Accurate Clock Models for Simulating Wireless Sensor Networks

F. Ferrari, A. Meier, L. Thiele
TIK Institute – ETH Zurich
Motivations: Hardware Clocks of Sensor Nodes

- **Digital clocks**
  - A counter counts time steps of an ideally fixed length
  - Reading of the counter at real-time $t: h(t)$
  - Ideal rate: $f(t) = dh(t) / dt \equiv 1$

- **Sensor nodes are equipped with cheap oscillators**
  - Rate fluctuates over time
    - due to changes in supply voltage, temperature, and aging
  - Drift: $\rho(t) = dh(t) / dt - 1$
  - Drift variation: $\vartheta(t) = d^2 h(t) / dt^2$
Motivations: Time-Critical Protocols

- Time-critical protocols require accurate clock models for realistic simulation results
  - Several MAC protocols (e.g., TDMA) assume perfectly synchronized clocks
    - Clock drift has to be taken into account
  - Synchronization protocols (e.g., FTSP) estimate the drift of the HW clock
    - Linear regression on time values retrieved from neighbors
    - A model for HW clocks that assumes constant drift would lead to an unrealistic perfect synchronization
Hardware Clock: Models

- **Tuning-fork** \( \rho(t) = -A \cdot (T(t) - T_0)^2 \)
  - Drift as a direct function of temperature

- **Bounded-drift** \(|\rho(t)| < \hat{\rho}\)
  - Drift limited by known bounds

- **Bounded-drift-variation** \(|g(t)| < \hat{g}\)
  - Drift variation limited by known bounds

- **Combined** \((|\rho(t)| < \hat{\rho}) \land (|g(t)| < \hat{g})\)
  - Most general model, describes also the previous ones
Hardware Clock: Linear Piecewise Approximation

- Time is divided into intervals of length $t_{\text{int}}$ with constant drift
  - Initial HW time and drift are sufficient for computing the approximated HW time within an interval
Hardware Clock: Linear Piecewise Approximation

- Maximum error introduced by the approximation:
  \[ \varepsilon = \varrho \cdot t_{\text{int}}^2 / 8 \rightarrow \varepsilon \approx 0.125 \mu \text{sec with } t_{\text{int}} = 10 \text{sec} \]
- Validity range for the approximation:
  \[ t_{\text{int}} \ll \hat{\rho} / \hat{\varrho} \approx 10,000 \text{sec} \]
Clock Translation

- From simulation to HW time
e.g., to provide current HW time
  1. $k = \left\lfloor \frac{(t - t_0)}{t_{\text{int}}} \right\rfloor$
  2. $h^*(t) = h_k^*(\tilde{t}) = h_k(0) + \tilde{t} \cdot (1 + \rho_k^*)$

- From HW to simulation time
e.g., to schedule an event by using HW time as the time reference
  1. find $k$ such that
     $h_k(0) \leq h^*(t) \leq h_k(t_{\text{int}})$
  2. $t = t_0 + k \cdot t_{\text{int}} + \left( h^*(t) - h_k(0) \right) / (1 + \rho_k^*)$
Our Case Study

  - WSNs simulator based on OMNeT++
    - Accurate model of the wireless channel and HW components
    - Provides time with constant drift to each node
      - Not sufficient for simulating time-critical applications
      - Manual translation to the OMNeT++ simulation time

- A node only knows the time provided by its HW clock
  - Simulation time hidden from the application
  - \textit{HW time} \rightarrow \textit{simulation time} before scheduling events
Proposed Approach for Castalia

- **Software Clock**
  - getHWtime()
  - adjustTime()
  - adjustDrift()

- **Hardware Clock**
  - getHWtime()

- **Synchronization Manager**
  - adjustTime()
  - adjustDrift()
  - getEstimatedDrift()

- **Application**
  - scheduleAt()
  - scheduleAtHWtime()
  - remove()
  - isScheduled()

- **OMNeT++ Scheduler**
  - scheduleAt()
  - sendDelayed()
  - cancel()
  - "expired()"

- **Clock Translator**
  - scheduleAtHWtime()
  - remove()
  - isScheduled()
Event Scheduling Interface

- New interface for scheduling events
  - Applications schedule events using HW time, the only time available on a real sensor node
  - Time translation is completely hidden
    - Clock Translator translates HW time to simulation time
    - OMNeT++ event scheduling methods are eventually called

```
scheduleAtHWtime()
scheduleAt()
sendDelayed()
cancel()
remove()
isScheduled()

Clock Translator

Application
Communication Stack
Synchronization Manager

OMNeT++ Scheduler

"expired()"
```
**Clock Translator: Sliding Storage Window**

- **Sliding storage window**
  - Storing values for all intervals would generate a prohibitively high memory overhead

- Only one window of $s$ intervals is stored at a time
- Window updated every $u$ intervals, $0 < u \leq s$
- Events beyond the current window are kept in a local queue and scheduled when the time window is updated
Evaluation

• Evaluated on four built-in Castalia applications
  – Memory overhead
    • \((16 \cdot N_{\text{nodes}} \cdot s)\) Bytes
      – e.g., 150 nodes, 1000 intervals per window: 2400 KBytes
  – Execution time overhead
    • About 11% when using an accurate drift clock model
Conclusions

• Our framework provides realistic clock models for simulation
  – Allows simulation of time-critical applications
    • MAC and synchronization protocols
  – Provides well-defined interfaces for scheduling events
    • Simulation time hidden from the application
  – Introduces minimal overhead
  – Easily extendable to other network simulators