SENSOR NETWORK USING TELOSB AND MICAZ SENSORS

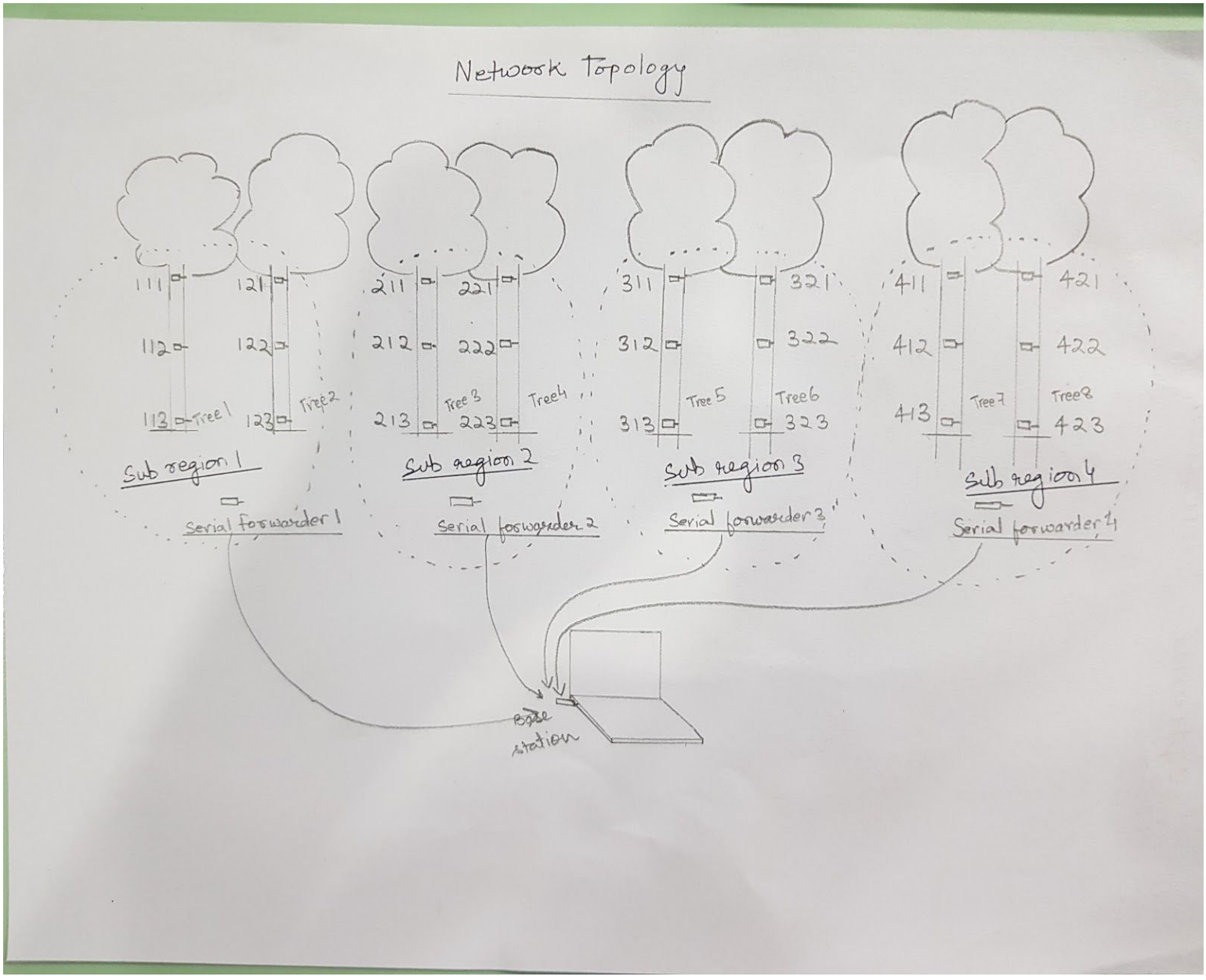
Monitoring a habitat is one of the most important applications of any embedded sensor network. We can use a sensor network for monitoring wildlife patterns in a particular region, for monitoring soil quality and

For this project we have to design a network of TelosB and MicaZ sensor motes to collect and send the temperature and humidity data values at various heights on a tree. These values should be accessible to us remotely. The user can also pole any random mote remotely to get its temperature and humidity value at any point in time.

We need to decide on the -

1. Network topology
2. Time synchronization
3. Routing
4. Communication protocol

Network Topology



Since the entire region was too big to cover we decided to split it into 4 subregions, each comprising of 2 trees. Each tree will have 3 nodes on it, each separated by 7 meters.

The bottom most node of each tree acts as a cluster head for that particular tree. The above two nodes give data to it.

The numbering of the nodes will be XYZ - where X is the subregion, Y is the tree in that subregion (either tree 1 or tree 2) and Z is the level for that node on that tree (either 1, 2 or 3).

For example - node number 413 refers to the bottom most node on tree number 1 in subregion 4. This is how it has been referred to in the code as well.

Each region further has a serial forwarder, which is a MicaZ node, but it's purpose is not to sense the temperature and humidity but forward the data that it gets from the trees to the base station.

The basestation is again a TelosB mote which is attached to a computer that has an internet connection. This node receives the data from all of the serial forwarders and send it to a server. Anyone who wishes to access the data can then extract it from the server via our website. Our website is called habitatmonitor.esy.es

Time Synchronisation

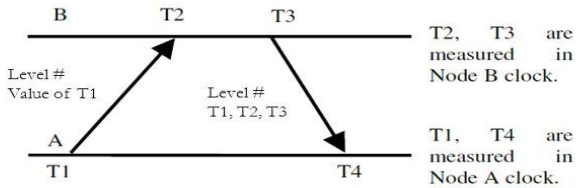
Time synchronisation is important in this case because all the nodes send data according to a TDMA algorithm and their times need to be synced to avoid collision of packets.

We used TPSN algorithm for time synchronisation. We consider the base station as the Level 0 node, Serial forwarder as level 1 node, and node on the trees as level 2, level 3 and level 4 nodes according to their heights respectively.
Our system performed time synchronisation every 5 minutes.

The algorithm works something like this in theory -

Error Analysis: TPSN & RBS

- Analyze sources of error for the algorithms
- Compare TPSN and RBS
- Trade-Offs



$$\Delta = \frac{(T2 - T1) - (T4 - T3)}{2}$$ Clock drift

The base station as Level 0 node which has access to the UTC, first coordinates time with the Serial forwarder, which in turn coordinate with the cluster heads that synchronise time throughout the tree.

Code explained -

We have set 3 stages for reception of packages.

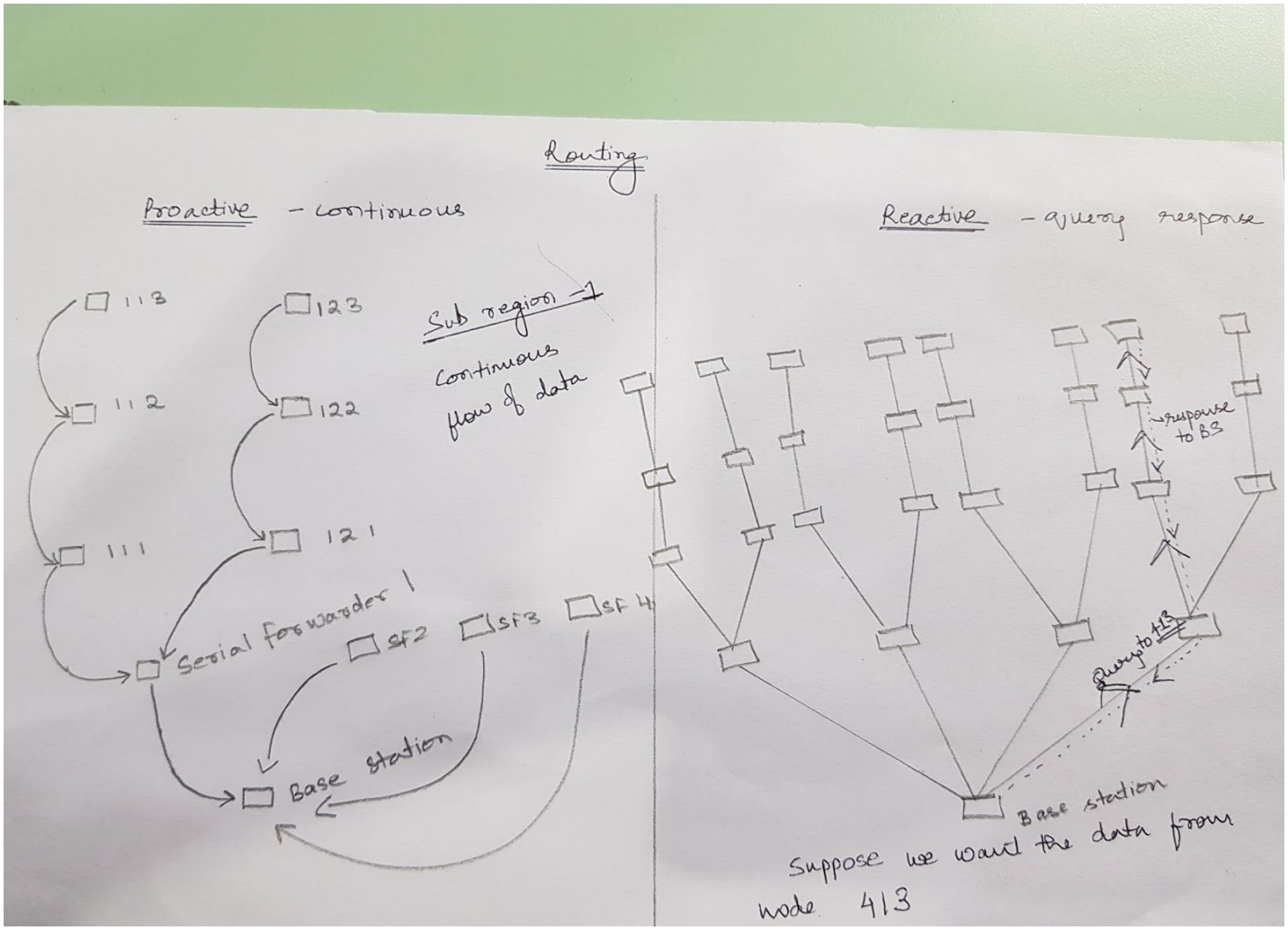
1. Stage 0 - If a node receives a package with stage 0, this means the node is receiving a time sync message from the node just below it.(A in terms of the figure). Then the node sends T2 and T3 to the node below.
2. Stage 1 - If a node receives a package with stage 1, it is receiving the time sync packet from the node below (A) with calculated phase and propagation delay values that it can use.
3. Stage 2 - If a node receives a package with stage 2, it is receiving a message from a node above it (B in terms of figure), so it has to calculate phase and propagation delay and send it back.

The figure below is an example of the time synchronisation happening between serial forwarder 1 (SF1) and cluster head of tree 1.
We can clearly see stage 0 and stage 1 processes happening when the cluster head has to send T2 and T3 to SF1 in case of level 0. And it has to take it the prop and phase delay calculated by SF1 in case of level 2.

```
152
153     event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len)
154
155     {
156
157             call Leds.set(2);
158         if(len==sizeof(TimeSync))
159         {
160
161
162             //IMPORTANT NOTE ABOUT TIMESYNC->stage
163             //IF IT IS 0 IT MEANS NOTE DOWN T2,T3 AND SEND DOWN
164             //IF IT IS 1 IT MEANS YOU HAVE RECEIVED PROP AND PHASE DELAYS
165             //IF IT IS 2 IT MEANS YOU HAVE RECEIVED T2,T3 AND MUST NOTE DOWN T4 AND CALCULATE PROP AND PHASE DELAYS
166
167             TimeSync* btrpkt = (TimeSync*)payload;
168             TimeSync* newpkt = (TimeSync*)(call Packet.getPayload(&pkt, sizeof (TimeSync)));
169
170             //FIRST OF ALL YOU WILL ONLY DO TIME SYNC WITH NEIGHBOURS
171             if(btrpkt -> nodeid_t == 1112 || btrpkt -> nodeid_t == 1100)
172
173             {
174                 //IMMEDIATELY SET YOUR NODE ID IN THE PACKET YOU WILL SEND AS OTHER NODES NEED TO KNOW WHO YOU ARE
175                 newpkt -> nodeid_t = 1111;
176
177                 //YOU HAVE TO NOTE DOWN T2,T3 AND SEND DOWN
178                 if(btrpkt->stage==0 && btrpkt->nodeid_t==1100)
179                 {
180
181                     newpkt->T2 =call LocalTime.get();
182
183                     //THE MOTE ABOVE SHOULD KNOW THAT YOU ARE SENDING T2,T3 DOWN
184                     newpkt->stage =2;
185
186                     if(!busy)
187                     {
188                         if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(TimeSync)) == SUCCESS)
189                         {
190                             call Leds.set(1);
191                             newpkt->T3 = call LocalTime.get();
192                             busy=TRUE;
193                         }
194                     }
195
196                 }
197
198
199                 //YOU HAVE RECEIVED PROP AND PHASE DELAYS FROM NODE BELOW
200                 if(btrpkt->stage == 1 && btrpkt->nodeid_t==1100)
201                 {
```

```
200                 if(btrpkt->stage == 1 && btrpkt->nodeid_t==1100)
201                 {
202
203                     call Leds.set(7);
204                     phase_delay=btrpkt->phase_delay;
205                     prop_delay=btrpkt->prop_delay;
206                 }
207
208
209                 //YOU HAVE RECEIVED T2,T3 FROM NODE ABOVE AND MUST IN TURN SEND IT PROP AND PHASE DELAYS
210                 if(btrpkt->stage ==2 && btrpkt->nodeid_t==1112)
211                 {
212
213                     T4=call LocalTime.get();
214                     T2=btrpkt->T2;
215                     T3=btrpkt->T3;
216                     newpkt->phase_delay = ((T4-T3-T2+T1))/2;
217                     newpkt->prop_delay = ((T4-T3)+(T2-T1))/2;
218                     call Leds.set(3);
219                     //THE MOTE ABOVE SHOULD KNOW THAT YOU ARE SENDING PROP DELAYS
220                     newpkt->stage =1;
221
222                     if(!busy)
223                     {
224                         if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(TimeSync)) == SUCCESS)
225                         {
226                             call Leds.set(7);
227                             busy=TRUE;
228                         }
229                     }
230                 }
231         }
232     }
```

ROUTING


Routing
Proactive - continuous
113
123
112
122
111
121
Sub region
Continuous flow of data
Serial forwarder 1
SF2
SF3
SF 4
Base station
Reactive - query response
response to BS
query to 413
Base station
Suppose we want the data from node 413

Two types of routing are used here -

1. Proactive routing
2. Reactive routing

Proactive routing - This is the routine routing that happens for the periodic data collection. The path for proactive routing is fixed. The topmost node of any particular tree(113) sends data data to the middle node(112) which aggregates the its own data with that of 113 and sends it to the bottommost node (111). 111 also aggregates the data received from 112 with its own data and sends it to the serial forwarder whose only job is to forward that data to the base station. The MAC protocol being used in this process is TDMA based to ensure that the packages don't collide at the node.

Reactive routing - This type of routing is ad-hoc. The query message is flooded by the base station to all the serial forwarders. The serial forwarder will find out whether the requested node is part of its subregion and then send the query to the bottomost node. The bottomost node will forward the query to the required node (if it's not the intended recipient itself). Then the intended node forwards the data via the same path to the base station.

DATA AGGREGATION AND ROUTING

We used a combination of a TDMA for data aggregation at the clusterhead level and then transfer the data forward through multihop routing.

Each node in this case is given a particular time slot, only during which it can collect and send its data to the next node. For example the node 111 can only collect and send data between 0 - 400mSecs. The node 112 can collect, aggregate and send data between 400 - 800mSecs and similarly node 113 can collect, aggregate and send data between 800 - 1200mSecs.
So node 113 will send its data 112 during it's time slot. Node 112 will aggregate and send the data to node 111. Finally node 111(cluster head) will aggregate send that data to the serial forwarder. We implement this functionality using the inbuilt timer function after time synchronisation is established in the network.

The cluster heads aggregates the data that they receive from the top two nodes of the cluster using TDMA. Then the cluster heads send this data to the base station using a multihop routing algorithm, wherein it sends the data to the serial forwarder which finally forwards it to the base station.