dtimer: Desynchronized timers as an operating system service

Raejoon Jung, Francis Y. Yan, Philip Levis, Kyle Jamieson, Kumar Balachandran, Dennis Hui, Ivana Maric
Periodic actions in wireless sensor networks

- Homogeneous nodes
- Periodic action with wide range of interval
Case study: MAC duty cycling

- Saves energy
- ~ 100 ms interval
- Receiver beacons or data frames can collide.
- Random wake up schedule? Sender waste energy for rendezvous.
Case study: Network Control frames

- Multi-purpose
  - Time sync.
  - Routing information
  - Link estimation
- Broadcasts
- ~ seconds interval
- Underlying MAC can aggravate the situation.

“Announcement: local time is XX:YY:ZZ”

Collision!
Case study: Data sampling

- 10 sec. ~ minutes interval
- No communication
- Spatio-temporal locality
- Improved resolution-energy tradeoff
Time desynchronization

Separate periodic events with identical interval.

- (Global) Offset: timing representation of a series of periodic events.
  - An offset of $O$ means events occur at time $O, O + I, O + 2I, O + 3I, \ldots$
- A set of periodic events $\rightarrow$ List of offsets visualized on a ring

![Diagram showing time desynchronization with nodes A, B, C, D connected in a cycle and a ring diagram visualizing periodic events.](image-url)
Desynchronization as an OS service

- Multiple applications benefit from desynchronization.
- Desynchronized state heavily depends on the topology of the network.*
- Desynchronization requires message exchanges. (network traffic and energy)

→ Let the operating system provide desynchronized timers efficiently.

- Assume homogeneity; All nodes run identical applications.
- Guarantee:
  Timing of the callback function of the timer is spread apart from its neighbors’ timers with the same interval.
Outline

1. Periodic applications in wireless sensor networks
2. Case for time desynchronization as an operating system service
3. OS service requirements
4. Design
   4.1. dtimer API for multiple interval support
   4.2. Scheduling
   4.3. Multi-hop network desynchronization
   4.4. Solo algorithm
5. Evaluation
6. Conclusion
Requirements

1. Support wide **range of intervals**; from hundreds of milliseconds to hours.
2. Support **multiple timers** simultaneously on each node.
3. **Spread timers** of a given interval across neighboring nodes in time.
4. Support **arbitrary network** topologies and cope with lossy links.
5. **Scale well** with the size of the network and the number of timers.
dtimer: Desynchronized timers

- Timers with different intervals built on top of a single state machine.
- Multiplexes onto a single underlying message exchange
- Solo algorithm desynchronizes each node’s offsets across one-hop neighbors.
dtimer API

- `dt = dtimer_init(interval, client_offset*, callback)`
  - Initiate a dtimer which executes a callback. Returns a dtimer handle.

- `dtimer_start(dt)`
  - Start the timer. Registered callback executes and the timer resets when the timer expires.

- `dtimer_stop(dt)`
  - Stop the timer.

To sample data every 10 seconds for an hour...

```c
count = 0; dt = NULL; interval = 10*CLOCK_SECOND;

void callback() {
    buffer.write(sensor.value())
    count++;
    if (count == 360) dtimer_stop(dt);
}

void main() {
    dt = dtimer_init(interval, 0, callback);
    dtimer_start(dt);
}
```
dtimer API

- \texttt{dt = dtimer\_init(interval, client\_offset*, callback)}
  - Initiate a dtimer which executes a callback. Returns a dtimer handle.

- \texttt{dtimer\_start(dt)}
  - Start the timer. Registered callback executes and the timer resets when the timer expires.

- \texttt{dtimer\_stop(dt)}
  - Stop the timer.

To sample data every 10 seconds for an hour...

```c
count = 0; dt = NULL; interval = 10*CLOCK\_SECOND;

void callback() {
    buffer.write(sensor.value());
    count++;
    if (count == 360) dtimer\_stop(dt);
}

void main() {
    dt = dtimer\_init(interval, 0, callback);
    dtimer\_start(dt);
}
```
Scheduling multiple timers

How can multiple dtimers of different intervals be scheduled efficiently?
Scheduling multiple timers: Strawman

- Desynchronize dtimers of different intervals individually.
- Number of message exchanges increase linearly.
- Example: fully connected 3 nodes.

- But, the timing of the dtimers in a single node share commonality.
Scheduling multiple timers: Multiplexing

- Each node runs a single desynchronization algorithm with a fixed interval
- Algorithm updates a single variable, \( O \), offset within its interval \( I \).
- Schedules of each dtimer can be computed by scaling the offset.
- Each dtimer’s next expiry is calculated when it expires. (Interval: \( I_d \))

\[
t_{\text{expiry}} = \left\lfloor \frac{t - O \cdot I_d / I}{I_d} \right\rfloor I_d + O \cdot I_d / I
\]

- dtimers share the underlying desynchronization algorithm.
- Only a single set of message exchanges (of interval \( I \)) is used.

* Requires time synchronization.
Multi-hop network desynchronization

How can we desynchronize nodes across a multi-hop topology?
Desynchronization in multi-hop networks

- Challenge 1: Definition in single-hop network does not extend well...
- Nodes have different degrees and belong to multiple single-hop subgraphs.
- Example: Network with maximal cliques of size 3 and 4,
Desynchronization in multi-hop networks

- Example: Network with maximal cliques of size 3 and 4,

- Can we say either state is better than the other?
- Multiple possible definitions (clique-, node-, pair-, demand-based, etc.)
- Application performance may justify but dtimer assumes multiple applications.
Desynchronization in multi-hop networks

- Challenge 2: Maximally separating the offsets requires two-hop link state.
- A node needs to know how their neighbors are connected with each other and also with its two-hop neighbors.
- Example 1: *Node A knows its identical neighbors in both cases.*
Desynchronization in multi-hop networks

- Example: *Node A knows identical neighbors with their identical degrees in both cases.*

A has 3 neighbors of degree 3.

- The more a node knows about its neighbors’, the better it can position itself.
Goal of Solo

- Link state of two-hop neighbors is expensive, both in comm. and memory.
- Nodes in wireless networks share many of the same neighbors.
- Solo exploits this to make the algorithm simple.

Given an interval $I$, each node,

- defines *separation* as distance between its offset and its predecessor’s offset.
- *target destination* is defined as $I/(d + 1)$ where $d$ is the predecessor’s degree.
- adjusts its offset to achieve the target destination with its predecessor.

* degree is broadcasted as part of the message exchange.
Solo: Neighbor list

- Collect one-hop neighbor link quality.
  - 1. degree 2. predecessor 3. successor.
- Updated at every beacon reception.
- Each entry contains,
  - Node id
  - Latest beacon timestamp
  - (Exponential) Average interval
- An entry is removed when either,
  - Time since latest beacon timestamp exceeded a threshold.
  - Average interval exceeded a threshold.
Solo: Pulse coupled oscillators (PCOs)

- Solo use PCOs to iteratively achieve target separation.
  - Broadcast a beacon once per interval at a particular offset $O$
  - When a neighbor fires at time $r$, it checks if it achieves target separation.
  - If not, delays its offset by the following,
    \[ \delta = \alpha \cdot \left( r - \left( O - \frac{I}{d+1} \right) \right) \]

- Allows “best effort” desynchronization under network perturbation.
- Achieves optimal state in (static) single-hop case.
- But can it converge with any topologies?
Solo: Loop detection

- PCOs fails when reactions causes nodes to drift over time.
- In general,
  1. a group of nodes in the network form a cycle.
  2. the offsets in the group follow the order on the cycle.
  3. the sum of target separations is greater than the sum of actual separations of the cycle.
Solo: Loop detection

- Solo detects the loop and break it by resetting a node in the loop.
- Path vectors of chained adjustments are propagated across the network.

For each node,
- when the beacon timer expires, broadcasts beacon with its path vector.
- When a node receives a neighbor’s beacon,
  - If needs to adjust, check if its id is in the beacon’s path vector.
    - If yes, reset its offset to a random value.
    - If not, store the beacon’s path vector with its id appended.
  - Otherwise, reset the nodes’ path vector to an empty vector.
Solo: Loop detection
Solo: Target Clamping

- Neighbors of a single node can constantly push the node too aggressively.
- Solo limits the target offset to the successor’s offset.
- Guarantees an adjusted node to always broadcast.
Summing up

Client Application

dtimers

Interval
Callback function
Software timer

Neighbor List

Node id
Latest beacon timestamp
Average interval

Solo Algorithm

Interval, Offset
Software timer

Communication Stack
Evaluation

- Solo algorithm
  - Discrete event simulator in Python
  - Evaluate convergence of Solo algorithm in arbitrary topologies.

- dtimer with Solo
  - C library in Contiki OS.
  - Deployed in 27 Tmote Sky motes in Flocklab
  - Evaluate robustness, accuracy, multiplexing capability, and efficiency of dtimer

- Applications with Solo
  - Receiver initiated MAC, Wi-Fi Power save mode implementation in C++ for Linux machines.
  - Deployed in OrbitLab Wi-Fi testbed
  - Evaluate application benefits from Solo-based desynchronization.
Solo: Desynchronization

- Monte Carlo simulation with random geometric graphs and initial offsets.
- PCOs itself cannot achieve Solo’s target separation.
- Path-vector loop detection and target clamping improves robustness.
Solo: Network Perturbation

- Flocklab deployment. Start with 4 fully connected nodes.
- Fifth node joins at time 150 seconds and leaves after 150 seconds.
dtimer: Interval Accuracy

- dtimers of intervals ranging from 250 milliseconds to 16 seconds.
- 98 % of the dtimer intervals are within 1 millisecond of the target interval.
- Error increases as target interval increases.
  - Caused by underlying Contiki’s callback timer. (Baseline)
dtimer: Multiplexing

- 27 node deployment in Flocklab.
- Each node has three dtimers with intervals of 3, 10, and 14 seconds.
- Snapshot after Solo converges.
- Relative timing of the nodes are identical across different dtimers.
dtimer: Multiplexing

- Run multiple dtimers simultaneously.
  Measure number of broadcasts per Solo’s timer interval.
- Communication cost is fixed regardless of the number of dtimers.
- Delayed beacon broadcast with shorter Solo timer interval
  - (1) Software timer error (2) Bursty nature of wireless links
dtimer: Multiplexing

- Communication cost is fixed regardless of the dtimer intervals.
Application: RI-MAC

- Original design: Random duty cycle intervals to avoid collision.
  - Sender must turn on the radio and wait until it hears from the receiver.
- With Solo: Desynchronized static duty cycle schedule.
  - Sender can anticipate the receiver’s wake up time and save energy.
Application: WiFi Power save mode

- Original design: Client only wake up intermittently to receive AP beacons.
  - If beacon indicates queued data, data transmission follows. (Beacons = xmit opportunities)
- With Sleepwell: AP beacon offsets are adjusted to avoid contention.
  - Adjustment made to achieve target separation immediately, ranging the entire offset ring.
- With Solo: Adjusted iteratively.
  - Maintain transmission opportunities and reduce latency.

<table>
<thead>
<tr>
<th>Client-side beacon interval</th>
<th>AP-side beacon count (10 seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit: ms</td>
<td>Baseline</td>
</tr>
<tr>
<td>Min</td>
<td>100</td>
</tr>
<tr>
<td>Median</td>
<td>100</td>
</tr>
<tr>
<td>Max</td>
<td>100</td>
</tr>
</tbody>
</table>
Conclusion

- Multiple periodic applications benefit from time desynchronization.
- Desynchronization of multiple timers can be aggregated to per-node basis.
  - Reduce energy and network traffic cost for desynchronization.
- Pulse coupled oscillators used for multi-hop network desynchronization.
  - Cope well with network perturbations (packet loss, node join/leave)
  - Requires loop detection and target clamping.

- Limitations:
  - Assumes homogeneous deployment. Will this be true for future sensor networks?
  - Some applications can embed time desynchronization with existing broadcast packets.
  - Solo’s desynchronization can be suboptimal, depending on the application.