European Seventh Framework Programme
FP7-288879-Collaborative Project
www.ict-calipso.eu

D6.64 Evaluation Report for the Smart Toys and Smart Parking applications

The CALIPSO Consortium

TCF, Thales Communications, France
CNRS, Centre National de la Recherche Scientifique, France
SICS, Swedish Institute of Computer Science, Sweden
UPA, University of Parma, Italy
DRZ, Disney Research Zurich, Switzerland
WOS, Worldsensing, Spain
CISCO, Cisco Systems International B.V., Netherlands

Document Information

Contract Number	288879
Deliverable Name	Evaluation Report for the Smart Toys and Smart Parking applications
Deliverable number	D6.64
Editor(s)	Lito Kriara (DRZ)
Author(s)	Lito Kriara (DRZ), Màrius Montón (WOS) Paolo Medagliani, Jérémie Leguay (TCS) Pietro Gonizzi, Simone Cirani (UPA) Andrzej Duda, Martin Heusse (CNRS) Simon Duquennoy (SICS)
Reviewer(s)	Jérémie Leguay (TCS)
Dissemination level	Public
Contractual date of delivery	09/2014
Delivery date	09/2014
Status	Final
Keywords	CALIPSO, prototypes, experimentation

This project is funded by European Community's Seventh Framework Program, area "Internet Connected Objects", under Grant No. 288879, CALIPSO Project – Connect All IP-based Smart Objects. The work reflects only the authors views; the European Community is not liable for any use that may be made of the information contained herein.

Contents

1. Overview

This deliverable describes evaluation of the CAL*IP*SO protocol stack for the integrated applications of Smart Parking and Smart Toy prototypes. For each prototype we present experimental results with metrics predefined in previous deliverables. Finally, the deliverable describes the demonstration of each prototype that will take place during the final project review.

2. Smart Parking

The code of the Smart Parking prototype is available at https://github.com/sics-iot/calipso-integrated/tree/wos-integration/examples-calipso/wos-example

2.1. Demonstrator overview

The implementation of the Smart Application has followed the design described in previous Deliverables: **D6.62 Trial and Demonstration specifications** and **D6.63 Trial and Demonstration Specifications**. We will describe the different parts of the design for the prototype.

As planned, two motes has been interconnected to build the prototype. A Parking detection sensor board by Worldsensing has been connected to a standard Tmote Sky. The parking sensor board does the sensor task and the Tmote Sky runs the CAL*IP*SO radio stack.

2.1.1. Sensor system board

The Parking detection sensor board is the main component of the FastPrk™product by Worldsensing. It consists of a digital system with a magnetic sensor and MCU able to detect the presence or not of a car above it. It also contains a sub-GHz radio for point-to-point communication purposes. The communication between the two system is done through a 2-way serial port (Figure 1).

The Parking detection sensor board been used is running the same firmware than the commercial version used by Worldsensing in its FastPrk™product. This system every time it detects a change in the state of the parking place sends a pre-defined message through the serial port to the communication system.

The TMote Sky then receives the message that must be send to the base station. This task is performed by CAL*IP*SO stack running in the TMote Sky.

CALIPSO stack

As explained in **D6.62 Trial and Demonstration specifications** the CAL*IP*SO stack runs on top con Contiki OS and provides full communication capabilities. The modules used in this demonstrator are:

- RAWMAC [7] this module enhances ContikiMAC minimizing the delay of the collection process. Each node aligns the wake-up phase to its parent in the routing tree.
- Reactive RPL enhances RPL with mechanism for on-demand route search. This mechanism works better in networks with variable link quality or event with disappearing motes.
- ORPL is an extension to RPL designed to work on heavy duty-cycled networks.
- CoAP client with resources for the Fastpark sensor data and performance metrics.

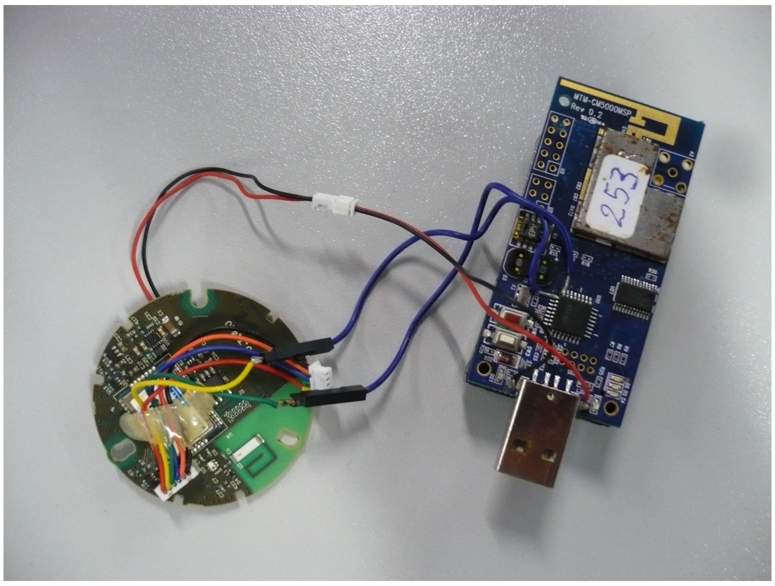


Figure 1: FastPrk™(left) and TMote Sky (right) connected. FastPrk™board periodically senses the electromagnetic field to detect the presence or absence of a car in the parking place. This information is sent to the TMote Sky (using a standard serial port) in order to be sent to the base station (and the cloud servers) by using the CAL*IP*SO communication stack.

2.1.2. Base station

To use the CAL*IP*SO stack, a base station supporting it is required. Worldsensing has provided one of its base stations to have a TMote Sky connected and usable by the project. This base station runs the base station part of the CAL*IP*SO stack, being in charge of receive the messages sent by the sensors and route them to the Worldsensing's servers in the cloud using the Internet.

The base station has two different components: a RPL Border Router and a standard embedded linux box. The RPL Border router is based on the same TMote Sky and Contiki OS with full CAL*IP*SO stack:

- Same components as before (RAWMAC, Reactive RPL, ORPL)
- Statistics module to gather and output metrics information.

The SW part of the base station is running in Java above Linux and have the following modules:

- CoAP Server: receives CoAP messages from the Park Sensor CoAP Client, extracts the payload and passes it to the Resource Manager module and to the HTTP Server.
- Resource Manager: receives CoAP payload from the CoAP Server. The Fastpark sensor data is passed to the HTTP Client. The performance metrics are generated from the CoAP messages and the data gathered from the Border Router. The data is also passed to the Database for storage
- Database (DB): stores all data received from the Data Normalizer. Data is also passed to the HTTP Server for visualization
- HTTP Client: sends HTTP request to the WOS HTTP Server containing the data received from the Data Normalizer

- HTTP Server: this local server runs on the gateway and is used for data visualization and diagnostic. The HTTP Server polls data from the Database

2.1.3. Cloud system

As the messages delivered by the base station are identical to the current Worldsensing commercial system, no changes in the cloud system has been required.

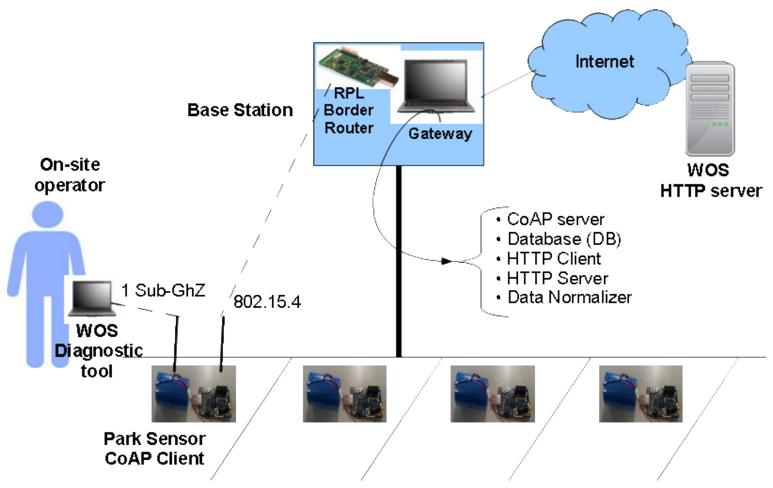


Figure 2: Block Diagram of the Smart Parking demo

2.2. Performance evaluation

The performance indicators chosen for this demo are presented in D6.62.

These parameters can be sent by the CoAP client running in the CAL*IP*SO mote to the base station CoAP server.

Sensor

- Energy consumed by the node
- RPL information (parent id, ETX metric)
- Packet delivery ratio (PDR)
- Routing protocol transmission overhead (number of non-application packets transmitted by the routing layer to maintain the routing tree)

Base station

- Routing tree convergence time
- Node link latency
- Node hop count

Experimental setup

In order to test and validate the CAL*IP*SO implementation, a scenario with 6 nodes communicating in a multihop fashion to the sink has been used, forming a tree with a maximum depth of three hops.

In Figure 3, we show the registration procedure the nodes follow before starting collecting statistics.

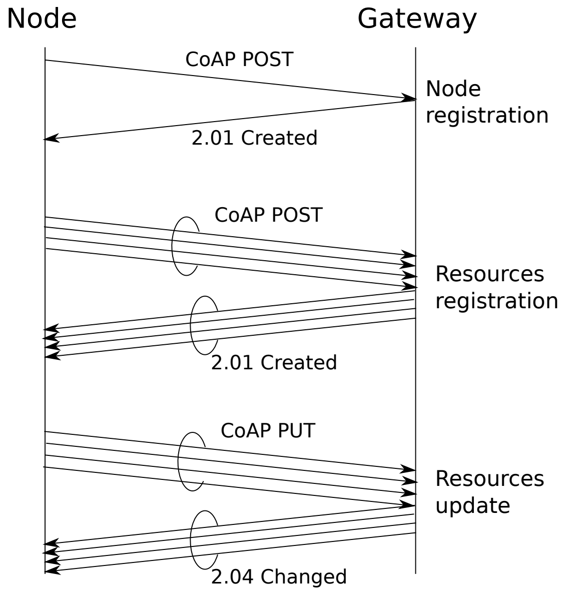


Figure 3: Communication scheme between the nodes and the gateway.

Once a node enters the network, it issues a CoAP POST to a specific resource of the gateway which, in turns, reply with a 2.01 CREATED message with an indication of the ID assigned to the node and the URI that the node must use to collect statistics. At this point, the node can start registering each resource for which it wants to collect statistics. Exploiting the URI received at the previous step, the node builds a new URI for each resource and issues a new CoAP POST to the gateway to notify it of the starting data collection. The gateway replies with a 2.01 CREATED message for each POST. Once this phase is terminated, the node can update the status of the resources via a CoAP PUT message, to which the gateway replies with a 2.04 CHANGED message. These updates are sent periodically by a node. Clearly, this frequency has an impact on energy consumption and network traffic. In order to have sufficient granularity for the tests, the data collection frequency has been set to 30 s.

The resources that a node collects are

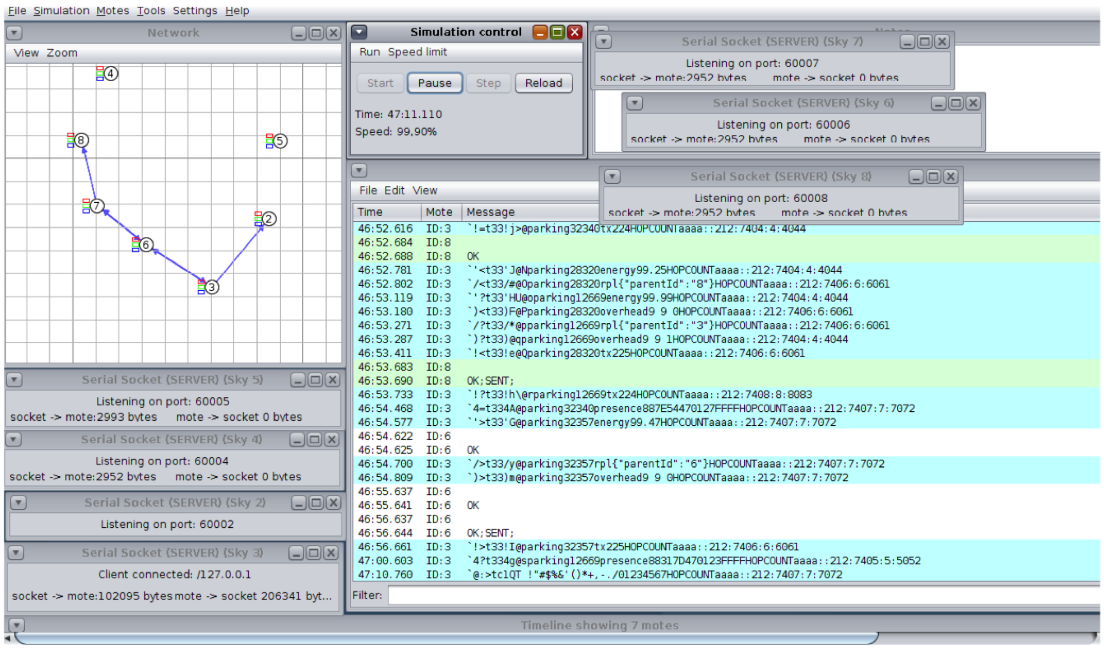
1. Presence of a node
2. Transmitted packets
3. Overhead messages
4. Consumed energy
5. Preferred parent

In addition, the border router notifies the gateway about the convergence time of each node and the number of hops traversed by a packet issued by a node.

The gateway extrapolates from this information statistics about energy consumption, the tree depth of a node, the total per-node overhead, the packet delivery ratio, and the convergence time. In addition, to measure the round-trip delay, the gateway cyclically sends ping messages to the nodes. Once the experiment is concluded, the gateway stores all the statistics in a log file for further processing.

2.2.1. Simulations

The entire system as described before has been simulated prior to the final deployment in order to validate the code and the results gathered. The simulator chosen has been Cooja as shown in Figure 4. In this simulator, the CAL*IP*SO stack runs as well all radio communications and devices. A model for the SmartPrk sensor board has been written from scratch in order to have the complete system running in the simulation.



```
Time: 47:11.110
Speed: 99,90%

Serial Socket (SERVER) (Sky 7)
Listening on port: 60007
socket -> mote:2952 bytes      mote -> socket 0 bytes

Serial Socket (SERVER) (Sky 6)
Listening on port: 60006
socket -> mote:2952 bytes      mote -> socket 0 bytes

Serial Socket (SERVER) (Sky 8)
Listening on port: 60008
socket -> mote:2952 bytes      mote -> socket 0 bytes

Serial Socket (SERVER) (Sky 5)
Listening on port: 60005
socket -> mote:2993 bytes      mote -> socket 0 bytes

Serial Socket (SERVER) (Sky 4)
Listening on port: 60004
socket -> mote:2952 bytes      mote -> socket 0 bytes

Serial Socket (SERVER) (Sky 2)
Listening on port: 60002

Serial Socket (SERVER) (Sky 3)
Client connected: /127.0.0.1
socket -> mote:102095 bytes mote -> socket 206341 byt...

Time       Mote  Message
46:52.616  ID:3  `!=t33!j>@parking32340tx224HOPCOUNTaaaa::212:7404:4:4044
46:52.684  ID:8
46:52.688  ID:8  OK
46:52.781  ID:3  `'<t33'J@Nparking28320energy99.25HOPCOUNTaaaa::212:7404:4:4044
46:52.802  ID:3  `/<t33/#@Oparking28320rpl{"parentId":"8"}HOPCOUNTaaaa::212:7406:6:6061
46:53.119  ID:3  `'?t33'HU@oparking12669energy99.99HOPCOUNTaaaa::212:7404:4:4044
46:53.180  ID:3  `)<t33)F@Pparking28320overhead9 9 0HOPCOUNTaaaa::212:7406:6:6061
46:53.271  ID:3  `/?t33/*@pparking12669rpl{"parentId":"3"}HOPCOUNTaaaa::212:7406:6:6061
46:53.287  ID:3  `)?t33)@qparking12669overhead9 9 1HOPCOUNTaaaa::212:7404:4:4044
46:53.411  ID:3  `!<t33!e@Qparking28320tx225HOPCOUNTaaaa::212:7406:6:6061
46:53.683  ID:8
46:53.690  ID:8  OK;SENT;
46:53.733  ID:3  `!?t33!h\@rparking12669tx224HOPCOUNTaaaa::212:7408:8:8083
46:54.468  ID:3  `4=t334A@parking32340presence887E54470127FFFFHOPCOUNTaaaa::212:7407:7:7072
46:54.577  ID:3  `'>t33'G@parking32357energy99.47HOPCOUNTaaaa::212:7407:7:7072
46:54.622  ID:6
46:54.625  ID:6  OK
46:54.700  ID:3  `/>t33/y@parking32357rpl{"parentId":"6"}HOPCOUNTaaaa::212:7407:7:7072
46:54.809  ID:3  `)>t33)m@parking32357overhead9 9 0HOPCOUNTaaaa::212:7407:7:7072
46:55.637  ID:6
46:55.641  ID:6  OK
46:56.637  ID:6
46:56.644  ID:6  OK;SENT;
46:56.661  ID:3  `!>t33!I@parking32357tx225HOPCOUNTaaaa::212:7406:6:6061
47:00.603  ID:3  `4?t334g@sparking12669presence88317D470123FFFFHOPCOUNTaaaa::212:7405:5:5052
47:10.760  ID:3  `@:>tc1QT !"#$%&'()*+,-./01234567HOPCOUNTaaaa::212:7407:7:7072
Filter:
```

Figure 4: Screenshot of Cooja simulator for Smart Parking evaluation.

The Cooja simulator also allows engineers to do a mixed simulation with real devices and simulated parts of the system, so the development of any of the parts of the system can be validated in HW in a quick and easy way.

For this evaluation phase, the following five scenarios has been compared:

1. RPL + NullRDC (i.e., radio always on)
2. RPL + ContikiMAC
3. RPL + RAWMAC
4. ORPL + ContikiMAC
5. RRPL + ContikiMAC

Since in ORPL, a node opportunistically sends to the first available parent, for this protocol, the concept of tree depth is meaningless. In order to provide a fair comparison, we averaged the results on all the nodes in the network.

Protocol	Delay [ms]
RPL + NullRDC	156.0
RPL + ContikiMAC	576.7
RPL + RAWMAC	612.0
ORPL + ContikiMAC	459.9
RRPL + ContikiMAC	1780.8

Table 1: Average delay [ms]

Considering the delay performance, in Table 1, the lowest delay is achieved with NullRDC since there is no duty cycling and the radio interfaces are always on to receive packets.

As with real nodes, there is no way to efficiently measure the time it takes to send a message from a node to the gateway, the delay measurements have been carried out via ping messages issued by the gateway, collecting statistics on two-ways delay. For such a reason, RAWMAC has better performance than ContikiMAC for upward traffic since it aligns wake-up phases of nodes, but this is achieved at the price of a larger delay for downward data transmission. The delay of ORPL instead is lower since a node sends to the first available node to reach the sink, without waiting for a predefined parent that relays data to the destination. Finally, RRPL is the one with largest delay since nodes need to find a route before sending data.

In Figure 5, we provide a comparison between these protocols in terms of (a) overhead, (b) energy consumption, and (c) packet delivery ratio averaging the results on all the nodes in the network.

From the overhead point of view (Figure 5a), ORPL broadcasts more packets than RPL-based protocols since it relies on DIO messages (as RPL) plus some protocol-specific ORPL broadcasts. RRPL instead is the one transmitting more control packets since it requires to create a route to transfer data. From the energy point of view (Figure 5b plotted on a logarithmic scale), having the interface always on leads to elevated energy consumption. In addition, due to the higher number of exchanged packets, ORPL and RRPL have a slightly higher energy consumption than RPL-based protocols. Finally, all the protocols have the same performance in terms of packet delivery ratio (Figure 5c) that remains close to 1.

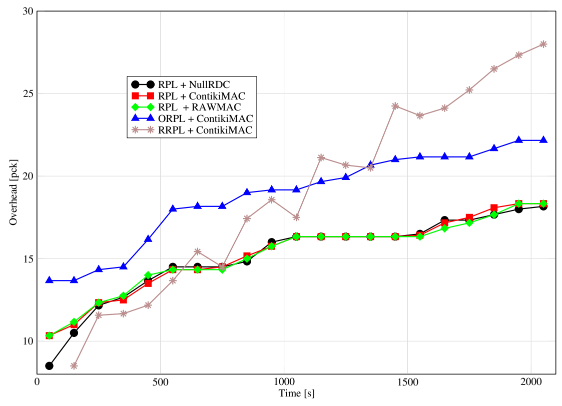
2.2.2. Field trials

The field trials has been performed as planned in the 22@ quarter in Barcelona. In this quarter, Worldsensing selected a load/unload corner zone. This zone has a high rotation of cars, truck or vans (a rotation is the change of the status of a sole parking spot i.e: it becomes free when a car outs)

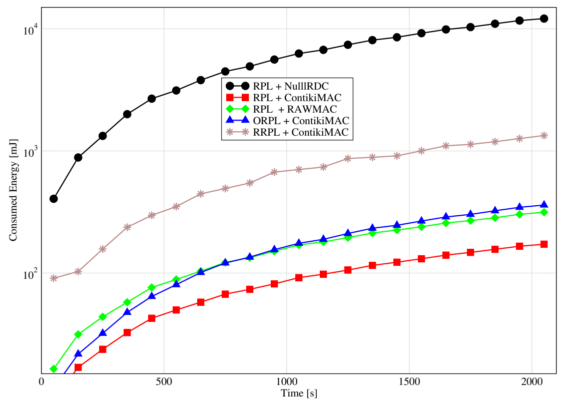
For the gathering of results in the field, 6 modified Fastprk™nodes have been installed in a load/unload parking area with around 1 meter spacing between them. The nodes are aligned in a straight line with the gateway nearby installed in a pole at 4 meters high, as shown in Figure 6. As per the specifications of the Fastprk™product, the nodes are installed in boxes below the tarmac surface, the only visible part in the tarmac being the top side. A view of this installation is provided in Figure 7.

A diagram of the aerial view is shown in Figure 8.

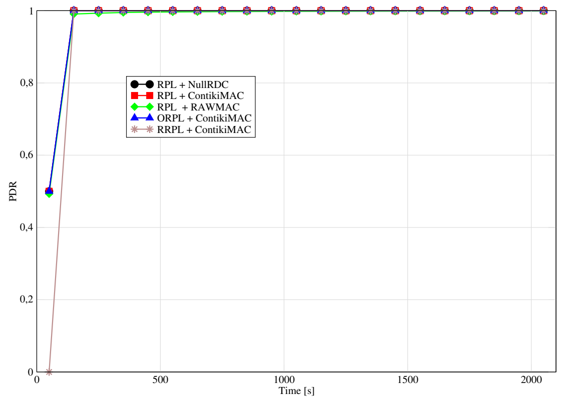
To reproduce a multihop scenario in this installation, we have tuned the receiver sensibility of the border router to -65 dBm, so that only the two nodes close to the pole can communicate directly with it. The other nodes, instead, could only reach the sink via a multi-hop path. The target setup is also shown in the Figure 8 where the arrows represent the ideally preferred next



(a) Overhead



(b) Energy consumption



(c) Packet delivery ratio

Figure 5: Performance comparison of the five protocols in terms of (a) overhead, (b) energy consumption, and (c) packet delivery ratio averaged on the nodes in the network.

hop device for the given node. It should be noted here that this in no way means the other nodes are not reachable by the given device, in fact nearby nodes to the preferred parent are reachable and the nodes may use them instead of the preferred parent for communication. The circumstances in which this would happen are down to changes in the environment, such as cars parking on the monitored parking spots, or even cars passing or stationed nearby, interferences from a nearby public WiFi hotspot, etc.

Since tests have been performed in a public street, the interference of ongoing traffic, radio interference from personal devices, and weather conditions are common problems and cannot be controlled in any way. For such a reason, unexpected behaviors may arrive, such as temporal

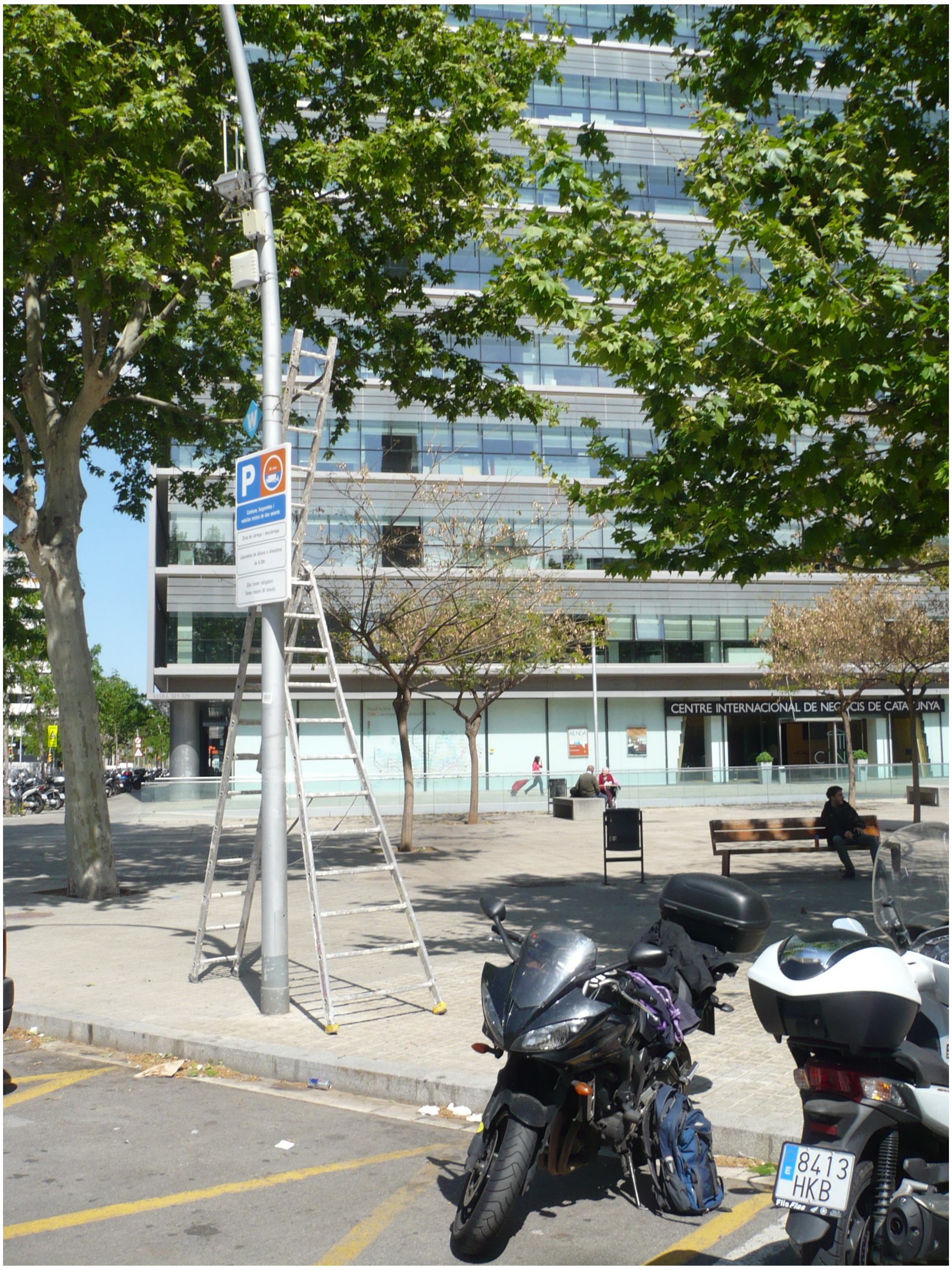


Figure 6: Picture of the base station with the border router (inside the white box) in the light pole.

weaker connectivity, jitters, extra overhead, or extra energy consumption.

Since nodes run in a real environment, we have carried out a performance comparison between the protocols developed in CAL*IP*SO and the standard protocols present in Contiki in terms of delay, overhead, energy consumption, and packet delivery ratio. The evaluated protocols are as in Subsection 2.2.1 (i) RPL+NullRDC, (ii) RPL+ContikiMAC, (iii)

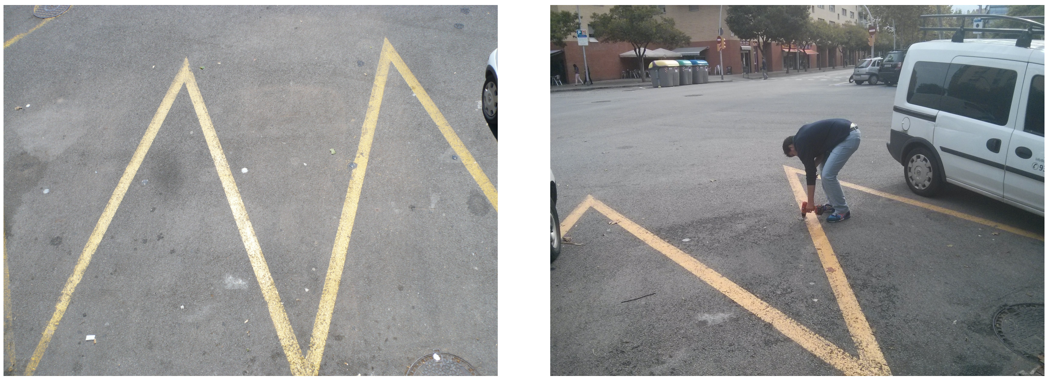

(a) Sensor in tarmac (b) Operation on sensors

Figure 7: Picture of the sensors embedded in the tarmac.

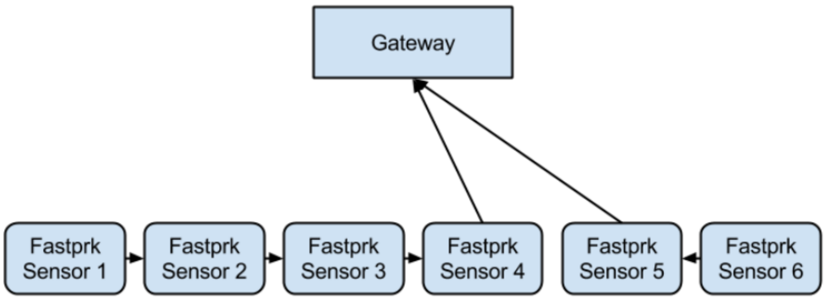
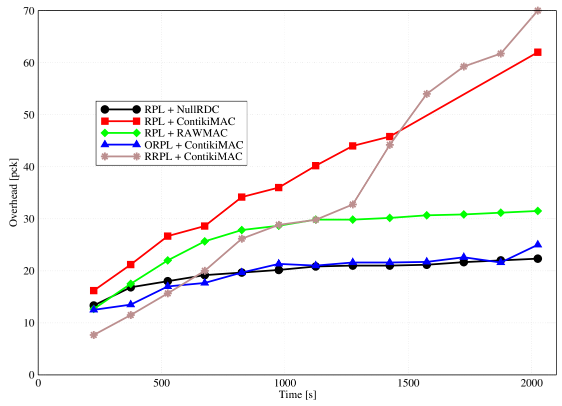


Figure 8: Schematic aerial view of the deployment.

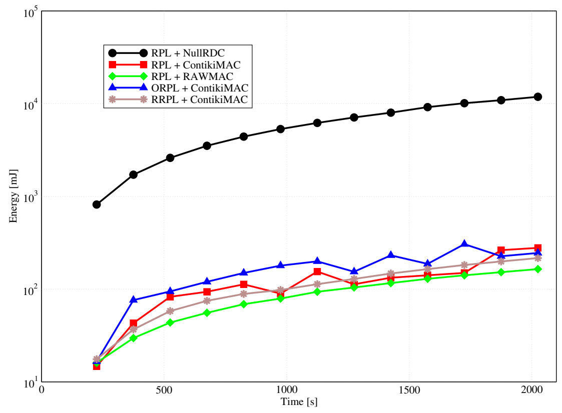
RPL+RAWMAC, (iv) ORPL+ContikiMAC, and (v) RRPL + ContikiMAC. In order to stress the network behavior especially during the startup phase, we present results over the first 40 minutes of the experience, since at that time network has already converged and its behavior is stable.

In Table 2, we compare the delay performance of the protocols. For each protocol, we show the average delay and the standard deviation. As for Table 1, we measured the round-trip delay, since it is not possible to have clock synchronization between nodes and know the exact packet transmission instant. The best performance is obtained with NullRDC, since radio interfaces are always on. RAWMAC, basic ContikiMAC, and ORPL, instead have approximately the same delay performance. In fact, the benefit of phase alignment in RAWMAC is partially lost since ping messages accumulate the whole round-trip delay to reach the node and return to the sink. In ORPL, instead, the low density of nodes bounds the effects of opportunistic transmissions. The difference between these protocols are mainly due to the number of cars that arrived or left during the experience. In fact, each car arrival perturbs the link conditions and requires some additional time to re-adapt the routing tree. Finally, RRPL due to its reactive nature experiences the highest delay.

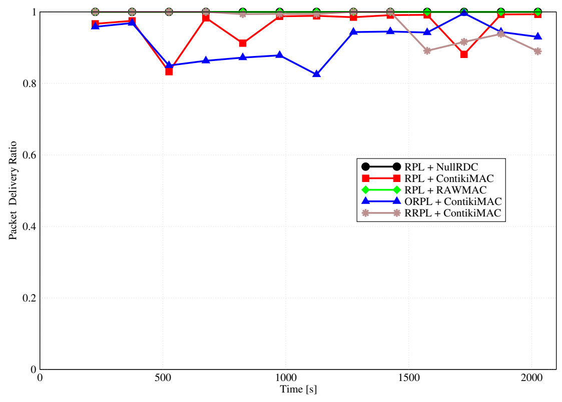
We want to point out that some experiences, such as RRPL + ContikiMAC and RPL + ContikiMAC, have been strongly affected by the presence and the rotation of cars. These three protocols, in fact, have been validated during a normal working day with several cars and vans parking. We noticed that some delay peaks correspond to the arrival of shipping-service vans



(a) Overhead



(b) Energy consumption



(c) Packet delivery ratio

Figure 9: Performance comparison through on-the-field experiments of the five protocols in terms of (a) overhead, (b) energy consumption, and (c) packet delivery ratio averaged on the nodes in the network.

that strongly interfered with the transmission of packets. On the contrary, RPL + NullRDC and RPL + RAWMAC have been carried out during a local holiday, so the car rotation was considerably smaller and the experimental conditions were quite stable. Finally, the experience with ORPL + ContikiMAC has been carried out during the local holiday as well. Together with the fact that during the experience some cars parked above the nodes closer to the gateway, the higher standard deviation can be explained by considering the opportunistic nature of the protocol.

Protocol	Delay [ms]	Standard deviation [ms]
RPL + NullRDC	99.3	66.7
RPL + ContikiMAC	441.4	286.0
RPL + RAWMAC	321.3	141.8
ORPL + ContikiMAC	542.3	886.2
RRPL + ContikiMAC	840.3	1025.1

Table 2: Field evaluation of average delay [ms] and standard deviation [ms].

In Figure 9, instead, we compared the five protocols in terms of overhead, energy consumption, and packet delivery ratio. Referring to Figure 9a, the lowest overhead among the protocols running RPL is achieved with NullRDC since nodes are always on and the effects of duty cycles on the RPL metric is absent. Similar overhead performance is achieved with ORPL where only a few broadcast messages are transmitted to diffuse information about each node phase. As shown in Figure 9b, the fact that ORPL uses many broadcast messages leads to similar energy consumption compared to RPL+ContikiMAC and RPL+RAWMAC, where both unicast and broadcast messages are sent, thus balancing the lower overhead. RRPL instead, pays for route creation. Finally, RPL+NullRDC has an energy consumption ten times higher since nodes have interfaces always on. Concerning the packet delivery ratio, as shown in Figure 9c, even if we experienced some losses for ORPL + ContikiMAC, RRPL + ContikiMAC, and RPL + ContikiMAC due to the above-mentioned experience conditions, all the solutions have a very interesting packet delivery ratio performance that makes the protocols developed in CAL*IP*SO suitable for the use in realistic applications.

Simulation and experimental results allow us to draw some conclusions about the protocols we considered and their applicability in smart parking applications.

RPL + NullRDC: This protocol has very interesting performance in terms of delay and pdr since nodes are always on to receive incoming messages. However, the extremely-elevated energy consumption makes it definitively unsuitable for smart parking applications that target important node lifetimes.

RPL + ContikiMAC: This solution shows interesting performance for smart parking applications. However, it is not adapted for converging traffic, since it has been conceived for a generic traffic pattern, without privileging a specific traffic direction.

RPL + RAWMAC: RAWMAC specifically targets upwards traffic since it aligns the wake-up phase of a node with that of its preferred parent. Performance shows that this protocol is quite adapted to smart parking applications since it has reasonable energy consumption, small delay, and high packet delivery ratio.

ORPL + ContikiMAC: ORPL is another potentially good candidate for smart parking applications as it conceived to guarantee small delay and it proved to have a reduced overhead and very good energy consumption. However, ORPL performs best when a high density of nodes is considered. In our testbed we only have 6 nodes communicating, so the benefits of ORPL are not completely evident.

RRPL + ContikiMAC: Since RRPL is a reactive protocol, its higher signalization is not adapted for pure data collection in smart parking applications. However, this protocol can be exploited for any other application is smart parking where rapid topology reconfiguration are required for instance in the presence of mobility.

3. Smart Toys

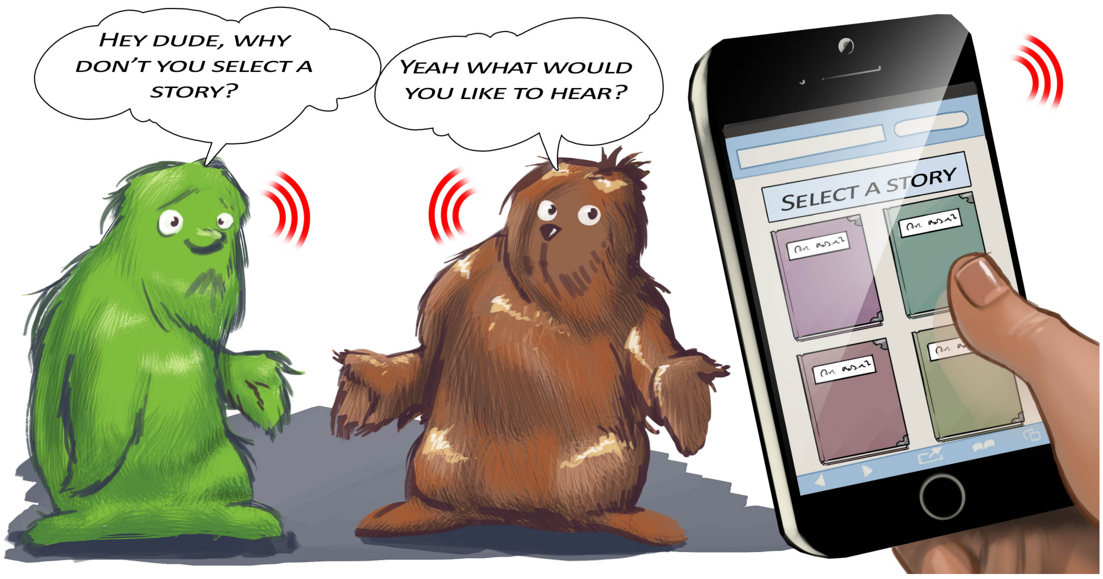

HEY DUDE, WHY DON'T YOU SELECT A STORY?
YEAH WHAT WOULD YOU LIKE TO HEAR?
SELECT A STORY

Figure 10: Smart Toy concept art to illustrate a target play pattern (exaggeration). The IoT connectivity enables seamless connectivity between toys and wireless devices such as smartphones. It is expected that –thanks to the new software modules of the CAL*IP*SO project– the path towards commercial exploitation and new revenue streams will be facilitated.

The Smart Toy prototype described in this section is a platform for the pet- or companion-doll category of toys. The Smart Toy concept is depicted in Figure 10. In the following, we describe the implementation, evaluation, and demonstration overview of the Smart Toy prototype. The implementation report is the updated version of Section 3 of Deliverable D6.62. Certain specifications have changed, like for example parts of the hardware used. We have removed the LCD screen from the Smart Toy prototype, because of its high power consumption and unreasonable CPU load on the microntroller unit. After extensive trials and measurements, we realized that the constant updating of the images on the display disrupts the wireless connectivity of the toys, because of the low RAM and memory capabilities of the Arduino Due board used.

3.1. Prototypes

The hardware of the Smart Toy prototype is based on the Arduino Due board. All the components are shown in Figure 11. The Arduino Due is a microcontroller board based on the 32-bit Atmel SAM3X8E ARM Cortex-M3 CPU. The board provides 96 KB of SRAM and 512 KB of ash memory. An Atheros AR9170 Wi-Fi interface, which is connected to the board via USB, provides Wi-Fi connectivity to the toy prototype, as Figure 12 also shows. The prototype also includes an MP3 board with an SD card slot and a speaker, which are used to play audio. The prototype is powered from a 5 V 5000 mAh LiPo battery that outputs up to 1.5 A, sufficient to support all hardware components. The hardware is placed into a 3D printed protection enclosure, which in turn is placed inside a plush toy body. The MP3 board and speaker enable the toy to speak, sing, and express emotions through sounds. The toys have a humanoid or animalistic appearance that resembles some fantasy creatures.

There is also a gateway which is an Internet-connected laptop that runs a border-router

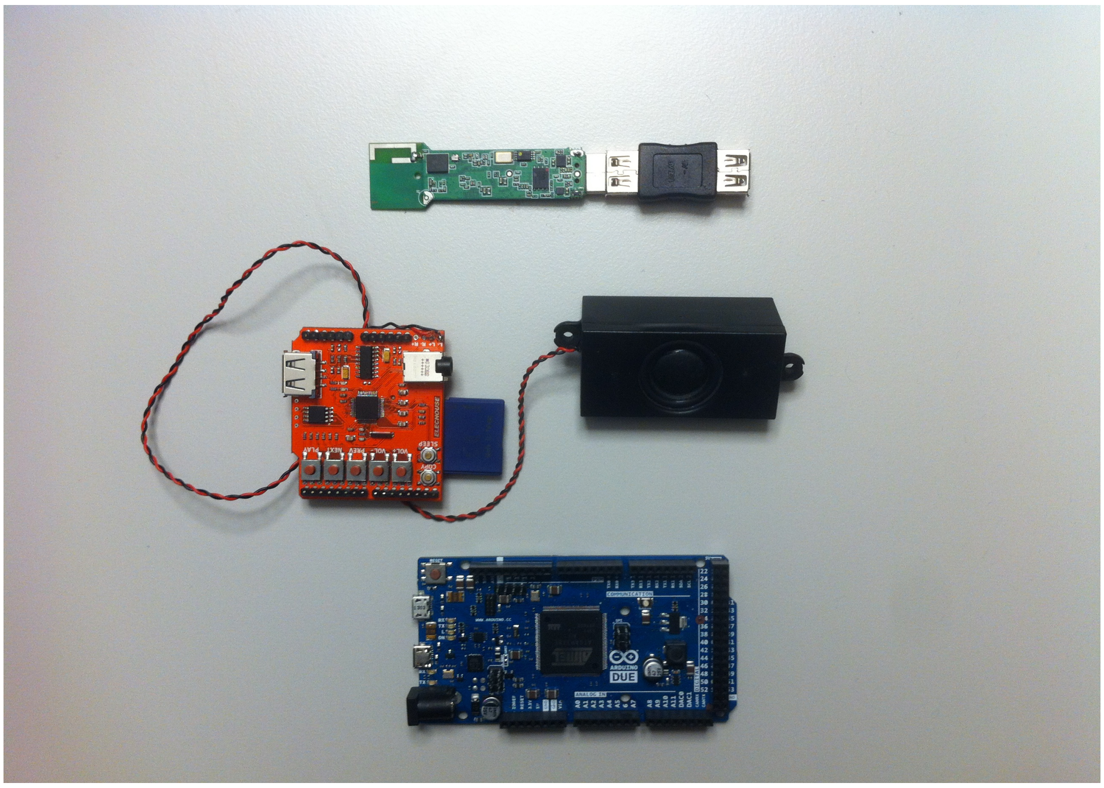


Figure 11: Hardware components of the Smart Toy prototype

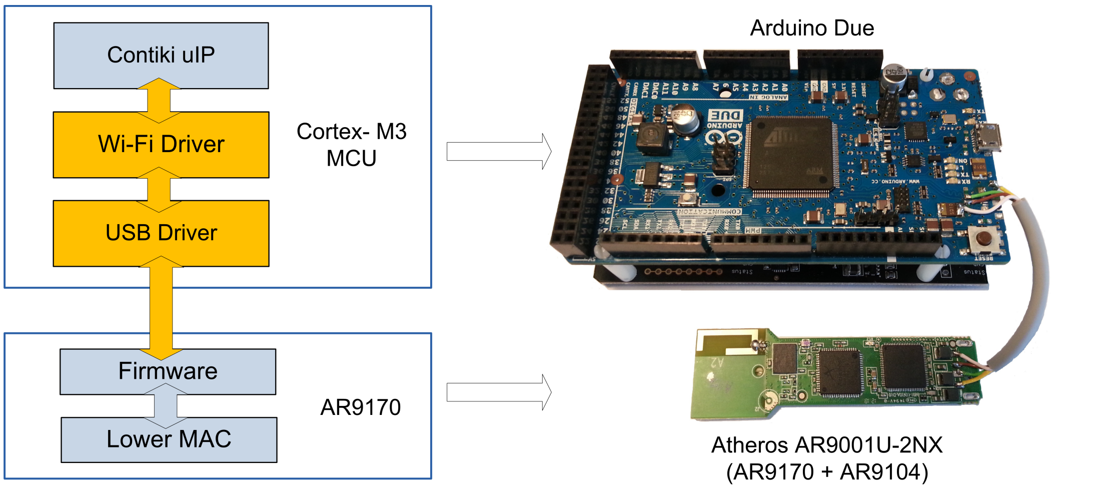


Figure 12: HW/SW platform for MH-PSM evaluation consists of an Arduino Due board and an Atheros 802.11n transceiver. We ported Contiki OS to the Arduino Due and implemented WLAN and USB drivers for the Atheros transceiver.

and other components of the CAL*IP*SO protocol stack. The toys and the gateway create an 802.11 IBSS (ad hoc) network. We have implemented an application that enables toys to speak together in coordination to demonstrate the functionality of the CAL*IP*SO protocol stack in such a Smart Toy scenario. The coordination requires signaling between toys. The gateway runs a web service allowing certain features of the application to be controlled using a web browser on a phone. The web application creates signaling traffic between the phone and toy(s), which traverses the gateway. Figure 13 shows a simple setup of the whole prototype and the final demonstration setup.

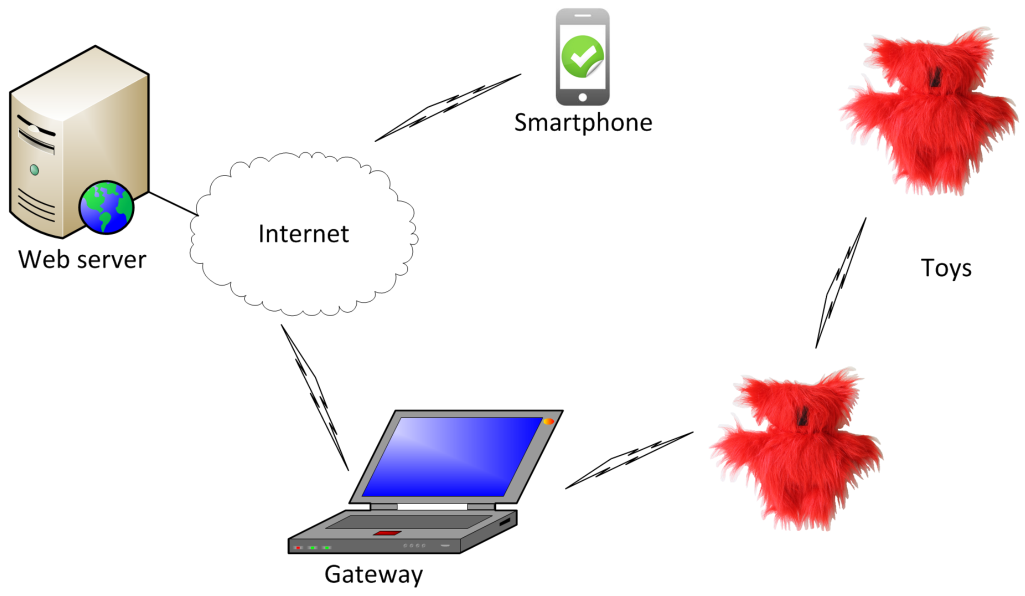


Figure 13: Smart Toy demonstration setup.

The CAL*IP*SO software components run on the Smart Toys, on the gateway, and on smartphones. For a more complete description of the CAL*IP*SO protocol stack for Smart Toys, refer to Section 3.3. of Deliverable D6.61. The Smart Toy software runs on Contiki OS, which we ported to Arduino Due boards. The toy software integrates the following CAL*IP*SO modules as also described in Deliverable D6.62 for experimental evaluation purposes:

- 802.11 with multi-hop power save mode (MH-PSM) [6, 10] provides lower latency and lower hop ad hoc networks of toys compared to the standard 802.11. 802.11/Wi-Fi is a preferred choice over 802.15.4/Zigbee because Smart Toys need to interact with consumer electronics. Also, Wi-Fi access points, which are common in many homes, and public hotspots provide almost ubiquitous Internet connectivity to the toys.
- Reactive RPL [9] enhances RPL with mechanisms for on-demand (reactive) route search. This makes it better suited for mobile ad hoc networks of toys and point-to-multipoint communication in toy networks. The reactive RPL protocol also implements link reversal mechanism for fast repair of link failures, which are common in case of node mobility.
- Featurecast [8] uses a data-centric approach to communicate the same information to a group of toys that all have some specific features (e.g. support certain service that is needed for the play experience, such as audio playout). Featurecast selects relevant recipients by collecting the IPv6 addresses of toys that posses the set of required features.
- Distributed key certification and optimized Datagram Transport Layer Security (DTLS) provide security. Distributed key certification eliminates the need for certificates and certification authorities and, therefore, streamlines the process of public key verification, which is important for toys. DTLS provides data confidentiality and integrity on top of UDP, and includes a key exchange mechanism for dynamic session establishment [5].

The overall integrated software architecture of the Smart Toy prototype is presented in Figure 14.

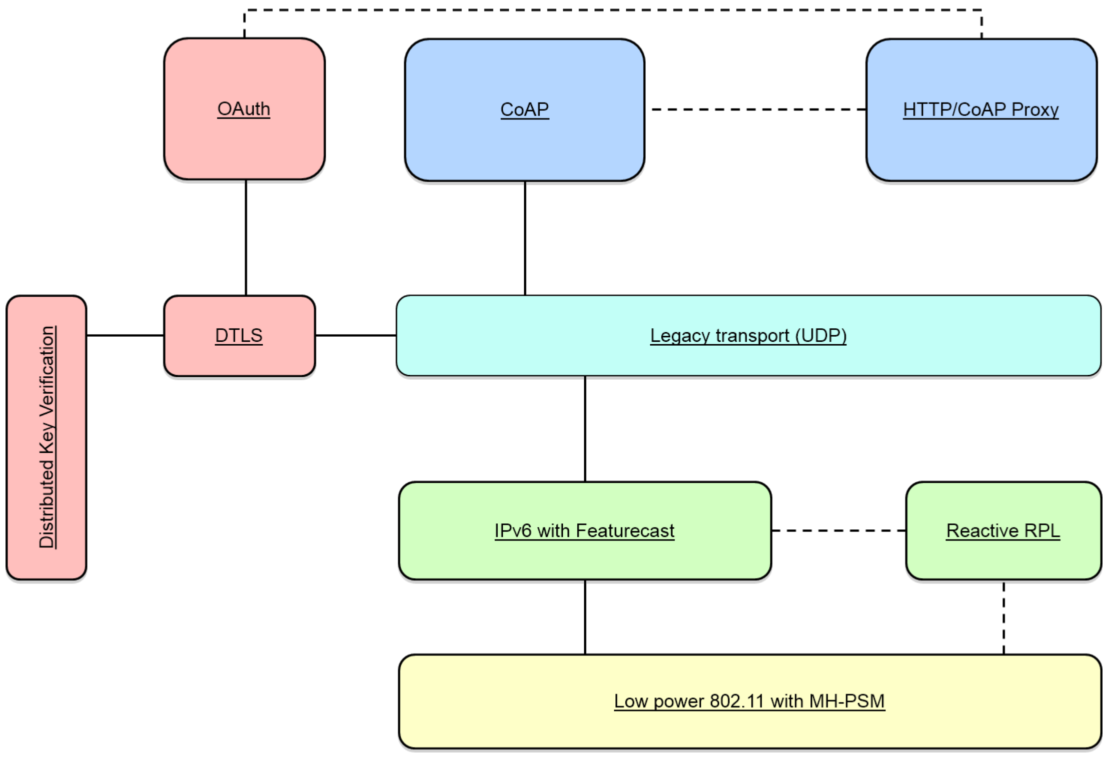


Figure 14: Software architecture of the Smart Toy prototype.

3.2. Evaluation

3.2.1. Initial Experimentation

This subsection includes results also published in [10].

We implemented the MH-PSM as a part of our WLAN driver module for Contiki [2], an open source operating system for the Internet of Things. The used hardware platform consists of an Arduino Due board (Cortex-M3 MCU, 96 KB of SRAM) connected via USB interface to an 802.11n transceiver based on Atheros AR9001U-2NX chipset [1], as shown in Fig. 12. The AR9001U-2NX chipset contains an AR9170 MAC/baseband and an AR9104 (dual-band 2×2) radio chip. Atheros has released the firmware of AR9170 as open source, which enables us to write the Contiki WLAN driver. The open source firmware provides a direct access to the lower-MAC program that runs on the AR9170 chip, which greatly simplifies driver debugging. The used Contiki driver is partially based on the otus driver [3], a depreciated Linux driver for Atheros devices (replaced by the carl9170 driver as of kernel version 2.6). It is fully integrated with the Contiki's uIP protocol stack for TCP/IP and supports standard PSM and MH-PSM in ad hoc mode. Our goal was not only to validate MH-PSM, but to build a flexible open-source platform for experimentation with 802.11 MAC Layer Management Entity (MLME) algorithms (power saving, beaconing, time synchronization, scanning, association, authentication) for future IoT-ready WLAN-enabled embedded devices.[1] While there is abundance of low-power WLAN modules for embedded devices (e.g., Roving Networks RN-131C, Gainspan GS2100M, Texas Instruments CC3000, Broadcom BCM4390, etc.) and WLAN enabled development boards for IoT applications (WiSmart EC32Lxx, RTX41xx, Spark Core, Flyport WLAN, etc.) they are not suitable for experimentation with 802.11 MAC layer management algorithms: Typically, their proprietary network stack implementations are provided as binary

[1] Lower-MAC functions with strict timing constraints, such as DCF, are implemented on the AR9170 chip.

firmwares and only high-level communication and configuration APIs are disclosed. Moreover, their stack implementations often do not support IBSS mode and/or IPv6.

Our experimental setup consists of up to seven WLAN nodes lined up 10 m apart from each other in a quiet alley at the back of an office building compound, as shown in Fig. 15. The experiments were run overnight, when the interference from the neighboring access points was negligible: WLAN spectrum monitors picked up only control and management traffic from other networks at those hours. The transmit power of the nodes was reduced to the minimum of 0 dBm, which resulted in the transmission range of roughly 30 m. Therefore, nodes that were up to three hops away could still observe and decode each others transmissions.

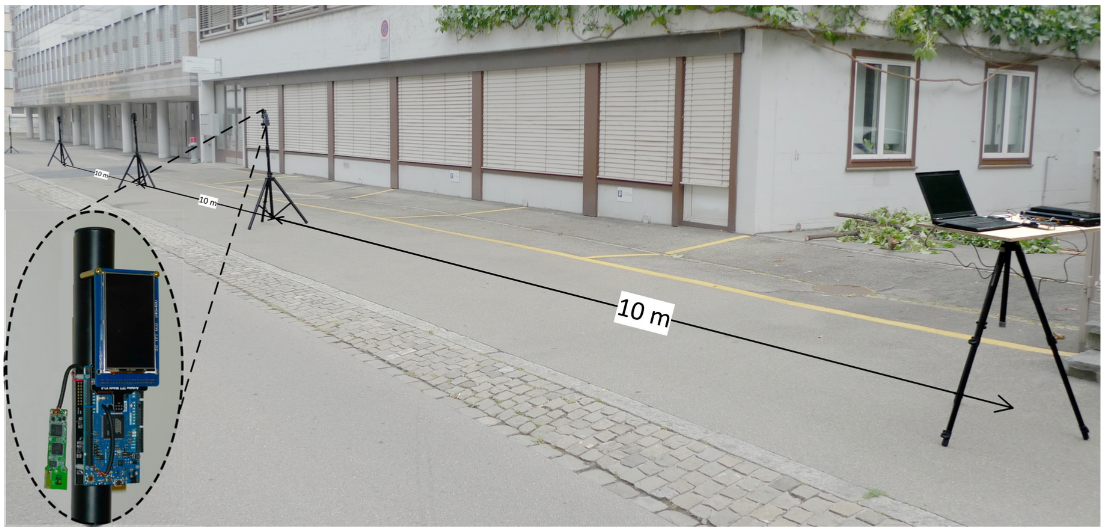


Figure 15: Experimental setup: Seven WLAN nodes lined up 10 m apart from each other. The experiments were run overnight to minimize the interference from surrounding access points. An LCD screen is connected to each node to monitor the status.

We first consider scenarios with two, three, and six hops that are created by placing, respectively, three, four, and seven nodes in a line. The beacon interval and unicast Announcement Traffic Indication Messages (ATIM) window size were set to 200 ms and 20 ms. We used the same Poisson traffic generator with fixed packet sizes as in the simulations. The sender (rightmost node in Fig. 15) generated on average one frame every 200 ms ($\lambda = 5$ frames per second). For each scenario we performed multiple (typically three to five) runs with 300 packets each. The results were then averaged over those runs.

The average end-to-end delay, doze time ratio, and ATIM overhead are shown in Fig. 16. The results follow the pattern seen in the simulation [6]. For the standard PSM it takes almost N beacon intervals to forward a frame over N hops, while for the MH-PSM the delay is much shorter. The doze time ratio is still significantly higher with MH-PSM than with standard PSM. The absolute values are higher than in the simulations presented in [6] because nodes have more neighbors (e.g., node A is able to hear not only B, but also C). This reduces the number of beacon intervals in which a node wins the beacon transmission and, therefore, the number of beacon intervals in which the node must stay awake. Hence, the doze time ratio is higher in the experiments. For the same reason, the gain of Sleep On Beacon Transmission (SoBT) is lower, but it is still significant. The SoBT overheads for 2, 3, and 6 hops are, respectively, 0.14, 0.15, and 0.15 intra-beacons per beacon interval. The ATIM overhead of MH-PSM is comparable to that of standard PSM in the two-hop and almost 30% lower in the six-hop case, as predicted by the simulations.

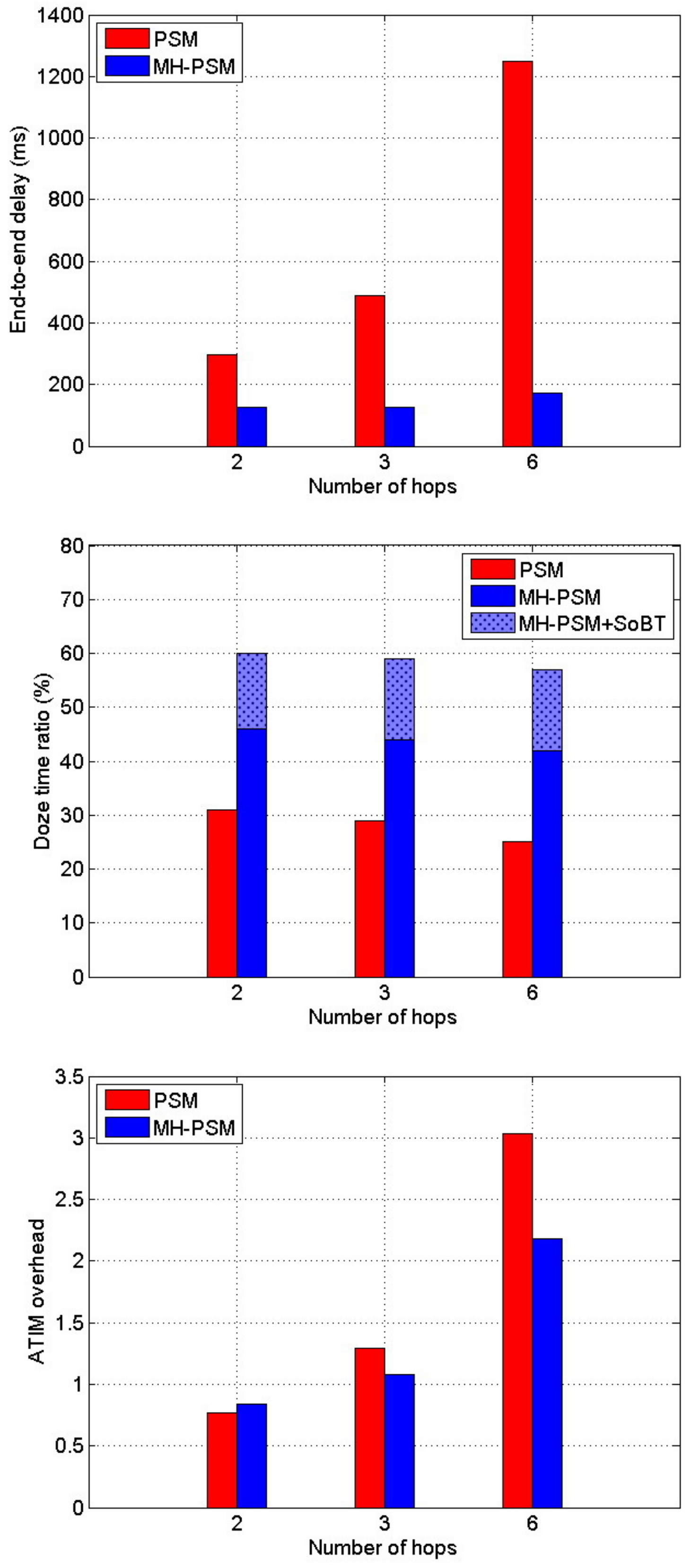


Figure 16: End-to-end delay, doze time ratio, and ATIM overhead for different number of hops.

BI (ms)	Delay (ms)		Doze time (%)		ATIM overhead	
	PSM	MH-PSM	PSM	MH-PSM	PSM	MH-PSM
100	586	125	40	52+17	4.11	2.78
200	1249	173	25	42+15	3.03	2.18
400	2234	281	19	41+15	2.12	1.23

Table 3: Performance of standard PSM and MH-PSM for different beacon intervals. Frame interarrival time is 200 ms. Frames are forwarded over six hops.

IAT (ms)	Delay (ms)		Doze time (%)		ATIM overhead	
	PSM	MH-PSM	PSM	MH-PSM	PSM	MH-PSM
100	1226	166	17	37+14	1.89	1.32
200	1249	173	25	42+15	3.03	2.18
400	1285	185	40	52+17	4.11	3.03

Table 4: Performance of standard PSM and MH-PSM for different frame interarrival time. Beacon interval is 200 ms. Frames are forwarded over six hops.

We also measured the performance of PSM and MH-PSM for different beacon intervals. The results presented so far assume the beacon interval of 200 ms. We changed the interval to 100 ms and 400 ms and repeated the measurements for the six-hop scenario. The average frame interarrival time is 200 ms regardless of the beacon interval. The results are summarized in Table 3. All our observations based on the simulation results (Table 3 apply to measurement results too: While the frame delay increases linearly with the beacon interval for the standard PSM, it increases moderately and remains comparably short for the MH-PSM. Even without SoBT, MH-PSM outperforms standard PSM in terms of the doze time ratio. As expected, the doze time decreases with the length of beacon intervals because fewer intervals are idle. The improvement in terms of ATIM overhead is also significant.

Finally, we varied the average frame Interarrival Time (IAT) $1/\lambda$ to evaluate the impact of traffic load in the six-hop scenario. The beacon interval is 200 ms regardless of the IAT. The results are summarized in Table 4. With MH-PSM, close to 80% of frames are transmitted end-to-end within a single beacon interval regardless of the traffic load (0% with standard PSM). The slight increase in the delay for longer IATs is due to the initial hold-up at the sender, as discussed in [6]. As expected, the doze time increases when the traffic load decreases (i.e., for longer IATs). We see that MH-PSM significantly outperforms standard PSM in all scenarios considered.

3.2.2. Initial Integrated Example

Once the integration of MH-PSM, RRPL and Featurecast is done we run an initial example with 3 nodes that can be up to two hops away. In our experiment we use the topology shown in Figure 17. The terminal view of each of those nodes is shown in Figure 18. We can see the Featurecast and the RRPL information available on each node. For Featurecast we see that the address of the next available node is shown, as well as the overhead imposed. For example, in Figure 18b we see that the intermediate node has sent 10 Featurecast advertisements and has received 3, making it a total packet overhead of 13 packets. We also see that the RRPL packet overhead is 70 control packets at that point, including the number of OPT, RREQ, RREP and other RRPL control packets described in detail in [9]. In the sink/gateway side (Figure 18a) we can see that the correct path to both nodes is described correctly by RRPL. "HC 0" and

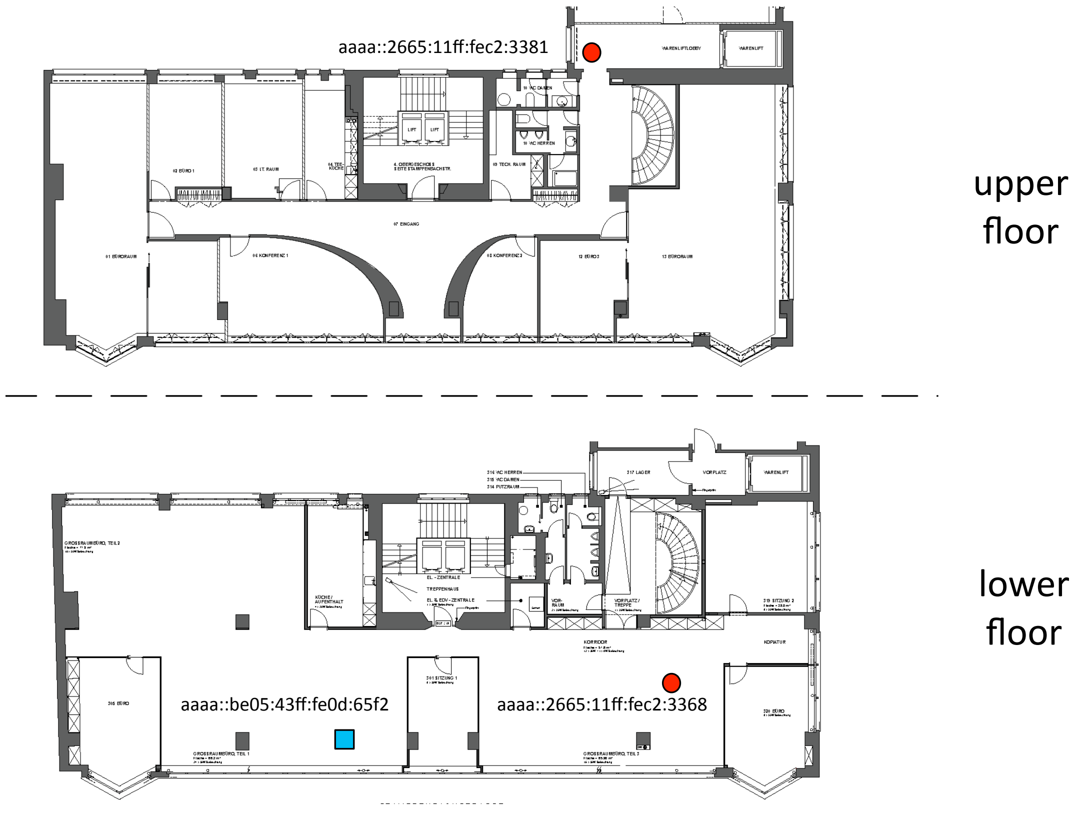

aaaa::2665:11ff:fec2:3381
upper floor
lower floor
aaaa::be05:43ff:fe0d:65f2
aaaa::2665:11ff:fec2:3368

Figure 17: Initial testbed setup for integration validation.

"HC 1" indicate the number of hops. We see that indeed node with "aaaa::2665:11ff:fec2:3368" is directly connected to the sink and therefore "HC 0" is assigned. On the other hand, for "aaaa::2665:11ff:fec2:3381" we see that the sink knows that the next node to reach in order to eventually reach it is "aaaa::2665:11ff:fec2:3368", and therefore "HC 1" is assigned it the route, since one hop is required.

3.2.3. Final Integrated Evaluation

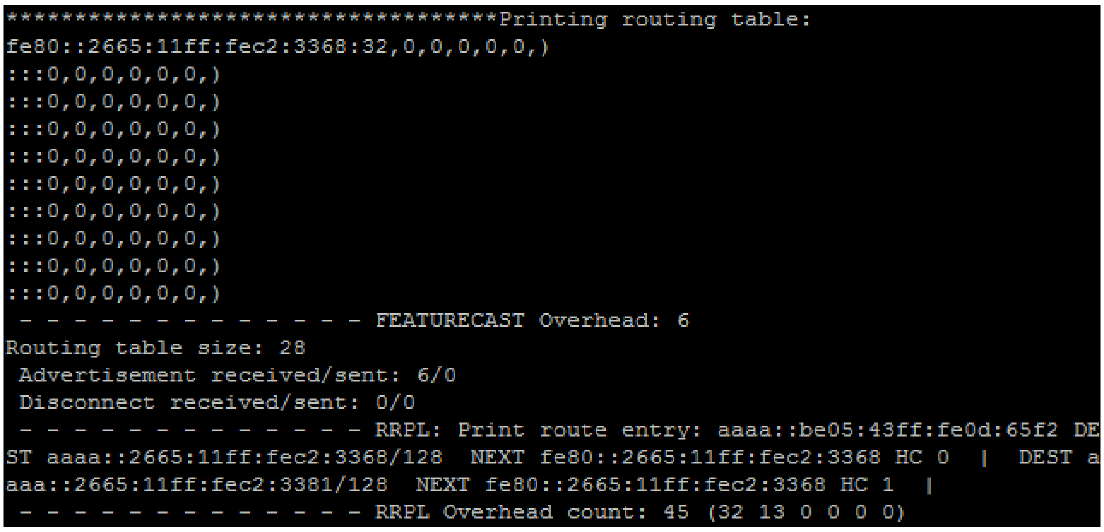
Following the experimental design proposed in Deliverable D6.62, the performance of the following modules will be measured during the experiments: 802.11 with MH-PSM, RRPL, and Featurecast.

The Smart Toy testbed includes 10 toy prototypes deployed at the DRZ's office space in Zurich, as shown in Figure 19. The gateway node is placed at fixed location. The rest are placed into portable enclosures and can be carried around. In the scenario examined (as shown in Figure 19) the maximum depth of the topology tree (maximum route length) is 5 hops.

RPL	**RPL & Featurecast**	**RRPL**	**RRPL & Featurecast**
736 bytes	1800 bytes	2816 bytes	3864 bytes

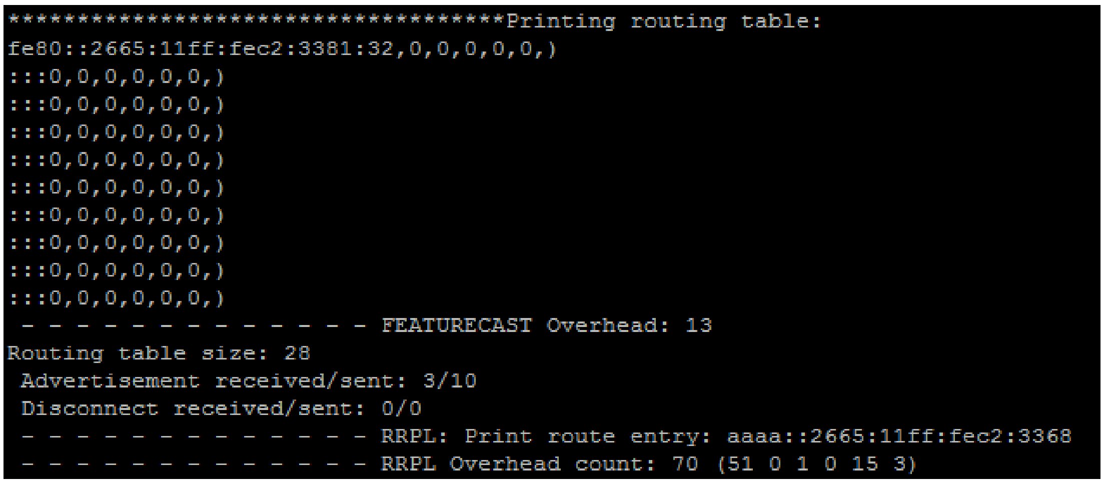
Table 5: Code footprint of RPL, RRPL and Featurecast.

The traffic scenario we employ is similar to that of a storytelling application (as described further in section 3.3.2), where we assume that packets need to be sent every 1 second. Therefore, we expect the numerical results to be different than what reported in section 3.2.1, where we show an extremely loaded network scenario.



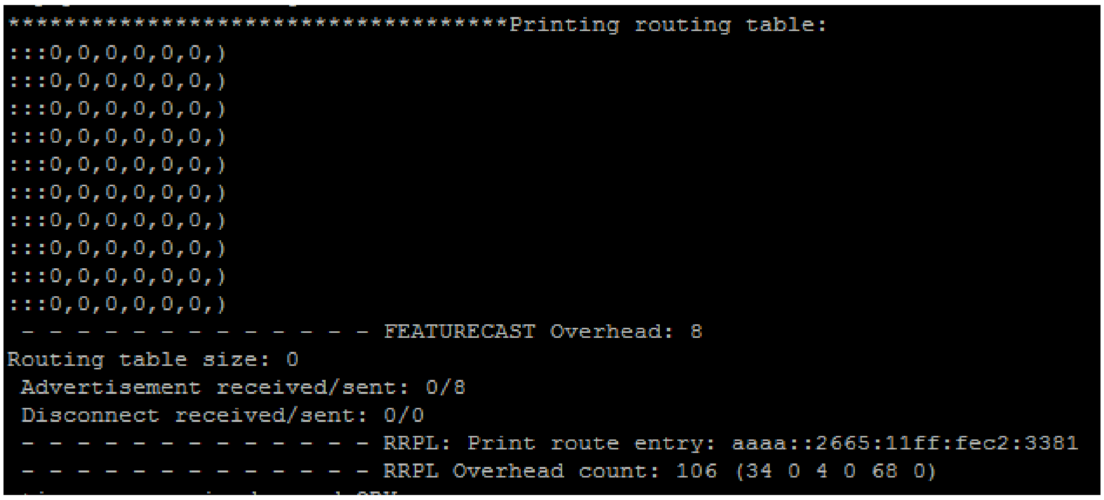
```
************************************Printing routing table:
fe80::2665:11ff:fec2:3368:32,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
 - - - - - - - - - - - - - - FEATURECAST Overhead: 6
Routing table size: 28
 Advertisement received/sent: 6/0
 Disconnect received/sent: 0/0
 - - - - - - - - - - - - - - RRPL: Print route entry: aaaa::be05:43ff:fe0d:65f2 DE
ST aaaa::2665:11ff:fec2:3368/128  NEXT fe80::2665:11ff:fec2:3368 HC 0  |  DEST a
aaa::2665:11ff:fec2:3381/128  NEXT fe80::2665:11ff:fec2:3368 HC 1  |
 - - - - - - - - - - - - - - RRPL Overhead count: 45 (32 13 0 0 0 0)
```

(a) Sink - aaaa::be05:43ff:fe0d:65f2



```
************************************Printing routing table:
fe80::2665:11ff:fec2:3381:32,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
 - - - - - - - - - - - - - - FEATURECAST Overhead: 13
Routing table size: 28
 Advertisement received/sent: 3/10
 Disconnect received/sent: 0/0
 - - - - - - - - - - - - - - RRPL: Print route entry: aaaa::2665:11ff:fec2:3368
 - - - - - - - - - - - - - - RRPL Overhead count: 70 (51 0 1 0 15 3)
```

(b) Node in 0 hops - aaaa::2665:11ff:fec2:3368



```
************************************Printing routing table:
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
:::0,0,0,0,0,0,)
 - - - - - - - - - - - - - - FEATURECAST Overhead: 8
Routing table size: 0
 Advertisement received/sent: 0/8
 Disconnect received/sent: 0/0
 - - - - - - - - - - - - - - RRPL: Print route entry: aaaa::2665:11ff:fec2:3381
 - - - - - - - - - - - - - - RRPL Overhead count: 106 (34 0 4 0 68 0)
```

(c) Node in 1 hops - aaaa::2665:11ff:fec2:3381

Figure 18: Integrated results in terminal.

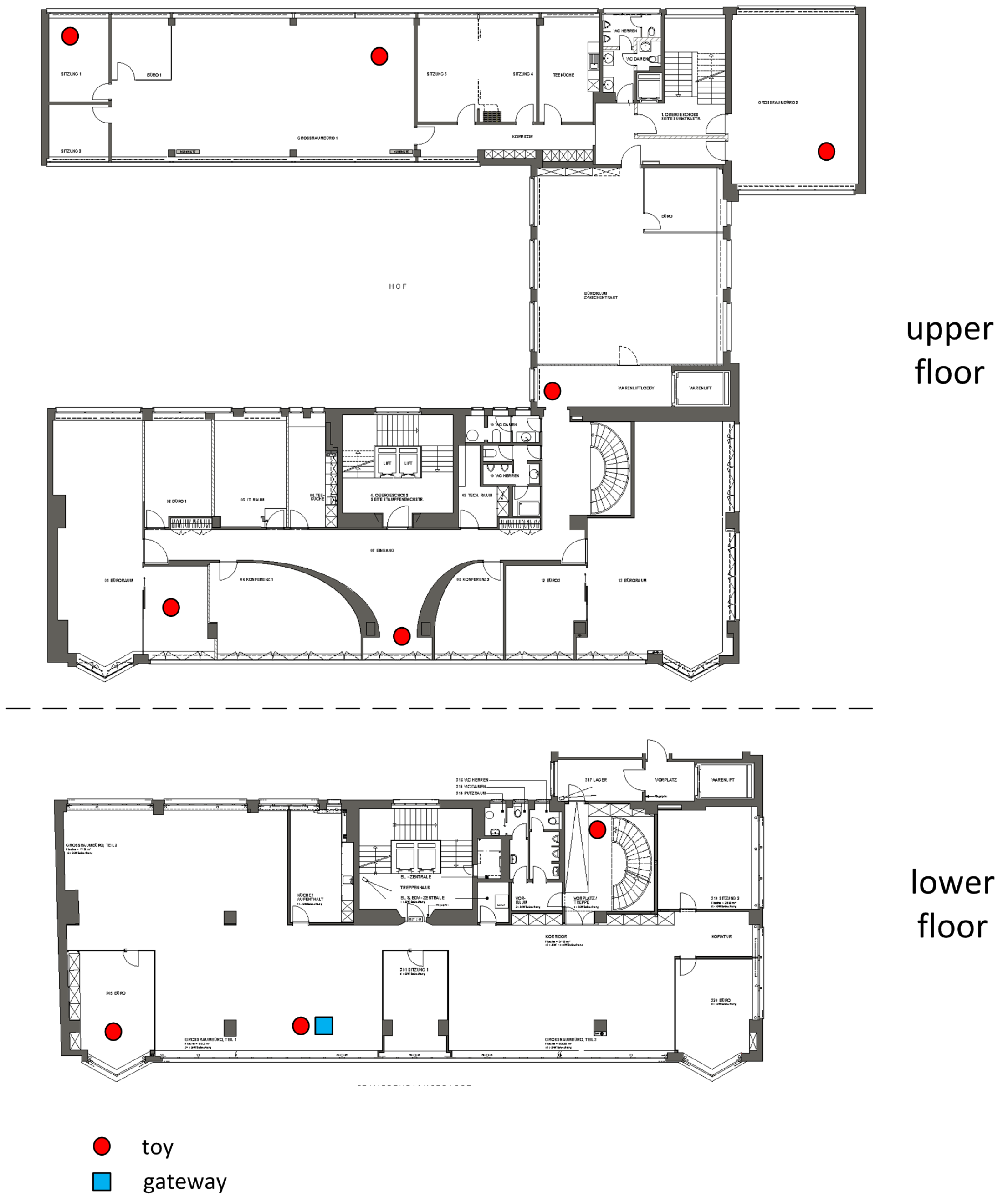
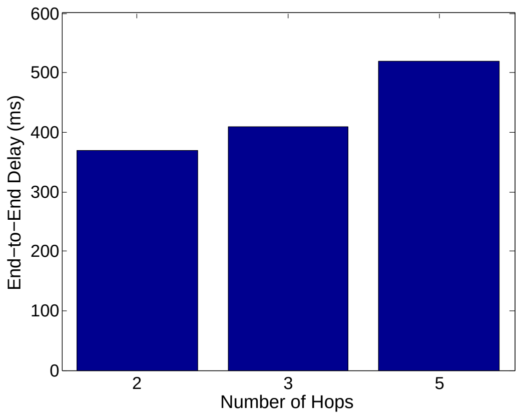
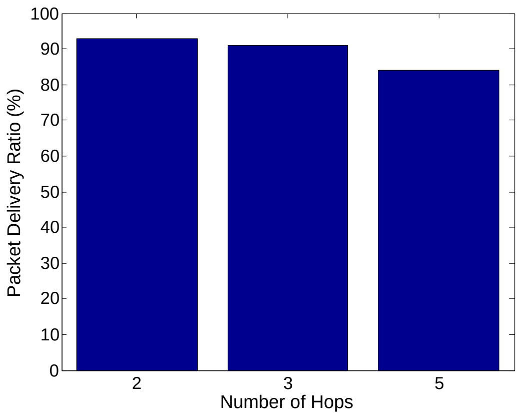


Figure 19: Testbed of the Smart Toy prototype evaluation.

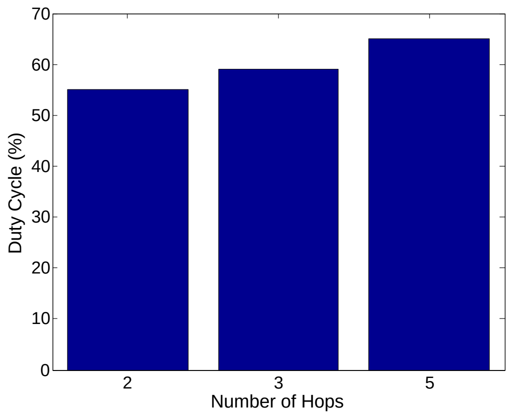
To evaluate the performance of 802.11 MH-PSM, we measure the average end-to-end packet delay, the average packet delivery ratio and the average duty cycle across all nodes of the network. At the same time we evaluate the performance of RRPL and Featurecast. We measure the average convergence time while RRPL is establishing new routes, the lengths of the established routes/paths, as well as the traffic overhead of RRPL. Moreover, for evaluating Featurecast we report the memory used by a node to store the routing tables, the average overhead to fill in the routing tables with the route information for "feature addresses" and the number of packets to reach the set of nodes defined by the "feature address". Finally, we measure the code footprint of RRPL and Featurecast, which is the amount of flash memory each protocol uses when compiled into the toy firmware. We run the tests three times for about 5 minutes



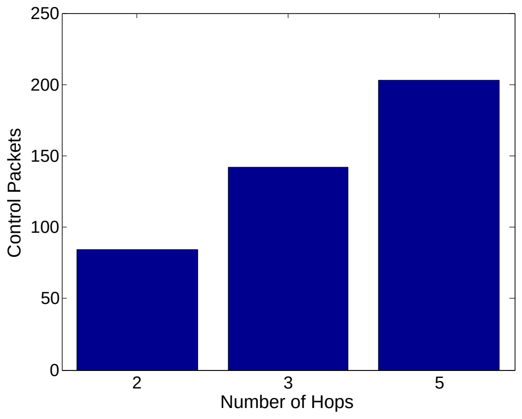
(a) Delay



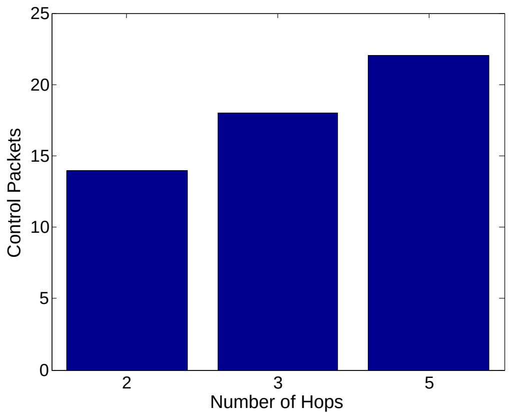
(b) PDR



(c) Duty Cycle



(d) RRPL Control Packet Overhead



(e) Featurecast Control Packet Overhead

Figure 20: Performance comparison in terms of (a) delay, (b) packet delivery ratio, (c) duty cycle, (d) RRPL control packet overhead and (e) Featurecast control packet overhead for establishing and maintaining connectivity for nodes in 2-, 3- and 5-hops away from the sink.

each time, and we report the average numbers for the metrics.

The code footprint of both RPL and RRPL alone and in coexistence with Featurecast are presented in Table 5. We see that RRPL has the highest footprint, especially in combination with Featurecast. Furthermore, via experimentation we find the convergence time of RRPL to be approximately 3 minutes to establish the routes in the network. The memory used by Featurecast to store the routing tables on a node depends on the topology and how many children each node has. However, for storing one child the node needs 28 bytes (as also shown in Figures 18a and 18b), the storing space grows linearly for stations with more children. As far as the flooding cost of Featurecast is concerned, it can vary again based on the topology scenario. In the fixed scenario examined in this section and shown in Figure 19 the flooding cost is 23.

In this fist scenario we use fixed placement of all nodes in the testbed. Figure 20 presents the experimental results of the performance of the Wi-Fi 802.11 with MH-PSM, and RRPL with Featurecast connectivity performance in the scenario of Figure 19. The gateway or sink coordinates the routing of the rest of the nodes according to RRPL and Featurecast. The rest of the non-sink nodes are only receiving routing information upon request from the sink.

We see that the delay is much higher in these results compared to the results shown in section 3.2.1. We believe that this extra time added is because of the routing information provided by RRPL, whereas in the results of section 3.2.1 the route was predetermined and not built at runtime. The duty cycle results are also different than what shown in the previous section, because the transmission frequency was much higher (every 200 ms) than the current results (every 1 sec). Moreover, in our scenario all 10 nodes transmit unicast traffic towards all other nodes, as if in a real world game application, creating a different congested environment than the one in section 3.2.1.

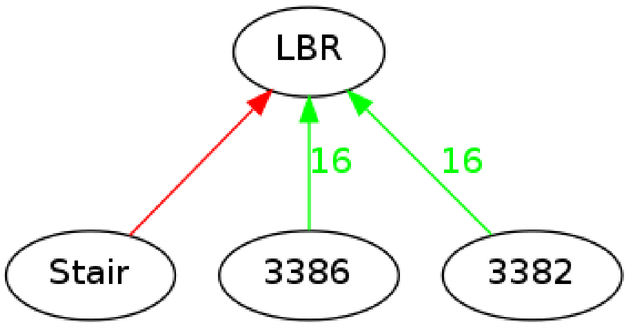
3.3. Demonstration overview

The final demonstration will be divided in two parts. The technical and the storytelling demonstration.

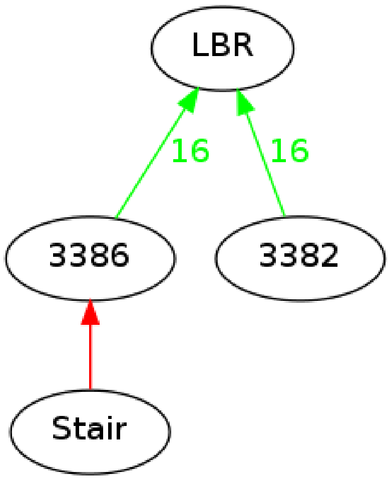
3.3.1. Technical Demonstration - Mobility

In the technical part of the demo we demonstrate the connectivity of our ad hoc sensor network. We generate a topology visualizer in the form of a tree that also includes routing information. The routing information is acquired by the RPL or Reactive RPL (RRPL) routing protocol [9]. RRPL is based on RPL, which is an IPv6 Routing Protocol for Low power and lossy networks. Routing is based on a Destination Oriented Directed Acyclic Graph (DODAG). This organizes the optimal routes between sink and other nodes for both collection and distribution of data traffic. Redundant equivalent routes are kept for reliability in case of link or node failure. Each DODAG can have a different optimization criterion. In our case, we create a DODAG with the criterion of the strongest link according to the specifications of RPL/RRPL. We also depict the cost of each link (e.g., ETX). The current RPL implementation for the Contiki operating system adopts, by default, the Expected Transmission Count (ETX) metric. According to ETX [4], the objective function to be minimized is the expected total number of packet transmissions required to successfully deliver a packet to the ultimate destination.

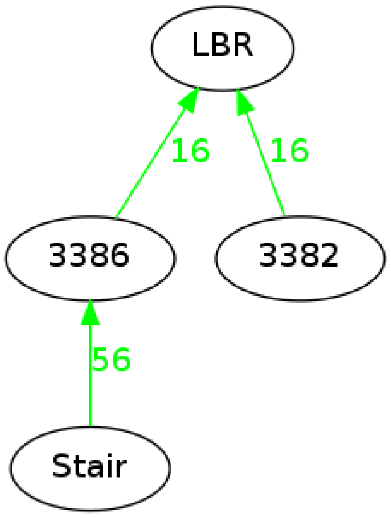
We use the Contiki UIP, RPL, RRPL, Erbium, CoAP13 libs. Moreover, we also use the



(a) Time elapsed: 35 sec



(b) Time elapsed: 230 sec



(c) Time elapsed: 250 sec

Figure 21: Technical demonstration instance example. Green line indicates a stable connection in DODAG. Red colour indicates the opposite.

Native Border Router[2], javagraphviz[3], and californium[4]. Each node sends CoAP messages with information about its parent and link ETX to the Visualizer via the gateway/sink node. The gateway/sink node then feeds this information to the tree visualizer.

Figure 21 depicts an instance of how we technically demonstrate the wireless connectivity and the routing developed as part of the CAL*IP*SO protocol stack to enable applications like the proposed Smart Toy prototype. As toy-nodes move then the tree adapts and re-organizes itself according to the movement of the nodes in the network. In the case of Figure 21a, node LBR is the gateway/sink node. Two nodes with IP addresses ending in "3386" and "3382" are in one hop distance from the gateway. Another node called "Stair" is placed within range but the connection to LBR is not stable once 35 seconds have elapsed. Then we move the "Stair" node further away reaching the stairs (as shown in the floorplan of Figure 19) when 230 seconds have elapsed from the beginning of the experiment (see Figure 21b). The "Stair" node is then connected to the network with the "3386" node as parent. Still the link is not stable until another few seconds have passed (Figure 21c) and the DODAG information is passed to the gateway/sink node and the tree established the connection of "Stair" node as stable in two-hop distance from the gateway. The ETX metric is also part of this visualization, showing that 16 packet transmissions are required to successfully deliver a packet to the next hop, while 56 are needed for the more faraway station (i.e., "Stair").

3.3.2. Storytelling Demo

We use the hardware depicted in Figure 11 and place it inside each toy. The toys are placed in a suitcase as in Figure 22. When the suitcase is open the Smart Toys look as shown in Figure 23a. Specifically, we use an Arduino Due, Elechouse MP3 Shield, Wi-Fi Dongle, and mp3 files divided into multiple stories. Each story consists of 2 to 7 characters. The mp3 files are stored on an SD card attached to the Elechouse MP3 Shield, as shown in Figure 11. The user selects the story to be told by the Smart Toys via a mobile web application, similarly to Figure 23b. Note that this is an initial design of the mobile application and is subject to change. Once the story is selected on the mobile web application, then the Smart Toys start interacting with each other saying the story, until the user decided to stop it from the web application.

The algorithm of the storytelling demo follows:

```
1. Handshaking of nodes:
   If not connected -> go to step 1
   If connected -> go to step 2
2. Wait until TCPIP/internal events (yield other processes)
3. If receive an event notification (from Web app or other toy)
        - get current track number
        - increase current track number current_track++
        - activate MP3 Shield
4. Once finish, send the current track number to the next toy
        back to step 2 (waiting)
```

[2]Contiki /examples/ipv6/native-border-router
[3]https://github.com/verto/java-graphviz
[4]https://github.com/mkovatsc/Californium

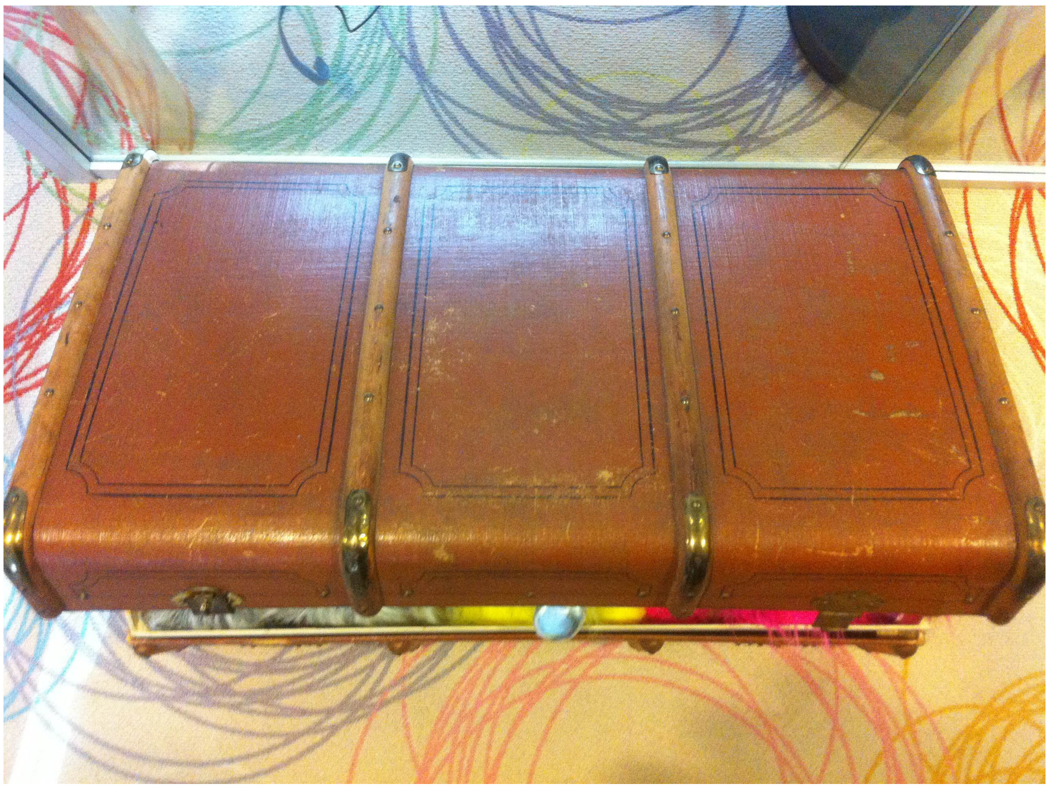
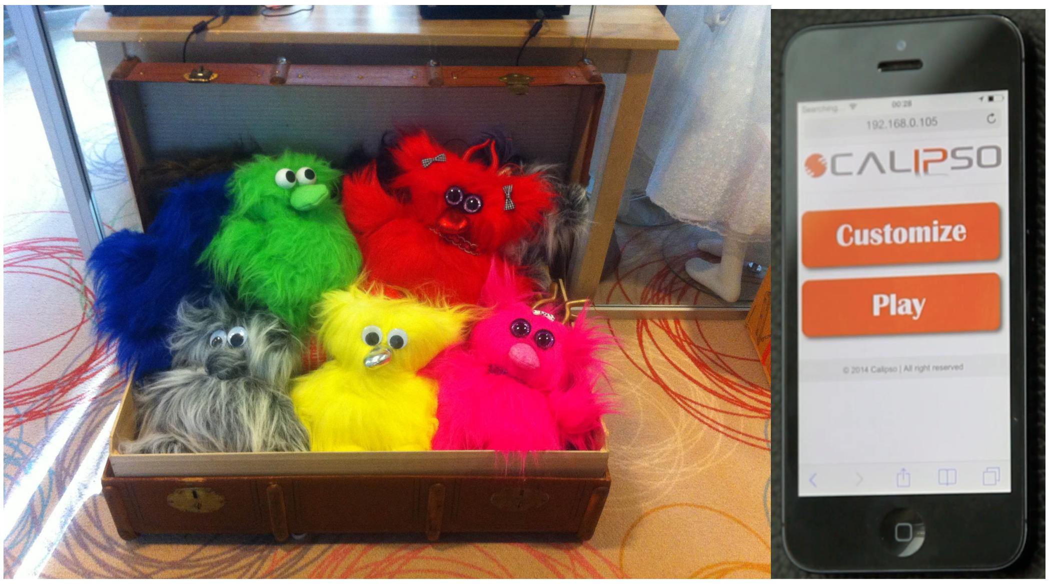


Figure 22: Storytelling demonstration suitcase containing the Smart Toys.



(a) Smart Toys (b) Smart Toys' Application

Figure 23: Storytelling demonstration for the Smart Toys.

4. Conclusions & Perspectives

The CAL*IP*SO software modules enable seamless connectivity between wireless devices such as toys, smartphones and other sensors. It is expected that –thanks to the new software modules of the CAL*IP*SO project– the path towards commercial exploitation and new revenue streams will be opened.

In the case of Smart Parking, CAL*IP*SO project has delivered a set of modules to build up an entire communication stack that is suitable for this application. Although new radio techniques has appeared in the last years named (LPWA)[5] like LoRa by Semtech, SigFox, Weighless and others that they seem to likely fit to these smart-applications in cities, it is still to be tested and studied the best solution versus classical WSNs (ZigBee-like networks). These new radio technologies lack the feature of having IP (or any of its variants) in the network they build, disallowing the use of upper protocols like CoAP. CAL*IP*SO brings a complete IP-based stack up to CoAP layer, bringing a competitive advantage among the LPWN technologies. Also, the experience acquired working with the entire CAL*IP*SO stack will be applied to other products by Worldsensing, as SHM[6] wireless products that nowadays are not using any standard stack.

Finally, adapting the story telling of Smart Toys with the help of CAL*IP*SO-enabled software (based on certified, free and standard-compliant solutions) is a much-demanded feature in the toy industry: Companion toys can now grow with the child. Moreover, such new features must not come at the cost of extensive energy consumption, a challenge that has been addressed by CAL*IP*SO already in the design phase. Also, the connectivity among the toys (i.e., toy-to-toy communication) is a desirable feature when crowds or families of toys are targeted (such as with so-called die cast collectibles that are popular at the Disneyland theme parks). With many Disney franchises and their related toy merchandise, like Lucas Art with the upcoming Star Wars 7, the Marvell Universe, Pixar Cars and Pixar Toystory, play patterns relay often on a larger set of (wirelessly connected) devices, instead of just a single toy. Fourth, and finally, additional complexity in the experience design of consumer products should come at lowest cost. It is easy to dream up new creative story telling of toys (for example listening and talking devices) when cost considerations are not taking into account. This relates to an important achievement of CAL*IP*SO's overall efforts. Due to the IoT connectivity of toys, new experiences such as tracking, speech recognition or speech synthesizing can now be offloaded, and the CAL*IP*SO toys can mimic a smartness that unlocks the much demanded novel play-patterns.

References

[1] Atheros AR9001U. http://wikidevi.com/files/Atheros/specsheets/ AR9001U.pdf. [Apr-2013].

[2] Contiki OS. http://www.contiki-os.org/. [Apr-2013].

[3] Otus. http://linuxwireless.org/en/users/Drivers/otus. [Apr-2013].

[4] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A High-Throughput Path Metric for Multi-hop Wireless Routing. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2003.

[5] LPWA stands for Low-Power Wide Area networks

[6] Structural Health Monitoring

[5] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. Let the Tree Bloom: Scalable Opportunistic Routing with ORPL. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2013)*, Rome, Italy, November 2013.

[6] Ioannis Glaropoulos, Stefan Mangold, and Vladimir Vukadinovic. Enhanced IEEE 802.11 Power Saving for Multi-hop Toy-to-Toy Communication. In *IEEE Internet of Things (iThings)*, 2013.

[7] Pietro Gonizzi, Paolo Medagliani, Gianluigi Ferrari, and Jérémie Leguay. Rawmac: A routing aware wave-based mac protocol for wsns. In *International Workshop on the GReen Optimized Wireless Networks (GROWN 2014), in conjunction with the 10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2014)*, Larnaca, Cyprus, 8-10 October 2014.

[8] Michał Król, Franck Rousseau, and Andrzej Duda. Représentation compacte des adresses pour le routage par caractéristiques. *ALGOTEL 2014–16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2014.

[9] Chi-Anh La, Martin Heusse, and Andrzej Duda. Link reversal and reactive routing in low power and lossy networks. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*, pages 3386–3390, 2013.

[10] Vladimir Vukadinovic, Ioannis Glaropoulos, and Stefan Mangold. Enhanced Power Saving Mode for Low-Latency Communication in Multi-Hop 802.11 Networks. In *Elsevier Ad Hoc Networks*, 2014.