Unified Service-oriented Access for WSNs and Dynamically Deployed Application Tasks

Th. Fronimos¹, S. Lalis¹,², M. Koutsoubelias¹,², Th. Bartzanas¹

¹ IRETETH/CERTH, Greece
² University of Thessaly
The IoT Universe

- Numerous interconnected devices
- Key challenge is to make such devices into active parts of the IoT
- Integrate WSNs into IoT
- Enable more advanced monitoring & control applications
WSNs in a nutshell

Characteristics

- Power supply
- Limited system resources
- Low level programming
- Communication constraints
- Often application specific

Approaches

- OS
  - TinyOS, Contiki
- Middleware
  - macro-programming, VMs, task chains
- Minimal communication stacks
  - 802.15.4, Zigbee
- Remote reprogramming / reconfiguration
  - firmware updates, replacement or re-linking of application-level code and components
Our focus

- Architecture allowing WSN to operate in two different modes
  - **standard mode**: control/query nodes as per their firmware
  - **application mode**: let the application deploy its own tasks (code) to perform custom data processing tasks directly on the nodes, beyond what is supported by the pre-installed firmware

- Co-existence of both modes at the same time

- Service-oriented, platform-neutral access for both modes
Integration Approach: Standard WSN Functions

- configure system parameters
- manage system/application files
- receive node status information
- reset nodes
- query node sensors once/periodically
- remotely install application tasks
# Standard Interface

<table>
<thead>
<tr>
<th>Request (HTTP)</th>
<th>Reply (JSON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodestat?addr0=&lt;int&gt;&amp;addr1=&lt;int&gt;</td>
<td>`{rimeaddress: [0,255].[0,255], timestamp: 'YYYY:MM:DD:hh:mm:ss', deluge_status: 'true'</td>
</tr>
<tr>
<td>query?addr0=&lt;int&gt;&amp;addr1=&lt;int&gt;&amp;sensors=&lt;string&gt;*</td>
<td>`{ ... values: { temp: (float), //if selected hum: (float), //if selected light: (float), //if selected light2: (float), //if selected } }</td>
</tr>
<tr>
<td>reboot?addr0=[int]&amp;addr1=[int]</td>
<td>None</td>
</tr>
</tbody>
</table>

*eg. temp.hum.light*
# Standard Interface

<table>
<thead>
<tr>
<th>Request (Websocket)</th>
<th>Reply (JSON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
<td>{</td>
</tr>
<tr>
<td>rimeaddress: [4,1],</td>
<td>rimeaddress: [4,1],</td>
</tr>
<tr>
<td>service: &quot;periodic&quot;,</td>
<td>deluge_status: 'true'</td>
</tr>
<tr>
<td>rates:</td>
<td>txpower: [0,32],</td>
</tr>
<tr>
<td>{</td>
<td>version: [0,65],</td>
</tr>
<tr>
<td>temp: 60,</td>
<td>lqi: [0,108],</td>
</tr>
<tr>
<td>hum: 90,</td>
<td>rssi: [-95,0],</td>
</tr>
<tr>
<td>light: 60</td>
<td>values: {</td>
</tr>
<tr>
<td>//light2 sensor not selected</td>
<td>battery: (float),</td>
</tr>
<tr>
<td>}</td>
<td>temp: (float), /*every 60 seconds</td>
</tr>
<tr>
<td>}</td>
<td>hum: (float), /*every 90 seconds</td>
</tr>
<tr>
<td></td>
<td>light: (float), /*every 60 seconds</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
Integration Approach: Application Tasks

Application data objects DTD

- <!ELEMENT datamessages (datamsg*)>
- <!ELEMENT datamsg (uname,code,par*)>
- <!ELEMENT uname (#PCDATA)>
- <!ELEMENT code (#PCDATA)>
- <!ELEMENT data (uname,valtype,size?)>
- <!ELEMENT valtype (#PCDATA)>

Application commands DTD

- <!ELEMENT commands (command*)>
- <!ELEMENT command (uname,code,par*)>
- <!ELEMENT uname (#PCDATA)>
- <!ELEMENT code (#PCDATA)>
- <!ELEMENT par (uname,valtype)>
- <!ELEMENT valtype (#PCDATA)>
Installing Application Tasks

**Installation steps**

1. Upload XML description files
2. Upload binary on server
3. Upload binary on sink
4. Activate transfer protocol on nodes
5. Start transfer
6. Upon completion, deactivate transfer protocol on sink
7. Transfer protocol is automatically deactivated on nodes but can also be done explicitly in case of failure
   - Transfer protocol deactivation to avoid background network traffic

**Service examples**

<table>
<thead>
<tr>
<th>URL</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>copytosink?file=test</td>
<td>Load file on sink</td>
</tr>
<tr>
<td>startinstall?addr0=12&amp;addr1=25&amp;file=test&amp;size=2048</td>
<td>Activate protocol on target node</td>
</tr>
<tr>
<td>install?file=test&amp;size=2048</td>
<td>Install</td>
</tr>
<tr>
<td>stopinstall?addr0=12&amp;addr1=25</td>
<td>Deactivate protocol on target node</td>
</tr>
</tbody>
</table>
Application Programming Model

- Hides the details of OS from user
- C language
- Implementation of event handlers
  - initialization
  - message arrival
  - new sensor readings
- Simple API for
  - marshalling data into messages
  - un-marshalling data from messages
  - accessing sensors
  - sending data
  - basic math functions
- Programmer should follow the data/command XML models

<table>
<thead>
<tr>
<th>Event handlers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>on_init(void)</td>
<td>Initialize task state</td>
</tr>
<tr>
<td>on_sensordata(int mask, float *vals[])</td>
<td>Handle new sensor values</td>
</tr>
<tr>
<td>on_command(char id, char *buf)</td>
<td>Handle config. command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor primitives</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>set_sensors(int mask, int rates[])</td>
<td>Set the sensors to use and their sampling rates</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outgoing data primitives</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>setType(char *buf, int type)</td>
<td>Set message buffer type</td>
</tr>
<tr>
<td>writeUInt8(char *buf, int *pos, int v)</td>
<td>Data marshalling routines</td>
</tr>
<tr>
<td>writeUInt16(char *buf, int *pos, int v)</td>
<td>Data marshalling routines</td>
</tr>
<tr>
<td>writeFloat(char <em>buf, int</em>pos, float f)</td>
<td>Data marshalling routines</td>
</tr>
<tr>
<td>send(char *buf, int len)</td>
<td>Send message buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Incoming command primitives</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>readUInt8(char *buf, int *pos, char *v)</td>
<td>Data unmarshalling routines</td>
</tr>
<tr>
<td>readUInt16(char *buf, int *pos, int *v)</td>
<td>Data unmarshalling routines</td>
</tr>
<tr>
<td>readFloat(char <em>buf, int</em>pos, float *f)</td>
<td>Data unmarshalling routines</td>
</tr>
</tbody>
</table>
Application Example

- WSN for monitoring environmental parameters
- Feed data into greenhouse control model
- Some of the monitored parameters may be a non-trivial function of basic parameters
  - Vapor Pressure Deficit (VPD)
    - function of Temperature (T) and Relative Humidity (RH)
- Compute on-board, and send an update only in case of significant deviation from previous reports
- Flexible application-level implementation
  - 40 lines of code
  - executable of 2Kbytes
```c
#define SETTHRESHOLD_CMD 1

float diff;

... on_init() {
    int rates[] = {60, 60};
    diff = 0.5;
    set_sensors(TEMP | HUM, rates);
}

... on_command(int id, char *buf) {
    int pos = 0;
    if (id == SETTHRESHOLD_CMD) {
        readFloat(buf, &pos, &diff);
    }
}

... on_sensordata(int mask, float vals[]){
    int pos, update = 0;
    float vpd;
    if (TEMP | mask) {
        tmp = vals[TEMP-1]; update = 1;
    }
    if (HUM | mask) {
        hum = vals[HUM-1]; update = 1;
    }
    if (update) {
        vpd = calcVPD(tmp, hum);
        if (abs(vpd - lastvpd) > diff) {
            setType(mbuf, UPDATE_MSG);
            pos = 1;
            writeFloat(mbuf, &pos, &vpd);
            send(mbuf, pos);
            lastvpd = vpd;
        }
    }
}
```

**Application command description file**

```xml
<command>
  <uname> setthreshold </uname>
  <code>1</code>
</command>

<param>
  <uname> absdiff </uname>
  <valtype> float </valtype>
</param>
</datamsg>

**Application data description file**

```xml
<datamsg>
  <uname> update </uname>
  <code>1</code>
  <data>
    <uname> vpd </uname>
    <valtype> float </valtype>
  </data>
</datamsg>
```
Send application command

```c
#define SETTHRESHOLD_CMD 1

... on_command(int id, char *buf) {
    int pos=0;
    if (id == SETTHRESHOLD_CMD) {
        readFloat(buf, &pos, &diff);
    }
}
```

**Application command description file**

```xml
<command>
    <uname> setthreshold </uname>
    <code>1</code>
    <param>
        <uname> absdiff </uname>
        <valtype> float </valtype>
    </param>
</command>
```

/GW/TranslaMon/Engine

Node

```
/taskcmd?adrr0=12&addr1=25&cmd=setthreshold&absdiff=0.95
```

I4T 2016, Berlin, Germany
Receive application data

... on_init()
...  
10     set_sensors(TEMP | HUM, rates);
11     
12     on_sensordata(int mask, float vals[]){
13         int pos, update = 0; float vpd;
14         if (TEMP | mask) {
15             tmp = vals[pos]; update = 1;
16         }
17         if (HUM | mask) { hum = vals[pos]; update = 1; }
18         if (update) {
19             vpd = calcVPD(tmp, hum);
20             if (abs(vpd - lastvpd) > diff) {
21                 setType(mbuf, UPDATE_MSG);
22                 pos = 1;
23                 writeFloat(mbuf, &pos, vdp);
24                 send(buf, pos);
25                 lastvpd = vpd;
26             }
27         }
28     }

Application data object

{rimeaddress: [12,25],
timestamp: 'YYYY:MM:DD:HH:mm:ss',
version: [0,65],
lqi: [0,108],
message: {'update',
rssi: [-95,0],
values: {
    vpd: (float)
}}

Application data description file

<Data>
  <uname> update </uname>
  <code> 1 </code>
  <data>
    <uname> vpd </uname>
    <valtype> float </valtype>
  </data>
</Data>
Implementation Details

- Middleware implemented on top of Contiki
- Protocol support (ContikiMAC, Rime)
- Application code is transformed into an ELF executable by a pre-processor
- Support for storing ELF binaries on external flash and loading them on RAM

- AdvanticSys TelosB nodes (48K ROM, 10K RAM)
- OLinuXino mini-PC as gateway (Linux)
- Gateway software written in Python
- CherryPy server
Future Work

- Support for mesh or cluster-based topologies
  - new services, extend the API
- Support for broadcast messages to control multiple nodes at once
- Design high-level and more user-friendly programming language
  - which can then be mapped to lower-level C code
- Alternative implementation of gateway software using NodeJS
Acknowledgment

This work was performed in the framework of EPEA-TH / KRIPIIS project within GSRT’s KRIPIIS action, funded by Greece and the European Regional Development Fund of the European Union under NSRF 2007-2013 and the O.P. Competitiveness and Entrepreneurship.
Thank you!

Questions?