Distributed Algorithms

Communication Channels in Practice

24.10.2016
Processes/Channels

Processes communicate by message passing through communication channels.

Messages are uniquely identified and the message identifier includes the sender’s identifier.
Fair-loss links

FL1. Fair-loss:

FL2. Finite duplication:

FL3. No creation:
Fair-loss links

**FL1. Fair-loss:** If a message is sent infinitely often by \( p_i \) to \( p_j \), and neither \( p_i \) or \( p_j \) crashes, then \( m \) is delivered infinitely often by \( p_j \).

**FL2. Finite duplication:** If a message \( m \) is sent a finite number of times by \( p_i \) to \( p_j \), \( m \) is delivered a finite number of times by \( p_j \).

**FL3. No creation:** No message is delivered unless it was sent.
Stubborn links

1. **Stubborn delivery:** if a process $pi$ sends a message $m$ to a correct process $pj$, and $pi$ does not crash, then $pj$ delivers $m$ an infinite number of times

2. **No creation:** No message is delivered unless it was sent
Algorithm (sl)

- Implements: StubbornLinks (sp2p).
- Uses: FairLossLinks (flp2p).

upon event < sp2pSend, dest, m> do
  while (true) do
    trigger < flp2pSend, dest, m>;
  upon event < flp2pDeliver, src, m> do
    trigger < sp2pDeliver, src, m>;;
Reliable (Perfect) links

Properties

- **PL1. Validity:**

- **PL2. No duplication:** No message is delivered (to a process) more than once

- **PL3. No creation:** No message is delivered unless it was sent
Reliable (Perfect) links

Properties

PL1. Validity: If pi and pj are correct, then every message sent by pi to pj is eventually delivered by pj

PL2. No duplication: No message is delivered (to a process) more than once

PL3. No creation: No message is delivered unless it was sent
Algorithm (pl)

- Implements: PerfectLinks (pp2p).
- Uses: StubbornLinks (sp2p).
- upon event < Init> do delivered := ∅;
- upon event < pp2pSend, dest, m> do
  - trigger < sp2pSend, dest, m>;
- upon event < sp2pDeliver, src, m> do
  - if m ∉ delivered then
    - trigger < pp2pDeliver, src, m>;
    - add m to delivered;
We shall assume reliable links (also called perfect) throughout this course (unless specified otherwise)

Roughly speaking, reliable links ensure that messages exchanged between correct processes are not lost
Reliable FIFO links

✓ Ensures properties PL1 to PL3 of perfect links

✓ *FIFO*. The messages are delivered in the same order they were sent.
Algorithm (fl1)

✓ Implements: Reliable FIFO links (fp2p).
✓ Uses: Reliable links (pp2p).
✓ Relies on acknowledgements messages.
✓ Acknowledgements are control messages.
Algorithm (fl1)

✓ upon event <init> do
  ✓ nb_acks[*] := 0
  ✓ nb_sent[*] := 0

✓ upon event <fp2pSend, dest, m> do
  ✓ wait nb_acks[dest] = nb_sent[dest]
  ✓ nb_sent[dest] := nb_sent[dest] + 1
  ✓ trigger <p2pSend, dest, m>
Algorithm (fl1)

✓ upon event <pp2pDeliver, src, m> do
✓ trigger <pp2pSend, src, ack>
✓ trigger <fp2pDeliver, src, m>

✓ upon event <pp2pDeliver, src, ack> do
✓ nb_ack[src] := nb_ack[src]+1
Algorithm (fl2)

✓ Implements: Reliable FIFO links (fp2p).
✓ Uses: Reliable links (pp2p).
✓ Relies on sequence numbers attached to each message.

✓ upon event <init> do
  ✓ seq_nb[*] := 0
  ✓ next[*] := 0
Algorithm (fl2)

√ upon event <fp2pSend, dest, m> do
  √ fifo_m := ( seq_nb[dest], m )
  √ trigger <pp2pSend, dest, fifo_m>
  √ seq_nb[dest] := seq_nb[dest]+1

√ upon event <pp2pDeliver, src, (sn,m)> do
  √ wait next[src] = sn
  √ trigger <fp2pDeliver, src, m>
  √ next[src] := next[src]+1
(fl1) vs. (fl2)

✓ (fl1) uses 2 messages per applicative message.
✓ (fl1) artificially limits bandwidth if latency is high.

✓ (fl2) increases the size of messages.
✓ Sequence numbers in (fl2) have an unbounded size.
Algorithm (fl3)

✓ Implements: Reliable FIFO links (fp2p).
✓ Uses: Reliable links (pp2p).
✓ Combines acknowledgements and sequence numbers mechanisms.
✓ An acknowledgement is sent every ack_int messages received.
✓ The sequence numbers are reset when they reach ack_int x win_size.
✓ The sender has to block at the right moment.
Algorithm (fl3)

upon event <init> do

seq_nb[*] := 0

next[*] := 0

ack_nb[*] := 0
Algorithm (fl3)

✓ upon event <fp2pSend, dest, m> do
  ✓ wait ack_nb[dest] > seq_nb[dest] - win_size
  ✓ fifo_m := ( seq_nb[dest], m )
  ✓ trigger <pp2pSend, dest, fifo_m>
  ✓ seq_nb[dest] := seq_nb[dest]+1
Algorithm (fl3)

✓ upon event <pp2pDeliver, src, (sn,m)> do
  ✓ wait next[src] = sn
  ✓ trigger <pp2pSend, src, ack>
  ✓ next[src] := next[src]+1
  ✓ trigger <fp2pDeliver, src, m>

✓ upon event <pp2pDeliver, src, ack> do
  ✓ ack_nb[src] := ack_nb[src]+1
Algorithm (fl4)

✓ upon event <init> do
  ✓ seq_nb[*] := 0
  ✓ next[*] := 0
  ✓ ack_nb[*] := 0
Algorithm (fl4)

✓ upon event <fp2pSend, dest, m> do
  ✓ wait ack_nb[dest] x ack_int > seq_nb[dest] - win_size x ack_int
  ✓ fifo_m := ( seq_nb[dest] mod (win_size x ack_int), m )
  ✓ trigger <pp2pSend, dest, fifo_m>
  ✓ seq_nb[dest] := seq_nb[dest] + 1
Algorithm (fl4)

upon event <pp2pDeliver, src, (sn,m)> do

  wait next[src] = sn

  if (sn+1) mod ack_int = 0
    trigger <pp2pSend, src, ack>

  next[src] := (next[src]+1) mod (win_size x ack_int)

  trigger <fp2pDeliver, src, m>

upon event <pp2pDeliver, src, ack> do

  ack_nb[src] := ack_nb[src]+1
Fair-loss links

**FL1. Fair-loss:** If a message is sent infinitely often by \( \pi \) to \( \rho \), and neither \( \pi \) or \( \rho \) crashes, then \( m \) is delivered infinitely often by \( \rho \).

**FL2. Finite duplication:** If a message \( m \) is sent a finite number of times by \( \pi \) to \( \rho \), \( m \) is delivered a finite number of times by \( \rho \).

**FL3. No creation:** No message is delivered unless it was sent.
Stoppable Stubborn links

\textit{SL1. Stubborn delivery:} if a process pi sends a message m to a correct process pj, and pi does not crash, then pj delivers m an infinite number of times unless pi receives a stop event for m

\textit{SL2. No creation:} No message is delivered unless it was sent
Algorithm (ssl)

- Implements: StoppableStubbornLinks (ssp2p).
- Uses: FairLossLinks (flp2p).

upon event <init> do
  sending = ∅
Algorithm (ssl)

upon event < ssp2pSend, dest, m> do
add m to sending
while (m in sending) do
  trigger < flp2pSend, dest, m>;

upon event < flp2pDeliver, src, m> do
  trigger < ssp2pDeliver, src, m>;
Algorithm (ssl)

upon event < flp2p Deliver, src, m> do
  trigger < ssp2p Deliver, src, m>;

upon event < ssp2p Stop, m>
  remove m from sending
Perfect Stoppable Links

Properties

PL1. Validity: If $pi$ and $pj$ are correct, then every message sent by $pi$ to $pj$ is eventually delivered by $pj$ unless $pi$ receives a stop event for $m$.

PL2. No duplication: No message is delivered (to a process) more than once.

PL3. No creation: No message is delivered unless it was sent.
Algorithm (psl)

- Implements: PerfectStoppableLinks (psp2p).
- Uses: StubbornStoppableLinks (ssp2p).

upon event < Init> do delivered := ∅;

upon event < psp2pSend, dest, m> do
  trigger < ssp2pSend, dest, m>;

upon event < ssp2pDeliver, src, m> do
  if m ∉ delivered then
    trigger < psp2pDeliver, src, m>;
  add m to delivered;
Algorithm (psl)

upon event < psp2pStop, m> do

trigger <ssp2pStop, m>
Algorithm (fl5)

✓ Implements: Reliable FIFO links (fp2p).
✓ Uses: Perfect Stoppable Links (psp2p).
✓ Relies on acknowledgements messages.
✓ Acknowledgements are control messages.
Algorithm (fl5)

✓ upon event <psp2pDeliver, src, (sn,m)> do
  ✓ wait next[src] = sn
  ✓ if (sn+1) mod ack_int = 0
    ✓ trigger <psp2pSend, src, ack>
    ✓ next[src] := (next[src]+1) mod (win_size x ack_int)
  ✓ trigger <fp2pDeliver, src, m>
✓ upon event <psp2pDeliver, src, ack> do
  ✓ ack_nb[src] := ack_nb[src]+1
  ✓ trigger psp2pStop for all messages associated with ack
Reliable Broadcast in Practice

✓ What is the problem with (rb) on top of (beb) in practice?
  ➔ scalability
Reliable Broadcast in Practice

✓ What is the problem with (rb) on top of (beb) in practice?
  → scalability

✓ upon event <bebBroadcast, m> do
  ✓ forall pi in S do
    • trigger <pp2pSend, pi, m>
Problem with rb/beb

✓ 1 process does all the work!
✓ We need to parallelize
Algorithm (gossip)

✓ Implements: ReliableBroadcast (rb).
✓ Uses: Perfect Links (pp2p).
✓ Relies on spreading messages in a randomized way
✓ Every process forwards messages to random peers
✓ Probabilistic guarantees
  -> liveness with probability 1
Algorithm (gossip)

Upon event <init> do

1. delivered = ∅
2. while (true)
   - for each m in delivered do
     - p = random process
     - trigger <pp2pSend, p, m>
Algorithm (gossip)

✓ upon event <rbBroadcast, m>
   ✓ add m to delivered
   ✓ trigger <rbDeliver, self, m>

✓ upon event <pp2pDeliver, src, m> do
   ✓ if m \notin delivered then
     • add m to delivered
     • trigger <rbDeliver, src, m>
Gossip

Experiment