

Distributed Algorithmics – TD1 - M2 IFI, Ubinet-CSSR, English version, Oct. 2013 ELEMENTS of SOLUTION 2013

Exercise 4 – INCOMPLETE CORRECTION and SOME EXPLANATIONS

A wireless sensor network is being used to monitor the maximum temperature in a region (current temperature, ie goal is not to collect the highest temperature even from the past). Each node monitors the temperature of a specific point in the region. Propose an algorithm for computing the maximum temperature and broadcasting the maximum value to every sensor node. Assume that communication is by broadcasting only, and the broadcasts are (NOT?) interleaved (i.e., they do not overlap), so two sensor nodes never send or receive data at the same time.

First, my own interpretation of the assumptions, as I directly proposed the original exercise as it appears in the book of Ghosh. “*Broadcasts are interleaved, ie they do not overlap, so two sensor nodes never send or receive data at the same time.*” could mean a lot of things! I think that the most reasonable is to assume we are in the Atomic Model of Mattern. So, when a process is executing the code guarded by a specific guard, which can contain a broadcast operation, this broadcast is atomic in the sense that when you trigger the broadcast, it is immediate, and unblocks the one that is broadcasting immediately. So, the process finishes its code also immediately (because remember the action is atomic). Then, it can come again to check if another guard is true, which can be the case for a guard containing a statement like “a message has arrived”. In this case, it is clear that while sending it can not be receiving a message at the same time !

Any other interpretation could be as valid as the one I just gave. So, I was not strict about any proposition you made.

A first, simple solution under this assumption of atomic model, and described using the notation we suggested to use in the first course, is:

```
Var mytemp;
{Max=undefined} : Max := mytemp; broadcast(Max);

{A message <j> has arrived}
  If j > Max, then Max=j; // and you do not need to bcast that because all other
//processes will receive also a message <j> so, eventually, they will also know <j>

// As the maximum temperature is just an estimation of the larger one in the region, you can
// assume that this is done. If you print the temperature on a screen, you will update its printed value
// regularly each time you receive a greater value. But what happens if Value Max decreases...
```

Something which I thought was clear from the subject, even if totally implicit: you cannot assume you know the size of the network, because a network of sensors is usually a bit volatile, and contains lot of sensors. This is why we adopted an approach following the Approximation Method. HOWEVER... none of you had the reflex (which I myself had!) to assume that the temperature is something that is totally fluctuating. So, when you have computed an estimation of the highest temperature in the region, you should include in your algorithm what is needed in case your own temperature changes. And in this case, your algorithm must be able to update the maximum temperature of the region (including the more tricky case where the maximum temperature is now going lower than the preceding one). In this case, the exercise really becomes interesting and more challenging because it

does not follow simply an approximation scheme, where the approximation allows each participant to end up with the knowledge of a shared, common stable, and non-changing value (like which process has the maximum ID in the network)! Don't forget the fact also that several sensors may sense the same value. Below is a proposition that is trying to handle the case, but not perfectly however.

```

Var mytemp; oldTemp;
Var Max, initially undefined, like mytemp, oldtemp. Or alternatively, we can initialize with a non
realistic value, like -10000

{Max=undefined} : Max := mytemp; broadcast(Init, Max); oldTemp=mytemp;
// this guarded code could probably be merged with the next guarded code
// but is kept separate here for clarity= it is the initialization code

{oldTemp <> mytemp} : // means my own temperature has just changed, as my sensor tells me

    If (mytemp >Max) then Max=mytemp ; broadcast(Update, Max) ;
                        // my new temp can make the highest one in the region increase
    If (mytemp < oldTemp) && (oldTemp == Max) then // I may be guilty of the fact that
// the maximum temperature in the region may now decrease, because my old temp
// was == to the Max, but now my temperature has decreased.
        Max = mytemp;
        broadcast (Update,Max); // I broadcast the potentially new max; but what happens if
another has still the Max as its 'mytemp' value ? In this case, my approximation of the Max is now
incorrect. So we will need a correcting action from the process identifying this case (see next guard)

        oldTemp=mytemp; //Thanks of this assignement, I just run this action once whenever my
temperature changes

{A message <j> has arrived and is tagged "Init"}: // I receive a msg from the first time from a process
    If (j>Max) Max=j; // improve the knowledge about the Max – ie , my current approx. of the
//highest value was not yet precise enough.
    //For the homework: Check here what happens if Update messages –not from this same
process, by FIFO channels assumption – have already been treated?

{A message <j> has arrived, and is tagged "Update"}: // I receive an update msg (from Pi in the
sequel) that may :

    //1)either improve the knowledge about the Max – ie , the Max has effectively increased
    If (j>Max)
        Max = j;

    else if (j<=Max) && (mytemp>j)
        //2) Or, I must inform the others that despite Pi thinks it has the highest value,
        // and despite the fact that my own value has not yet changed,
        // I'm now the one which may have the highest value in the region
        Max=mytemp;
        broadcast(Update,Max); //note that Pi will receive this Max, and correct its own value
    else // j<=Max but this value j is to be considered effectively as the new max
        Max= j;

```

In some situations due to message transmission indeterminism, this algorithm is incorrect. Because, if 2 update messages are broadcasted at the same moment like “Update, 11” and “Update, 14” (so the right maximum temperature in the region is indeed 14), one process (with value < 11) may receive 11 then 14, so the Max on it is equal to 14 which is correct. But, another process (with value < 11) may receive 14 and then 11, and will keep 11 (last else case), which is incorrect.

So, we need to find another, probably ad-hoc, approach.

Or, we need to add some assumptions about the communication network behavior.

Exercise 4 BIS– for MONDAY October 7 2013

Propose a correct solution –if any! (There might be a genius student in the class!) Or, study the effect of the following assumption on the given algorithm. I.e., does the following assumption allows the algorithm to become correct, and why (or why not if you believe this assumption is not sufficient)?

ADDITIONAL ASSUMPTION: when a broadcast is initiated by any process, the resulting message gets always treated at each receiver node in the same order. Say another way, if two processes broadcast “at the same time”, the 2 sent messages are always ordered in the same order in any receiver queue. Start from a situation where two sensor nodes detect in parallel that their ‘mytemp’ variable has changed.

Additional question: do you think the differentiation between Init and Update messages is worth keeping. Justify your answer. Say another way, is it correct to only tag broadcasted messages with Update and treat them as proposed by the given algorithm, and if yes, what is the impact on the resulting performance (counted as total number of exchanged messages) ?

Proposed Answer

The assumption (see ADDITIONAL ASSUMPTION) about total order broadcast property (even if kept anonymous!) ensures that the measured maximum temperature is consistent over the whole sensor network. Consistent, but not mandatorily perfectly correct!! Let us start with explaining why it might be still incorrect.

Remember (from Group Communication course) that it could happen that total order is not compliant with FIFO order. On our example, it could happen that the maximum temperature rises up to 14 (on one process), then latter changes to 11 (on this same process, so it broadcasts a second message “Update 11” to all, and keep max=11). But, “Update 11” message gets treated on each sensor node (so, if the max was previously 7 on the whole network, all processes increase it to 11), before “Update 14” message, due to total order group communication delivery order. Because “Update 14” message is greater than 11, all processes, including the process on which it had effectively measured 11, gets updated to 14. So the measured max temperature is consistent on the whole network, but indeed it is incorrect (as it is not 14 but 11 indeed)!

The problem only happens if the two messages are originating from the same process.

If the two Update messages are concurrently emitted (it is the situation I suggested to start from) => whatever the delivery (same) order on each process, the process holding the highest temperature will rebroadcast its own, so the max value held by any process will be the right one. As an example: max temperature was 7. Then, concurrently, it rises to 11 at some site, and to 14 at another site. Assume the delivery total order is 11 then 14: all sites where the max was 7 update their max to 11. On the site where it is now 14, receiving an “Update 11”

message will inform all the others that 14 is the right maximum temperature. So eventually, only value of 14 will be kept on all sites.

Moreover, you can notice that the second “Update 14” message is a direct consequence of having processed “Update 11” message on the site where the temperature is indeed 14. So, whatever the total order delivery (which affects only concurrent messages broadcast on the group), there is no risk at all that the two “Update 14” messages get delivered before the “Update 11”. Consequently, it seems that the proposed solution correctly handles the temperature modification, except in the case discussed at first.

For this first case, a correct solution requires a FIFO order on the channel on which a same emitter puts the message to be delivered by the group communication primitive.

Having point-to-point FIFO communication channels is realistic. If not possible to make this assumption, the only solution is to use a sequence numbering for each channel. (like TCP which relies upon such technic to detect packet disordering). If any process receives a message from any other one, and from the associated sequence number can detect that one message is missing in the sequence, it can wait the missing one. Be careful where to implement this additional waiting, as it should not imply that delivery order is not total within the group of processes! We could assume this is achieved within the underlying group communication primitive (see the correction of exercise 3 from the Group Communication course, where we add such assumption to a total order broadcast protocol).

Additional question: there is an interest of using Init message. When an “Init x” message is received by a process that has a temperature y s.t. $y > x$, it does not trigger the broadcast of “Update y” messages that are useless. Indeed, the “Init y” messages have also been received by all processes (eventually), and have allowed to keep ‘y’ as the maximum temperature if relevant. Avoiding the Init message type and simply using the Update one, would have triggered many rebroadcast operations: from any process receiving a message not in line with known max and own temperature, according to the corresponding guard/action code.

Having Init messages is thus saving lot of initial and not useful broadcasts.. More precisely, if the N-1 initial temperatures except the maximum one, are broadcast and then the process owning the maximum one is “replying” by broadcasting back its own, you see that these N-1 additional broadcasts are useless.