Using Meta-Code for Building Task-Specific WSNs

Igor Talzi, Christian Tschudin / University of Basel, Computer Science Dept
Andreas Hasler, Stephan Gruber / University of Zürich, Dept of Geography
Jan Beutel, Mustafa Yücel, Roman Lim / Computer Engineering and Networks Lab, ETH Zurich

Introduction

Meta-code definition:
- Meta-code deals with the meta-data, or
- Meta-code describes behaviour, the actual implementation is still based on the native code.

The potential use:
- Auto-configuration (react to changes in topology, sensor attachments, etc)
- Self-documenting (report changes)
- Proactive actions

Node Architecture

Basic principles:
- Stack-based VM
- Shared memory for vars (heap)
- Memory protected capsule execution
- FIFO-like scheduling, blocking operations (e.g. `delay N` or `send_dt up`), direct access to the queue
- Errors are reported to the meta-data layer
- Dynamically extensible dictionary-based ISA

Basic modules:
- CEM – Central Execution Module
- STORAGE / QUEUE – a queue containing capsules in a decoded form
- DICT – operand and opcode dictionary
- COMP – Code Compressor
- DECOMP – Code Decompressor
- MUX – Packet Multiplexer
- DEMUX – Packet Demultiplexer
- FEC ENC – FEC Encoder
- FEC DEC – FEC Decoder
- CVM – Code Validation Module

Instruction Set

Basic instructions:
- Stack ops («push», «pop»)
- Heap ops («read», «write», «move», etc)
- Arithmetic ops («add», «mult», etc)
- Binary/Logic ops («and», «xor», etc)
- Jumps («jmp L1», «jmp L2», etc)
- Debug («led green,off»)
- Macro-instructions

Macro-Instructions

<table>
<thead>
<tr>
<th>Command</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>send_dt/code updown</code></td>
<td>Code: comment to overwrite code for the data directed or send up/down.</td>
</tr>
<tr>
<td><code>get_meta_dt F</code></td>
<td>Get meta-data field F from the current position.</td>
</tr>
<tr>
<td><code>mod_meta_dt F,V</code></td>
<td>Modify meta-data field F with value V.</td>
</tr>
<tr>
<td><code>split</code></td>
<td>Split the capsule into 2 at the current position.</td>
</tr>
<tr>
<td><code>copy A,B</code></td>
<td>Copy the capsule from A to B.</td>
</tr>
<tr>
<td><code>copy LADDR</code></td>
<td>Copy the code from the current position to the delimiters (e.g. ADDR).</td>
</tr>
<tr>
<td><code>rb</code></td>
<td>Round up bounds.</td>
</tr>
<tr>
<td><code>link_meta_dt</code></td>
<td>Link meta-data to/from a specific set.</td>
</tr>
<tr>
<td><code>write/erase</code></td>
<td>Write/erase a record from the instruction dictionary.</td>
</tr>
<tr>
<td><code>get AADDR</code></td>
<td>Get a capsule.</td>
</tr>
<tr>
<td><code>get_children</code></td>
<td>Get the list of available children nodes.</td>
</tr>
</tbody>
</table>

Show Cases

Show Case 1: toggle a led
- `get_children (嘘, 1)`
- `if we have children (嘘, 1)`
- `send up`
- `# do it periodically`
- `# delay 1000`
- `# sleep for 1s
- `# toggle the red led`
- `# turn off`

Show Case 2: count the nodes
- `get_children (嘘, 1)`
- `if we have children (嘘, 1)`
- `report ADDR`
- `get_nodeid`
- `push 3`
- `jmp 0.11`

Code Compression

`ping ADDR`
- `i2: push 2`
- `get_nodeid`
- `ADDR delay 1000`
- `read`
- `jmp 0.11`

Future Work

- Obtaining initial results on simulation
- More complex show cases (e.g. skew-balance time sync protocol)
- Power profiling