switch is on or off (because the switch is up or down). Every once in a while the hangman will let one arbitrary prisoner into the switch room. The prisoner may throw the switch (on to off, or vice versa), or leave the switch unchanged. Nobody but the prisoners will ever enter the switch room. The hangman promises to let any prisoner enter the room from time to time, arbitrarily often. That is, eventually, each prisoner has been in the room at least once, twice, a thousand times or any number you want. At any time, any prisoner may declare "We have all visited the switch room at least once". If the claim is correct, all prisoners will be released. If the claim is wrong, the hangman will execute his job (on all the prisoners). Which strategy would you choose...

a) ...if the hangman tells them, that the switch is off at the beginning?

b) ...if they don't know anything about the initial state of the switch?

Assignment Preview

2.1 Consensus with Edge Failures

In the lecture we only discussed node failures, but we always assumed that edges (links) never fail. Let us now study the opposite case: Assume that all nodes work correctly, but up to f edges may fail.

Analogously to node failures, edges may fail at any point during the execution. We say that a failed edge does not forward any message anymore, and remains failed until the algorithm terminates. Assume that an edge always simultaneously fails completely, i.e., no message can be exchanged over that edge anymore in either direction.

We assume that the network is initially fully connected, i.e., there is an edge between every pair of nodes. Our goal is to solve consensus in such a way, that *all* nodes know the decision.

a) What is the smallest f such that consensus might become impossible? (Which edges fail in the worst-case)

b) What is the largest f such that consensus might still be possible? (Which edges fail in the best-case)

c) Assume that you have a setup which guarantees you that the nodes always remain connected, but possibly many edges might fail. A very simple algorithm for consensus is the following: Every node learns the initial value of all nodes, and then decides locally. How much time might this algorithm require?

Assume that a message takes at most 1 time unit from one node to a direct neighbor.