On Energy Consumption and Drifting Clocks

and why models and algorithms are needed to assess their interplay

Holger Hermanns

Dependable Systems and Software
Saarland University, Germany

VASY
INRIA Rhône-Alpes, France
Claim (my version of the MLQA mission)

There is an enormous spectrum of intriguing research questions out there ... in embedded system design

Many of them

- are crying for mathematically well-founded investigations.
- are very tough to investigate with current technology.

Often, there is no other way.

So, let’s invest in better modelling and analysis techniques for them.
How the practical guys look at the problems

Remember the time when “compilers” didn’t yet have error/warning messages?

Jan Beutel, ETHZ
PermaSense project @ Matterhorn

and many others
Example
Example: ZigBee

- Low-Rate Wireless Personal Area Network
- 250, 40, or 20 kbit/s data rate

Application Profiles
Application Framework
Network & Security Layer
MAC Layer
PHY Layer

ZigBee Alliance

IEEE 802.15.4

Holger Hermanns
ZigBee

Characteristics:

+ very low power consumption
+ small range
+ simple comm. protocol
- low data rate
ZigBee MAC Layer: IEEE 802.15.4

Configurable channel access types

- Guaranteed time slots
  - TDMA
  - Slotted CSMA/CA
  - Unslotted CSMA/CA

- Probabilistic medium access

- Beacon determines frame boundaries
IEEE 802.15.4

Superframe Structure

CSMA/CA  TDMA

Active  Inactive

Beacon Interval

Time →
Sleep, but do not oversleep

Principle:
- Coordinator sends periodic beacon signal to indicate slot boundaries.
- Beacon interval ranges from 0.015 sec to 251.657 sec.
- Sensors go sleep in between whenever possible.
- Sensors are responsible for waking up on time, have their own clocks.
Advance of clocks

Sensor clock

coordinator clock

active stand by sleeping
Wake up!

Sensor clock

Advance of clocks

coordinator clock

active

stand by

sleeping

Holger Hermanns
Wake up!

Sensor clock

overslept!

Real clocks

Ideal clocks

active

stand by

sleeping

coordinator clock

Holger Hermanns
Wake up!

Sensor clock

Ideal clocks

Real clocks

overslept!

active

stand by

sleeping

Holger Hermanns
Holger Hermanns

Wake up!

Sensor clock

Ideal clocks

Real clocks

overslept!

active

stand by

sleeping

coordinator clock

coordinate clock
Wake up!

Sensor clock

Possible ‘drift’

Ideal clocks

Real clocks

overslept!

active

stand by

sleeping

Holger Hermanns
Holger Hermanns

Make it better!

1. Prepare sensors for worst case.
2. Re-synchronize clocks each time.
3. ...
4. ...
5. ...

Sensor clock

coordi
ator
clock

active

stand by

sleeping
Results
Bad clocks cost energy

More frequent wakeups cost energy

Guaranteed time slots save energy

Unslotted communication appears best

Sensor clock

Active

Stand by

Sleeping
Does Clock Precision Influence ZigBee’s Energy Consumption?

- Well, yes.
  - To a marginal extent.
- Drift is amplified by a factor of up to 60.

Half a day per year battery lifetime.

[Gross, H, Pulungan: OPODIS 2007]
The particular question considered here cannot be answered by lab experiments, mainly because clock drift is uncontrollable.

Furthermore, battery lifetime effects take long to manifest themselves. Lab conditions are very difficult to establish.
Modelling
process coordinator() {
    alt{:: do_act1
        :: do_act4
    }
}

process station() {
    do{:: do_act2
        :: do_act3
    }
}

par{
    ::coordinator()
    ::station()
}
Modeling
Probabilistic Choice

```plaintext
process coordinator() {
    palt{:1: do_act1
         :9: do_act4
    }
}

process station() {
    do{::: do_act2
        :: do_act3
    }
}

par{
    ::coordinator()
    ::station()
}
```
process coordinator() {
    do{:: do_act1
        :: comm_starts ; comm_ends
    }
}

process station() {
    clock c;
    do{:: do_act2
        :: comm_starts {= c=0 =} ;
        when (c==10) comm_ends
    }
}

par{
    ::coordinator()
    ::station()
}
process coordinator() {
    do{:: do_act1
        :: comm_starts ; comm_ends
    }
}

process station() {
    clock c;
    do{:: do_act2
        :: comm_starts {= c=0 =} ;
        when (c==Exponential(10)) comm_ends
    }
}

par{
    ::coordinator()
    ::station()
}
Modeling
State Variable

process coordinator() {
    do{ :: do_act1
        :: comm_starts ; comm_ends
    }
}

process station() {
    clock c;
    int sent = 0;
    do{ :: do_act2
        :: comm_starts {= c=0 =} ;
        when (c==Exponential(10))
            comm_ends {= sent+= 1 =}
    }
}

par{
    ::coordinator()
    ::station()
}
Modeling
The Real System Model

par{  ::coordinator()
  ::station_1()
  ::station_2()
  ...
  ::station_10()}

Holger Hermanns
Modeling
Relabeling

par{
    ::coordinator()
    ::relabel{...} by {....} station(1)
    ::relabel{...} by {....} station(2)
    ...
    ::relabel{...} by {....} station(10)
}
process coordinator() {
    clock btimer, c;

    do{:: sendb_start
        {= c=0, btimer=0, bintheair=true =} ;
        when(c==52) sendb_end
        {= bintheair=false =} ;
        when(btimer==binterval)
    }
}
process station(int id) {
    ...
    do{::when(bintheair)
        beacon_received = mainclock=0, ttosend=sendingtime =
    }
    when(!bintheair) attempt_sending;
    do{::when(ttosend==0 || !enoughtime)
        do_nothing = enoughtime=true =
    } break
    ::when(ttosend>0 && enoughtime)
        start_csmaca
    ...
    \ Slotted CSMA/CA Code Here
    ...
}

\ Holger Hermanns
Holger Hermanns

Modeling

The Station (contd.)

... ...

17 alt{
18   ::when(mainclock>=CAP-backofftime-ccatime) (= enoughtime=false =) ; break
19   ::when(mainclock<CAP-backofftime-ccatime)
20     when(c==backofftime) (= c=0 =) ;
21     do{::when(c==ccatime)
22       alt{:when(sending>0)
23         alt{:when(NB<=maxbackoff)
24           channel_busy (=CW=2,NB=NB+1,BE=(BE<6)?BE+1:6=) ; break
25           ::when(NB>maxbackoff) (= restart=true =) ; break }
26         ::when(sending==0 && mainclock<CAP-attosend-960)
27           count_down_CW (= CW=CW-1 =) ;
28           alt{:when(CW==0) wait_for_boundary ;
29             when (c==320) send_message_start (= sending+=1, c=0 =) ;
30             when (c==attosend) send_message_end
31               (= ttosend-=attosend, sending-=1, restart=true =) ; break
32             ::when(CW>0) wait_for_boundary ;
33             when(c==320) (= c=0 =) }
34     ::when(sending==0 && mainclock>=CAP-attosend-960)
35       (= enoughtime=false, restart=true =) ; break }
36    } ; alt{:when(restart) (= restart=false =) ; break
37    ::when(!restart)
38 } } } } }
...do{:: when (CW==0) start_on_next_backoff_boundary ;
when (c1==320) start_send_message {= sending+=1, c2=0 =} ;
when (c2==actual_to_send) finish_send_message
  {= still_to_send-=actual_to_send, sending-=1,
   messages_sent[id]+=1, continue_backoff=0,
   message_to_send=0,
   time_in_tx_mode[id] += actual_to_send + 194
   =} ; break :: ...
}
Modeling Clock Precision

- Inaccuracy $p$ ppm means at time $t$:
  
  \[ [t - p \cdot 10^{-6} \cdot t, t + p \cdot 10^{-6} \cdot t] \]

- Maximal clock drift allowed is 40 ppm

- Waiting time:
  
  \[ W' = W - W \cdot p \cdot 10^{-6} + W \cdot p' \cdot 10^{-6} \]

...when(c1 == wait_in_backoff
   - wait_in_backoff * precision[id]
   + wait_in_backoff * random_drifts[id])

perform_cca {= c2=0, ... =}...
Modest, a language for stochastic timed system

- Rooted in classical process algebra:
  - atomic actions,
  - sequential & parallel composition,
  - nondeterministic choice,
  - looping
- Probabilistic choice,
  - exception handling, and
  - hard and soft real-time aspects

- Conventional programming constructs

- Semantics: Stochastic Timed Automata (STA)

[Bohnenkamp, D'Argenio, H, Katoen: IEEE TSE 32(10)]
Stochastic Timed Automata

- finite automata
- with clocks
- and with costs
- modular: composition of automata

- with continuous probability distributions
- and with discrete probabilistic branching

[Bohnenkamp, D'Argenio, H, Katoen: IEEE TSE 32(10)]
Analysis
Analysis Trajectory

- STA are as yet too expressive to be analysable (by us)
  - no nondeterminism: Discrete event simulation
  - no probabilism: real-time model checking
  - discrete probabilism only: PTA model-checking

What we do:

- **Semi-automatic abstraction**
  to arrive at a timed automata model.

- **UPPAAL** to generate schedules with worst cost.

- Derive insights on worst-case clock drifts

- Code this worst case into the model

- **Motor/Möbius** to simulate scheduled Modest specs.
Results

Perfect Clock

![Graphs showing performance metrics for different network configurations.](image)
Results

Drifting Clock
Epilogue
There is an enormous spectrum of intriguing research questions out there ... in embedded system design

Many of them

- are crying for mathematically well-founded investigations.
- are very tough to investigate with current technology.

Often, there is no other way.

So, let's invest in better modelling and analysis techniques for them.
## How to validate a hypothesis?

<table>
<thead>
<tr>
<th>Model validation</th>
<th>Simulation</th>
<th>Emulation</th>
<th>Empirical studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPAAL</td>
<td>TOSSIM</td>
<td>EmStar</td>
<td>...</td>
</tr>
<tr>
<td>PRISM</td>
<td>Avrora</td>
<td>Bee</td>
<td>...</td>
</tr>
<tr>
<td>Motor</td>
<td>Glomosim</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>20-sim</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Modelica</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Matlab</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
How to validate a hypothesis?

<table>
<thead>
<tr>
<th>Model validation</th>
<th>Simulation</th>
<th>Emulation</th>
<th>Empirical studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>General?</td>
<td>Bad</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>Time/cost effective?</td>
<td>Bad</td>
<td></td>
</tr>
<tr>
<td>Bad</td>
<td>Accurate?</td>
<td>Good</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>Reproducible?</td>
<td>Bad</td>
<td></td>
</tr>
<tr>
<td>Bad</td>
<td>Scalable?</td>
<td>Bad</td>
<td></td>
</tr>
</tbody>
</table>
What we need

Model validation  Simulation  Emulation  Empirical studies

From Models to Experimentation and Back
Why we need that
Why we need that

Remember the time when “compilers” didn’t yet have error/warning messages?

Jan Beutel, ETHZ
PermaSense project @ Matterhorn

and many others
What we need

Model validation  Simulation  Emulation  Empirical studies

From Models to Experimentation and Back

The holy grail:

Sound models and sound abstractions
Go!