Kernel-based ARchitecture for safetY-critical cONtrol

# KARYON

**FP7-288195**

# D3.3 – Working prototype of adaptive middleware

| | | | |
|---|---|---|---|
| Work Package | WP3 | | |
| Due Date | M24 | Submission Date | 2013-12-03 |
| Main Author(s) | Jörg Kaiser (OVGU), José Rufino (FFCUL) | | |
| Contributors | Christoph Steup, Tino Brade (OVGU), José Rufino (FFCUL), Jeferson Souza (FFCUL), Rui Caldeira (FFCUL), André Guerreiro (FFCUL) | | |
| Version | 1.0 | Status | Final |
| Dissemination Level | Public | Nature | Software |
| Keywords | Wireless protocols, inaccessibility analysis, adaptive middleware, mixed reality | | |

Part of the Seventh

Framework Programme

Funded by the EC - DG INFSO

## Version history

| Rev | Date | Author | Comments |
|---|---|---|---|
| V0.1 | 2013-11-18 | Jörg Kaiser (OVGU) | Preliminary Draft release. |
| V0.2 | 2013-11-29 | Jeferson Souza (FFCUL) | FFCUL contributions. |
| V0.3 | 2013-11-30 | António Casimiro (FFCUL) | Made minor corrections. |
| V1.0 | 2013-12-03 | António Casimiro (FFCUL) | Final review and delivery. |

## Executive Summary

This document provides a brief description of the software that has been developed in WP3. The software is made available in the KARYON svn as release packages under public licence. It may also be accessible in local version management systems to allow shared development.

The following packages have been made available:

1. IEEE 802.15.4 Network Inaccessibility Evaluation Tool. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
2. NS-2 Simulator module with GTS (Guaranteed Time Slots) support for frame transmissions on IEEE 802.15.4 wireless networks. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
3. NS-2 fault injector and timeliness evaluator component for IEEE 802.15.4 wireless networks. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
4. Wireshark extension modules for fault-injection, monitoring and evaluation of IEEE 802.15.4 wireless networks. It are available through https://bitbucket.org/rpcaldeira/karyon-wireshark/get/master.zip and https://bitbucket.org/rpcaldeira/karyon-adapter/get/master.zip.
5. FAMOUSO Middleware. This is the central part of the KARYON publish subscribe middleware. It is available through the FAMOUSO repository at https://svn-eos.cs.ovgu.de/repos/staff/mschulze/Research/MIKRO/famouso.
6. For deeply embedded systems we provide two hardware abstraction libraries for AVR and ARM-based microcontrollers. These are available in respective repositories at https://github.com/steup/AVR-HaLib and https://github.com/steup/ARM-HaLib.
7. A toolset for testing, validating and integrating sensors in our prototyping system based on ROS. The individual tools are currently available at http://eos.cs.ovgu.de/de/crew/dietrich.

# Table of Contents

# 1. Introduction

This document provides a brief description of the software that has been developed in WP3. The software is made available in KARYON repositories as a set of release packages under public licence. The software may also be accessible in local version management systems to allow shared development.

The following packages have been made available:

1. IEEE 802.15.4 Network Inaccessibility Evaluation Tool. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
2. NS-2 Simulator module with GTS (Guaranteed Time Slots) support for frame transmissions on IEEE 802.15.4 wireless networks. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
3. NS-2 fault injector and timeliness evaluator component for IEEE 802.15.4 wireless networks. It is available through the KARYON web site, at http://www.karyon-project.eu/documents/software-tools/.
4. Wireshark extension modules for fault-injection, monitoring and evaluation of IEEE 802.15.4 wireless networks. It are available through https://bitbucket.org/rpcaldeira/karyon-wireshark/get/master.zip and https://bitbucket.org/rpcaldeira/karyon-adapter/get/master.zip.
5. FAMOUSO Middleware. This is the central part of the KARYON publish subscribe middleware. It is available through the FAMOUSO repository at https://svn-eos.cs.ovgu.de/repos/staff/mschulze/Research/MIKRO/famouso.
6. For deeply embedded systems we provide two hardware abstraction libraries for AVR and ARM-based microcontrollers. These are available in respective repositories at https://github.com/steup/AVR-HaLib and https://github.com/steup/ARM-HaLib.
7. A toolset for testing, validating and integrating sensors in our prototyping system based on ROS. The individual tools are currently available at http://eos.cs.ovgu.de/de/crew/dietrich.

# 2. Predictability and Resilience in Embedded Networks

A set of tools were developed in the context of Task 3.1 of WP3 to help in the analysis and evaluation of embedded networks, concerning aspects related to their predictable and resilient operation. Those tools to perform theoretical, simulation-based, and experimental evaluation of IEEE 802.15.4 networks were specifically designed to analyse and evaluate: the impact of network inaccessibility on the timeliness of the network; and the network behaviour under error conditions.

## 2.1 IEEE 802.15.4 Network Inaccessibility Evaluation Tool (LibreOffice)

The network inaccessibility evaluation tool supports the theoretical study of network inaccessibility on IEEE 802.15.4 networks. This tool is a LibreOffice spreadsheet built-in, helping the offline calculation of the network inaccessibility duration, as given by our theoretical model described within the D3.1 document.

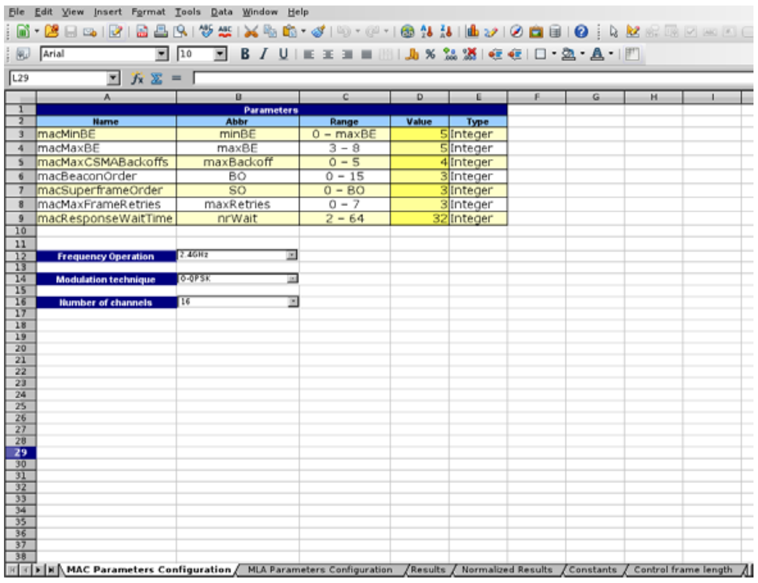
The tool is composed by different sheets, presenting the network configuration parameters in an intuitive way, and the numerical and graphical results of network inaccessibility for a given configuration. Only the sheets/cells specifying the network configuration parameters can be modified.

The utilisation of the KARYON network inaccessibility evaluation tool requires a basic knowledge of LibreOffice calc and computer networks, more specifically about the medium access control (MAC) sublayer and the IEEE 802.15.4 wireless standard. Each parameter has an additional explanation to help the user to understand it's semantic and purpose, and how such parameter influences the temporal behaviour of the network. Such explanations show up when a parameter value is selected.

The first thing that has to be done is the configuration of the MAC parameters, using the interface of the *MAC Parameters Configuration* sheet, depicted in Figure 1. The parameters presented in this interface defines the temporal behaviour of the network, including, e.g., the network cycle (i.e., the beacon interval, $\mathcal{T}_{BI}$) and the number of transmission retries in case of the presence of faults.

The other important parameters are related to the durations of management operations, performed by the MAC sublayer and represented by the acronym *MLA*. The duration regarding the execution of such operations is not explicitly defined in the IEEE 802.15.4 standard. By default, each *MLA* operation is set to an uniform value of $T_{MLA}(action) = T_{BI}/10$, as illustrated in Figure 2. However, these values can be changed and replaced by the real processing times of each IEEE 802.15.4 specific platform.

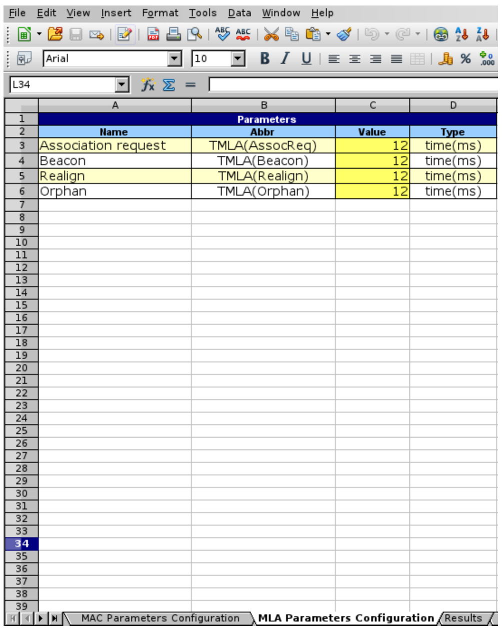
Assuming that MAC and *MLA* parameters were configured, the duration of network inaccessibility scenarios can be evaluated and visualised as absolute values in milliseconds (*ms*), or as normalised values of $\mathcal{T}_{BI}$ units of time. Figure 3 presents an example of the absolute (Figure 3a) and normalised (Figure 3b) results obtained from the tool.



| Parameters | | | | |
|---|---|---|---|---|
| Name | Abbr | Range | Value | Type |
| macMinBE | minBE | 0 – maxBE | 5 | Integer |
| macMaxBE | maxBE | 3 – 8 | 5 | Integer |
| macMaxCSMABackoffs | maxBackoff | 0 – 5 | 4 | Integer |
| macBeaconOrder | BO | 0 – 15 | 3 | Integer |
| macSuperframeOrder | SO | 0 – BO | 3 | Integer |
| macMaxFrameRetries | maxRetries | 0 – 7 | 3 | Integer |
| macResponseWaitTime | nrWait | 2 – 64 | 32 | Integer |

| | |
|---|---|
| Frequency Operation | 2.4GHz |
| Modulation technique | O-QPSK |
| Number of channels | 16 |

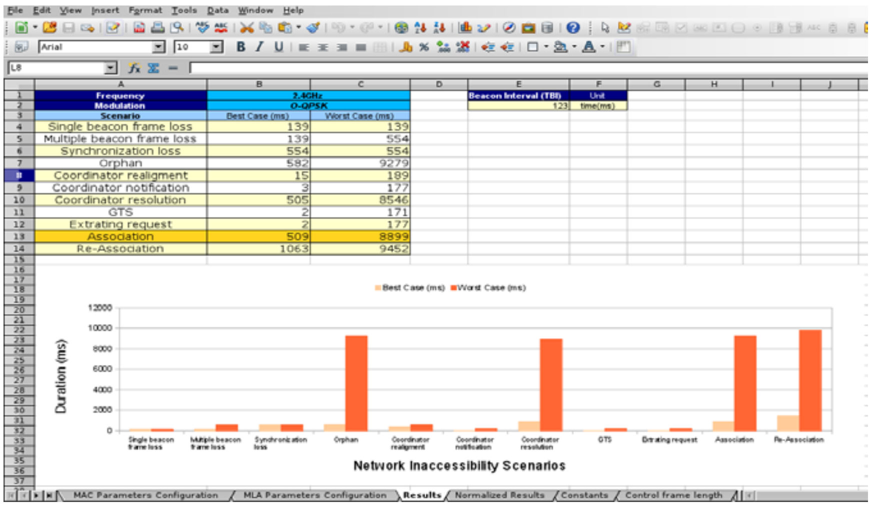
**Figure 1 - The interface for the MAC parameter configuration.**



| Parameters | | | |
|---|---|---|---|
| Name | Abbr | Value | Type |
| Association request | TMLA(AssocReq) | 12 | time(ms) |
| Beacon | TMLA(Beacon) | 12 | time(ms) |
| Realign | TMLA(Realign) | 12 | time(ms) |
| Orphan | TMLA(Orphan) | 12 | time(ms) |

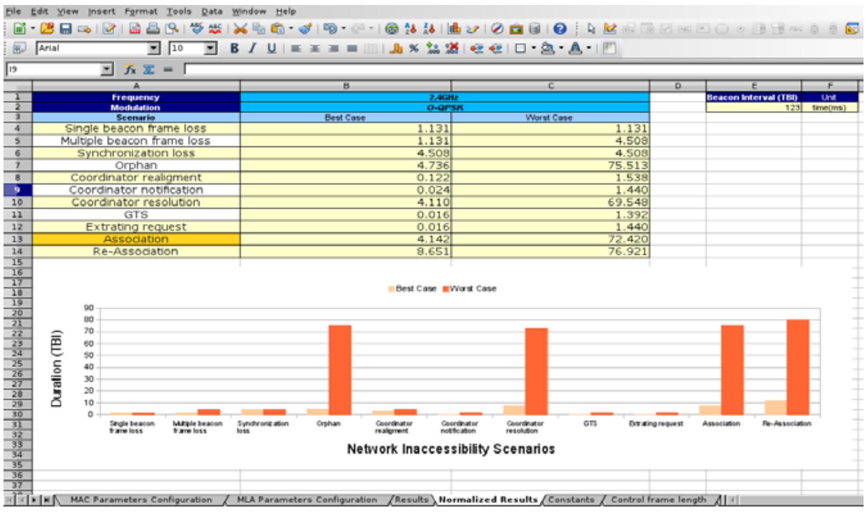
**Figure 2 - The MLA Parameter Configuration sheet.**

Additionally, the tool also incorporates mechanisms to evaluate and apply reduction policies to minimise the impact of network inaccessibility on the IEEE 802.15.4 network operation, as illustrated in Figure 4. The scientific explanation and detailed description of each reduction policy, as the impact in the network dependability and timeliness, is presented in [2] . The tool is available at the KARYON website (http://www.karyon-project.eu/wp-content/uploads/2013/09/Inaccessibility_IEEE802.15.4_Beacon-enabled-Karyon.ods). The LibreOffice suite has to be installed to use the tool, and can be downloaded at its official website (http://www.libreoffice.org/).



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Frequency | 2.4GHz | | | Beacon Interval (TBI) | Unit |
| 2 | Modulation | O-QPSK | | | 123 | time(ms) |
| 3 | Scenario | Best Case (ms) | Worst Case (ms) | | | |
| 4 | Single beacon frame loss | 139 | 139 | | | |
| 5 | Multiple beacon frame loss | 139 | 554 | | | |
| 6 | Synchronization loss | 554 | 554 | | | |
| 7 | Orphan | 582 | 9279 | | | |
| 8 | Coordinator realigment | 15 | 189 | | | |
| 9 | Coordinator notification | 3 | 177 | | | |
| 10 | Coordinator resolution | 505 | 8546 | | | |
| 11 | GTS | 2 | 171 | | | |
| 12 | Extrating request | 2 | 177 | | | |
| 13 | Association | 509 | 8899 | | | |
| 14 | Re-Assocation | 1063 | 9452 | | | |

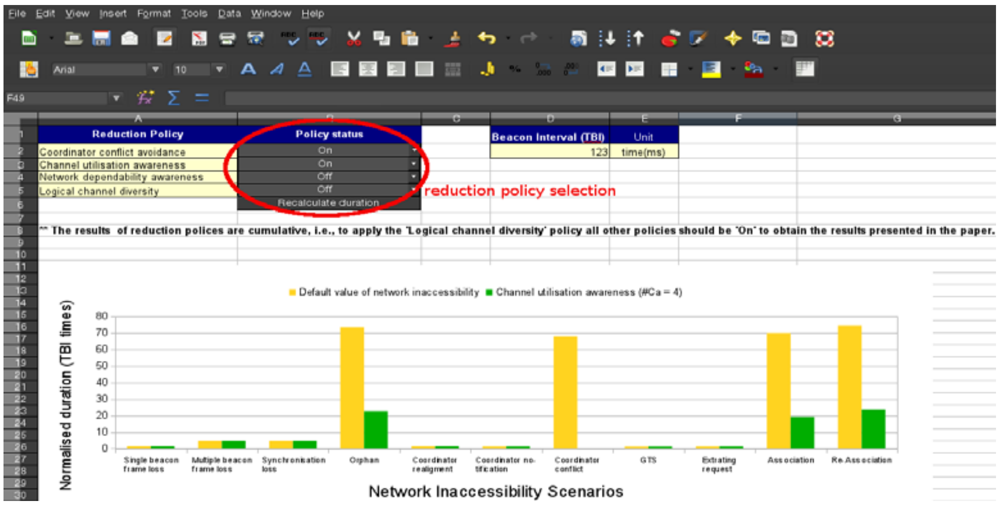
a) Results



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Frequency | 2.4GHz | | | Beacon Interval (TBI) | Unit |
| 2 | Modulation | O-QPSK | | | 123 | time(ms) |
| 3 | Scenario | Best Case | Worst Case | | | |
| 4 | Single beacon frame loss | 1.131 | 1.131 | | | |
| 5 | Multiple beacon frame loss | 1.131 | 4.508 | | | |
| 6 | Synchronization loss | 4.508 | 4.508 | | | |
| 7 | Orphan | 4.736 | 75.513 | | | |
| 8 | Coordinator realigment | 0.122 | 1.538 | | | |
| 9 | Coordinator notification | 0.024 | 1.440 | | | |
| 10 | Coordinator resolution | 4.110 | 69.548 | | | |
| 11 | GTS | 0.016 | 1.392 | | | |
| 12 | Extrating request | 0.016 | 1.440 | | | |
| 13 | Association | 4.142 | 72.420 | | | |
| 14 | Re-Assocation | 8.651 | 76.921 | | | |

b) Normalised results

**Figure 3 - The duration of network inaccessibility scenarios for IEEE 802.15.4 wireless networks.**


File Edit View Insert Format Tools Data Window Help
Arial
10
F49
reduction policy selection
Default value of network inaccessibility
Channel utilisation awareness (#Ca = 4)
Normalised duration (TBI times)
80
70
60
50
40
30
20
10
0
Single beacon frame loss
Multiple beacon frame loss
Synchronisation loss
Orphan
Coordinator realigment
Coordinator notification
Coordinator conflict
GTS
Extrating request
Association
Re-Association
Network Inaccessibility Scenarios

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Reduction Policy | Policy status | | Beacon Interval (TBI) | Unit |
| 2 | Coordinator conflict avoidance | On | | 123 | time(ms) |
| 3 | Channel utilisation awareness | On | | | |
| 4 | Network dependability awareness | Off | | | |
| 5 | Logical channel diversity | Off | | | |
| 6 | | Recalculate duration | | | |

** The results of reduction polices are cumulative, i.e., to apply the 'Logical channel diversity' policy all other policies should be 'On' to obtain the results presented in the paper.

**Figure 4 - The Reduction policies sheet.**

## 2.2 NS-2 Simulator module with GTS (Guaranteed Time Slots) support for frame transmissions on IEEE 802.15.4 wireless networks

Guaranteed time slots (GTS) are time slots allocated to provide contention-free access to the network, being designed to support communications with real-time requirements. The GTS support is not implemented in the native version of the IEEE 802.15.4 module of the network simulator 2 (NS-2), being not possible to perform simulations of networks with real-time traffic. This module provides the GTS support for the NS-2 IEEE 802.15.4 module.

The provision of the GTS support involves the adaptation of a previous GTS implementation proposed by [3] with enhancements developed by the KARYON team. Real-time traffic, supported by the introduction of the GTS support, is transmitted within IEEE 802.15.4 networks according the Algorithm 1. Every transmission is only allowed within the allocated GTS for a given node, resulting in a time division multiple access (TDMA) approach to control the access to the network.

```
1: Begin.
2: MAC.Data.Send.Request(data);
3: when allocated GTS is reached do
4: MAC.Data.transmit(data);
5: end when
6: End.
```

**Algorithm 1 - Data transmissions using GTS**

The referenced implementation proposed in [3] prevents the use of MAC management commands such as Orphan Notification and Coordinator Realignment. The KARYON team create a path to fix it, allowing the activation of the GTS mechanism from the NS-2 script without affecting other MAC sublayer services.

Figure 5 presents the main classes of the IEEE 802.15.4 module, evidencing the modified and implemented methods related to the GTS mechanism incorporated by the KARYON team.

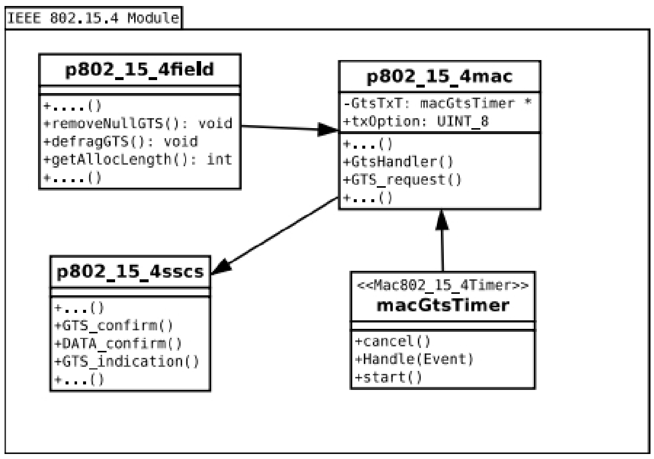


**Figure 5 - Class diagram of changed classes on native IEEE 802.15.4 module.**

The methods of the *p802_15_4field* and the *macGtsTimer* classes are utilised by the *p802_15_4mac* class within its management procedures related to the GTS extension. The *p802_15_4sscs* class represents the connection between the MAC and the logical link control (LLC) sublayer, providing a way to access all the methods present in the MAC exposed service interface. The source code of the IEEE 802.15.4 module with the GTS support can be downloaded at: http://www.karyon-project.eu/wp-content/uploads/2013/11/ns-2_v2_35_802_15_4_gts_extension.zip. Instructions about how to install this module within the NS-2 (version 2.35) are found inside the available package.

## 2.3 NS-2 Simulator extension module for fault-injection and timeliness evaluation of IEEE 802.15.4 wireless networks under error conditions

The KARYON project complements the NS-2 with the integration of a new fault injector and a temporal analysis component, which are utilised to evaluate IEEE 802.15.4 wireless networks, as represented in Figure 6.

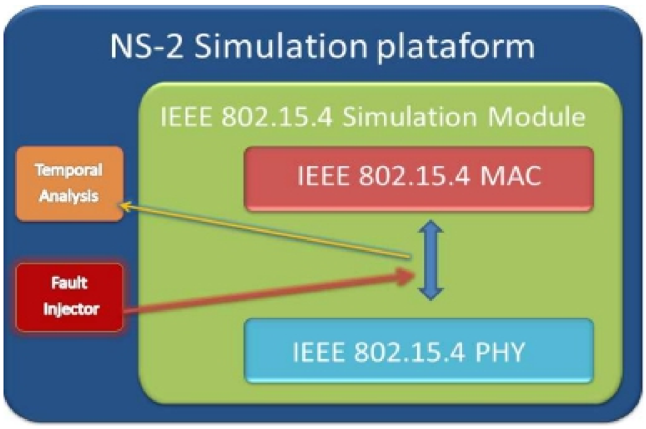


**Figure 6 - New Features in IEEE 802.15.4 module.**

The KARYON NS-2 fault injector is capable to use a fault pattern to introduce within networking communications, during the simulation. The criteria to define the fault pattern is totally configurable, allowing the definition of deterministic or probabilistic fault patterns. The fault injection scheme is depicted in Figure 7.

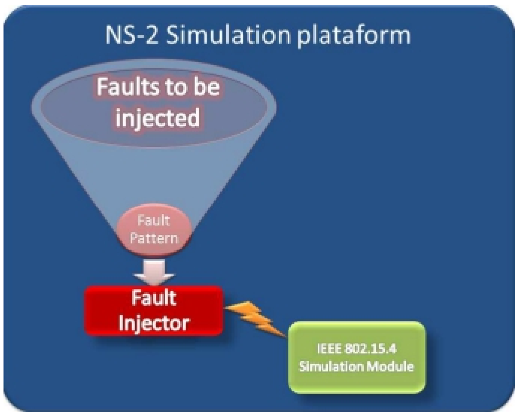


**Figure 7 - Fault injector scheme.**

The fault injector can be customised regarding the type of frame to be corrupted, the injection rate, and the duration of the fault injection campaign. Random noise or interferences are simulated according to the random function implemented in the fault injector, as described in Algorithm 2.

```
1: Begin.
2: randomTime = randomGenerator(seed);
3: NewRandomEvent = faultInjector(frameToCorrupt);
4: Scheduler :: instance().schedule(NewRandomEvent,randomTime);
5: CorruptNode.Update();
6: End.
```

**Algorithm 2 - Fault Injector - A random function.**

Random noises are generated through a pseudo-random generator, described in line 2. A new event to perform the frame corruption is created, as indicated in line 3. The created event is inserted in the NS-2 scheduler and executed at the defined instant of time, as illustrated in line 4. An information about the corruption occurred in a specific node is recorded, as described in line 5.

The fault injector achieves the frame corruption as described in Algorithm 3. The faults are injected by changing a bit in the frame content, which implies the drop of these frames in the MAC level of the receiving nodes. The parameter *frameToCorrupt*, represented in line 3, is previously defined and if desired all the received frames can be affected.

An information about the corruption occurred in a specific node is recorded, as represented in line 6. This information is used for a better control of the simulation events. The corruption of the frames can be disabled, through the deactivation of the fault injector on a tcl script, being the normal behaviour of the network restored at any time.

```
1: Begin.
2: MAC.Receive(frame);
3: if frame = frameToCorrupt then
4: when selected Fault Pattern do
5: CommandHeader(frame)− > error() = 1;
6: CorruptNode.Update();
7: end when
8: end if
9: End.
```

**Algorithm 3 - Fault Injector Mechanism.**

Additionally, the KARYON team created the temporal analysis component to measure the effects of fault injection campaigns on the simulated network. This component is responsible to evaluate time events, in special the duration of periods of inaccessibility associated with the corruption of specific MAC control frames, such as the beacon frame.

The temporal analysis component produces a report regarding the recorded events during the simulation, which is stored within a log file. The log file is used as input to a gnuplot script that produces a graphic analysis of recorded events.

The source code of the fault injector and the temporal analysis component can be downloaded at: http://www.karyon-project.eu/wp-content/uploads/2013/11/ns-2_v2_35_fault_injector_module.zip. Instructions about how to install the additional tools in the NS-2 (version 2.35) are found in the available package.

## 2.4 Wireshark extension module for fault-injection, monitoring and evaluation of IEEE 802.15.4 wireless networks

In order to perform an experimental evaluation of the IEEE 802.15.4 networks, the KARYON team have built a traffic analysis tool, as an extension the well-known network protocol analyser tool dubbed WireShark (http://www.wireshark.org/). This extension allows the use of the Atmel REB232ED-EK Evaluation Kit (Figure 8) with the WireShark, which is an IEEE 802.15.4 compliant wireless node. The general view of the modules developed by the KARYON team, and integrated with the WireShark, is illustrated in the Figure 9.

**Figure 8 - Atmel REB232ED-EK Evaluation Kit**

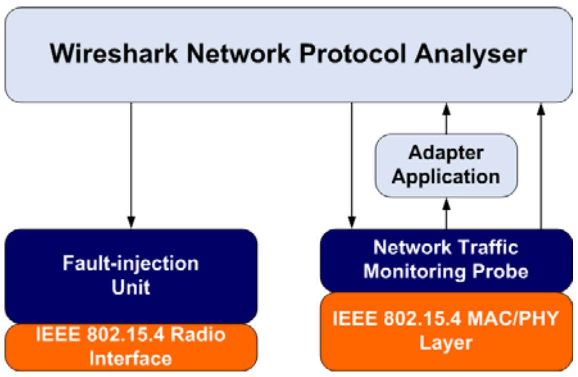

Wireshark Network Protocol Analyser
Adapter Application
Fault-injection Unit
IEEE 802.15.4 Radio Interface
Network Traffic Monitoring Probe
IEEE 802.15.4 MAC/PHY Layer

**Figure 9 – Integration of developed modules with WireShark**

The wireless nodes illustrated in Figure 8 has the promiscuous mode activated, enabling the capture of all traffic flowing through the network, even if such traffic contains errors. The captured traffic is written to the serial connection on the computer where the WireShark is in execution, converting the raw data obtained from the network to a pcap-savefile, which can be interpreted by the WireShark tool.

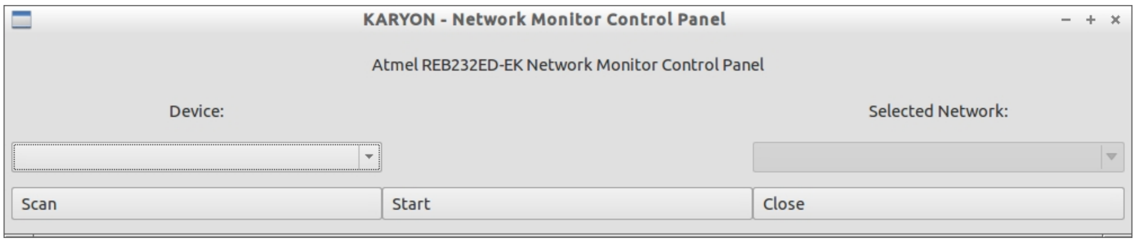
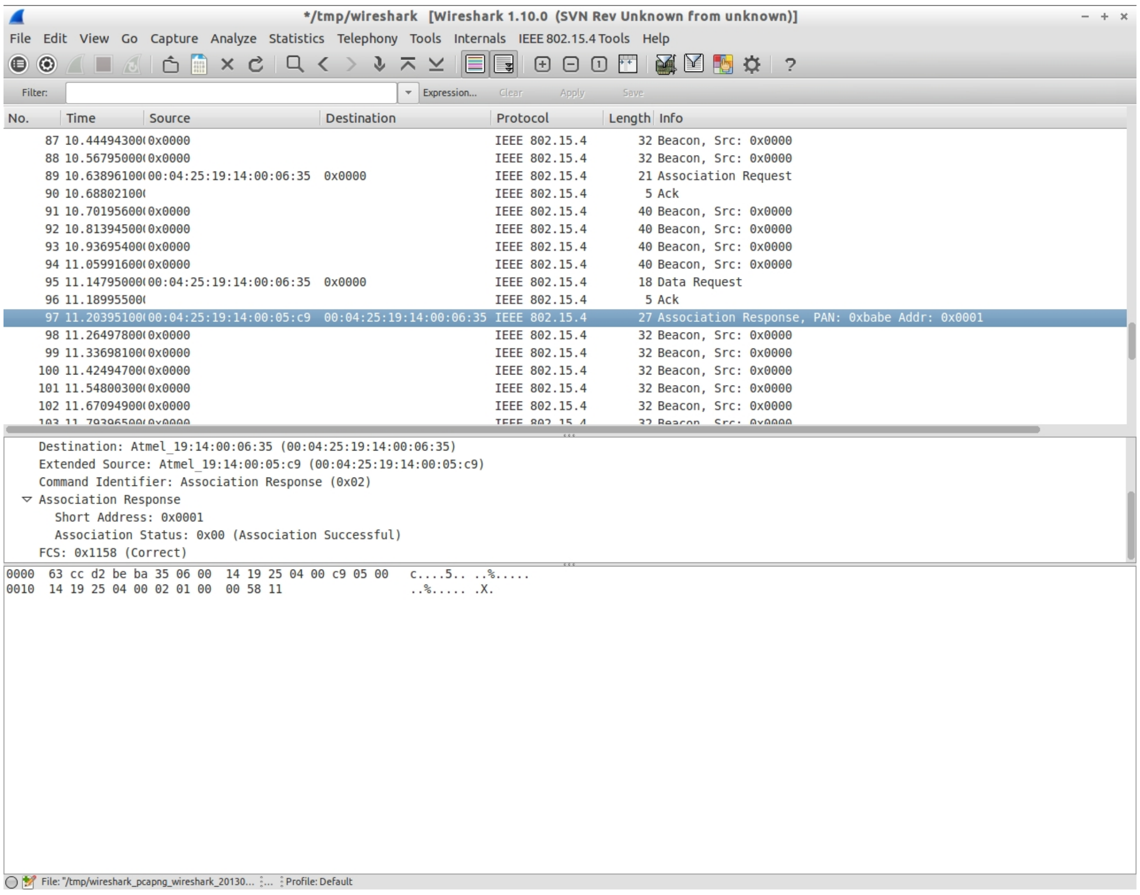

KARYON - Network Monitor Control Panel
Atmel REB232ED-EK Network Monitor Control Panel
Device:
Selected Network:
Scan
Start
Close

**Figure 10 - Network Monitor Control Panel.**

Figure 10 presents the control panel of the network monitor, accessed in the IEEE 802.15.4 Tools menu of the WireShark tool. In this control panel the user can select the serial port where the wireless node is connected, being therefore able to star the scan of communication channels supported by the IEEE 802.15.4 standard. The scan procedure is started pressing the scan button. This scan procedure searches for IEEE 802.15.4 networks, inserting the found networks in the dropdown list below to the label "Selected Network". If the network intended to be monitored was found during the performed scan, the monitoring process can be initiated selecting the network on such list and pressing the start button.

The analyses of the captured data from the experiments can be done directly on the WireShark (depicted in Figure 4), using the already existent features to help such analyses, which includes embedded statistics, expert info, graphical representation of the frames (representation in ASCII), and the possibility to save the captured data for future analysis.


*/tmp/wireshark [Wireshark 1.10.0 (SVN Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals IEEE 802.15.4 Tools Help
Filter:
Expression...
Clear
Apply
Save
File: "/tmp/wireshark_pcapng_wireshark_20130...
Profile: Default

```
No.  Time          Source                   Destination              Protocol      Length Info
  87 10.44494300(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
  88 10.56795000(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
  89 10.63896100(00:04:25:19:14:00:06:35  0x0000                   IEEE 802.15.4     21 Association Request
  90 10.68802100(                                                  IEEE 802.15.4      5 Ack
  91 10.70195600(0x0000                                            IEEE 802.15.4     40 Beacon, Src: 0x0000
  92 10.81394500(0x0000                                            IEEE 802.15.4     40 Beacon, Src: 0x0000
  93 10.93695400(0x0000                                            IEEE 802.15.4     40 Beacon, Src: 0x0000
  94 11.05991600(0x0000                                            IEEE 802.15.4     40 Beacon, Src: 0x0000
  95 11.14795000(00:04:25:19:14:00:06:35  0x0000                   IEEE 802.15.4     18 Data Request
  96 11.18995500(                                                  IEEE 802.15.4      5 Ack
  97 11.20395100(00:04:25:19:14:00:05:c9  00:04:25:19:14:00:06:35 IEEE 802.15.4     27 Association Response, PAN: 0xbabe Addr: 0x0001
  98 11.26497800(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
  99 11.33698100(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
 100 11.42494700(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
 101 11.54800300(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
 102 11.67094900(0x0000                                            IEEE 802.15.4     32 Beacon, Src: 0x0000
```

```
    Destination: Atmel_19:14:00:06:35 (00:04:25:19:14:00:06:35)
    Extended Source: Atmel_19:14:00:05:c9 (00:04:25:19:14:00:05:c9)
    Command Identifier: Association Response (0x02)
  ▽ Association Response
      Short Address: 0x0001
      Association Status: 0x00 (Association Successful)
    FCS: 0x1158 (Correct)
```

```
0000  63 cc d2 be ba 35 06 00  14 19 25 04 00 c9 05 00   c....5.. ..%.....
0010  14 19 25 04 00 02 01 00  00 58 11                  ..%..... .X.
```

**Figure 11 - The WireShark capture window.**

The KARYON team also developed a fault injector to emulate accidental faults in IEEE 802.15.4 networks, such as loss or corruption of legitimate data. This fault injector makes possible to subject a network to inject faults according the following parameters:

- *Minimum Inter-Injection Time*: the minimum time separation between fault injections;
- *Fault-Injection Duration*: The duration of the fault injection;
- *Fault Pattern*: The pattern of the fault injected on the network. The minimum pattern length is constituted by 1 byte and the maximum pattern length is 127 bytes. The bytes included in the pattern can be generated randomly, or can constitute a valid IEEE 802.15.4 frame;
- *Frame arrival notification* – This notification enables the fault injector to analyse the network, using the timestamp and type of received frames as inputs to a specific fault scenario.

A set of five pre-defined fault injection modes were specified to help the evaluation of IEEE 802.15.4 wireless networks. In summary, these pre-defined modes set values for the parameters *Minimum Inter-Injection Time*, *Fault-Injection Duration*, *Fault pattern*, and *Frame arrival notification,* which are utilised to control the fault injection. Each one of these pre-defined modes is described as follows:

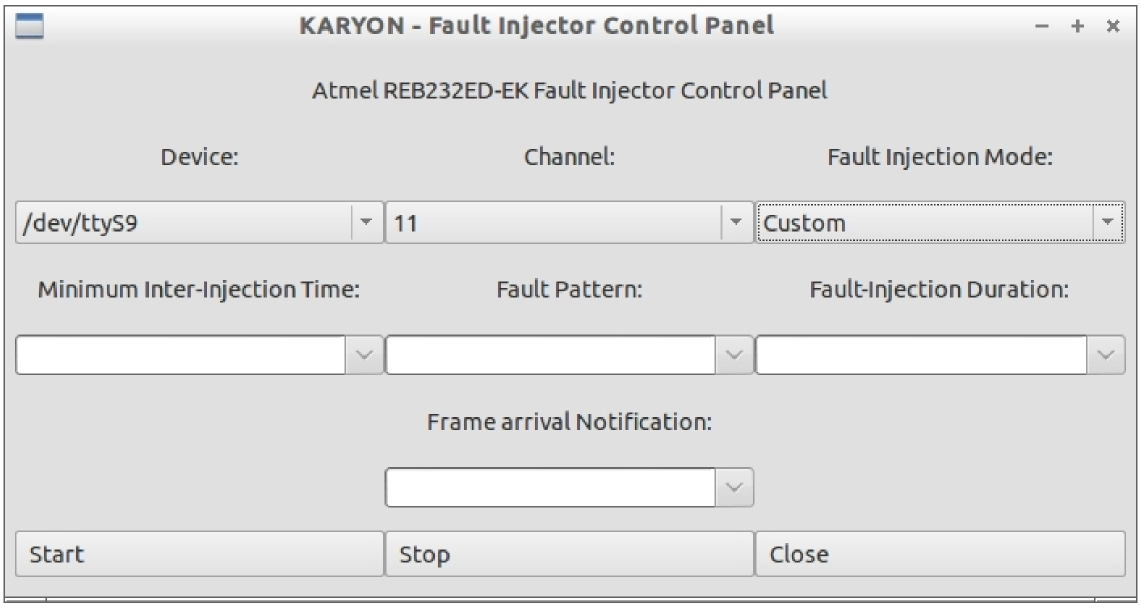
- **Constant** - This fault scenario emits a constant noise to the network. This scenario block all communications through the network while active;

- **Random** - This scenario defines a random minimum inter-injection time for each fault injection. The duration of the fault injection is also random;

- **Deceptive** - This fault scenario constantly sends out valid frames keeping the remaining nodes in a constant listening mode;

- **Adaptive** - The adaptive scenario used the received Frame arrival notification receives a notification to inject faults, according the frame arrival pattern into a list of the timestamps of most recent twenty received frames;
- **Frame-type Adaptive** - The frame-type adaptive scenario is an extension of the Adaptive scenario, which use a specific frame type to perform the fault injections.

Additionally, the user of the fault injector can select a custom fault injection mode, where all the aforementioned parameters can be configured manually.


KARYON - Fault Injector Control Panel
Atmel REB232ED-EK Fault Injector Control Panel
Start
Stop
Close

Device: /dev/ttyS9
Channel: 11
Fault Injection Mode: Custom
Minimum Inter-Injection Time:
Fault Pattern:
Fault-Injection Duration:
Frame arrival Notification:

**Figure 12 - Fault Injector Control Panel.**

Figure 12 represents the fault injector control panel. The fault injection mode is chosen from the right dropdown list (below the label "Fault Injection Mode"), which is composed by the pre-defined fault injection modes, plus the custom mode. The start button sends commands to the same wireless node utilised in the analysis traffic tool, initiating the fault injection accordingly to the given fault injection mode.

The source code of the modules integrated into the WireShark tool can be downloaded from a protected repository (access to these tools has to be explicitly requested) at: https://bitbucket.org/rpcaldeira/karyon-wireshark/get/master.zip. The communication module responsible to connect WireShark with the ATMEL hardware is available in a separated repository, and can be downloaded at: https://bitbucket.org/rpcaldeira/karyon-adapter/get/master.zip. Instructions about how to compile those tools are found in the available packages.

# 3. Adaptive Middleware for Advanced Control Systems

The software provided has been developed based on the concepts elaborated in T3.2 of WP3. The first prototype version includes the complete publish/subscribe communication package for Linux and Windows operating systems. There are language bindings for C, C++, Python and Simulink.

## 3.1 How-To Install and use the KARYON Middleware

### 3.1.1 Overview

The KARYON Middleware consists of multiple layers of supporting technologies. This preliminary release delivers the communication abstracting platform independent publish subscribe mechanism of the FAMOUSO middleware. This document guides users through the steps necessary to use the software and integrate it in applications.

### 3.1.2 Sources

There are multiple distribution systems available for downloading the software. If development of the middleware is considered a version controlled download is available through:

https://svn-eos.cs.ovgu.de/repos/staff/mschulze/Research/MIKRO/famouso

This version contains the latest features and bug fixes from the development. It can be checked out using any SVN tool. Additional information may be obtained through the official webpage http://famouso.sourceforge.net/ as well as the wiki and bug tracker https://ivs-pm.ovgu.de/projects/famouso.

Alternatively, a direct download link is available. The link is provided by sourceforge and is a packed archive of the latest stable release of the software. I is available through:

http://sourceforge.net/projects/famouso/files/

### 3.1.3 Dependencies

All versions provided are source distributions, which need to be compiled on the target machine. The build process depends on some external tools, which need to be present in specific versions. The needed dependencies for **Linux** are:

- gcc = 4.7
- Boost = {1.52,1.53,1.54} 1.53 will be fetched automatically during setup
- make >= 3.80
- wget
-

On **Windows** these dependencies can be fulfilled through the installation of Cygwin www.cygwin.com.

### 3.1.4 Building the software on Windows

**Cygwin**

To successfully build the KARYON Middleware on Windows the user need to provide an appropriate infrastructure. Therefore, the installation of the Cygwin environment is necessary. Download the 32bit setup for Cygwin from the Cygwin homepage (www.cygwin.com). During installation choose your preferred install directory, but make sure it contains no spaces. When the installer asks for packages to install choose:

- gcc-g++
- make
- wget

The dependencies of these packages will be resolved automatically. Since Cygwin provides a Linux like environment on Windows all following steps are identical for both systems.

### 3.1.5 Building the software on Linux

**Download and Extraction**

The software can be fetched from the sources listed above. Depending on the selected source, the procedure differs slightly. If a VCS is used nothing needs to be done additionally. If the software was acquired through an archive, it needs to be copied to the Home directory of Cygwin. This is done by copying the file to C:\cygwin\home\<username>\. If cygwin was installed in another path, the command needs to be changed accordingly. If the downloaded file has a .zip extension it can be extracted from the Cygwin command line with “unzip <file.zip>”. If it has a .tar.* ending, then it can be extracted with “tar –xf <file.tar.*>”.

**Starting the build**

To start the build process, issue “make” in the Cygwin command line in the root folder of the extracted software. For a successful build internet access is needed, since some dependencies are fetched automatically. If you build on Windows change the used configuration to windows/cygwin. This can either be done by editing the file make/config.mk or on the command line by issuing “make config=windows/cygwin”. If the configuration is stated in the command line all calls to make need to be adapted similarly.

Afterwards the needed Event-Channel-Handlers supporting the used communication mechanism need to be built. They are located in ECHs subdirectory of the root directory of the software.

Currently the following ECHs are supported:

- ech: ECH for local event dissemination without access to networked PCs
- ech-UDP-BC: ECH for broadcast event dissemination within a single hop environment
- ech-UDP-MC: ECH for multicast event dissemination over multiple hops if routers support IP multicast
- ech-CAN: ECH for event dissemination over a CAN network using PEAK CAN Dongles

To build an individual ECH the command: “make <ech-name>” may be used.

## 3.2 Overview of the Software

### 3.2.1 KARYON Middleware Stack

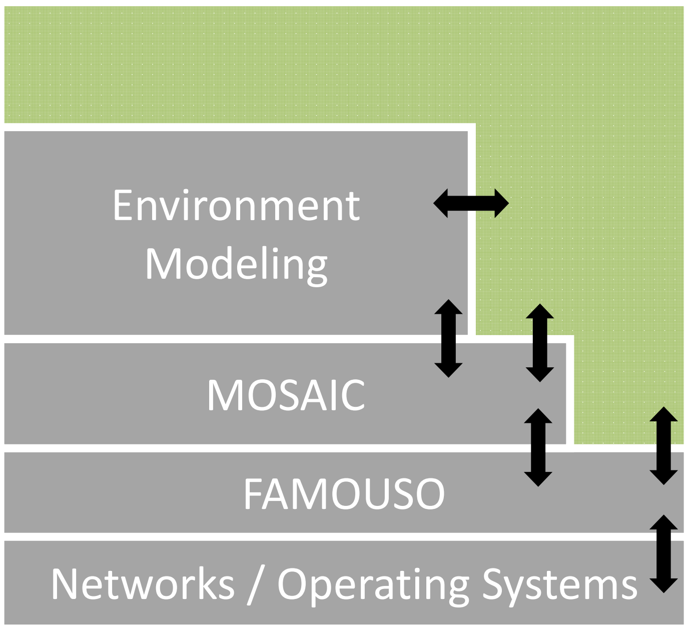


**Figure 13 – KARYON middleware stack.**

The KARYON middleware is subdivided in 3 major parts, as illustrated in Figure 13. The Mosaic and Environment Modeling parts are currently not included in the released stack. This release considers only the FAMOUSO communication abstractions.

### 3.2.2 Pub/Sub Communication

Publish Subscribe is a communication paradigm decoupling sender and receiver of data. This is done through subject-based routing mechanism. Therefore, a subscriber requiring a specific type of data may subscribe through the middleware to the subject of this data. Subsequently, the communication middleware disseminates generated events of this subject to the registered subscribers. This enables run-time binding between publishers and subscribers and consequently, an independent development of publisher and subscriber software. During run time, a virtual channel is established between each Publisher and Subscriber dynamically. This channel may have attributes attached, describing quality (e.g. latency) and context (e.g. time, location) parameters. At the moment, the attribute framework in not included in the distribution. It will be added at a later stage.

### 3.2.3 Layered Communication Stack

The communication middleware consist of a layer stack. Each layer is responsible for a certain abstraction in the communication as can be seen in Figure 14.

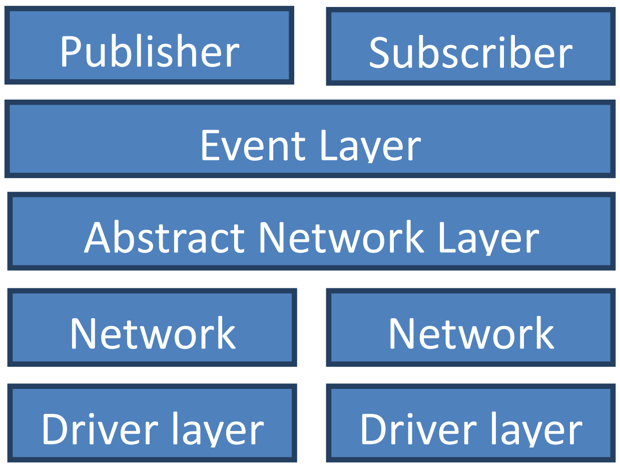


**Figure 14 – Layered communication stack.**

The Event Layer provides the notion of an event and handles the dissemination of events the local system. If an event needs to propagate through the network it is passed to the Abstract Network Layer which handles the selection of the appropriate output networks for the events. The specific Network Layer converts the event to one or multiple network packages. During this step the subject may be replaced by a network specific representation. Afterwards the packet stream is delivered to the driver of the network interface. Each layer may be subject to a user-specified configuration. In the following list, we describe configuration options:

- **Event Layer**: Base Event Layer forwarding events to networks or Event Layer Stubs binding multiple Applications to a single communication stack
- **Abstract Network Layer (ANL)**: The ANL may support an optional adaptive fragmentation protocol to disseminate large events. The ANL also handles multi-network configurations through the network adapter mechanism
- **Network Layer**: handling network specific bindings. Supported networks are currently: IP and CAN.
- **Driver Layer**: individual driver for the networks. Currently IP uses asynchronous sockets whereas CAN relies on PEAK CAN dongles.

## 3.3 Examples

The software is accompanied by different examples. In the examples directory are more complex examples. The Bindings directory contains the basic examples for the different language interfaces. As a starting point the basic publish and subscribe bindings for C++ will be discussed next.

### 3.3.1 A basic configuration

The following code represents a basic configuration of the middleware enabling interprocess communication on a single PC, depending on the selected event channel handler additional network support may be included.

```
#ifndef _famouso_bindings_h_
#define _famouso_bindings_h_

#include "mw/el/EventLayerClientStub.h"
```

```
05: #include "mw/api/EventChannel.h"
06: #include "mw/api/PublisherEventChannel.h"
07: #include "mw/api/SubscriberEventChannel.h"
08:
09: #include "mw/common/Event.h"
10: #include "famouso.h"
11:
12: namespace famouso {
13:   class config {
14:     public:
15:       typedef famouso::mw::el::EventLayerClientStub EL;
16:       typedef famouso::mw::api::PublisherEventChannel  < EL > PEC;
17:       typedef famouso::mw::api::SubscriberEventChannel < EL > SEC;
18:   };
19: }
20:
21: #endif
```

In line 01-02 and 21 an include guard is created to enable robust inclusion of this header file as a general configuration. Lines 04-07 include the individual components of the configuration:

- EventLayerClientStub as Event layer to support IPC with the event channel handler.
- PublisherEventChannel to support Publishers
- SubscriberEventChannel to support Subscribers

Additionally Lines 09-10 include generally needed headers of the middleware. Lines 12-19 contain the actual configuration which is included in the namespace of the middleware. The class *config* contains the definition needed by the middleware. The only needed definitions are *PEC* and *SEC*. These represent the configured event channels for publishing or subscribing. For each subject an instance of these channels need to be instantiated. The *EL* is only a helper definition to increase code readability.

This code is included in the release of the middleware as Bindings/include/famouso_bindings_config.h.

### 3.3.2 A simple publisher

The following code realizes a simple publishing application that periodically transmits the ASCII string “Publish” on the subject “__Test__” it uses the already defined middleware configuration.

```
01: #include "debug.h"
02: #include "famouso_bindings.h"
03:
04: #include <boost/thread/thread.hpp>
05: #include <boost/thread/xtime.hpp>

06: int main(int argc, char **argv) {
07:   famouso::init<famouso::config>();
08:   famouso::config::PEC pec(famouso::mw::Subject(“__Test__“);
09:   pec.announce();
10:
11:   famouso::mw::Event e(pec.subject());
12:   e.length = 7;
13:   e.data = (uint8_t*)"Publish";
14:
15:     while (1) {
```

```
16:       pec.publish(e);
17:       boost::xtime time;
18:       boost::xtime_get(&time, boost::TIME_UTC_);
19:       time.sec += 1;
20:       boost::thread::sleep(time);
21:     }
22: }
```

Lines 01-02 contain famouso specific headers like the previously defined configuration.

In lines 04-05 boost headers are included to enable platform independent sleeping. Beginning from line 06 the publisher application starts. In line 07 the middleware is initialized with the configuration *famouso::config*. In line 08 the needed publisher event channel is instantiated with the subject "__Test__", afterwards it will be announced to the network with line 09.

In line 11 a middleware event is created. It is initialized with the length of the data and a pointer to the data itself in line 12 and 13. The data pointer is expected to be of type *uint8_t** which is not the case for static strings resulting in a cast. Also the content length needs to be defined manually. In this case it is 7 because the contained string contains 7 ASCII-Letters. Therefore the 0-byte finishing the string will not be transmitted.

Lines 15 to 21 contain the periodic publication of this data. In line 16 the data is actively published to the network. Line 17 to 20 contain boost code to enable a platform independent sleep of 1s.

This code is available in the release of the middleware as Binding/C++/Publisher.cc

### 3.3.3 A simple subscriber

The following code realizes a subscriber listening for the data transmitted by the publisher of the previous section. Therefore the defined subject is the same.

```
01: #include "debug.h"
02: #include "famouso_bindings.h"
03:
04: #include <boost/thread/thread.hpp>
05: #include <boost/thread/xtime.hpp>
06:
07: void cb(famouso::mw::api::SECCallBackData& cbd) {
08:     ::logging::log::emit() << FUNCTION_SIGNATURE << " Length="
09:    << cbd.length << " Event data="
10:    << cbd.data << ::logging::log::endl;
11: }
12:
13: int main(int argc, char **argv) {
14:
15:   famouso::init<famouso::config>();
16:   famouso::config::SEC sec(famouso::mw::Subject("__Test__"));
17:   sec.subscribe();
18:   sec.callback.bind<cb>();
19:
20:   while (1) {
21:     boost::xtime time;
22:     boost::xtime_get(&time, boost::TIME_UTC_);
23:     time.sec += 100;
24:     boost::thread::sleep(time);
25:   }
26: }
```

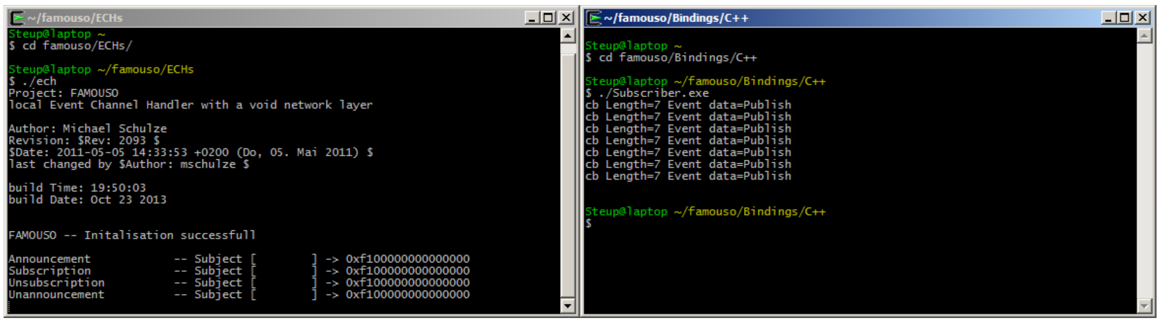
Lines 01 - 06 are equal to previously defined publisher. Line 07 to 11 defines the handling callback function for a reception of an event. Since we expect strings in the event we print the signature of the function we are currently in as well as the length of the data received and the data as a *uint8_t* array. This data is therefore printed as a basic ASCII string. Lines 13 to 15 are also equal to the publisher example. However in line 16 we define a subscriber event channel listening on the subject "__Test__" this time. Following this the subscription is announced to the network in line 17 and the defined callback function is bound to the channel in line 18. Lines 20 to 25 are similar to the publisher example, with the difference being the lack of any publication and a shorter sleep time of 100ms.

### 3.3.4 Running an example communication

To run the examples the ech needs to be started first. This is done by navigating in terminal to the ECHs folder of the middleware and issue the command *./ech*.

Afterwards the publisher and subscriber can be started in additional terminals in any order by issuing ./publisher or ./subscriber from the Bindings/C++ folder.

If everything went well the following output will be visible in the individual terminals, as shown in Figure 15.



```
Steup@laptop ~
$ cd famouso/ECHs/

Steup@laptop ~/famouso/ECHs
$ ./ech
Project: FAMOUSO
local Event Channel Handler with a void network layer

Author: Michael Schulze
Revision: $Rev: 2093 $
$Date: 2011-05-05 14:33:53 +0200 (Do, 05. Mai 2011) $
last changed by $Author: mschulze $

build Time: 19:50:03
build Date: Oct 23 2013


FAMOUSO -- Initalisation successfull

Announcement            -- Subject [        ] -> 0xf100000000000000
Subscription            -- Subject [        ] -> 0xf100000000000000
Unsubscription          -- Subject [        ] -> 0xf100000000000000
Unannouncement          -- Subject [        ] -> 0xf100000000000000
```

```
Steup@laptop ~
$ cd famouso/Bindings/C++

Steup@laptop ~/famouso/Bindings/C++
$ ./Subscriber.exe
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish
cb Length=7 Event data=Publish


Steup@laptop ~/famouso/Bindings/C++
$
```

**Figure 15 – Terminals in the communication example.**

The left terminal shows the ech. Here the announcement of the publisher fallowed by the subscription of the subscriber can be seen. After this on terminating the programs respective unsubscriptions as well as unannouncements are visible.

The right terminal shows the output of the subscriber while the publisher is running. Periodically events are received containing data of length 7 as well as the ASCII string "Publish".

All programs can be terminated through CTRL-C.

## 3.4 Next Release

The next release will comprise three major additional features, as described next.

### 3.4.1 Support for Sensor Event Attribute Schemes

This feature adds support for user specified sensor event attribute schemes. These schemes define a contract between publishers and subscriber on what data is communicated. A scheme is a set of defined attributes including data type, scaling, physical units and endianess. The middleware provides transparent marshaling and unmarshaling of events obeying the scheme. Therefore nodes can communicate typed events easily and transparently.

### 3.4.2 Contiki Support

The Contiki embedded OS is tailored towards small embedded microcontrollers. It allows easy hardware abstraction of basic hardware features like timers, sensors and communications facilities. The support for Contiki within the middleware allows Contiki applications to use the middleware API natively. To achieve this, the middleware is integrated into the build system of Contiki. Additionally the communication stack of Contiki called RIME is integrated into the middleware exploiting the configurable component system. Contiki supports multiple activity models like threads, protothreads and events. However, only the event model is currently integrated into the middleware support.

### 3.4.3 Directed Diffusion Routing

Through the Contiki support communication hardware allowing single-hop 802.15.4 is supported by the middleware. To extend this communication to multi-hop scenarios a directed diffusion based routing will be implemented.

This routing is distance vector based routing, where each topic has its own distance vector in each relay. Therefore the routing is content based. The route will be established through the flooding of the initial subscription message through the network. Each node forwarding the message remembers the first node it received the subscription from. On the publication of an event with the same topic the receiving nodes transmit the message to this stored node.

The currently envisioned version has not yet support for mobility. However a mobility extension is in the conception phase.

## 3.5 Known Bugs

**Double Publish using ECH-UDP-BC on Window**

On Windows using Cygwin the broadcast UDP event channel handler transmits every event twice. This seems to be a bug in the ASIO library of boost. A fix is currently not yet ready. It is recommended to switch to the multicast UDP event channel handler providing correct behaviour.

**Alignment Error on ARMv7a based targets**

On ARMv7a architecture based systems like the Pandaboard the middleware is currently not usable at all because of an alignment exception on accessing boost mutex. A fix is currently developed and will be deployed with the next release.

# 4. References

[1] D3.1 First Report on Supporting Technologies, KARYON Technical Report, 26.10.2012

[2] Jeferson L. R. Souza and José Rufino, **“**Analysing and Reducing Network Inaccessibility in IEEE 802.15.4 Wireless Communications**”.** In *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN 2013),* October 2013.

[3] WoongChul Choi and SeokMin Lee. “A Novel GTS Mechanism for Reliable Multihop Transmission in the IEEE 802.15.4 Network” In *International Journal of Distributed Sensor Networks.* vol. 2012, Article ID 796426, 10 pages, 2012. doi:10.1155/2012/796426.