```
/**
 * This file contains functions that implement asymmetric neighbor discovery.
 *
 * PROBLEM- Given 3 nodes with periods t1=10, t2=8, and t3 = 10, and that node 1 and 2
 * happen to discover each other first. Secondly, node 2 and 3 discover each other.
 * Q: How can node 3 indirectly discover node 1 using information received from node 2 ?
 * A:
 * (1) When nodes 1 and 2 discover each other, each node computes the offset to the other
 * D_ij = Poffset(i)-Poffset(j).
 * IF D_ij < 0 THEN
 *   D_ij = D_ij + tj
 *  In realist this happens in two fucntions: input(), and neighs_register(a,b,c).
 *  The first function is called from device driver indicating the reception of a packet.
 *  The second fucntion is called inside inpput() to "ADD THE CONTENTS OF THE RECEIVED PACKET"
 *  At "neighs_register", D_ij is computed.
 *
 * (2) HOW WE UDATE K ? - neighs_jfactor_update()
 * INITIALLY: the offset is D_ij, however, for each following anchor slot each node computes
 * its offset to neighbors as.
 * ta_i - node_i anchor slot time/position
 * ta_j - node_j anchor slot position/time
 * t_i  - period of node_i
 * t_j  - period of node j
 *
 * IF MY ANCHOR TIME ta_i IS AHEAD OF NEIGHOR anchor time ta_j:
 * IF ta_i > ta_j:
 *   IF t_i <= t_j:
 *        K_ij = K_ij + 1
 *   else:
 *        K_ij = K_ij + t_i/t_j
 *
 * (3) Which offset we send ? - neighs_jfactor_update()
 * WE COMPUTE a new estimative of ta_j..
 * IF hop_count_ij == 1 AND ta_j > ta_i AND t_i NOT EQUAL t_j:
 *   D_ij(send) = ta_j - ta_i
 *
 * (4) Indirect offset... IMPLEMENTED IN FUCNTION: neighs_register(a,b,c)
 * There is a discovery 2 and 3 and node 2 conveys information about node 1 to node 3.
 * HERE again we start by computing the offset between nodes 2, and 3 (see case 1)
 * ..SECOND: we compute the indirect offset: D_iz
 * D_iz = Poffset(i) - Poffset(j) + Djz (conveyed offset information)
 * Offset less than zero
 * IF D_iz < 0:
 *      D_iz = D_iz + tz
 * ...This is IMPLEMENTED IN FUCNTION: neighs_register(a,b,c)
 */

/**
 * At every time slot, each node updates its offset to all its 1-hop neighbors
 * This is done at the main thread function.
 */
static char power_cycle(struct rtimer *rt, void* ptr){


   //start protothread function
   PT_BEGIN(&pt);

   while(1){

        //This part is used to implement the random offset.. During the offset
        // slot, a node does NOTHING... once this period ends, the algorithm begins
        if(rand_offset_wait_flag){

           for(slot_counter = 0; slot_counter < node_slots_offset; slot_counter++){
               rtimer_clock_t tnow_wait = RTIMER_NOW();

               schedule_fixed(rt, tnow_wait + TS);
               PT_YIELD(&pt);
          }
```

```
          //random slots offset is over.. disable flag
          rand_offset_wait_flag = 0;
          COOJA_DEBUG_PRINTF("END oF RANDOM OFFSET:%u..%u\n", rounds_counter,node_slots_offset);
        }//else{}

        static rtimer_clock_t t0;
        //rtimer_clock_t tnow;

        //for(slot_counter = 0; slot_counter < slot_counter_upperB+2; slot_counter++){
        for(slot_counter = 0; slot_counter <= slot_upperBound; slot_counter++){

            //this offset is transmitted
            probe_offset = slot_counter%period_length;

            /// ======> Mikael and Themis <==============
            ///@note: HERE EVERY t, we call the neighs_jfactor_update() function.
            /// This function is used to implement the offset computation.
            if(slot_counter != 0 && (slot_counter % period_length) == 0){

                  ///anchor time is computed right here..
                  anchor_time = slot_counter - probe_offset;

                  //update j coeficient of neighbor nodes
                  neighs_jfactor_update();
            }

            //....... Remaining part omitted.....................................
/**
 *
 *
 */
void neighs_jfactor_update(){
    struct nodelist_item *localp = NULL;

    localp = list_head(neighs_list);

    for( ; localp != NULL; localp = list_item_next(localp)){
        if (localp->node_id != rimeaddr_node_addr.u8[0]){

            //local variable enables tracking offset to neighbors
            ///get the anchor time here
            uint16_t ta_i = get_anchor_time();
            ///compute the anchor time of the neighbor
            uint16_t ta_j = time_neighbor_anchor(localp);

            ///get my period here
            uint16_t tp_i   = get_node_period();
            /// get the period of the neighbor here
            uint16_t tp_j   = localp->period;

            ///if my anchor is ahead of my neighbor's annchor and that
            if(ta_i > ta_j){
                ///and my period is larger than my neighbor's period..
                ///increment j_factor (k)
                /// X_ij = ta_i + D_ij + k*tj
                ///HERE we are just incrementing K
                if (tp_i <= tp_j){
                    localp->j_factor = localp->j_factor + 1;
                }else{
                    localp->j_factor = localp->j_factor + tp_i/tp_j;
                }
            }

            ta_j = time_neighbor_anchor(localp);

            ///WHICH OFFSET >TO SEND ?
            ///if my period is different to neighbor's period and anchor of my neighbor
            ///is ahead of my anchor.. compute the new difference between anchors
            ///Dij_send = ta_j - ta_i
            if ((localp->hopcount == 1) && (ta_j > ta_i) &&  tp_j != tp_i){
```

```
                uint8_t offsetnode = ta_j - ta_i;
                //set the new offset for broadcast.
                localp->offsetj    = offsetnode;
            }
        }
    }
}
}
/** This FUNCTION is used to compute the time of my neighbor anchor and
 * returns it.
 */
uint16_t time_neighbor_anchor(struct nodelist_item *n_item){
  uint8_t  periodL = n_item->period;
  uint16_t ta = n_item->t_anchor + n_item->offset + periodL*n_item->j_factor;

  return ta;
}
/**Whenever we receive a packet, we know that we have an OVERLAP.. therefore,
 * we call this function to compute the the OFFSETS and update other variables.
 * FUNCTION CALLED from DEVICE driver indicating that a packet have been received
 * and IT IS on the receiving buffer
 */
static void input(){
  int len = 0;

   //read packet from the radio driver
   ///used as a MUTEX to PREVENT  send and receive
   radio_read_flag = 1;

   len = NETSTACK_RADIO.read(packetbuf_dataptr(), PACKETBUF_SIZE);

   radio_read_flag = 0;

   ///packet NOT VALID
   if(len <= 0){
       return ;
   }

   ///access the data BUFFER
   data_packet_t *inpkt = (data_packet_t*)packetbuf_dataptr();

   ///CHECK IF THE PACKET TYPE IS OK!
   if((inpkt->type & 0x0F) == DATA_PKT){

       //get the discovery time..
       discovery_time = node_slots_offset + slot_counter + 1;

       ///add new nodes to the list
       //neighs_sregister(&inpkt->data[0]);

       //add new nodes
       ///HERE...OVERLAP FUCNTION..
       ///We CALL this function to compute the OFFSETS to our 1h neighbor
       ///and to all possible 2-HOP neighbors he might have.
       neighs_register(inpkt, len-DATAPKT_HDR_LEN, probe_offset);

       ///returns the number of 1 hop neighbors
       uint8_t tmp_num_neighs = neighs_xhops(1);

       ///If there is NO UPDATE in the number of 1 hop neighbor, do not print anything
       ///check if we have discovered all our neighbors
       if(curr_frac_nodes < tmp_num_neighs){

           ///set current number of neighbors
           curr_frac_nodes = tmp_num_neighs;

           /////discovery_time = node_slots_offset + slot_counter + 1;

           process_post(&output_process,PROCESS_EVENT_CONTINUE, NULL);
       }
```

```
   } ///inpkt->type == DATA_PKT
}
/** @note: HERE D_ij IS Computed
 *
 */
uint8_t
neighs_register(data_packet_t *pkt_hdr, int pldLen, uint8_t probe_counter){
    uint8_t k = 0;
    int16_t offsetH1, offsetH2;

    //computes the offset to the sender of the packet.
    /// HERE.... D_ij
    int16_t ownOffset  = ((int16_t)probe_counter - pkt_hdr->offset);

    //uint8_t periodLength = compute_node_period(pkt_hdr->energy);

    offsetH1 = ownOffset;

    //if the offset is negative, we compute the positive offset by summing
    //the sender's period length.
    /// D_ij < 0, we and sender period..
    if (ownOffset < 0){
          offsetH1 = pkt_hdr->period + ownOffset;

          //COOJA_DEBUG_PRINTF("dc:%u\n",compute_node_period(pkt_hdr->energy));
    }

    //go though all items in the packet payload and add them accordingly..
    for ( k = 0; k < pldLen; k++){

        uint8_t dpos = k*DATA_ITEM_LEN;

        struct data_item_t *ditem = (struct data_item_t*)(&pkt_hdr->data[dpos]);

        //filter packets based on hop-count number, remove also my id
        if ((ditem->node_id != 0 && (ditem->dc_hopc & HOPC_MASK) ) &&
                        (ditem->node_id != rimeaddr_node_addr.u8[0]) &&
                       (ditem->dc_hopc  <= MAX_HOPCOUNT)){

             //COOJA_DEBUG_PRINTF("rcd %u,%u,%u,%d,%u, %d\n",ditem->node_id,
            //ditem->hopcount, ditem->offset,ownOffset, offsetH1, periodLength);

             struct nodelist_item *nli = NULL;

             //check if the nodeID is already registered/received..
             nli = neighs_get(ditem->node_id);

             if(nli == NULL){

                 //we allocate memory for a new element..
                 nli = memb_alloc(&neighs_memb);

                 if(nli != NULL){
                     //set the id of this node
                     nli->node_id    = ditem->node_id;
                     //extract the hopcount..
                     nli->hopcount   = ditem->dc_hopc;
                     //we do not know yet :)
                     nli->max_txhop2 = 0;

                     //get the time of reception
                     nli->tknown     = get_discovery_time();
                     //equal time of confirmation if node is discovered
                     //without help of epidemics
                     nli->tconfirmed = get_discovery_time();

                     //if node is received as hop=2, jfactor is what allows
                     //to locate it.. we explain later..
                     nli->j_factor   = 0;
```

```
                //what is the anchor time when this node was received
                nli->t_anchor   = get_anchor_time();

                //retrieve the period of node.
                nli->period = ditem->period;

                ///@TODO add something if offset greater than T/2
                //offset = T-offset.

                ///INITIALLY WE HAVE JUST discovered a neighbor,
                ///the offset to send is equal to the offset..
                if(ditem->dc_hopc == 1){
                     nli->offset  = offsetH1;
                     nli->offsetj = offsetH1;
                     COOJA_DEBUG_PRINTF("%u ADD_d(%u) offset:%2u(%d)\n", rimeaddr_node_addr.u8[0],
                    nli->node_id, offsetH1,
                                                                        ownOffset);

                }else{

                    if(ditem->dc_hopc == 2){

                        //compute offset of a hop 2 neighbor.
                        /// D_iz offset to a HOP-2 neighbor
                        offsetH2    = ( ditem->offset + ownOffset);

                        ///D_iz negative ? Add H2-neighbor period..
                        if(offsetH2 < 0){
                            offsetH2 =  nli->period + offsetH2;
                        }

                        nli->offset = offsetH2;

                        COOJA_DEBUG_PRINTF("%u Epid(h2)-> %u offset:%2d\n",rimeaddr_node_addr.u8[0],
                        ditem->node_id, offsetH2);
                    }
                }

                nli->next     = NULL;

                //add new element to the list..
                list_add(neighs_list, nli);
            }
        }else{

            //we update an existing element..PROBLEM HERE...
            /// NODE EXISTS.. HERE IS AN UPDATE... I.E, WE have discovered it now..
            if( (nli->node_id == ditem->node_id) &&
                            (ditem->dc_hopc  < nli->hopcount)){

              //COOJA_DEBUG_PRINTF("NULL#%u:%u-%u\n", nli->hopcount, ditem->hopcount, ditem-
              >node_id);
              //node already known and gets confirmation now.
              if(ditem->dc_hopc  < nli->hopcount){

                  nli->hopcount   = ditem->dc_hopc ;
                  nli->tconfirmed = get_discovery_time();
                  nli->offset     = offsetH1;
                  nli->offsetj    = offsetH1;

                  //generic discovery
                  nli->j_factor   = 0;
                  nli->t_anchor   = get_anchor_time();

                  COOJA_DEBUG_PRINTF("%u ADD_i(%u) offset:%2u(%d)\n", rimeaddr_node_addr.u8[0],
                  nli->node_id, offsetH1,
                                                                      ownOffset);

              }
```

```
                }
                //neighs_update(ditem);
            } //else

        } //(ditem->node_id != 0 || (ditem->hopcount <= MAX_HOPCOUNT))

    } // for ( k = 0;

    return 0;
}
```