OBJECT: MAC-timestamping errors

We use MAC-timestamping in our distributed time-synchronization algorithm. We use a broadcast-based data dissemination, so an intensive use of the radio chip is required. We noted that the values provided by the MAC-timestamping are affected by an increasing error as a function of data traffic. We used the 32kHz external oscillator of the Tmote Sky by Moteiv.

DESCRIPTION

Our algorithm is consensus-based, so every node broadcasts to all its neighbors a estimation of several parameters so that the whole network can converge to a global virtual clock (which depends on many factors, as the initial conditions). Every node estimates both the skew, $\alpha_{ij}$ and the offset $\beta_{ij}$.

\[ \alpha_{ij} = \frac{\tau_j(t_2) - \tau_j(t_1)}{t_2 - t_1} \]

Figura 1: Local clock plot between two nodes.

The convergence of the algorithm requires a good measure of the MAC-timestamps: in fact, for the relative skew estimation, we use the ratio of two subsequent transmissions or receptions,timestamped at the sender and at the receiver’s side.

The algorithm has a timer that every TIMESYNC seconds fires and send in broadcast several parameters (transmitter node $j$), as you can see in the code below (task sendMsg). There is a field in the packet that should be filled by the sender MAC-timestamper ($\tau_j(t_i)$), but there is a bug, which appears more frequently in tinyos 1.x, but is also present in Boomerang 2.04. This bug results in no update of the field that remains set to 0. This bug
can be caught at the receiver’s side flushing the packet if the value is 0 (not a time-stamping value).

Once a node receives a packet (receiver node \( j \)), it stores its MAC-timestamp value \( \tau_j(t_{rc}^1) \), that should be stamped at the same instant of the sender, i.e. \( t_{tr}^1 = t_{rc}^1 \) (in general \( t_{tr}^1 - t_{rc}^1 \) is only transmission time, typical several nanoseconds, negligible for our purposes), and process the packet updating several parameters (see the code, `processMsg` task and `calculateConversion` function).

The main problem is that both in Tinyos 1.x and in Boomerang, the intensive use of the radio chip leads to a incorrect interpretation of SFD Interrupts used to update the MAC-timestamps. In fact, if we compute the relative skew \( \alpha_{ij}(t_2) = \frac{\tau_j(t_2) - \tau_j(t_1)}{\tau_i(t_2) - \tau_i(t_1)} \) as indicated in Figure 2 for consecutive packets arrivals, then this value should be almost constant over time, i.e. \( \alpha_{ij}(t) \approx \text{const.} \) This is indeed the case when we used 2 nodes as indicated in Figure 2 (left), where we plotted \( \alpha_{ij}(t_k) - \text{mean}(\alpha_{ij}) \) for consecutive packet arrival times \( t_k \); in fact in this the variance around the mean is on the order of \( 10^{-10} \). When we try with 26 nodes, then this variance increases dramatically up to \( 10^{-3} \), i.e. 7 order of magnitude larger !!!!, as indicated on the right panel of Figure 2. However, the code is exactly the same and therefore also the two plots should show the same error variance. This means that when traffic is high, the MAC-timestamping is not performed correctly, and most likely \( t_{tr}^1 \neq t_{rc}^1 \) in this case.

Figure 2: Estimated relative skew \( \alpha_{ij} \) when using 2 nodes (left) and when using 26 nodes (right). The mean has been subtracted to rescale graphs around zero.

QUESTIONS

- Q1: Do we use in the correct way the MAC-timestamping? We call the sender timestamping right after the send return SUCCESS and the receiver’s on the receive event.
• Q2: How can we modify the interrupt handler in the CC2420RadioM module to solve this problem?

• Q3: We think that probably other algorithms like the FTSP, based on a hierarchical topology, are not so disturbed by this problem. Have anyone who have studied distributed algorithms based on broadcast encountered these problems?

```c
/*
... MODULES
... VARIABLES AND STRUCTS AND SOME ROUTINES
... */

/*************************************************************/
float calculateConversion(AverageTimeSyncMsg *mp,uint8_t myindex)
{
    uint32_t num,den;
    uint8_t i;

    float old_ratio=eta[myindex].ratio;
    float new_ratio,max,min;
    float epsilon=2e-6;

    //update eta if got almost 2 values
    if ( (table[myindex].state == ENTRY_FULL) ) {
        atomic{
            num = mp->tau - table[myindex].old;//remote..if negative truncate to 0
            den = tau - table[myindex].local_old;//local..if negative truncate to 0
        }

        //...
        //...
        //...

        if (num>0 && den >0){//first control
            new_ratio=((float)num-(float)den) / ((float)den);
            /*
             use the normalized form (1 subtracted) also for eta
             */

            /*.....
             ...some filtering to clear outliers...and other controls
             */
        }
    }
    //...
    //...
    //...
}```
```c
void timeSyncMsgSend()
{
    //msg dispatch...
    state |= STATE_SENDING;
    post sendMsg();
}

*/TASKS**************************************************************************/

task void processMsg()
{
    /*****************************************1: relative skew estimation*******************/
    HERE CalculateConversion is called to get the relative skew estimation
    /*****************************************2: skew compensation***************************/
    ...
    /*****************************************3: offset compensation*************************/
    ....
    OTHER UPDATES
*/
}

task void sendMsg()
{
    //send using MAC timestamp
    atomic {
        outgoingMsg->tau = 0;/tau_j will be stamped by MAC layer
        /*
        setting other fields of outgoing message
        */
    }
    /*Here, right after SendMsg returned SUCCESS (as stated in
    CC2420TimestampingM.nc,
    or the tinyos 1.x equivalent file), the sending side timestamping is called
    THERE IS A BUG... sometimes this field is left blank (0)...
    this bug is caught at the receiver’s side, flushing
    such a packet
    */
    if( call SendMsg.send(TOS_BCAST_ADDR, sizeof(AverageTimeSyncMsg),
                        &outgoingMsgBuffer) == SUCCESS ){
        call TimeStamping.addStamp(&outgoingMsgBuffer,
                                   offsetof(AverageTimeSyncMsg,tau));//ok->stamp tau_j
    }
}
```
else{
    state &= ~STATE_SENDING;//send failed, never mind
    signal TimeSyncNotify.msg_sent();
call Leds.redOff();
}

/*******************************RECEIVEMSG*******************************/
/*********************************************************************/
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr p)
{
    uint8_t incomingID;
    int8_t diff;
    AverageTimeSyncMsg* sync=(AverageTimeSyncMsg*)p->data;
    if (!(p->crc) || (p->group != TOS_AM_GROUP)) return p;
    /*ADDR control to simulate multihop...accept only pckts from nearest
    neighbours */
    if (sync->tau < sync->tau_star) return p;//sender timestamp bug-fix
    if((state & STATE_PROCESSING) == 0 ) {//go on only if not processing
        atomic state |= STATE_PROCESSING;
        atomic memcpy(processedMsgPtr,p,sizeof(TOS_Msg));
        /*HERE get receiver side-MAC time-stamp: in Boomerang get
directly
        the time field (filled with the head of the timestamps' queue by
        the CC2420RadioM' SFD interrupd handler),
        while in tinyos 1.x update the rcvtime variable stored in the
        timestamping module
        */
        atomic tau= call TimeStamping.getReceiveStamp(p);//reception
time
call Leds.greenOn();
        post processMsg();
        return p;
    }
    return p;
}

event result_t UpdateTimer.fired()
{
    /*send in broadcast our parameters