Assignment 2 Report

Yu Wang, yuwan@kth.se
Tao Jin, taojin@kth.se
2010-02-18

Exercise 1

Finding error:

a) pi is different from sm. So the startTimer (TimeDelay, pi, snm); is wrong. pi is who send the unDeliver which is itself the Unreliable Broadcast. While sm is who send the message.
b) When receiving a DataMessage from flp2p, it is only necessary to put the message in which the snm is after the current delivered message but ignore already delivered message. Otherwise, the pending has more data message. It may not be a fault. But it is not necessary.
c) The timeout event handler doesn't consider the missed messages which will block future deliver. It should deliver all pending messages and mark the lost messages.
d) When the node receives a delivery, it continues gossip for all the lost messages.

Algorithm Implementation (The changed part is marked by red ink and added green comments):

ProbabilisticBroadcast (pb).

Uses:
FairLossPointToPointLinks (flp2p);
UnreliableBroadcast (un).

```
upon event <Init> do
    for all pi ∈ Π do delivered[pi] := 0;
    lsn := 0; pending := stored := lost := ∅;
end event

procedure deliver-pending (s) is
    while exists [Data, s, x, snx] ∈ pending such that
    snx = delivered[s]+1 do
        delivered[s] := delivered[s]+1;
```

```
            pending := pending \ {[Data, s, x, snx]};
            trigger < pbDeliver | s, x >;
        end while
end procedure

function pick-targets(ntargets) is
        targets = ∅;
        if |Π | <= ntargets-1 then return  Π \{self};
        while |targets| < ntargets do
            targets := targets ∪ random(Π \{self}\targets);
        end while
        return targets;
end function

procedure gossip (msg) is
        for all t  ∈  pick-targets (fanout) do
            trigger < flp2pSend | t, msg >;
        end for
end procedure

upon event < pbBroadcast | m > do
        lsn := lsn+1;
        trigger < unBroadcast | [Data, self, m, lsn] >;
end event

upon event < unDeliver | pi, [Data, sm, m, snm] > do
        if (random() > store-threshold) then
            stored := stored  ∪  { [Data, sm, m, snm] };
        end if
        if (snm = delivered[sm]+1) then
            delivered[sm] := delivered[sm]+1;
            trigger < pbDeliver | sm,m >;
        else if (snm > delivered[sm]+1) then
            pending := pending  ∪  { [Data, sm, m, snm] };
            for all seqnb  ∈  [delivered[sm] + 1, snm − 1] do
                gossip ([Request, self, sm, seqnb, maxrounds−1]);
            end for
            startTimer (TimeDelay, sm, snm); // change from pi to sm
        endif
        for all [Data, sm, NIL, snm]  ∈lost do // continue to retrieve lost messages
            gossip ([Request, self, sm, snm, maxrounds−1]);
        end for
end event
```

```
upon event < flp2pDeliver | pj, [Request, pi, sm, snm, r] > do
    if ([Data, sm, m, snm] ∈ stored) then
        trigger < flp2pSend | pi, [Data, sm, m, snm] >;
    else if (r > 0) then
        gossip ([Request, pi, sm, snm, r − 1]);
    end if
end event

upon event < flp2pDeliver | pj, [Data, sm, m, snm] > do
    if (snm = delivered[sm]+1) then
        delivered[sm] := delivered[sm]+1;
        trigger < pbDeliver | sm,m >;
        deliver-pending (sm);
    else if snm > delivered[sm]+1 then // for already delivered do not need to add to pending
        pending := pending ∪ { [Data, sm, m, snm] };
    else // for gossip received even after the timeout, remove from lost and deliver
        while exists [Data, sm, NIL, snm] ∈ lost do
            lost := lost \ { [Data, sm, x, snm]};
            trigger < pbDeliver | sm,m >;
        end while
    end if
end event

upon event < Timeout | sm, snm > do // the Timeout event handler was re written.
    for all seqnb ∈ [delivered[sm] + 1, snm ] do
        if exists [Data, sm, x, seqnb] ∈ pending such that seqnb = delivered[sm]+1 then
            delivered[sm] := delivered[sm]+1;
            pending := pending \ {[Data, sm, x, seqnb]};
            trigger < pbDeliver | sm, x >;
        else
            lost := lost ∪ { [Data, sm, NIL, seqnb] };
        end if
    end for
end event
```

The parameters:

fanout: this parameter is how many processes the algorithm will gossip for lost messages which will impact the recovery phase. The more it is, the more chance a message can be retrieved.
store-threshold: this parameter is the chance of storing a received message which also impact the recovery phase. The more it is, the more chance a lost message can be retrieved.
maxrounds: this parameter is the max rounds of a gossip can have. It also impacts the recovery phase. The more it is, the more chance a lost message can be retrieved.

The scenario discussion:

(a) No message is lost (all nodes deliver a broadcasted message m).
If the lost rate for the flp2p link is 0 and all nodes are fully connected, then no message is lost

(b) A broadcasted message is lost in the unreliable broadcast but recovered by gossip for some node p.
If all nodes are fully connected and fanout>0, store-threshold<1 and maxrounds>0, then **eventually** lost message will be recovered.

(c) A broadcasted message is lost such that although it is stored on some node(s) in the network, a node p missed it in the unreliable broadcast and furthermore, p could not retrieve it via gossiping as well.
For the algorithm on the book, it is possible when the time out for gossip is smaller than the link delayed. However, in our algorithm, we record the lost message and continues to gossip. So like scenario (b), lost messages will eventually be recovered.

(d) A broadcasted message, after being delivered by some node(s) and missed by a node p, is completely lost such that p can never retrieve it through gossiping.
When the broadcasted message is the last broadcast, then node p will never know it missed a message. Or when the store-threshold is 1 then no message is stored.

Exercise 2

For not fully connected topology, the algorithm cannot work on that because the broadcast cannot be delivered by all nodes especially for the node not linked with others. For example, node p is not connected to node q, when node p broadcasts, node q will not receive it. Even worse, node q doesn't know p broadcasts so q will also not gossip for it.

The Implementation for Unreliable Broadcast on not fully connected topology. The basic idea is to forward delivered message to all neighbors except source and self and ignore the already delivered message on deliver event.
UnreliableBroadcast (un).

Uses:
FairLossPointToPointLinks (p2p).

upon event <Init> do
    for all pi ∈ Π do delivered[pi] := ∅;
    lsn := 0;

```
end event

upon event < unBroadcast | m > do
    lsn := lsn + 1;
    for all pi∈Π  do
        trigger <flp2pSend | pi, m, lsn >;
    end for
end event

upon event < flp2pDeliver | pi, m, lsn > do
    if lsn∈delivered[pi] then
        return;
    else
        delivered[pi] := delivered[pi]  ∪  {lsn};
        trigger <unDeliver | pi, m >;
        for all pj∈Π \{pi}\{self} do
            trigger <flp2pSend | pj, m, lsn >;
        end for
    endif
end event
```