On Communication-Efficient Failure Detection in Omission Environments

R. Cortiñas I. Soraluze M. Larrea A. Lafuente

University of the Basque Country, San Sebastián, Spain

JCSD 2010

1. Context of the research
   - Failure models in fault-tolerant systems
   - Failure detectors to solve Consensus
   - Communication-efficient failure detectors
   - From the Crash model to the Omission model
   - Contribution
2. The limits of detectability
3. System Model
   - The bidirectional link abstraction
   - Well-connected processes
   - Failure detector properties
   - Communication efficiency
4. The failure detector algorithm
   - Achieving communication efficiency
   - Implementing the FD algorithm
5. Discussion

Failure models in fault-tolerant systems

- The Crash failure model
- The Crash-recovery failure model
- The Omission failure model
- The Byzantine failure model

Failure detectors to solve Consensus

- The FLP impossibility result (Fisher-Lynch-Paterson, 1985)
  - Consensus cannot be solved in asynchronous systems if at least one process can crash
- The failure detector abstraction (Chandra-Toueg, 1996)
  - Encapsulating asynchrony to circumvent the FLP result
  - Partial synchrony (Dwork-Lynch-Stockmeyer, 1988)

Failure detector classes

- A process can be *correct* or *not correct*
- For every process p, its failure detector provides a list of suspected processes
- A number of failure detector classes have been defined (Chandra-Toueg)
- We focus on the *Eventually Perfect* failure detector class: $\Diamond\mathcal{P}$
- Properties of $\Diamond\mathcal{P}$
  - Eventual Strong Completeness
  - Eventual Strong Accuracy

Implementing failure detectors

- Processes monitor each other
- Every (correct) process build a list of suspected processes
- Monitoring mechanism:
  - Polling
  - Heartbeats
- Communication pattern:
  - All-to-all
  - One-to-one (e.g., arranging the processes in a ring)

Communication-efficient failure detectors

- Communication efficiency: at most $n-1$ links used permanently (Aguilera et al, 2001)
- Communication-efficient FDs:
  - Larrea et al: DISC 2005, JS 2008, JCSD 2006
- Communication-optimal FDs:
  - Using sporadic reliable broadcast (Larrea el al: DISC 2006, JCSD 2007)
  - Using sporadic one-to-m ($m << n$) communication (Lafuente et al: PODC 2008, JCSD 2008)

The General Omission failure model

- Processes can fail by
  - Crashing
  - Omit to send messages
  - Omit to receive messages
- In the General Omission model processes suffer
  - Only send omissions, only receive omissions, or both
  - Permanent omissions or transient omissions
  - Non-selective omissions or selective omissions

Questions to be answered

- Which omissions can/cannot be detected in the General Omission model?
- How can a failure detector class be defined in the General Omission model?
- Can a communication-efficient failure detector be implemented in the General Omission model?
- How can communication efficiency be defined in the General Omission model?

Contribution

- Definition of an eventually perfect failure detector class for the General Omission model
- A communication-efficient implementation of the failure detector

The limits of detectability in the General Omission model

Problem

p sends a message to q, but q does not receive it

- a send omission of p or a receive omission of q?

- A naive solution: consider both p and q as not correct
- Instead, we focus on *well-connected* / *not well-connected* processes

The limits of detectability in the General Omission model

Problem

p sends a message to q, but q does not receive it

- a send omission of p or a receive omission of q?

- A naive solution: consider both p and q as not correct
- Instead, we focus on *well-connected* / *not well-connected* processes

The limits of detectability in the General Omission model

Problem

p sends a message to q, but q does not receive it

- a send omission of p or a receive omission of q?

- A naive solution: consider both p and q as not correct
- Instead, we focus on *well-connected / not well-connected* processes

System Model

- Failure model: General Omission
- Majority of correct processes
- Timing assumptions: *Partially synchronous*
- Reliable links
- Bidirectional communication: the *b-link* abstraction

The *b-link* abstraction

- $b\text{-}link_{p,q} \equiv b\text{-}link_{q,p}$ represents the state of the bidirectional communication between processes p and q
  - $b\text{-}link_{p,q} = Active$: p and q are exchanging messages periodically (in both directions)
  - $b\text{-}link_{p,q} = Blocked$: p and q do not exchange messages periodically (in both directions)
  - $b\text{-}link_{p,q} = Paused$: p and q do not exchange messages periodically (in both directions)
- Note that *Paused* and *Blocked* *b-link*s exhibit the same behavior (we say that the *b-link* is *not Active*)
- *Paused* and *Blocked* *b-link*s differ in how they are reached

The bidirectional link abstraction

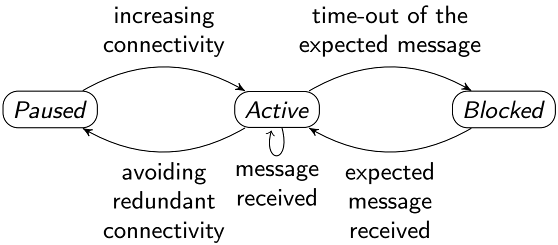
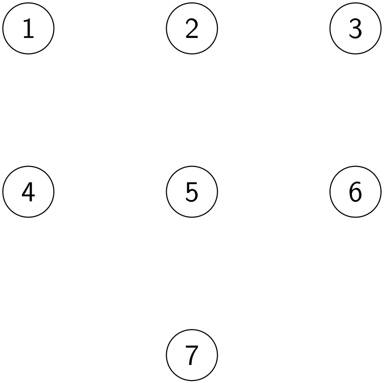


Figure: State diagram of a *b-link*.

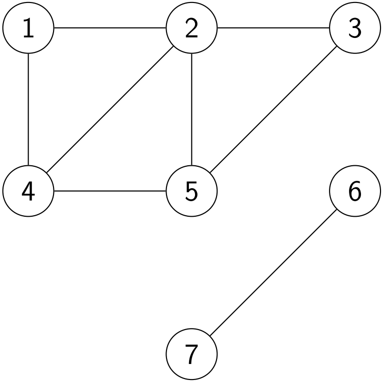
Well-connected processes

- Consider a graph of process and *Active b-links* $G = (V, E)$
- Due to crashes and omissions, G can be a disconnected graph with several connected components $S \subseteq G$
- Eventually and permanently, there will be in G a connected component S such that $|\ V(S)\ | \geq \lceil \frac{(n+1)}{2} \rceil$
- Every process $p \in V(S)$ is *well-connected*

Well-connected processes



Well-connected processes



Failure detector properties

- Strong Completeness: eventually every *not well-connected* process will be permanently considered as *not well-connected* by every *well-connected* process
- Eventual Strong Accuracy: eventually every *well-connected* process will be permanently considered as *well-connected* by every *well-connected* process

Communication efficiency

- An algorithm is *communication-efficient* in the General Omission model if it uses at most $n-1$ bidirectional links to send messages forever
- Note that in a connected graph with m nodes, exactly $m-1$ edges are needed
- In G there will be less than $n-1$ edges

Hint

- Calculate a *spanning tree* for every connected component

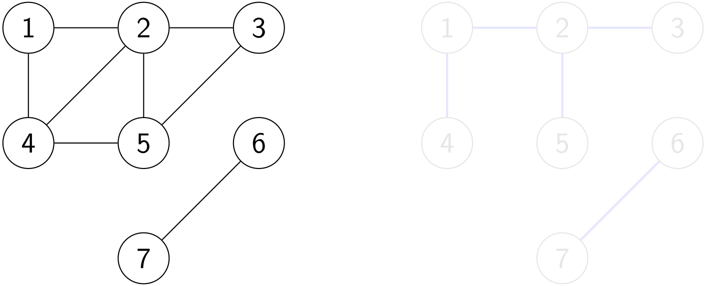
Communication efficiency

- An algorithm is *communication-efficient* in the General Omission model if it uses at most $n-1$ bidirectional links to send messages forever
- Note that in a connected graph with m nodes, exactly $m-1$ edges are needed
- In G there will be less than $n-1$ edges

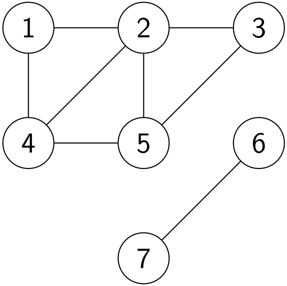
Hint

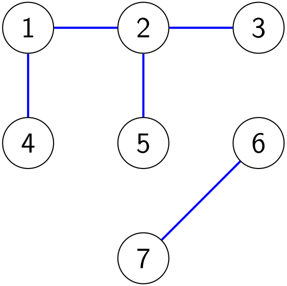
- Calculate a *spanning tree* for every connected component

Communication efficiency



Communication efficiency





Achieving communication efficiency

- Every process p computes a spanning tree T for the connected component $S \subseteq G$ it belongs to
- Using a deterministic implementation of a breadth-first search (BFS) algorithm
- If a $b\text{-}link_{p,q}$ is in S but not in T, then $b\text{-}link_{p,q}$ is set to *Paused*

Implementing the FD algorithm

- Every process p sends periodic heartbeat messeges m to the other processes
  - m includes the current connectivity information as viewed by p
- Upon the reception (or time-out) of a message m from q, a process p:
  - manages the state transition of $b\text{-}link_{p,q}$, if any
    - *Blocked* → *Active* (or *Active* → *Blocked*)
  - updates its connectivity information
  - recalculates the spanning tree for its connected component
  - updates the list of connected processes
  - manage the state transitions for its connected component
    - *Active* → *Paused* or *Paused* → *Active*
- Eventually there will be a permanent connected set including a majority of *well-connected* processes

Implementing the FD algorithm

- Every process p sends periodic heartbeat messeges m to the other processes
  - m includes the current connectivity information as viewed by p
- Upon the reception (or time-out) of a message m from q, a process p:
  - manages the state transition of $b\text{-}link_{p,q}$, if any
    - *Blocked* → *Active* (or *Active* → *Blocked*)
  - updates its connectivity information
  - recalculates the spanning tree for its connected component
  - updates the list of connected processes
  - manage the state transitions for its connected component
    - *Active* → *Paused* or *Paused* → *Active*
- Eventually there will be a permanent connected set including a majority of *well-connected* processes

Discussion

- In a previous FD algorithm for the General Omission model, we used all-to-all communication (Cortiñas et al, 2007)
- Now we have a communication-efficient algorithm with at most $n-1$ bidirectional links carrying messages forever
- What do we pay for that?
- Chandra-Toueg consensus algorithm is more dificult to adapt
  - Consensus messages are forwarded using the spanning tree
  - Connectivity should not change during a consensus round in order to avoid blocking

Discussion

- In a previous FD algorithm for the General Omission model, we used all-to-all communication (Cortiñas et al, 2007)
- Now we have a communication-efficient algorithm with at most $n-1$ bidirectional links carrying messages forever
- What do we pay for that?
- Chandra-Toueg consensus algorithm is more dificult to adapt
  - Consensus messages are forwarded using the spanning tree
  - Connectivity should not change during a consensus round in order to avoid blocking

Discussion

- In a previous FD algorithm for the General Omission model, we used all-to-all communication (Cortiñas et al, 2007)
- Now we have a communication-efficient algorithm with at most $n - 1$ bidirectional links carrying messages forever
- What do we pay for that?
- Chandra-Toueg consensus algorithm is more dificult to adapt
  - Consensus messages are forwarded using the spanning tree
  - Connectivity should not change during a consensus round in order to avoid blocking