Rover: Poor (but Elegant) Man's Testbed

Zacharie Brodard, Hao Jiang, Tengfei Chang, Thomas Watteyne, Xavier Vilajosana, Pascal Thubert, Géraldine Texier

Rover: Poor (but Elegant) Man's Testbed

Zacharie Brodard
Cisco Systems
Paris, France
brodazac@cisco.com

Hao Jiang
Cisco Systems
Paris, France
hjiang2@cisco.com

Tengfei Chang
Inria, EVA team
Paris, France
tengfei.chang@inria.fr

Thomas Watteyne
Inria, EVA team
Paris, France
thomas.watteyne@inria.fr

Xavier Vilajosana
Univ. Oberta de Catalunya
Barcelona, Catalunya, Spain
xvilajosana@uoc.edu

Pascal Thubert
Cisco Systems, France
Telecom Bretagne, France
pthubert@cisco.com

ABSTRACT

This paper presents the OpenVisualizer Rover testbed, a simple, easy-to-deploy and cheap testbed for the Internet of Things (IoT). The OpenWSN project provides a free and open-source implementation of a standards-compliant protocol stack for the IoT, as well as all the necessary network management and debugging tools. The network management software, OpenVisualizer, is portable across popular operating systems, and connects the OpenWSN low-power wireless mesh network to the Internet. In the current setup, motes are connected to the USB ports of the computer the OpenVisualizer runs on. The OpenVisualizer monitors the internal state of those motes, which it presents through a web interface. Rover extends the OpenVisualizer by allowing motes plugged into different computers to remotely connect to it. Once connected, a user monitors and manages the motes exactly as if they were connected locally. This offers endless experimentation possibilities, as the resulting testbed can be quickly (re)deployed in realistic environments. An example Rover testbed has been deployed at the Cisco Paris Innovation and Research Lab. This paper discusses the Rover architecture, the deployment, and the experimental research done with it.

CCS Concepts

•**Networks** → **Network experimentation;** *Network protocol design;* Network measurement;

Keywords

Internet of Things; Testbed; OpenWSN; OpenMote.

1. INTRODUCTION

With the development of new technologies in ICT, we are witnessing during the last few years the emergence and fast growth of the Internet of Things (IoT). To exploit, test and verify new IoT technologies and standards, several experimental testbeds have been developed and deployed to enable experimental research and benchmarking. FIT IoT-lab [1] is a testbed deployed in 6 sites across France, composed of 2728 low-power wireless nodes and 117 mobile robots, available for experimenting with large-scale wireless IoT technologies. Indriya [2] is another large-scale wireless sensor network testbed deployed at the National University of Singapore. It uses TelosB devices and is built on an active-USB infrastructure. Tutornet[1] is a testbed running in Ronald Tutor Hall at University of Southern California. It consists of 13 clusters, each cluster composed of a stargate and several motes attached via USB cables. A central PC can program any mote in the testbed through the communication with these stargates over WiFi.

Such shared testbeds have a couple of shortcomings. First, they are shared by nature: one has to wait for the motes to become available before being able to start an experiment. Second, they only feature a handful of types of motes (hardware) and its hard – if at all possible – to swap the hardware used. Third, they are deployed in a single location, which is not representative of the variety of environments low-power networks are deployed in. Forth, these testbeds are complex and expensive to install, and often require a team of

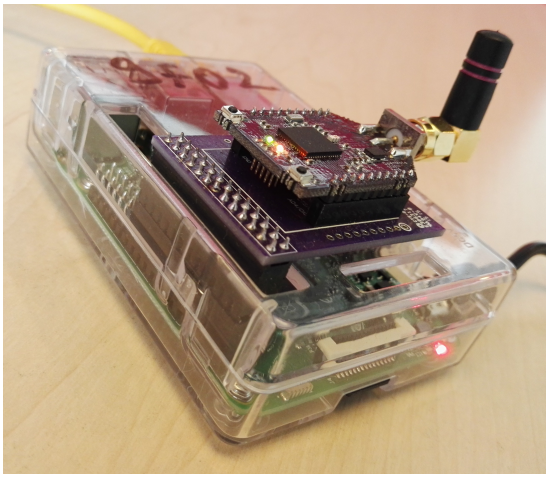

Figure 1: A Rover node.

[1] http://testbed.usc.edu/

PE-WASUN'16 13–17 November 2016, Valletta, Malta

DOI: XX.XXX/XXX_X

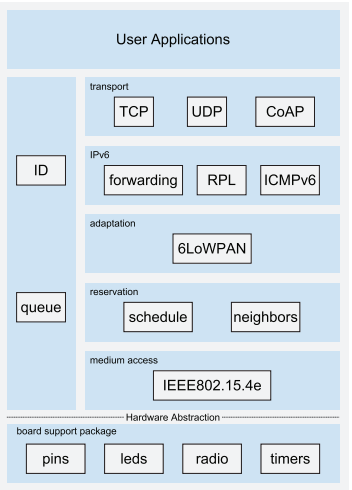

Figure 2: The OpenWSN protocol stack.

full-time engineers or grad students to keep them up and running. And finally, these testbeds are academic/open by nature, and not appropriate for in-house developments or proprietary technology. The goal of the Rover architecture presented in this paper is to address these shortcomings.

Rover combines the strengths of the OpenMote hardware and the OpenWSN software. **OpenMote**[2] [6] is an open-hardware prototyping ecosystem designed to accelerate the development of the Industrial Internet of Things (IIoT). The OpenMote-CC2538 is the core of this ecosystem. It is a 4 cm × 3 cm board which provides computation and communication capabilities. Its main component is the Texas Instrument CC2538, a System-on-Chip (SoC) with a 32-bit ARM Cortex-M3 micro-controller and a 2.4GHz IEEE802.15.4 radio transceiver. **OpenWSN**[3] [7] provides an open-source implementation of a fully standards-based protocol stack for the IoT, running on a variety of hardware and software platforms. With a suite of free and open-source debugging and integration tools, it aims to help academia and industry verify the applicability of these standards to the Internet of Things. It is the reference open-source implementation of the IEEE802.15.4-2015 Time Synchronized Channel Hopping (TSCH) standard and IPv6 over TSCH (6TiSCH). Fig. 2 depicts the protocol stack implemented in OpenWSN. OpenMote and OpenWSN have been designed together.

The Raspberry Pi[4] is a credit-card sized single-board computer, which aims at providing low-cost, high-performance computers to stimulate the education of basic computer science at schools. It features a Broadcom BCM2835 SoC, and uses an SD card for booting and long-term storage. The different Debian and Arch Linux flavors that run on it comes with a Python interpreter. It has become very popular gateway computer for IoT networks. Rover uses it as the default platform to connect OpenMote boards to.

The goal of Rover is to offer all the pieces necessary for building a testbed which is easily (re-)deployable, cheap and running cutting-edge IOT protocols and standards. The Rover testbed integrates the Raspberry Pi, OpenMote and OpenWSN projects.

The contribution of this paper is threefold:

- It presents Rover, an easy-to-deploy, cheap, standards-ready testbed for the IoT.
- It shows an example deployment of a Rover testbed at Cisco-Paris.
- It discusses cases for which this type of testbed in particularly suited.

The remainder of this paper is organized as follows. Section 2 introduces the OpenVisualizer, the software running on the PC controller the testbed. Section 3 presents the Rover architecture, highlights the components developed as an contribution to the OpenVisualizer, and lists the key features of the Rover testbed. Section 4 describes how a Rover tesbed is currently deployed at Cisco-Paris. Section 5 discusses example use cases in which Rover-based experimentation is particularly suitable. Finally, Section 6 concludes this paper and discusses future work.

2. OPENVISUALIZER

The OpenVisualizer is a Python-based debugging and visualization tool that runs on a Windows/Linux/OSX computer and interacts with the motes connected to it over serial ports. It includes 6LoWPAN Low-power Border Router (LBR) functionality, i.e. it translates 6LoWPAN packets into IPv6 packets, thereby connecting the low-power wireless mesh to the Internet. Its web-based user interface provides relevant information about the OpenWSN network. This includes the internal states of each mote (neighbor table, communication schedule, routing table, a.o.), the multi-hop routing structure, and the debug/error messages generated by the motes. The OpenVisualizer also sends commands and data to the motes, in particular packets coming from the Internet.

The OpenVisualizer is built following the "Event Bus" software design pattern. Components connected to the Event Bus use a publish-subscribe approach to send messages to one another. This modular software architecture makes the code maintainable, flexible and extensible. The Event Bus has proven to be a powerful tool to write applications and support the ongoing development of communication standards and protocols.

Fig. 3 depicts the general architecture of the OpenVisualizer. The Event Bus provides the messaging framework for the full chain of services between a mote and the external network. Components are conceptually grouped into the functional areas shown on the bus. Each component implements an event handler and provides a specific service. The `openTun` component creates and uses an IPv6 virtual `TUN` interface, allowing motes to communicate with external applications running on the Internet. `LBR` is the component that provides low-power border router translation between IPv6 packets on the external network, and 6LoWPAN packets in the low-power wireless mesh. The `RPL` component manages the multi-hop routes by implementing the RPL standard. It receives the RPL DAO messages sent by the motes, and maintains a central view of the RPL topology; it uses that view to build source routes for packet sent into the mesh. Each `moteConnector` is the primary interface for an individual mote; it pushes events out from the mote onto the bus, for other components to display/interpret.

[2] www.openmote.com
[3] www.openwsn.org
[4] www.raspberrypi.org

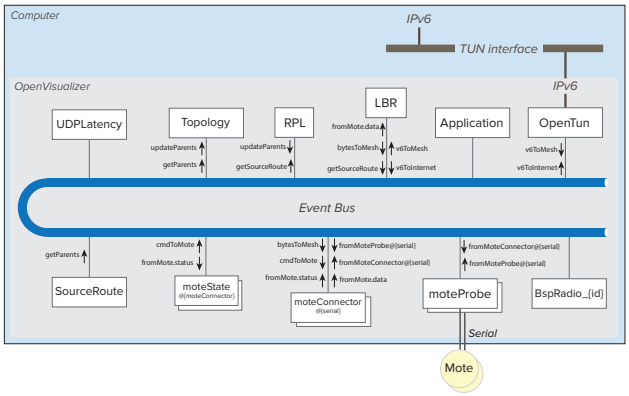

Figure 3: OpenVisualizer *without* Rover.

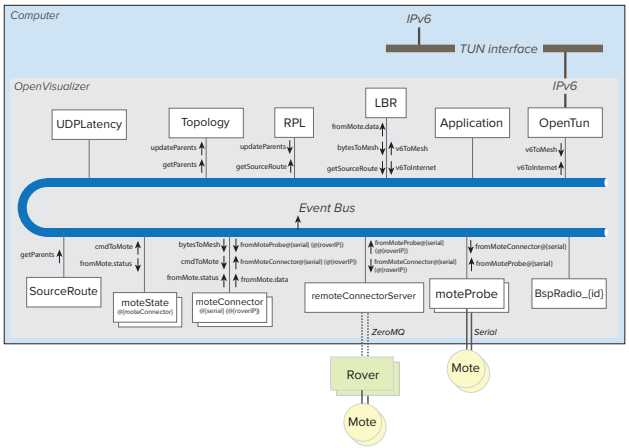

Figure 4: OpenVisualizer *with* Rover.

Each `moteConnector` communicates with a mote over a serial port using a `moteProbe` component. For each message received from the mote, the `moteProbe` encapsulates it as an event which it publishes on the Event Bus. Similarly, the `moteProbe` subscribes to `fromMoteConnector@{serial}` notifications at the Event Bus, to forward them to the mote over its serial port. The serial frames between the mote and the computer are framed using HDLC, implementing RFC1662 [4]. The payload of the HDLC frame has a leading 1-byte "type" field which indicates the format of the followed payload.

With each mote connected to the OpenVisualizer through the serial, and almost all the necessary information about the low-power wireless mesh fully visualized, it becomes easy to conduct IoT research and development activities. Yet, it requires for all the nodes to be physically connected to a same computer, severely limiting the scalability of the approach.

3. THE ROVER TESTBED

In any realistic deployment, the different motes of the mesh are not close enough to be connected directly to the same computer running the OpenVisualizer. What is needed is a mechanism to allow some motes to connect remotely. This paper presents the improvement to the OpenVisualizer which allows just that. We implemented the "Rover" module, a version of the `moteProbe` which is connected to the serial port of the mote, and connects to the main Event Bus of the OpenVisualizer remotely.

3.1 Architecture

The new architecture can be divided into two parts, with most components running in the central OpenVisualizer, and the `moteProbe` components running remotely on each Rover node. The architecture defines a new `remoteConnectorServer` component for the OpenVisualizer, and a `remoteConnectorRover` component for the Rover node. These bridge the signaling data between the motes and the central computer, without removing any of the OpenVisualizer functionality. Fig. 4 shows the resulting OpenVisualizer architecture.

The original `moteProbe` component still remains available; the new architecture supports both local and remote motes, concurrently. `RemoteConnnectorServer` is the new OpenVisualizer component which allows the mote to connect remotely. It maintains a TCP session between each Rover node and the main OpenVisualizer. Through that session, it transfers all the events destined to a remote Rover node. Simlarly, it receives incoming events from the Rover nodes, and dispatches them in the local Event Bus. No other changes to the core components of the OpenVisualizer were necessary.

The `remoteConnectorServer` communicates with each `remoteConnectorRover` using ZeroMQ[5]. ZeroMQ is an asynchronous messaging system which offers one-to-many communication between the OpenVisualizer and the Rover nodes. Components exchange messages using a publish-subscribe pattern: the `remoteConnectorServer` publishes events the Rover nodes subscribe to, and subscribes to all the Rover nodes to receive their events. The result is that the Event Bus of OpenVisualizer is extended remotely to each Rover node; all events are transparently transferred between components, whether local or remote.

3.2 Additional Features

The Rover functionality is built into the OpenPi platform[6], the OpenWSN-ready distribution for the Raspberry Pi. In the OpenPi, an OpenVisualizer Rover program runs as a service called "openrover", and serves socket connection requests from the central OpenVisualizer.

Once the remote connection is established, the OpenVisualizer is able to monitor and manage all the motes, regardless of whether they are local or remote. It also allows the user to remotely reset or reflash the motes with any arbitrary firmware.

The Rover code was contributed to the OpenWSN project, and is available under an open-source BSD license[7].

4. EXAMPLE DEPLOYMENT

Fig. 1 shows a Rover node, consisting of an OpenMote and a Raspberry Pi. Fig. 5 show the block diagram of

[5] http://zeromq.org/
[6] openpi.openwsn.org
[7] As an online addition to this paper, all source code is available at https://github.com/openwsn-berkeley/openpi under a BSD open-source license.

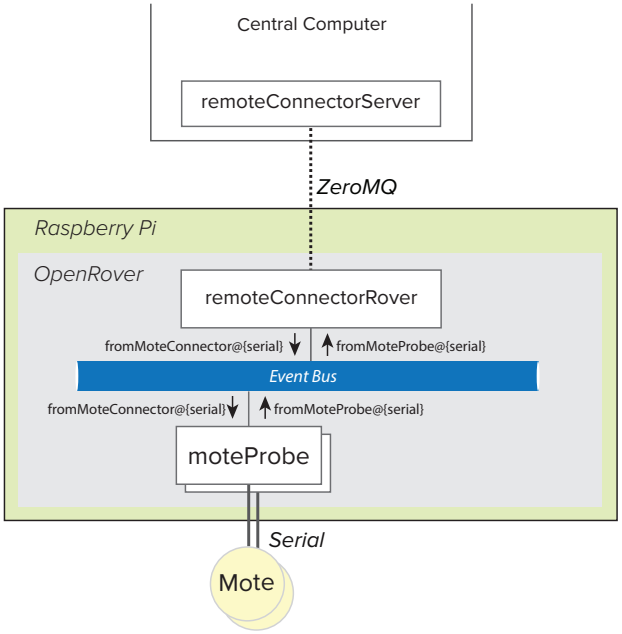

Figure 5: Functional block diagram of a Rover node.

OpenMote-CC2538	75 €
Raspberry Pi 3	35 €
enclosure	5 €
SD card	15 €
RJ45 cable	5 €
Total	135 €

Table 1: Approximate cost of a Rover node.

the software running on the Raspberry Pi The `moteProbe` component is equivalent to that running in the OpenVisualizer. It is the `remoteConnectorRover` component which maintains the (remote) connection to the OpenVisualizer The `remoteConnectorRover` component subscribes to the `fromMoteProbe@{serial}` events from the local Event Bus, and forwards them to the OpenVisualizer. Similarly, the `remoteConnectorRover` component republishes `fromMoteConnector@{serial}@{roverIP}` events received from the OpenVisualizer into local Event Bus.

The `remoteConnectorRover` communicates with the `remoteConnectorServer` on the OpenVisualizer using ZeroMQ. Each time a Rover node connects, the OpenVisualizer creates dedicated `moteState` and `moteConnector` components, identified by the "`@{roverIP}`" and "`@{serial}`" suffixes. From that moment on, Rover node and the OpenVisualizer exchange all relevant information.

Table 1 gives an approximate cost of a Rover node.

A Rover-based testbed is currently deployed on the sixth floor of the Cisco Innovation and Research Lab in Paris, France. Fig. 6 shows the 8-node topology, each label are the last 2 bytes of the OpenMote's MAC address, in hexadecimal notation. Mote `9E-D9` is the DAGroot, and is connected to the Raspberry Pi which serves as the central computer running the OpenVisualizer. The remaining nodes remote, each connected to a Raspberry Pi running the OpenRover software. For this experiment, The topology is hardcoded

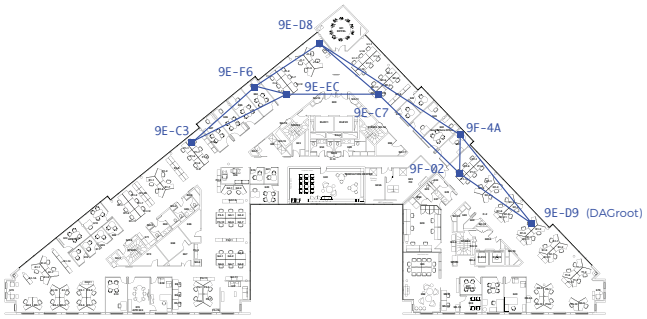

Figure 6: Bird view of the Rover testbed running at Cisco-Paris.

in a general shape of a ladder, and each mote check at layer 2 if each received packet comes from one of their neighbors, if not the packet is simply dropped.

5. EXAMPLE USE CASE

This section illustrates how the Rover testbed is used to conduct research on 6TiSCH [5] schedule management with Traffic Engineering for Bit Indexed Explicit Repliation (BIER-TE).

BIER (Bit Indexed Explicit Replication) is a new multicast forwarding protocol [8]. When a multicast data packet enters a BIER domain, the ingress router determines a set of egress routers to which the packet needs to be sent, and then encapsulates the packet in a BIER header. The BIER header contains a bitstring in which each bit represents exactly one egress router in the domain. To send a packet to a particular set of egress nodes, the ingress node sets the bits for each of those egress nodes, and clears other bits in the bitstring. Each packet can then be forwarded along the unicast shortest path tree from the ingress node to the egress nodes based on some routing protocol.

BIER-TE [3] is a Traffic Engineering technique inspired from BIER and being standardized. It forwards and replicates packets based on BIER-like bitstrings, but does not require an routing protocol. The key differences over BIER are: (1) BIER-TE replaces in-network, autonomous, and IGP-based path calculation by explicit paths calculated off-path by a controller; (2) in BIER-TE, every bitposition of the bitstring of a BIER-TE packet indicates one or more adjacencies – instead of a set of destination nodes as in BIER; (3) in BIER-TE, each node has no routing table but only a BIER-TE Forwarding Table (BIFT). The BIFT specifies how the node should replicate packets to its adjacencies according to the bitstring contained in the BIER header of the packet. BIER-TE can thus be used to enable path diversity by controlling replication and elimination by tuning the bitstring in the BIER header. The output bitstring from the last node in path can be used to identify transmission failures, and this information can be passed to the central controller, which in turn can modify the bitString for the next packets.

To explore the performance of BIER-TE on a 6TiSCH low-power wireless mesh, we implemented BIER-TE on OpenWSN and evaluated its performance on the Cisco-Paris Rover testbed. Fig. 7 depicts how the test is used. The OpenVisualizer shows the network topology in real time. Upon receiving a new flow, the OpenVisualizer computes a com-

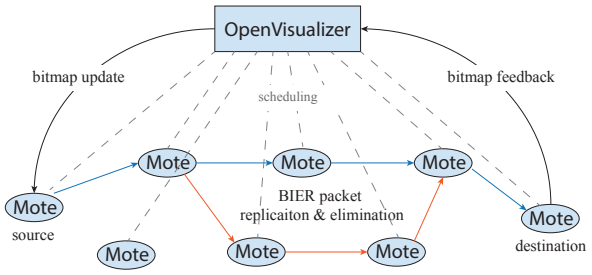

Figure 7: Using the Rover testbed to benchmark BIER-TE on OpenWSN.

plex path, reserves timeslots, and installs a BIFT into each node over their serial port. An associated bitmap for this flow is then assigned by the OpenVisualizer in the ingress node, and inserted into the BIER header of each packet. Each node along the path receiving the packet examines the bitmap against its BIFT and performs packet forwarding, replication or elimination operations. When the destination node receives the packet, it forwards the payload data to the upper layer and feeds the bitmap and delay back to OpenVisualizer. With the Rover testbed, various experiments are conducted to benchmark the performance of BIER-TE over 6TiSCH. For each experiment, the code is instrumented to measure jitter, latency, loss ratio and reliability.

The Rover testbed thus proved itself essential to conduct this real-world experimental study. It integrates both the advanced OpenWSN software and the state-of-the-art OpenMote hardware. It includes an open-source and up-to-date implementation of a complete protocol stack based on IoT standards, which makes the testbed complete and powerful. The real-time network visualization makes it simple and easy to use. The OpenVisualizer offers network monitoring and management capabilities, no matter whether the mote is local or remote, real or simulated. The fact that each Rover node can be positioned anywhere while maintaining the connection with the central OpenVisualizer is a critical feature. Because of their Ethernet connectivity, and the low cost of the nodes, the testbed can be redeployed in minutes. We strongly believe the Rover architecture is key tool for conducting benchmarking and feasibility studies, in particular for the standards for the Industrial IoT.

6. CONCLUSION

This paper introduces Rover, an IoT testbed which combines OpenWSN software and OpenMote hardware in an easy-to-use, flexible and cheap solution. It offers an ideal real-world experimentation environment for both academic and industry R&D activities, and will help shape the future of the Internet of Things. It improves to the OpenVisualizer software by allowing node to connect remotely. Through the connection, the OpenVisualizer and Rover node exchange signaling and data.

We are currently working on a mechanism for the OpenVisualizer to automatically discover the Rover nodes which are in the local network, either through mDNS, or by using a registration mechanism to a resource directory. We are also working towards a Rover node which uses an Intel Edison rather than a Raspberry Pi. This will allow Rover nodes to connect to WiFi (at 5 GHz) rather than Ethernet, offering even more flexibility.

Acknowledgment

This work was partly supported by the European Commission's Horizon 2020 Framework Programme, through the H2020 F-Interop and H2020 ARMOUR projects.

7. ADDITIONAL AUTHORS

Additional author: Geraldine Texier (Telecom Bretagne, email: `geraldine.texier@telecom-bretagne.eu`).

8. REFERENCES

[1] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. FIT IoT-LAB: A large scale open experimental IoT testbed. In *World Forum on Internet of Things (WF-IoT)*, pages 459–464, Milan, Italy, 14-16 December 2015. IEEE.

[2] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. Indriya: A Low-cost, 3D Wireless Sensor Network Testbed. In *International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, pages 302–316, Shanghai, China, 17-19 April 2011. Springer.

[3] T. Eckert, G. Cauchie, W. Braun, and M. Menth. Traffic Engineering for Bit Index Explicit Replication BIER-TE, 8 July 2016.

[4] W. A. Simpson. PPP in HDLC-like Framing. RFC1662, July 1994.

[5] P. Thubert. An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4. draft-ietf-6tisch-architecture-10 [work-in-progress], 10 June 2016.

[6] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister. OpenMote: Open-Source Prototyping Platform for the Industrial IoT. In *7th EAI International Conference on Ad Hoc Networks (AdHocNets)*, pages 211–222, San Remo, Italy, 1-2 September 2015. Springer.

[7] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister. OpenWSN: a Standards-based Low-Power Wireless Development Environment. *Transactions on Emerging Telecommunications Technologies (ETT)*, 23(5):480–493, 2012.

[8] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. K. Aldrin. Multicast using Bit Index Explicit Replication. draft-ietf-bier-architecture-04 [work-in-progress], 18 July 2016.