(1) Broadcast and Consensus
(2) Epidemic Broadcast

Algorithmique répartie avancée - ARA
Master2

Luciana Arantes

Plan

- **Consensus concept and properties**
- **Terminating reliable broadcast (*protocole de diffusion fiable avec terminaison*)**
- **Epidemic broadcast**
- **Gossip-based broadcast**

Consensus

- **Specified by two primitives:**
  - *propose*
    - Processes exchange their proposal values
  - *decide*
    - All correct processes decide on a single value through this primitive

Consensus Properties

1. Termination

- Every correct process eventually decides some value

2. Validity

- If a process decides *v*, then v was proposed by some process

3. Integrity

- No process decides twice

4. Agreement

- No two **correct** processes decide differently.

Uniform Consensus

1. **Termination**
   - Every correct process eventually decides some value
2. **Validity**
   - If a process decides *v*, then *v* was proposed by some process
3. **Integrity**
   - No process decides twice
4. **Agreement**
   - No two processes decide differently.

Terminating Reliable Broadcast

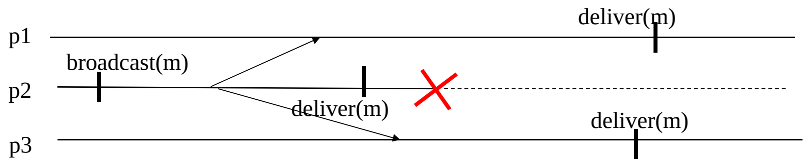
protocole de diffusion fiable avec terminaison

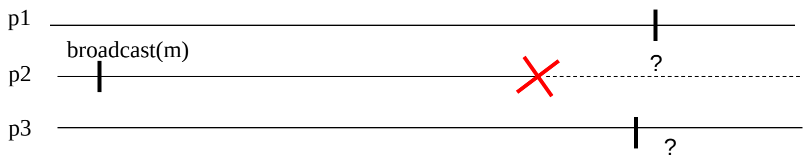
Terminating Reliable Broadcast

- **Motivation**
  - Consider that process p_i is known to have broadcast some message to all processes in the system.
    - Process p_i is an expected source of information and all processes must perform some specific task upon delivering p_i's message . Thus, all processes wait then for p_i's message.
  - The use of a *uniform reliable broadcast* will ensure that if some process deliver *m* then all correct processes will deliver *m*.
    - A process can not decide if it should wait for *m* or not.
    - Impossible for a process p_j to distinguish the case where some process has delivered *m* (p_j should wait for *m*) and the case where no process will ever deliver *m* (p_j should not keep waiting for *m*).

Terminating Reliable Broadcast

(Uniform) Reliable broadcast





Terminating Reliable Broadcast

- **Terminating Reliable broadcast (TRB) is a (uniform) reliable broadcast with a specific termination property.**
  - Ensures precisely that every process p_j either delivers a message *m* broadcast by p_i or some indication *F* that *m* will never be delivered by any process.
    - The indication *F* is given in the form of a specific message, but it does not belong to the set of possible messages that processes broadcast.
  - The TRB abstraction is a variant of consensus since all processes deliver the same message *m* or the message *F*.

Terminating Reliable Broadcast

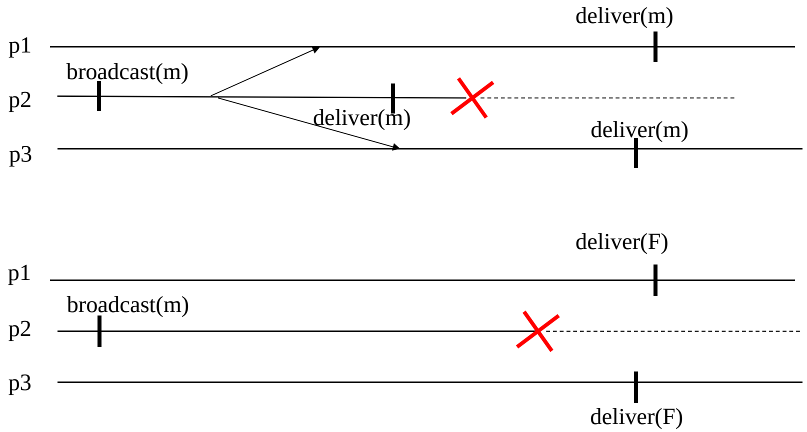
- Analogous to reliable broadcast, ***terminating reliable broadcast (TRB)*** is a communication primitive used to disseminate a message among a set of processes in a reliable way

- TRB is however strictly stronger than (uniform) reliable broadcast

Terminating Reliable Broadcast

- **src = process that broadcasts the messages**
- **Properties :**
  - *Validity*: If the sender *src* is correct and broadcasts *m*, then *src* eventually delivers *m*.
  - *Integrity* : If a correct process delivers a message *m* then either *m=F* or *m* was previously broadcast by *src*.
  - *(Uniform) Agreement* : For any message *m*, if a correct (any) process delivers *m*, then every correct process delivers *m*
  - *Termination*: Every correct process eventually delivers exactly one message.

Terminating Reliable Broadcast



Terminating Reliable Broadcast

- ***Analogous to*** reliable broadcast, correct processes in TRB agree on the set of messages they deliver
- ***Analogous to*** (uniform) reliable broadcast, every correct process in TRB delivers every message delivered by any process
- ***Contrary to*** reliable broadcast, every correct process delivers a message, even if the broadcast process *src* crashes

Terminating Reliable Broadcast

- **Algorithm**
  - Uses :
    - BestEffort_broadcast
    - Perfect Failure Detector *P* (synchronous system)
    - Consensus

  Init :
  prop = ⊥;
  correct = Π /* tous les processus */

Terminating Reliable Broadcast

Consensus-based Algorithm

TRB_broadcast (m) /* only called by src */
BestEffort_broadcast(m);

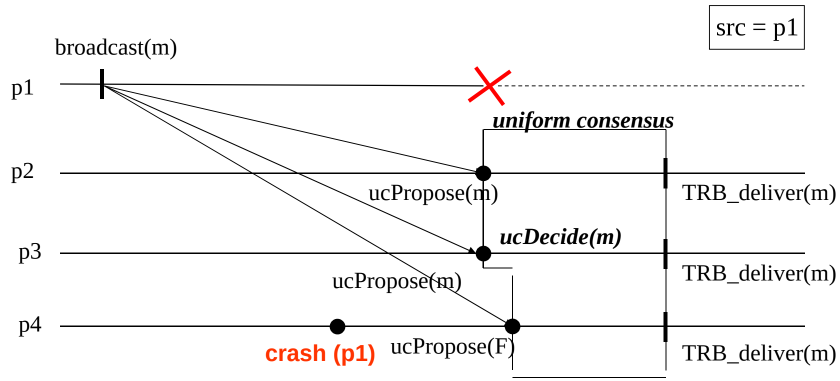
upon <crash | pj >
correct = correct \ {pj}

upon <(src ∉ correct) and (prop= ⊥)>
prop = F
ucPropose <prop>

upon < (BestEffort_deliver (m)) and (prop = vide)>
prop = m
ucPropose <prop>

upon <ucDecide, decision>
TRB_deliver (decision)
prop = ⊥

Terminating Reliable Broadcast



Epidemic Broadcast

Diffusion Epidémique

Epidemic Broadcast

- **The broadcast algorithms that we have seen till now are not scalable**
  - They consider a set of processes known by all processes from the beginning.
- **Epidemic algorithms are effective solution for disseminating in large scale and dynamic systems.**
  - They do not provide deterministic broadcast guarantees but just make probabilistic claims about such guarantees.
- **An epidemic broadcast uses a randomized approach where all the participants in the protocol should collaborate in the same manner to disseminate information.**

Epidemic Broadcast

- When a process *p* whishes to send a broadcast message, it selects *k* processes at random and sends the message to them
  - *k* is a typical configuration parameter called *fanout.*
- Upon receiving a message from *p* for the first time, a process *q* repeats the same procedure of *p*'s : *q* selects *k* gossip targets processes and forwards the message to them.
  - If a node receives the message twice, it simply discards the message
    - Each process needs to keep track of which messages it has already seen and delivered. The size of this buffer is also a scalable constraints.
- The step consisting of receiving a message and forwarding it is called a *round*.
  - An epidemic algorithm usually performs a *maximum number of rounds r* for each message.

Epidemic broadcast

- **Epidemic broadcast can only be applied to applications that do not require full reliability.**
  - The cost of full reliability is usually not acceptable in large scale systems.
  - However, it is possible to build scalable randomized epidemic algorithms which provide good reliability guarantees.
  - It exhibit a very stable behavior even in the presence of failures.

Epidemic Broadcast

- **Parameters associated with the configuration of gossip protocols :**
  - ***Fanout (k)***: number of nodes that are selected as gossip targets by a node for each message that is received by the first time.
    - Tradeoff associated between desired reliability level and redundancy level of the protocol.
  - ***Maximum rounds (r):*** maximum number of times a given gossip message is retransmitted by nodes.
    - Each message carries a *round value*, which is increased each time the message is retransmitted.
    - *Modes* :
      - *Unlimited mode:* the parameter maximum round is undefined
      - *Limited mode* : the parameter maximum round is defined with a value greater than 0.
    - Higher value: higher reliability as well as message redundancy.

Epidemic Broadcast

- **Probabilistic Broadcast**
  - **Properties**
    - *Probabilistic validity* : There is a given probability such that for any two correct processes p_i and p_j, every message broadcast by p_i is eventually delivered by p_j with this probability.
    - *No duplication* : No message is delivered more than once by a process
    - *No creation* : If a message *m* is delivered by some process p_j, then *m* was previously broadcast by some process p_i.

Epidemic Broadcast

- **Strategies**
  - ***Eager push approach*** : Nodes send message to selected nodes as soon as they receive them for the first time
  - ***Pull approach*** : Periodically, nodes query random selected nodes for information about recently received messages. When they receive information about a message they did not received yet, they explicitly request the message to their neighbors.
  - ***Lazy push approach*** : When a node receives a message for the first time, it gossips only the message identifier. If a node receives a identifier of a message it has not received, it makes an explicitly pull request.
  - ***Hybrid approach*** : First phase uses a push gossip to disseminate a message in best-effort manner. A second phase of pull gossip is used to recover messages not received in the first phase.

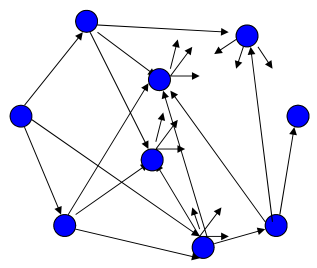
Eager Push Epidemic Broadcast

Algorithm

```
Init :
  delivered = ø

Epid_broadcast (m)
  gossip(self, m, maxrounds);

upon recv (pi, <src,m, r>)
  if (m ∉ delivered)
      delivered = delivered U {m}
      Epid_deliver(src,m)
  if (r > 0)
      gossip(self, m, maxrounds - 1);

Function chose-targets (ntargets)
   targets = ø
   while ( | targets| < ntargets ) do
        candidate = random (Π)
        if ((candidate ∉ targets) and
           (candidate != self) )
               targets = targets U {candidate};
   return targets

procedure gossip (src,msg,round)
   for i ϵ chose-targets(fanout) do
          send (i, msg, round);
```

Eager Push Epidemic Broadcast

Execution example



Fanout = 3 ; Maxround = 3

Epidemic Broadcast

- **Ideally, one would like to have each participant to select gossip targets at random from the entire system, as shown in the previous example.**
  - Realistic if it is deployed within a moderate sized cluster.
  - Such approach is not scalable :
    - High memory cost to maintain full membership information.
    - High cost of ensuring the update of such information.
- **Solution:**
  - Gossip-based (epidemic) broadcast protocols rely on *partial view*, instead of full membership information.

Epidemic Broadcast : Partial view

- **Partial view**
  - A process just knows a small subset of the entire system membership, from which it can selects nodes to whom relay gossip messages
  - The membership protocol establishes *neighboring* association among nodes.
    - It must maintain the partial view at each node in face of dynamic changes in the system membership.
      - Joining of new nodes, crashes of nodes, etc.
  - A partial view must be a tradeoff between *scalability* against *reliability*
    - Small views scale better, while large views reduce the probability that processes become isolated or that network partitions occur.
  - ***Overlay***
    - Partial views of all nodes of the system define a graph

Epidemic Broadcast : Partial view

- **Partial View Properties : related to the graph properties of the overlay defined by the partial view of all nodes**
  - *Connectivity* : the overlay should be connected : there should be at least one path from each node to all other nodes.
  - *Degree Distribution* : number of edges of the node.
    - *In-degree* of node *n* : number of nodes that have *n* in their partial view. It provides a measure of *reachability*.
    - *Out-degree* of node *n* : number of nodes in *n*'s view: measure of the importance of that node to maintain the overlay.
  - *Average Path Length* : the average of all shortest paths between all pair of nodes in the overlay.

Epidemic Broadcast : Partial view

- **Strategies to maintain partial view**
  - *Reactive strategy* : a partial view only changes in response to some external event such as a joining of a node, a crash of a node, etc.
  - *Cyclic strategy* : A partial view is update every ΔT units of time, as a result of some periodic process that usually involves the exchange of information with one or more neighbors.
  - *Mixing strategy* : the partial view membership is included in the epidemic broadcast protocol
    - Whenever a process forwards a message, it also includes in it a set of processes it knows. Process that receives this message can update its own list of known processes.
    - It does not introduce extra communication to maintain membership.

Gossip protocol in ad hoc Networks

- **An ad hoc network is a multi-hop wireless network with no fixed infrastructure**
  - Node broadcasts a message which is received by all nodes within one hop (neighbors)
- **Gossiping protocol Gossip(p)[HHL06]**
  - A source node sends the message *m* with probability 1.
  - Upon reception of *m*
    - first time,
      - it broadcasts *m* with probability *p*
      - it discards *m with* probability *1-p*
    - Otherwise it discards *m*

Gossip protocol in ad hoc Networks

- **If the source has few neighbors, chance that none of them will gossip and the algorithm dies.**
  - Solution : Gossip (p,k)
    - Gossip with probability 1 for the *k* hops before continuing to gossip with probability *p*.
      - Gossip (1,1) is equivalent to flooding.
      - Gossip (p,0) : even the source gossips with probability *p*.

Bibliographie

- R. Guerraoui, L. Rodrigues. *Reliable Distributed Programming*, Springer, 2006 .
- J. C. A. Leitão. *Gossip-based broadcast protocols.* Master thesis's. 2007.
- P.Eugster, R.Guerraoui,A. Kermarrec and L. Massoulié. *From Epidemics to Distributed Computing*. IEEE Computer, 37,pages 60-67.
- S. Voulgaris, D. Gavidia, M. Stten. *Cyclon : Inexpensive membership management for unstructured p2p overlays*. Journal of Network and System Management. Vol 13, pages 197-217, 2005.
- Z.J.Haas, J. Halpern, L. Li. *Gossip-Based Ad Hoc Routing*. IEEE Transactions on Network, Vol. 14, N. 13, pages 479-491, 2006