Efficient and scalable IPv6 communication functions for wireless outdour lighting networks

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

Efficient and Scalable IPv6 Communication Functions For Wireless Outdoor Lighting Networks

Surafel Teshome Mamo

September 2014

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

Efficient and Scalable IPv6 Communication Functions For Wireless Outdoor Lighting Networks

Surafel Teshome Mamo

September 2014

Efficient and Scalable IPv6 Communication Functions For Wireless Outdoor Lighting Networks

Eindhoven University of Technology Stan Ackermans Institute / Software Technology

By Surafel Teshome Mamo

Partners

Project Owner:
POLAR Research Team

Project Manager:
Oscar Garcia Morchon

Project Mentor:
Dr. Esko Dijk

Project Supervisor:
Dr. Andrei Jalba

September 2014

Contact Address Eindhoven University of Technology Department of Mathematics and Computer Science MF 7.090, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands +31402474334

Published by Eindhoven University of Technology Stan Ackermans Institute

Printed by Eindhoven University of Technology UniversiteitsDrukkerij

ISBN 978-90-444-1325-0

Keywords Beacon frame, Bootstrapping, Design, Network, Node, Wireless, 6LoWPAN, 802.15.4

Partnership This project was supported by Eindhoven University of Technology and Philips Research.

Preferred reference Efficient and Scalable IPv6 Communication Functions For Wireless Outdoor Lighting Networks: Mesh Bootstrapping. Eindhoven University of Technology, SAI Technical Report, December, 2013. (978-90-444-1325-0)

Foreword

In 2013, Philips Research started a project to explore a new architecture for connected outdoor lighting. The new architecture, making use of 6LoWPAN radio mesh networking, is designed to bring end-to-end IPv6 connectivity to every light pole. For this project we needed "network-minded" people who understand embedded systems and could contribute to the design, implementation and evaluation of various data-communication related functions. The exact functions to work on were not even defined at the time we wrote the project description! When Surafel started this project, he showed to be capable and motivated to dive right into the new subject of networking for Internet-of-Things, where quite different design choices are taken and protocols are used than in the "regular" IP networking domain. After a quest for the right topic/function to work on, with some detours due to the quite dynamic and agile nature of the research project, we arrived at the topic of node bootstrapping (also known as "secure joining") in the mesh network. This function is a key part of the new communication architecture.

Surafel managed to create an extensive overview of existing bootstrapping methods and standards, and provided us with a good overview of the design space. Also a preferred design is described in this report. One of the key functions of the design is the so-called DTLS Relay. For this function, Surafel provided us with a reference implementation in Java and also showed that it would work for our connectivity architecture. Our expectation is that we can take Surafel's implementation in Q4 and integrate it as part of the prototype system we are developing. Thanks Surafel for your valuable work and contribution to Philips!

Esko Dijk

September 15th, 2014

Preface

This report is a step by step description of my work at Philips Research from January to September 2014. It discusses about a particular design challenges in outdoor lighting connectivity. more specifically, bootstrapping of Nodes in an IPv6 based Personal Area Network (6LoWPAN). The work presented in this document is a collaborative work between POLAR research team of Philips Research and the author of this document. Dr. Esko Dijk (Project Mentor) has been involved in every steps of this document in clarifying research questions, suggesting and recommending design methodologies, reviewing my work, as well as being a great support through out the life time of the project.

September 15th, 2014

Acknowledgments

would like to thank my supervisors, Dr. Esko Dijk and Assistant Professor Andrei Jalba for their support and guidance throughout the course of this project. It has been a great pleasure and experience working with you.

I would also like to deliver my big appreciation to the working culture of The Netherlands which make me keep the belief that people should be arbitrated by the work they have done.

These two years of my PDEng study has not been too easy on me. I would finally like to tell myself that “I did it again! ”

September 15th, 2014

Executive Summary

Outdoor lighting today is becoming increasingly network-connected. The rapid development in wireless communication technologies makes this progress faster and competitive. Philips Research and Philips Lighting are part of the leading forces in exploration and development of a wide spectrum of low-maintenance, high-quality outdoor/indoor lighting systems that are state of the art. City Touch is a proprietary outdoor lighting connectivity system of Philips Lighting, which is based on a client-server architecture.

In an outdoor lighting context, an embedded computer (Node) is installed on a light pole and is connected to different sensors to provide connectivity for the luminaires. Thus, connectivity of luminaires generally refers to the computer network of Nodes.

In this report, I present a survey of mechanisms, protocols and technologies that are needed for bootstrapping of wireless Nodes to an IPv6 based personal area network (PAN). The survey indicates that there is no single off-the-shelf product or standard that meets all the requirements of Philips research for its future solution. Hence, I designed a thorough bootstrapping protocol that is custom tailored to Philips 's POLAR architecture. The design brings a solution from pre-deployment configuration to the point where a new Node successfully becomes a part of a wireless network. The design is partially demonstrated with two software implementations. Finally I provide recommendations for future work based on my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Outdoor lighting today is becoming increasingly network-connected. The main drivers for this development are cost-saving (on energy and maintenance activities), public safety, and the drive for "green" lighting solutions. Outdoor lighting connectivity mainly includes creating and maintaining an end-to-end network of luminaires. This enables remote operations such as dimming, monitoring, enforcing scheduled lighting, and maintaining luminaires. In an outdoor lighting context, an embedded computer (Node) is installed on a light pole and is connected to different sensors to provide connectivity for the luminaires. Thus, connectivity of luminaires generally refers to the computer network of Nodes.

Philips Research and Philips Lighting are part of the leading forces in exploration and development of a wide spectrum of low-maintenance, high-quality outdoor/indoor lighting systems that are state of the art.

1.2 Context

Due to the ubiquitous nature of luminaire deployment over a certain area (mainly a city), outdoor lighting connectivity can be used in a wide range of application contexts. This includes surveillance, connectivity, hosting, and lighting management systems.

This section discusses common systems and services that can be operational in the application context. As Figure 1-1 illustrates, entities with green boundary line are systems and infrastructures that could use all diversions of the City Touch network infrastructure in order to be operational while entities with blue boundary line are systems and infrastructures that the

City Touch could use to be operational. The directed dotted lines indicate that the interaction is not mandatory.

The diagram only demonstrates the most significant systems that interact with the City Touch Network infrastructure. However, there could be other domains in which the proposed network could be used.

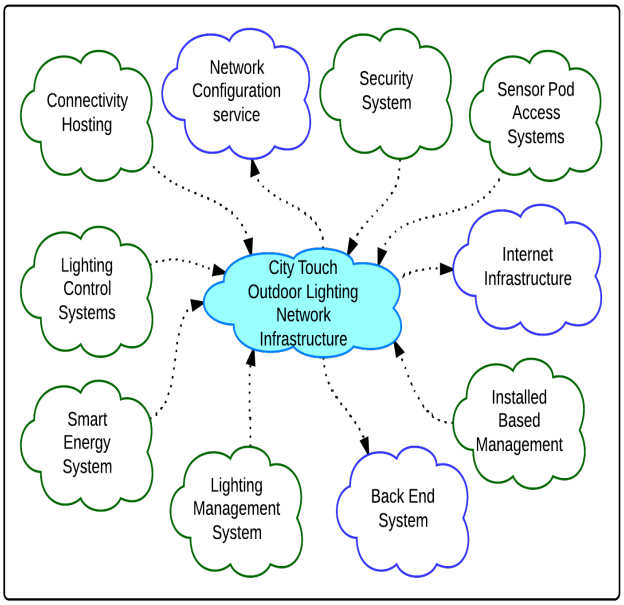


Figure 1-1: The POLAR Outdoor Lighting Context

1.3 Outline

The rest of the document is organized in the following manner. Chapter two discusses stakeholders, their interest, and contribution to the project. Chapter three explains about the problems and scope of the solutions. Chapter four brings a summery of selected approaches for solving the problems defined in earlier chapter. It investigates the feasibility of the approaches in context of this project. Chapter five defines the requirements of the project. Chapter six introduces design options and presents the selected technologies, procedures(methods) and standards.Chapter seven describes the demonstrator software programs developed in this project. Chapter eight presents validation and verification of the artifacts delivered. Finally, chapter nine concludes the findings of the project and points to directions for future studies.

Chapter 2

Stakeholder Analysis

The POLAR Research team of Philips Research owns this project. The team guides the design modifications, technology choices, and implementation of proof of concept. Philips Lighting is the owner of the already developed City Touch architecture and product. It is a direct client of the project since it is expected to use the deliverables of this project as an input for product creation. TU/e (Technical University of Eindhoven) holds a stake through the PDEng (Professional Doctorate in Engineering) trainee who is primarily conducting this project. Hardware/Software suppliers also hold a stake through their products, which are used for experimental, demonstration, and product creation purposes. All of the City Touch instances are set to be deployed in cities and municipalities. Service providers in those cities play an indirect role in the design of outdoor lighting connectivity through their lighting needs. Table 2.1 demonstrates stakeholders and their role in the conception, development, or deployment of the proposed network.

Role Index

Owner (O): An entity who have the proprietary right of the system to be developed.

Direct client (DC): An entity who is the direct user of the system to be developed.

Ultimate client (UC): An entity who is the final user of the outdoor lighting solution. High level requirements arise from this stakeholder.

Partner (P): Entities who do not have a direct contribution to the project but through the products and services that they provide.

Table 2.1: Stakeholders and interests

Stakeholder	Represented by	Interest	Role
Philips Research, Lighting Control systems	POLAR Research Team	Designing and making a demonstration of outdoor lighting connectivity options with proof of concept implementation of usecases.	O
CityTouch Group, Philips Lighting	City Touch Group	Developing and maintaining the current outdoor solution by Philips.	DC
PLS Outdoor, Philips Lighting	PLS Outdoor	Developing innovative outdoor lighting solutions.	DC
TU/e	OOTI	Educating competent professionals who can design and develop complex technological products. Ensure that the project is of high quality and offers sufficient design challenges for the PDEng candidate to prove themselves. Evaluate the knowledge, skill and performance of the PDEng candidate. Assess the potential of the project outcomes as a means for promoting the program towards students and industry.	P
Hardware and Software Suppliers	Redwire(Econotag) ST Microelectronics Contiki	Design and manufacture IEE 802.15.4 enabled devices. Designing boards and computers that are used in outdoor lighting networks. Developing and distributing operating systems for Internet of things. Developing and distributing protocol stack for Internet of things.	P
Municipalities and service providers	Lighting needs	Purchasing and using the final product of outdoor lighting solution.	UC

Chapter 3

Problem Analysis

3.1 Overview

In the POLAR architecture, Nodes create a wireless sensor network (WSN) of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, and ultimately lighting. Nodes cooperatively pass their data through the network to a main location and also receive control commands.

The driving questions that this project needs to answer ascend from this problem. These questions are the following:

- How can an mesh only Node receive bootstrapping information? This includes identifying and stating the necessary and optional steps that should be followed by a device to successfully acquire network information during startup.
- Where and how should the Node get the information from?
- What are the choices available?
  - What protocols are available ?
  - What technologies are provided by stack suppliers?
  - Which of these can be reused?
  - Which of these can be remodeled/redesigned?
  - Which features are not available?
- What are the necessary steps for a Node to become a part of mesh network?
  - are there standards or a previous designs by vendors?

  - How can these standards be adopted to the City Touch architecture?
  - What else can be added for a more robust and secured bootstrapping?

- What is the information a Node needs to receive and in what form?
- How can this process be secured?

3.2 Scope

This project aims to provide a detailed design of the bootstrapping process for POLAR architecture. It describes steps, procedures, and technology choices for a robust and thorough bootstrapping mechanism of Nodes. It mainly strives toward answering the driving questions mentioned in section 3.1.

Security is one of the essential topics in designing any network. However, it is only an optional component of this project. Other projects are being conducted separately to face the security requirement of the POLAR architecture. Thus, this project only focuses on offering an open architecture that could host different security designs.

Though out door lighting connectivity can be achieved by combining different link and network layer technologies such as, wifi, Zigbee, [2, 12, **?**], the design proposed in this document is exclusively based on 802.15.4 [5, 6, 7] and IPv6 [17, 10] technologies. These are chosen to allign POLAR with previous architectures and solutions.

A demonstrator is implemented that follows the high level design guideline (mesh bootstrapping.) However, it can only serve the purpose of proofing a concept. It does not follow the detailed design choices and flow of events described in this document.

The research included in this project does not guarantee an exhaustive list of already existing solutions. Solutions are selected and discussed based on availability.

Chapter 4

Domain Analysis

4.1 Overview of bootstrapping protocols

Bootstrapping can be defined as obtaining network and security resources to successfully become a part of a network. This section discusses procedures proposed by different bootstrapping protocols from different standardization bodies. It includes a set of steps followed by a joining Node in order to join an operational 6LoWPAN. Before going into the details of bootstrapping protocols, it is vital to understand the following technologies listed and discussed in sections 4.1.1-4.16.

4.1.1 IEEE 802.15.4

IEEE 802.15.4 [5, 6, 7] is an Institute of Electrical and Electronics Engineers (IEEE) standard. This standard defines the protocol and interconnection of devices via radio communication in a personal area network (PAN). The standard initially specified two physical layers (PHY), namely, the 868/915 MHZ direct sequence spectrum and the 2450 MHZ direct sequence spectrum. The 2.4GHZ PHY supports 250 kb/s and the 868/915 PHY supports 20 and 40 kb/s respectively. The standard describes the physical layer (PHY) and medium access control (MAC) sublayer specifications for low data rate wireless connectivity with fixed, portable, and moving devices with no battery or very limited battery consumption requirements typically operating in the personal operating space (POS) of up to 1km.

Since the release of the first standard in 2003, IEEE made number of amendments to the 802.15.4 standard. Currently, the standard defines more than 10 PHYs (partially shown in Table 4.1.) When designing and implementing a wireless network using an IEEE 802.15.4 device, the PHY chosen depends on local regulations and user preference about operating frequency.

Table 4.1: IEEE 802.15.4 PHY variations (Partial)

PHY (MHZ)	Frequency band (MHz)	Chip rate(kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)
780	779-787	1000	O-QPSK	250	62.5
780	779-788	1000	MPSK	250	62.5
865/915	868-868.6	300	BPSK	20	20
	902-928	600	BPSK	40	40
865/915 (Optional)	868-868.6	400	ASK	250	12.5
	902-928	1600	ASK	250	50
865/915 (Optional)	868-868.6	400	O-QPSK	100	25
	902-928	1000	O-QPSK	250	62.5
950	950-956		GFSK	100	100
950	950-956	300	BPSK	20	20
2450 DSSS	2400-2483.5	2000	O-QPSK	250	62.5

4.1.2 Internet Protocol Version 6 (IPv6)

IPv6 [17] is the replacement Internet protocol for IPv4. It is a standard developed by the Internet Engineering Task Force (IETF), an organization that develops Internet technologies. The IETF, anticipating the need for more IP addresses, created IPv6 to accommodate the growing number of users and devices accessing the Internet. IPv4 has proven to be robust, easily implemented, and interoperable, and has stood the test of scaling an internetwork to a global utility the size of the Internet. Some of the initial design goals of IPv6 are:

- **Larger Adress Space:** Recent exponential growth of the Internet and the impending exhaustion of the IPv4 address space. IPV6 solves this problem by introducing a 128 bit addressing scheme in contrast to the 32 bit addressing scheme of IPv4. This means IPv6 can allow 2^{128} or approximately $3.4*10^{38}$ addresses, or more than $7.9*10^{28}$ times as many as IPv4.

- **Better Management of Address Space:** It was desired that IPv6 not only include

more addresses, but also a more capable way of dividing the address space and using the bits in each address.

- **Elimination of "Addressing Kludges":** Technologies like NAT are effectively "kludges"that make up for the lack of address space in IPv4. IPv6 eliminates the need for NAT and similar workarounds, allowing every TCP/IP device to have a public address.

- **Requirement for security at the IP level (IPSec)**

- **Quality of Service:** Need for better support for real-time delivery of data, known as quality of service (QoS.)

Besides providing an almost limitless number of unique IP addresses for global end-to-end reachability and scalability, IPv6 has a simplified header format for efficient packet handling, larger payload for increased throughput and transport efficiency, and many interesting feature that makes it preferable to its predecessor.

All the advantages of IPv6 did not come without a few scarifies. Specifically, IPv6 suffers from a relatively high packet header due to bigger source and destination addresses. This becomes a highly degrading feature when the protocol is used in low data rate devices with a smaller MAC packet size such as IEEE 802.15.4. In addition, the Maximum Transmission Unit (MTU) of IPv6 packets (1280 octets) is too large to be carried by MAC payload (127 octets) of 801.15.4 packet. To overcome this drawback, the IETF 6LoWPAN working group designed and proposed an intermediate layer as discussed in the next section.

4.1.3 6LoWPAN

6LoWPAN [10, 16]is an acronym for Ipv6 over Low Power Wireless Personal Area Network. It is an IETF draft that describes transmission of IPv6 packets over IEEE 802.15.4 with the help of an adaptation layer which exist between MAC and the IP network layer (shown in figure 4-1). Furthermore, the 6LoWPAN specification adopts various IPv6 concepts such as Neighbor Discovery and Stateless Auto Configuration with slight modification so that it fits the 802.15.4 specification. In practice, 6LoWPAN stack implementations in embedded devices often implement the 6LoWPAN adaptation layer together with IPv6. Thus, they can alternatively be shown together as part of the network layer. The 6LoWPAN specification describes a fragmentation and reassembly mechanism to successfully transport IPv6 Packets in IEEE 802.15.4 MAC Payload.

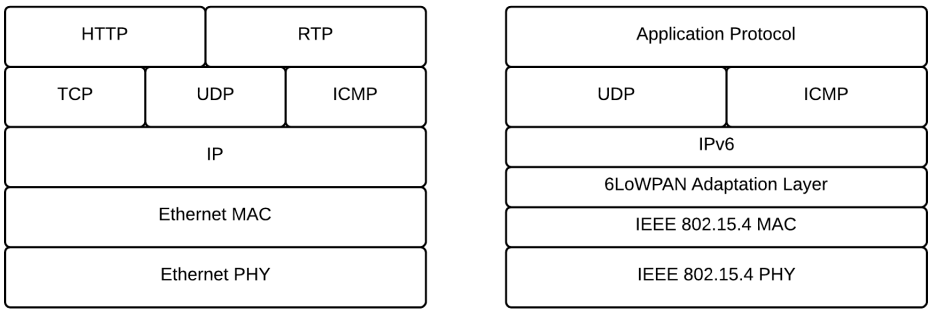


Figure 4-1: IP vs 6LoWPAN Stack

4.1.4 RPL

RPL [18] stands for Routing protocol for lossy and low power networks. RPL is defined in RFC 6550. It is a highly used routing protocol for low power and constrained networks such as 6LoWPAN. The main idea is that Nodes in constrained networks form a Destination Oriented Directed Acyclic Graph (DODAG) to route L3 and above packet to a root. RPL introduces a rank (shown in Figure 4-2) for each Node which denotes a Node's relative position within a DODAG with respect to the DODAG root. RPL defines a new ICMPv6 message with four possible types:

- **DAG Information Object (DIO):** carries information that allows a Node to discover an RPL Instance, learn its configuration parameters, and select DODAG parents.
- **DODAG Information Solicitation (DIS):** used to solicit a DODAG Information Object from a RPL Node.
- **Destination Advertisement Object (DAO):** used to propagate destination information upwards along the DODAG.
- **Destination Advertisement Object-Acknowladgment (DAO-ACK):** Is the acknowladgment message for DAO. The above messages are exchaned as discribed in RFC 6550 to form upward routes towards the DODAG root.

4.1.5 CoAP

Constrained Application Protocol (CoAP) [15] is an application protocol for devices in low power, low bit rate, and high loss rates. CoAP is designed in such a way that it can be used as a constrained web protocol, fulfilling machine-to-machine communication requirements. The

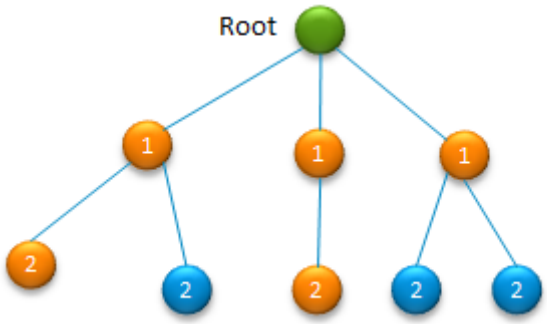


Figure 4-2: RPL DODAG

interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation. Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP.

4.1.6 MLE

Mesh Link Establishment Protocol (MLE) [8] is a link establishment protocol that is mainly used to dynamically configure and secure links in a mesh network. It is also used to transport network-wide changes to end devices. 802.15.4 links can be asymmetric in that a link between neighboring devices may be much more reliable in one direction than in the other. This limits the usefulness of unilateral link quality detection: a link that looks strong to one device may not be usable because it works poorly in the other direction. To avoid wasting effort configuring unusable links, devices can use MLE to send link-local multi-casts containing their local link quality indicator (LQI). Neighboring Nodes can then form an estimate of the two-way quality of their link to the sender.

4.2 Survey on bootstrapping protocols

4.2.1 Commissioning in 6LoWPAN

This document [9] releases a proposed 6LoWPAN bootstrapping protocol. The proposed draft starts with a core definition of bootstrapping as a process by which a LoWPAN Bootstrapping

Device (LBD) collects a LoWPAN Information Base (LIB) message from a LoWPAN Bootstrapping Server (LBS.) As depicted in Figure 4-3 and 4-4 a joining Node starts by scanning its radio range looking for an existing LoWPAN network. This is followed by broadcasting a LoWPAN Bootstrapping Agent (LBA) solicitation message. Based on the LBA solicitation message, all LBAs and LBSs that are one hop reachable to the LBD reply by sending a unicast LBA advertisement message addressed to the LBD. At this point the LBD have a list of devices which it can use to bootstrap. Thus, it will choose one of the LBSs (if there is any in the range) or LBAs. However, the document does not specify or recommend a guideline to choose the finest LBA or LBS. After selecting the LBA, the LBD sends a unicast Join Request (JR) to the chosen LBA. If the LoWPAN is an open network, then the LBA does two things(demonstrated in Figure 4-3):

- Reply with Join Reply (JRep) message, which contains all PAN-Specific Information (PSI). This information may include PANID, Server Address, and Prefix.
- Forward the JR to the LBS for Device-specific Information (DSI.)

The LBD now receives the JRep, configures its PSI, and waits for the server to send the JRep containing the DSI. In this case (Open LoWPAN) the LBS will ultimately sends back the JRep. It is then forwarded via the LBA to the LBD. The LBD can now configure its DSI and the bootstrapping procedure is completed. However, if the LoWPAN is a closed/secured (demonstrated in Figure 4-4), then the LBA must advice the LBS for an authentication, in order to send the PSI to the LBD. Thus, in a closed/secured LoWPAN, the LBA always forwards the JR to the LBS. Subsequently, the LBS transform the JR message to a valid authentication request and forward it to an EAP authentication server [1]. (LBD must encapsulate its authentication credentials in a JR.) When the authentication server indicates the success of the authentication, the LBS sends all network resources (PSI, DSI, Encryption key, etc.) along with the ACCEPTED code. In the case of failure in the authentication process, a DECLINE code is sent to the LBD. Upon the successful arrival of the JRep with an ACCEPTED code and all necessary network resources, the bootstrapping process is completed.

4.2.2 ZigBee IP (ZIP) bootstrapping protocol

The ZigBee IP bootstrapping protocol [2] is perhaps the most explicitly explained procedure. It is composed of 12 consecutive steps, shown in Figure 4-5, that specify a list of actions that

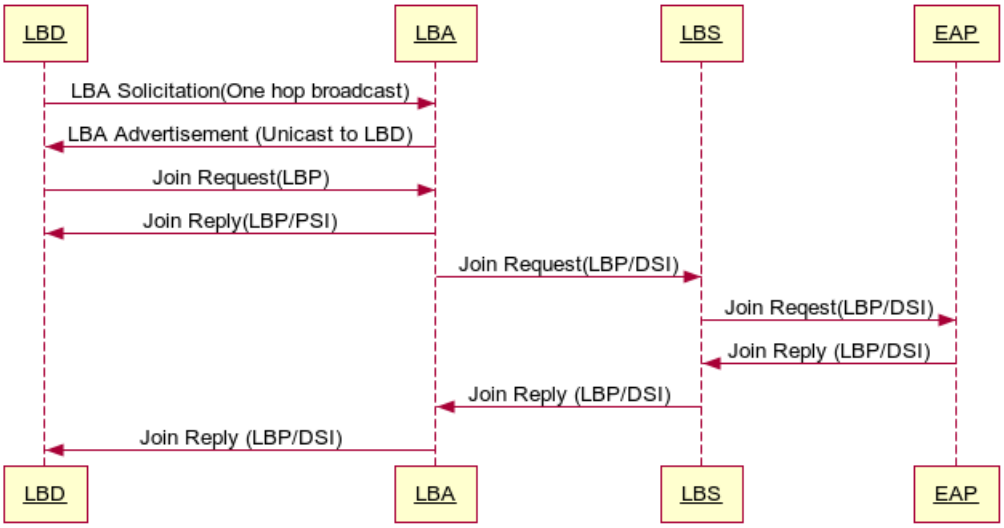


Figure 4-3: Commissionig in 6LoWPAN Bootstrapping: Open PAN

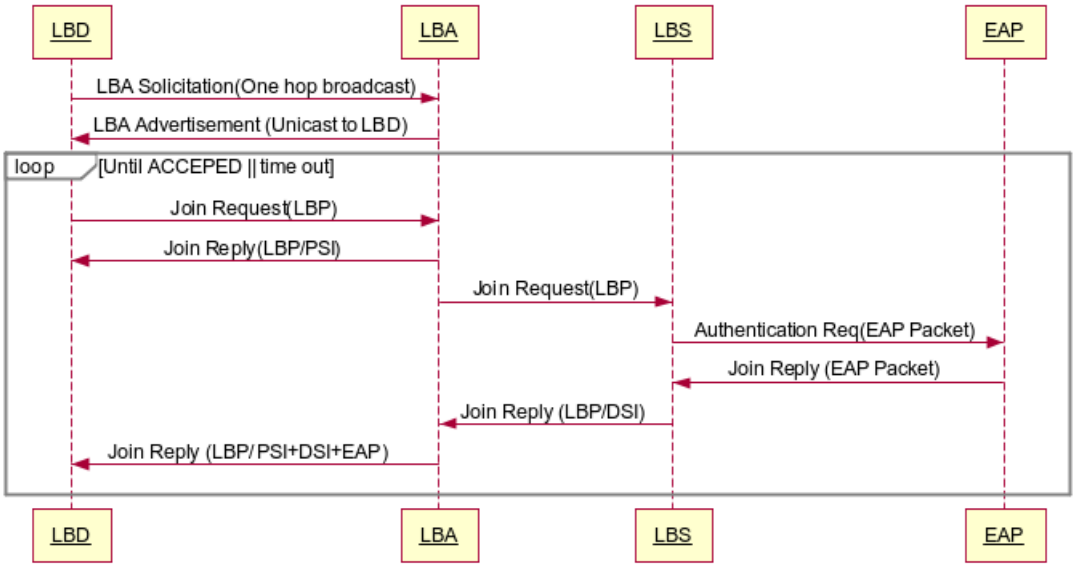


Figure 4-4: Commissionig in 6LoWPAN Bootstrapping: Secured PAN

should be taken by a joining device in order to join an operational PAN. These steps are presented and discussed below.

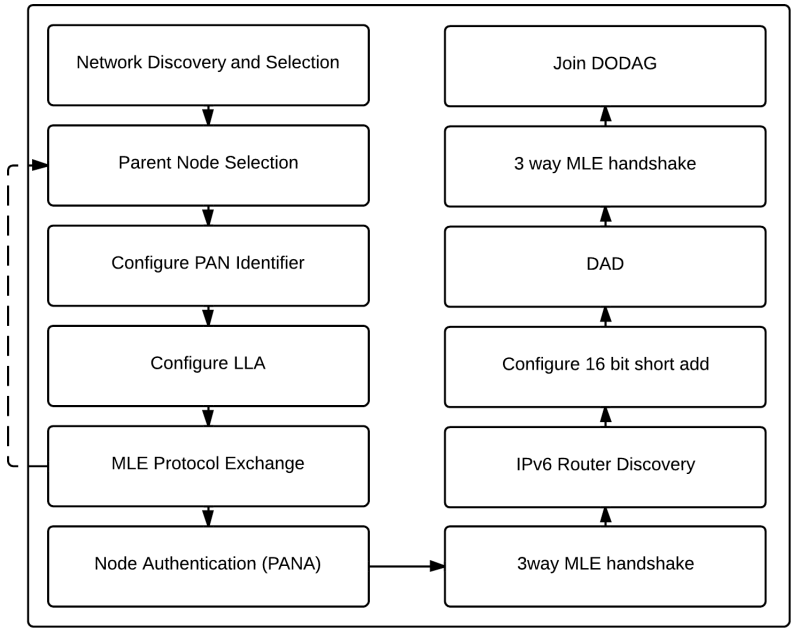


Figure 4-5: ZIP Bootstrapping

1. Network Discovery and Selection: This is a phase where a joining device scans for available ZIP networks and chooses which ZIP to join. It has two sub-phases in it.

   (a) Network Discovery: the joining device broadcasts a beacon request and collects all beacon frames sent by ZIP routers or coordinators.

   (b) Network Selection: when there is more than one ZIP network or even other IEEE 802.15.4 based networks, the joining device should choose which network it joins. This basically can be done in application specific means, varying on the implementation. One way of implementing a selection mechanism is by preconfiguring the desired ZIP parameters to the joining device. When a joining device receives a beacon frame, it only selects the originating network only if the information in the beacon checks out with its preconfigured parameters.

2. Parent Node Selection: Once the desired ZIP network is chosen, the next step is choosing the appropriate parent to use for bootstrapping. This is done by selecting a Node with the best Link Quality Indicator (LQI) and ability to accept a child Node.

3. Configuring PAN Identifier: The joining Node configures its 802.15.4 MAC PAN identifier

to that of the selected target network.

4. Configuring Link Local Address (LLA64): The Node configures an IPv6 link local address for its 802.15.4 interface using the LL64 address format.

5. Configuring MAC Polling (Optional): If the joining Node is a sleepy Node, it must use Mesh Link Establishment (MLE) protocol exchange. This is to inform the parent router that it is indeed a sleepy Node and uses the MAC Polling feature for Layer-2 packet transmissions. If the parent could not accept a sleeping Node, then the joining Node should select another parent (step B.)

6. Authentication: As security is a big aspect of bootstrapping, the joining Node is authenticated using PANA[1] [3].

7. Synchronize Packet Counter: The joining Node performs a 3-way secured MLE handshake to synchronize frame counters with the parent router.

8. IPv6 Router Discovery: At this phase, the joining Node is authenticated and its packet counter is synchronized with that of the parent router. The next step is finding an IPv6 router for packet routing to and from the coordinator. At this phase, the Node also receives the network prefix of the ZIP network. This is done by using the IPv6 Router discovery protocol as it is described in RFC 6775 [18].

9. Short Address Generation: The joining Node generates its own 16-bit short address. The generated addresses must not be one of the reserved values (e.g., 0xfffe.)

10. Duplicate Address Detection (DAD): As the joining Node generates its own 16-bit short address, it runs DAD to guarantee that the generated value has not already been taken by another Node. DAD is performed as it is described in RFC 6775.

11. Short Address Exchange: The joining Node and its parent exchanges their 16-bit short addresses using a 3-way MLE [8] handshake.

12. Join DODAG: The joining Node now is a part of the ZIP network. It can join the DODAG. If it is a host, then its parent must send the RPL DAO message to the DODAG root. This creates a downward root. (RPL is discussed in Section 6.1.4)

[1]Protocol for Carrying Authentication for Network Access

4.2.3 ZigBee association procedure

The ZigBee [12] association procedure describes a mechanism where a new device joins a ZigBee network. In order to join a pre-existing network, a joining device passes through the following phases:

1. Network Scanning: The joining Node performs an active/passive scan.

2. Beacon broadcast by router/coordinator: The coordinator broadcasts a beacon frame both periodically and by request from a joining device. The beacon frame includes information about PANID, Address of coordinator, extended PANID (optional), Allow Join, Router- Capacity, Host-Capacity, and Device Depth.

3. Association Request: Following the receipt of a beacon frame, the joining Node sends an association request command to a specific PANID or to any PAN by setting PANID Value 0x FFFF. This is followed by acknowledgment from the parent.

4. Association response: If the association request is accepted (if the Node is validated to be authentic.) then parent includes a 16-bit short address for the joining Node. This message is also followed by an acknowledgment.

4.3 Summary

When a Node joins a wireless sensor network, it passes through a series of steps before it is functional in the network.The seven selected procedures presented in sections 4.2.1-4.2.3 depicts a framework template for the Bootstrapping process. The process starts with discovering available wireless networks and ends with final stage of registration and becoming a part of the routing tree. These steps are:

1. Pre-deployment Configuration: Pre-installation and deployment configuration that enables the joining Node to have initial parameters. Example methods include factory configuration where initial parameters are burned into the Node during manufacturing, or administrative configuration, where those parameters are injected before installation.

2. Network Discovery: Detecting available wireless networks that are in a radio frequency range. It can be carried out by using a beacon broadcast from network coordinators and routers.

3. Network Selection: From the available wireless network discovered in the previous step, a Node has to select one as a desired one. If a beacon broadcast is used in earlier steps, the information contained in the beacon may be used to select the desired network.

4. Agent Selection:This step is to select a viable bootstrapping agent. When a Node is more than one hop away from a Gateway Node, It selects a bootstrapping agent that can facilitate the process. The Link Quality index and the depth of the candidate in routing tree are some of the parameters that may play a role in selecting a bootstrapping agent.

5. Authentication: As security is an enormous concern in a wireless network, a joining Node must be authenticated by a legitimate entity and receive security credentials. DTLS handshake and PANA relay are some of the protocols widely used for this purpose.

6. Acquiring Network Parameters: Assuming the authentication process is successful, the next step is to receive network parameters. The entity which is pushing those parameters and the type of parameters to be pushed vary among protocols. The parameters are stored and initialized.

7. Forming routing path: Finally a joining Node should be a part of routing tree for an end-to-end communication with other Nodes and a Back-end server.

Table 4.2 depicts a summary of the mechanisms used by the selected seven bootstrapping procedures.

4.3.1 Comparison of selected bootstrapping procedures

I also made a comparison of bootstrapping protocols based on criteria that are desirable to the development of POLAR architecture. These criteria are.

- Support for mesh bootstrapping: bootstrapping via gateway and routers.

- Support for third party AAA server : support for Third party authentication servers that may or may not be in the same location as the Back-end.

- Support for DTLS: DTLS being the preferred authentication mechanism, does the procedure have open architecture for DTLS?

- Distinction between phases: is there a clear distinction between phases of bootstrapping? This might be useful to combine different phases from different procedures.

- Agility:

    1. Re-association: the ability of an already joined Node to re-associate when needed (example, link broken, re-authentication is needed.)
    2. Changing gateway: what happens when a gateway (coordinator) is changed?

- Plug and play after pre-installation configuration: does it provide PnP after pre-installation configuration. (No human interaction during deployment)

- Security: Does it describe different levels of security (L2, upper layer)?

- Availability of implementation: Is there available implementation? Open or closed source? Full or partial?

- State of Standardization: is the protocol standardized

Table 4.1 illustrates the comparison of selected bootstrapping protocols based on the criteria mentioned above. Color codes are used for comparison. Green, Orange, and Red refers to fully supported, partially supported or expected, and not supported respectively. For the sake of the size of the table, letters from A-G are used to represent the bootstrapping procedures in Sections 4.2.1-4.2.3 in an orderly manner. Descriptions about the comparison are not included in the table.

Table 4.2: Comparision of bootstrapping protocols

Criteria	**Preference**	A	B	C
Support for mesh bootstrapping	Mandatory			
Support for third party AAA server	Preferred			
Support for DTLS	Encouraged			
Distinction between phases	Nice to have			
Re- Association	Preferred			
Changing Gateway	Mandatory			
PnP after pre-installation configuration	Mandatory			
Security	Mandatory			
Availability of implementation	Encouraged			
Standardization	Nice to have			

Fully Supported | Partially Supported or Expected | Not Supported

4.3.2 Conclusion

Our summery concludes by pointing out the following observational facts:

- The POLAR bootstrapping procedure supports the least amount of criteria. This indicates the need for re-designing the procedure for future (POLAR) Architecture.
- Zigbee IP provide support for most of the mandatory criteria.
- Zigbee IP are considered a better bootstrapping procedure due to the fact that they provides support for eight of the eleven criteria. Zigbee follows with seven.

Table 4.3: Summary of bootstrapping protocols

Step	**Commissioning in 6LoWPAN**	**ZIP**	**Zigbee**
Pre-deployment Configuration	Unspecified	Administrative configuration	Administrative configuration
Network Discovery	Beaconing	Beaconing	Beaconing
Network Selection	Unspecified	Application specific means	Application specific means
Agent Selection	LBS first otherwise LBA	LQI	Device depth
Authentication	EAP	PANA	Trust center
Acquiring Network Parameters	LBP message	IPv6 RD	Unspecified
Configuration	Unspecified	IPv6 RD	Unspecified
Forming Routing Path	RPL	RPL	Unspecified

Chapter 5

Requirements Analysis

5.1 Customer objectives

This section discusses the project goals and customer's expectation towards the project achievement. Six fundamental customer objectives are identified and discussed. They are also used to determine requirements of the proposed design. Customer objectives are stated below.

- Low-Cost Network: The research team wants to develop an alternative low-cost network to the POLAR network setup.
- Rich Functionality: Even though the proposed network is a low-cost alternative, it should support fundamental networking functionalities and features that are required in outdoor lighting connectivity.
- Ease of Setup: The network is required to be easy-to-setup with minimal configuration effort.
- Ease of Integration and Deployment:
  - Integration with existing infrastructure: as Philips owns a wide range of network infrastructure and services, the proposed network is anticipated to be integrated with the existing network infrastructure and services.
  - Integration with existing demo: the POLAR network setup is currently implemented as a proof of concept; the demonstrator for the proposed network is foreseen to be able to integrate with the existing implementation.
- Sustainability: Outdoor lighting connectivity is exposed to a wide variety of failures, dynamically changing deployment environment, and high chances of upgradability. The

proposed network is expected to handle these failures, changes, and upgrades with minimal human interaction.

- Security: The proposed network is expected to only sustain an authorized, authenticated, and accounted communication.

Clusters of requirements and design choices are generated from the above mentioned customer objectives. Figure 5-1 demonstrates how fundamental requirements of the proposed network are derived from customer objectives.

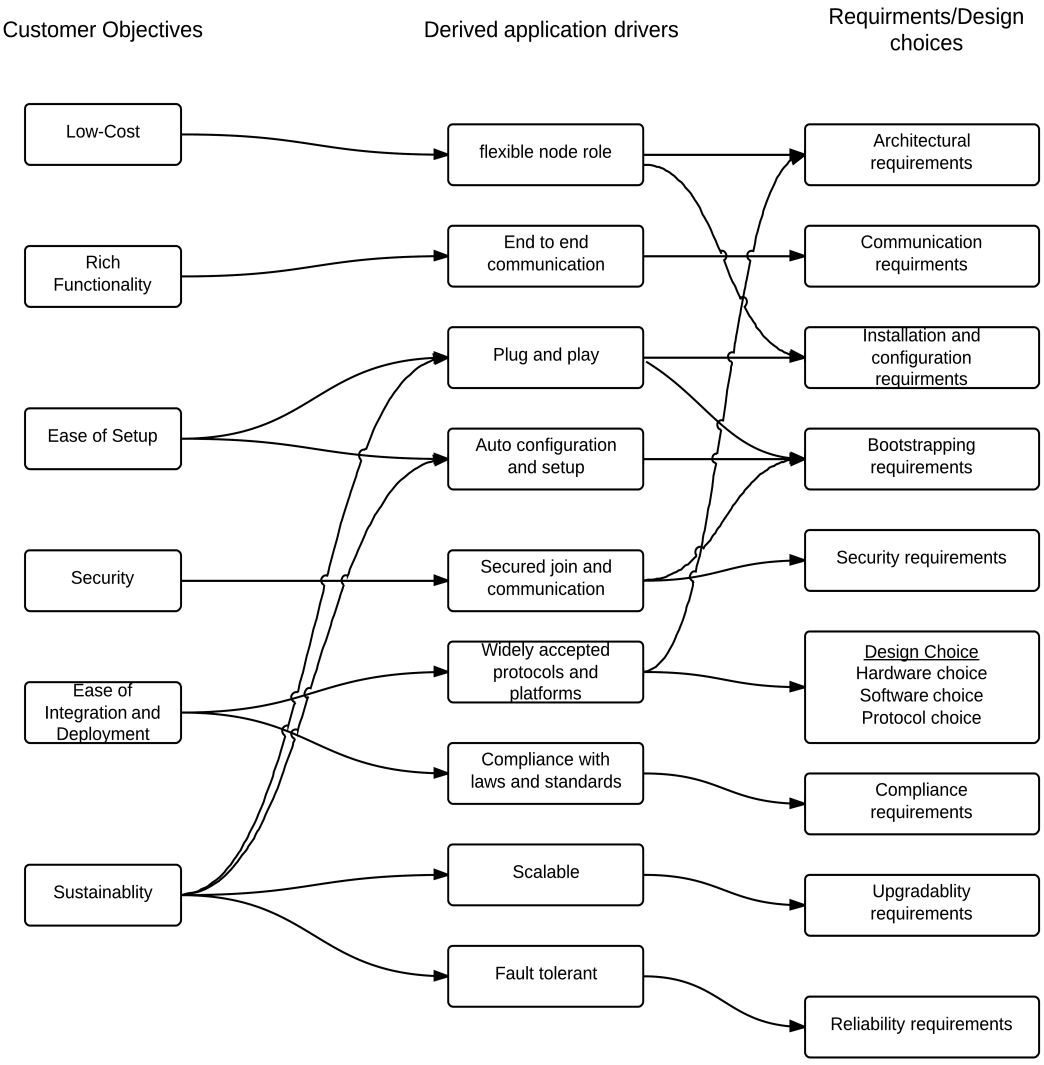


Figure 5-1: Customer Objectives and Requirements

5.2 Requirements

As discussed in the previous section, clusters of requirements are identified. These requirements guide the design goal of the POLAR architecture. Not all of these requirements are necessarily related to the bootstrapping module. They are obtained from POLAR team requirement list and discussed here to give a thorough understanding of the POLAR network objective.

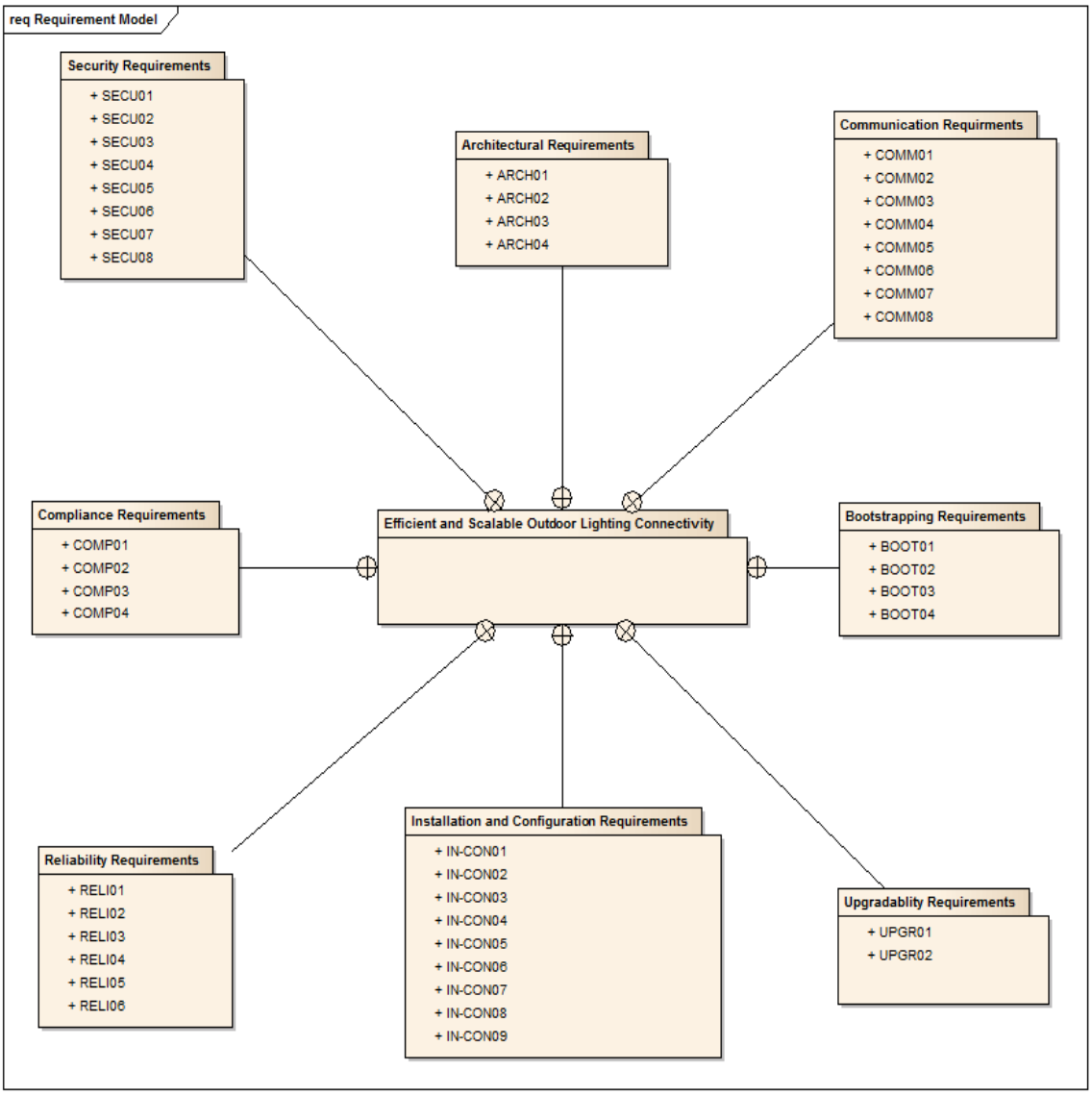


Figure 5-2: Requirement Overview

The requirements are :

- Architectural requirements Requirements that mainly specify the ability of hardware in the network and their execution environment.
- Bootstrapping requirements Requirements that state the necessities of a Node to be able to join a network.

- Communication requirements Requirements that guide the type of communication technologies that the network needs.
- Compliance requirements Requirements that suggest what kind of legal and technical compliance the network should follow.
- Installation and configuration requirements Requirements that set list of features that the network should have to start up.
- Reliability requirements Requirements that determine the robustness of the network.
- Security requirements Requirements that are revealing the security needs of the network. This includes the specific type of authentication mechanism and which communication should be secured.
- Upgradability requirements Requirements that describe the scalable nature of the network.The scalablity does not only refer to the growth in number but also the expansion in technology use (hardware and software).

The above list presents requirements for the entire POLAR domain. However, specific requirements of this project are those which describes conditions for commissioning and bootstrapping of Nodes. figure 5-3 demonstrates sets of requirements that are relevant in the design of bootstrapping mechanism for POLAR. As security generally is an optional concern for this project, the figure presents two requirement sets, namely bootstrapping requirements and secured bootstrapping requirements. The dotted arrows shows the dependency and the dotted triangular arrows represents the realization.

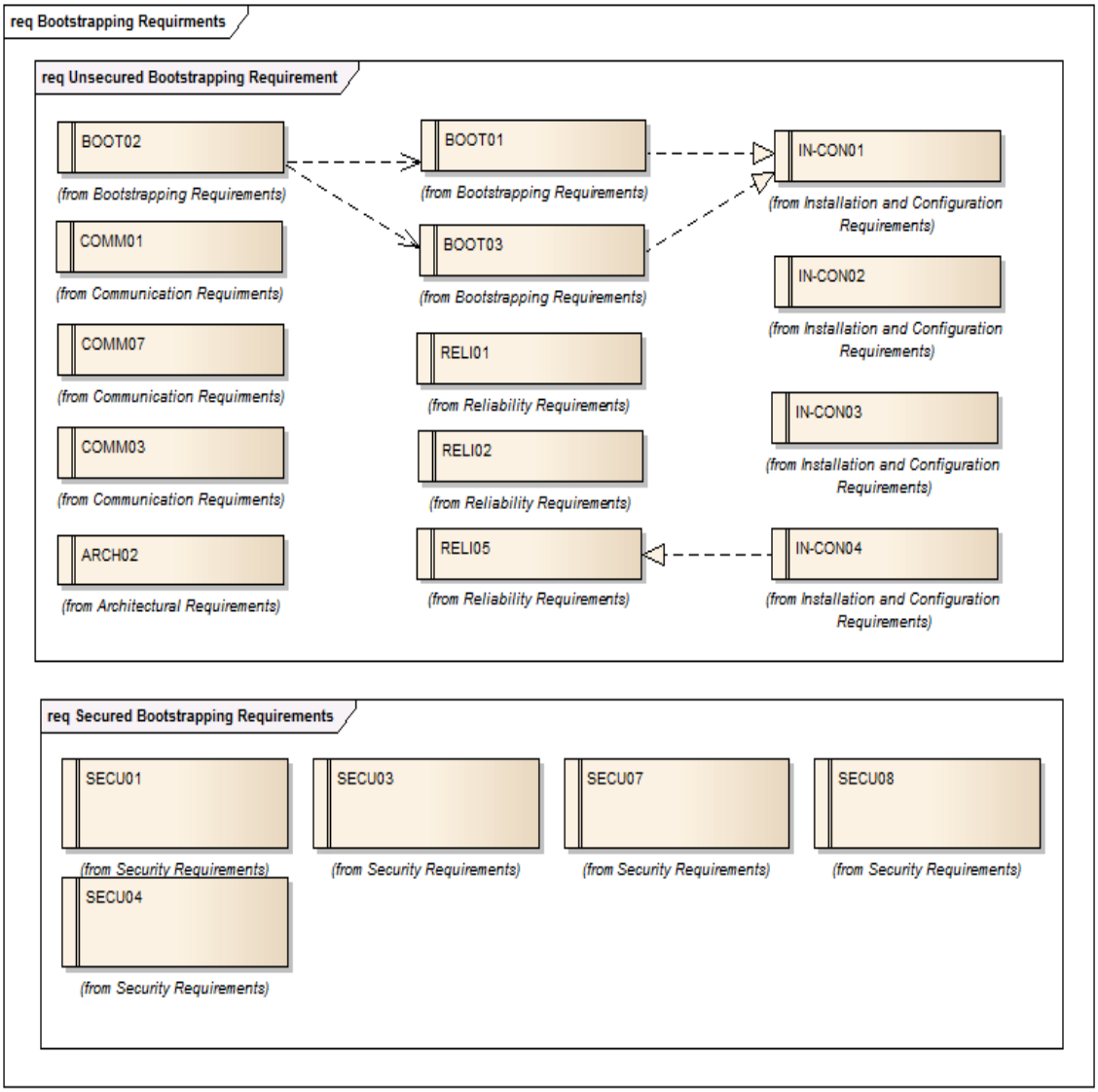


Figure 5-3: Bootstrapping Requirements

5.3 Usecases

As observed in the survey, a bootstrapping process is a collection of interdependent sets of message exchanges and configurations between a joining device and (set of) devices that are used for authenticating and supplying a network parameter. Although, bootstrapping can be regarded as a single usecase that describes an interaction of devices to allow/reject joining of a new device, the procedure slightly differs depending on the joining node's position with respect to the Gateway Node.That is, if a Gateway Node is one-hop reachable from a joining Node, then the bootstrapping procedure does not require a relay or proxy mechanism for requesting and receiving desired parameters. In contrast, when the joining Node is more than one-hop away from the Gateway Node, there must be a mechanism to transport the request and response to

and from the Gateway Node. Tables 5.1 and 5.2 depicts the above two scenarios.

Table 5.1: Single-hop bootstrapping

Usecase ID	001		
Usecase Name	Single-hop Bootstrap		
Created By:	Surafel	Last Updated By	Surafel
Date Created	January 23, 2014	Date Last Updated	May 5, 2014
Actor:	Node As JN, Node as Gateway Node		
Description	A Node joins an already existing PAN		
Precondition:	Gateway router is located in the Radio range of JN JN is a 6LoWPAN enabled device, Gateway is a 6LoWPAN FFD JN have all the required credentials and configurations to bootstrap		
Post Conditions:	JN is a part of the PAN. It can send IP packets to other Nodes and receive the once addressed to it.		
Course of Events:	1. Node: Discovers the desired PAN and its Gateway 2. Node: Request network parameters 3. Gateway: Sends network parameters 4. JN: Recieve parameter and configure itself 5. JN: Join a routing tree		
Alternative flow	Security 2a. Node request security credentials 2b. Gateway Authenticates JN (with BE or Locally) 2c. Gateway sends security and network parameters Gateway dose not accept new Node 2a. Gateway: reply with not accepted message 2b: JN: Go to step 1 and select another PAN or giveup after some time		
Exception	The flow of events does not apply when a Gateway Node bootstraps		
Special Requirements	6LoWPAN ND, RPL, CoAP and other protocols may be used		
Notes and Issues	This usecase is performed every time a new Node is joining the PAN		

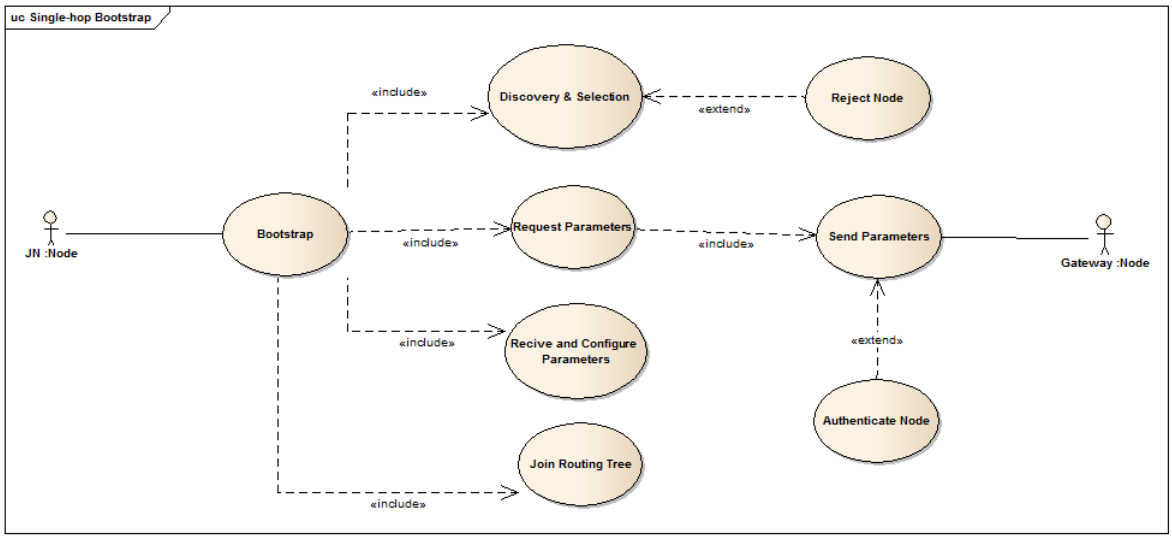


Figure 5-4: Single-hop Bootstrap

Table 5.2: Multi-hop bootstrapping

Usecase ID	002		
Usecase Name	Multi-hop Bootstrap		
Created By:	Surafel	Last Updated By	Surafel
Date Created	January 23, 2014	Date Last Updated	May 5, 2014
Actor:	Node As JN, Node as Gateway Node, Node as BA		
Description	A Node Joins an already existing PAN		
Precondition:	Gateway router is not located in the Radio range of JN JN is a 6LoWPAN enabled device, Gateway is a 6LoWPAN FFD BA is a 6LoWPAN FFD BA is located in the Radio range of JN JN have all the required credentials and configurations to bootstrap		
Post Conditions:	JN is a part of the PAN. It can send IP packets to other Nodes and receive the once addressed to it.		
Course of Events:	1. Node: Discovers the desired PAN and list of potential BAs 2. selects a BA from the list 3. Node: Request network parameters 4. BA: Forwards the request to Gateway 5. Gateway: Sends network parameters to BA 6. BA: Receives network parameters and forwards it to JN 7. JN: Receive parameter and configure itself 8. JN: Join a routing tree		
Alternative flow	Security 3a. Node request security credentials 3b. Gateway Authenticates JN (with BE or Locally) 3c. Gateway sends security and network parameters via BA Gateway dose not accept new Node 3a. Gateway: Reply with not accepted message 3b: JN: Go to step 1 and select another PAN or give up after some time BA does not accept new device 3a. BA: Reply not accepted message 3b. JN: Go to step 2 and select another BA or give up after sometime		
Exception	The flow of events does not apply when a Gateway Node bootstraps		
Special Requirements	6LoWPAN ND, RPL, CoAP and other protocols may be used		
Notes and Issues	This usecase is performed every time a new Node is joining the PAN		

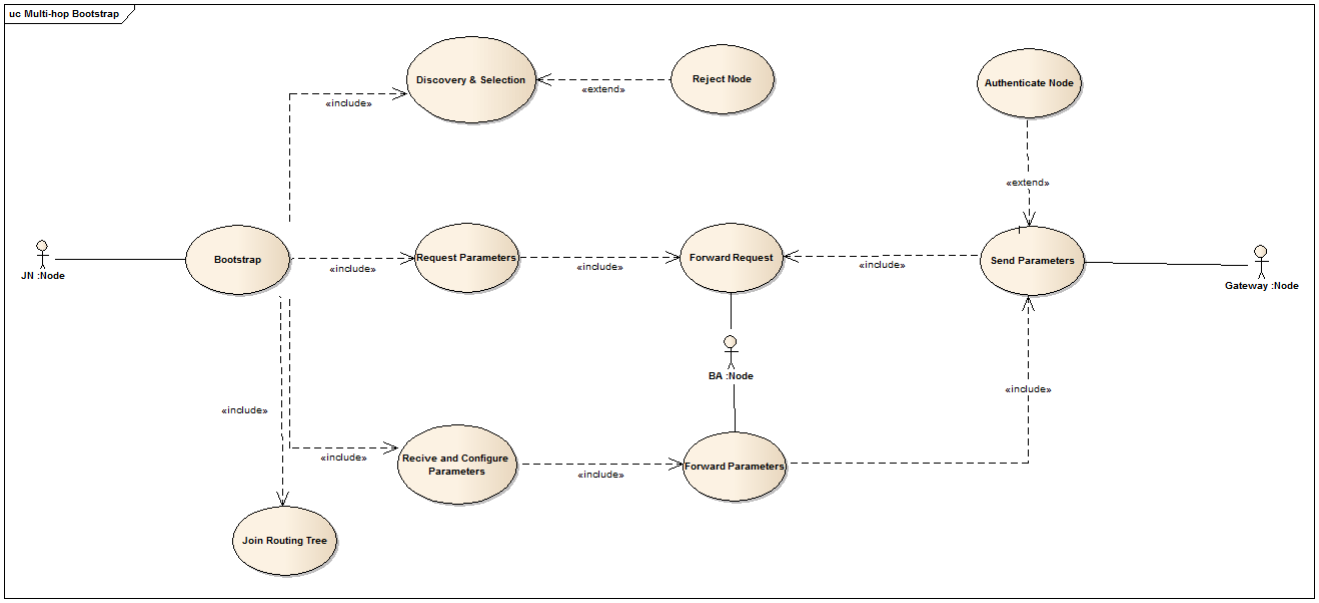


Figure 5-5: Multi-hop Bootstrap

Figures 5-4 and 5-5 diagrammatically represents the two use case scenarios discussed in the Tables 5.1 and 5.2.

Chapter 6

Design

6.1 Overview

A bootstrapping protocol for POLAR follows the same design principles of the bootstrapping protocols discussed in Chapter 4. Furthermore, the POLAR imposes certain specific guidelines that govern the design procedure. These specific guidelines are directly inherited from requirements of the system and preference of Philips Research. Basically, POLAR is a 6LoWPAN with a Back-end system controlling the operation and configuration of the network. Therefore, the bootstrapping procedure of the system is a custom-designed join procedure for an outdoor lighting connectivity network of 6LoWPAN.

6.2 Design Options

Bootstrapping can be described as obtaining network and security resources to successfully become a part of a functional PAN. An already bootstrapped Node can send and receive packets addressed to it or to the group it belongs to, successfully decode them, and interpret the content of the packet that it received and correctly encode messages before sending so that receiver(s) with the right security resources can unravel the content. Additionally, depending on the Node's role in the PAN, it may route packets.

Although bootstrapping is a single usecase in a wireless PAN, as discussed in Section 4.2 there are sequence of events and message exchanges between a JN and other Nodes which have already joined the PAN. In almost all of the existing bootstrapping protocols, these message exchanges are designed in a modular /step-by-step manner. This is particularly handy to construct a newer protocol by using some of the existing steps. The design of POLAR boot-

strapping protocol covers all the necessary steps for a new Node to join an existing wireless PAN. Thus, the design specifies the technology choices, message exchanges, and its contents for all steps except the authentication procedure.

Generally, a new Node passes through states that enables it to choose an active and desired PAN in a designated radio channel, select a bootstrapping agent, optionally authenticate itself to a security server, collect necessary resources, and join a routing graph (DODAG). In this project, we search and compare efficient technologies available for each step and chose the methodologies that are thorough and provide maximum flexibility for redesign. The chosen technologies are further explained in the following sections and customized to best fit the overall POLAR architecture and requirements.

6.2.1 Pre-deployment configuration

This is the first step where all parameters that describe the identity of the JN are configured to the Node. Initial values for pre-bootstrapping parameters such as join key and MAC address can sometimes be provided by a manufacturer or it is also possible to configure them administratively on/off site before deployment. The life path of an IEEE 802.15.4 is depicted in Figure 6.1. Administrative configuration refers to any configuration of nodes after leaving the manufacturer to the point it is deployed. Table 6.1 demonstrates the three options for the pre-deployment configuration phase. One can clearly see that a mixed method has a better potential than the other two. This is because it gives designers a better flexibility to choose which of the parameters should be configured locally at Philips and which should come configured from a manufacturer. Furthermore it does not require a special order from manufacturer (e.g. all 802.15.4 RF chips have a pre-configured unique MAC address)

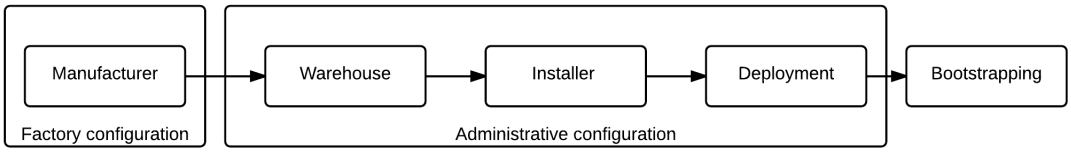


Figure 6-1: Life Path of POLAR Node

Table 6.1: Pre-deployment configuration options

Method	Pro	Con
Factory Configuration	Less/no configuration at Philips.	Not flexible, Philips may want to configure some parameters locally.
	Well-constructed parameters are already pre-configured upon receiving the Node from manufacturer.	Manufacturer may not configure all the necessary parameters.
	Presumably globally unique MAC address is constructed from EUI-64	May cost extra to order custom configured Nodes.
Administrative configuration	Flexible, Philips or an installer can configure localized parameters and keys.	Extra work for generation and configuration of pre-deployment parameters. Uniqueness of parameters can only be guaranteed locally, i.e., in Philips owned devices.
Mixed	Philips can take advantage of already pre-configured parameters without extra cost and configure other local parameters on/off site. Additionally, all the pros from the first two methods apply in this method.	Extra work for generation and configuration of local parameters.

6.2.2 Network discovery

The network discovery phase is a period at which the JN scans through channels looking for transmitted packets to discover available PANs. The prioritized list of channels can be configured to the JN at an earlier phase. The IEEE 802.15.4 MAC specification already describes mechanisms for network discovery. The network discovery is mainly based on beacon frames broadcasted by routers and a PAN coordinator. However, there are various kinds of beacon broadcasts depending on how and when these frames are broadcasted. In triggered beacon mode, beacon frames are only broadcasted by routing devices following a command from the PAN coordinator. The other variation is a beacon frame is broadcasted periodically and on request . Thus, the JN can receive this frame by passively listening to or actively asking for it. In some cases, the JN can also be pre-configured with the initial information of the desired network. Thus, the IEEE 802.15.4 network discovery mechanism is only mandatory if JN does not have a prior knowledge of the available or desired network. Table 6.2 compares the different

network discovery mechanisms. In practice, Passive scan is mostly enabled because it is also a means for Nodes to get information about the PAN throughout the lifetime. Accordingly, by combining a passive then an active scan, a better performance can be achieved. This means a Node first passively listens for beacon frames for certain amount of time then when time expires it can start actively requesting. As Figure 6.2 depicts, the IEEE 802.15.4 specification defines a MAC beacon frame that can be used in network discovery. The beacon payload may be designed to accommodate all the necessary information needed for network discovery.

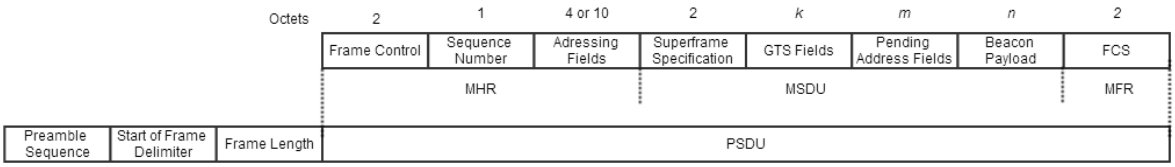


Figure 6-2: IEEE 802.15.4 MAC Beacon Frame
k,m,n = variable length

Table 6.2: Network discovery options

Method	Pro	con
Active beacon scan	No unnecessary beacon broadcasts. BA only responds to beacon solicitation. No broadcast is needed. Avoids periodic Advertisement.	Might create congestion, when a large number of Nodes are trying to join the PAN at the same time.
Passive beacon scan	Passive scan is always enabled to get information about the network. Less network trafic, speciially if there are large number of Nodes trying to join the PAN.	A Node may wait for a long time depending of the MAC superframe structure, before getting a beacon from desired network.
Triggered Beacon Mode	Very few beacon broadcasts. BA only broadcasts when it is triggered from Back-end	Human administration (or schedule) is needed to trigger on and off
Administrative Configuration	No need of beacon solicitation and advertisement. Avoids periodic advertisement.	Desired network must be known at first. Re-configuration is needed for different deployment in different network.
Passive-Active scan.	Takes advantage of the periodic beacon advertisement as well as active beacon scan Node does not have to wait for long period of time to receive beacon frames since it can request based on its own timer.	JN spends certain amount of time before actively requesting beacon frames.

6.2.3 Network selection

Once a Node discover a set of PANs, it has to make a selection. The selected PAN becomes the network on which the JN bootstraps to. Assuming that there could be a number of PANs available in the personal operating space (POS) of JN, the selection should filter out all the PANs that are not desired. Basically network selection is done by using the information collected in the previous phases. This information is found in the payload of a beacon frame that has been collected in the network discovery phase or a pre-configured (predefined selection) by using a PAN ID of the desired network which is flushed into the JN during pre-deployment configuration. Table 6.3 lists and compares the selection mechanisms. Automatic selection is a method where a JN makes a decision based on the beacon payload. The content of the payload is described in the detailed design. User selection is a means where a human operator manually makes a decision after discovering the entire available network. In some cases, the owner of the PAN may want to include proprietary tokens in the beacon payload to mark the Identity of PAN. In such cases a selection mechanism could use the presence of those tokens to choose the desired PAN.

Table 6.3: Network selection options

Method	Pro	Con
Predefined selection	Simple.	Prior knowledge of network is needed. Nodes must be reconfigured to be deployed in different network.
Automatic selection based on beacon payload	No further information is needed. Selection is made on the spot. Thus, no static configuration on the Node. No human or other commissioning device interaction except for the JN itself.	Malicious devices may broadcast forged beacon payload, hence JN could be tricked. Extra security is needed for a better confidence.
User selection	Better level of confidence since an operator is making the choice. No extra information in the beacon frame. No prior knowledge is needed by the joining Node.	Needs human supervision. Malicious devices may trick the human operator.
Automatic selection base on Proprietary tokens	Improved confidence becasue of the proprietary tokens that can be checked by JN. Can be used in combination with other information in the beacon payload.	Could increase the size of the beacon frame. Cryptographic algorithm is needed for generation and verification of tokens.

6.2.4 Parent selection

In multi-hop bootstrapping, selecting the PAN is not the only selection a JN has to make. It has to also select an FFD device (Full Functioning Device, a Node with routing ability), which can serve as a bootstrapping agent. During the network discovery and selection phases, a JN has gathered information about each routing device that are one-hop reachable. JN must select one of these devices as a bootstrapping agent. The rank or the link quality can be used to make a selection. Table 6.5 shows which parameters should be considered for this selection.

Table 6.4: Parent selection options

Method	Pro	Con
LQI	Gives a parent with the best link quality.	A parent may have the best link quality but might be far from Gateway.
Rank	Gives the parent that is close to the Gateway. RPL Ranks can be encapsulated on beacon payload. Therefore, no further calculation is needed.	Multiple parents with the same rank could exist. A parent could have a high rank but with broken or very low strength link.
Predefined	Simple	As it is predefined, it does not have the capacity to select the best parent dynamically. The pre-configured parent might have a broken link or be dysfunctional by the time JN wants to bootstrap.
Combined	Better confidence in finding a "better"parent. This is because it considers both link quality and rank. It can be tunned in different ways (one can set minimum acceptable link quality or maximum rank).	Might cause a process overhead since it has to compare rank and link quality for all available routers.

6.2.5 Authentication

Authentication mechanism is a topic that is being investigated and designed by security experts of the POLAR team. Hence, this document only lists the options in consideration. The options are: EAP [1], PANA [3], and DTLS [11, 13, 14]

6.2.6 Bootstrapping request

At this phase a JN Node is authenticated and has acquired all the security credentials needed. Therefore, it can now request network credentials from a coordinator. In all of the POLAR variations, these credentials arise from a cloud-based Back-end system. By definition, acquiring

these network resources enables the JN to become a part of the PAN. These parameters are listed in Table 6.5. The methods available for this phase are listed as follows:

Table 6.5: Bootstrapping parameters

Parameter	Meaning
isParent: bool	A boolean value indicating if the JN acts as a Router Node
serverAdress	IPv6 address of the Back-end server
serverPort:	Port number of the Back-end server
maxWaitTime	The amount of time in millisecond to wait before retransmission in case of failure.
maxRetries	The number of tries before giving up
updateInterval	The time interval Nodes should wait before asking for network information updates from Back-end server
IPv6 Address	A gloabally routable IPv6 address for the JN
dodagID	The Identifier of the routing tree to which the JN will should join to
RPLparentID	The Node Identifier of routing tree root Node
blackList	List of IP addresses that are blacklisted

Link-Local UDP request to a Gateway router: This is a simple and straightforward mechanism to collect bootstrapping information. A JN can easily obtain its link local IPv6 address by combining its EUI-64 to IPv6 Link-Local Prefix (fe80). Subsequently, JN requests parameters to the gateway Node (using gateway Node's Link Local Address as destination), which is in the same PAN as JN. However, this requires multi-hop unicast/multicast forwarding when the gateway router is out of the POS of the JN.

MLE for parameter dissemination: In most cases MLE is used for network wide parameter changes; Yet a JN can send a unicast update request to a neighbor. The neighbor responds by sending an update message containing the current values of the parameters. Since POLAR bootstrapping parameters originate from Back-end server, a modification to MLE is needed to supply these information from a Back-end to a neighbor of JN so that it will supply it to JN using a MLE parameter update message.

CoAP relay: A CoAP relay may also be used to request and receive bootstrapping parameters. In this method a JN requests parameters from its already joined one-hop neighbor (bootstrapping agent) using CoAP request. Since the bootstrapping agent already has a globally routable IPv6 address, it relays the CoAP request to a Back-end server via a multi-hop CoAP request. The neighbor also relays the response from the Back-end server to JN. This method can be implemented easily and can have larger packet size. In addition, the path from the neighboring device to Back-end is already secured.

DHCPv6: A DHCPv6 may also be configured to dynamically provide parameters to JN. In

this case, Custom DHCPv6 options need to be defined to accommodate the parameters listed in Table 6-5. This approach does not work together with the POLAR architecture. As discussed earlier parameters originate from a Back-end Server rather than being assigned dynamically by a DHCPv6 server. Another drawback is that most 6LoWPAN stacks do not natively implement this method.

6LoWPAN neighbor discovery: In this method, The joining Node interacts with the one-hop reachable router to get the network prefix of the PAN and generates its own IPv6 address by combining it with its EUI-64. This method only supplies the mesh interface IPv6 parameter of the bootstrapping configuration. Thus, JN must perform another request with another protocol (e.g. CoAP) to a Back-end server using its newly generated IPv6 Address to collect the full list of parameters.

6.2.7 Data representation

Table 6.6 lists the parameters that need to be transported to a JN for bootstrapping. Depending on the type of method we use for making the parameter request, a designer faces a number of options to represent those parameters. The parameters are partly static flags (e.g., isBorderRouter, isParent) and partly a structured data such as IPv6 address. Table 6.6 shows available data representation mechanisms to transport bootstrapping parameters. The selection is highly influenced by the development language, the platform of the client program, as well as the data representation chosen when developing the Back-end server.

Figure 6.2 demonstrate an exemplary (Json, XML, TLV [1], and ASN.1[2]) representation of initial parameters that are stated in Table 6-5.

6.2.8 Joining routing tree

This is the last phase of the bootstrapping process. At this stage, a JN has found and selected a PAN, authenticated itself to a Back-end system, and acquired all the necessary security and network credentials. Therefore, the next and the last thing is to be a part of routing tree. RPL as discussed in Section 6.1.4 creates a DODAG that a JN can join. Based on the parameters in the bootstrapping request phase, a JN undergoes to RPL message exchange as described in RFC 6550 [18].

[1] Type-Length-Value.
[2] Abstract Syntax Notation

Table 6.6: Parameter representation options

Method	Pro	Con	Supported in
Byte Array	Smaller in size Can represent any data type	Harder to parse Not human readable	All
JSON	Easy to parse Available libraries in many languages Medium size Human readable	Medium size Does not have support for all data types such as byte sequence	CoAP Relay Link-Local UDP MLE
XML	Easy to parse Available libraries in many languages Human readable Support for huge number of data types	Big size Large parser	CoAP Relay Link-Local UDP MLE
YAML	Medium size Human readable Support for binary data type Available libraries	Large specification	CoAP Relay Link-Local UDP MLE
TLV	Faster to parse Smaller size Support for any data type	Not human readable	All
ASN.1	Faster to parse Support for any data type	Not human readable	All

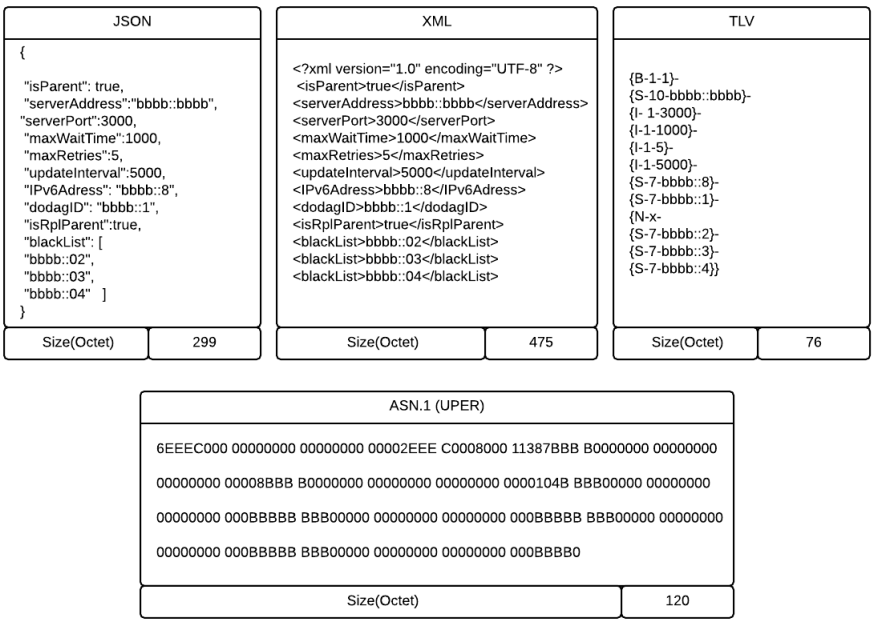


Figure 6-3: Example Parameter Representations

6.2.9 Summary

In the previous sections we discussed available technologies and methods for designing a bootstrapping protocol for POLAR outdoor lighting system. The comparisons stated in Section 6.2 are vital in picking and redesigning phases of the usecases. In this section we summarize and modularly present the available technologies. This guides my design choices and redesigning principles in the next section. Figure 6-4 diagrammatically shows the design choices discussed in the previous section.

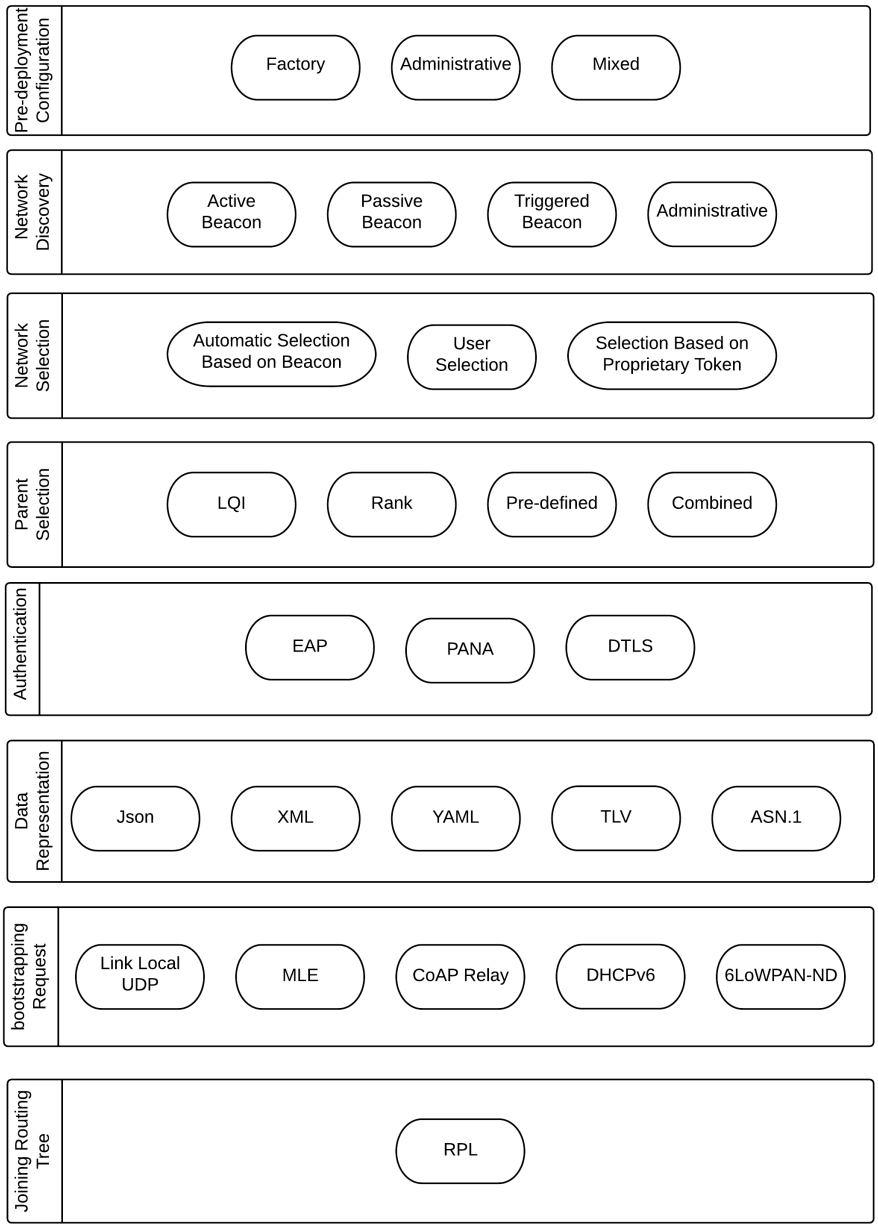


Figure 6-4: Design Options

6.3 Design details

In this section, we explain the design choices made for the design of the bootstrapping protocol for POLAR. The choices are further customized to fit the overall architecture of the entire system. As Figure 6-5 depicts, a technology (method) is chosen for each phase in the bootstrapping process.

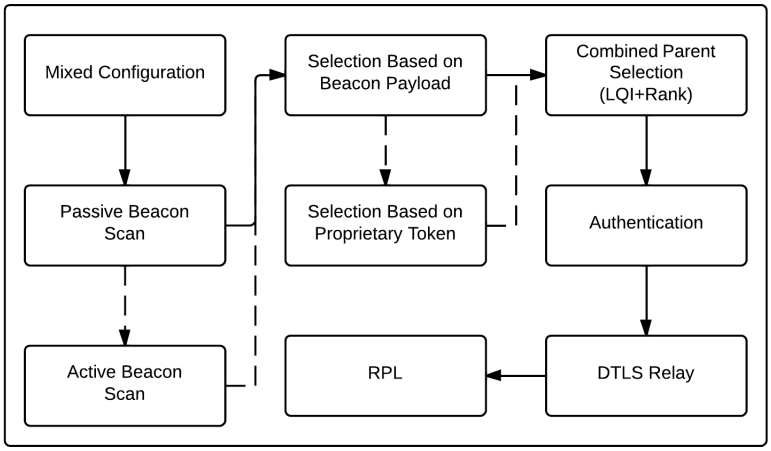


Figure 6-5: Summary of Design Choices

As discussed earlier, the bootstrapping process starts by flushing the Node with a pre-deployment configuration. The configured parameters are listed in Table 6.7.

Pre-deployment configuration: The Node is partly configured by the manufacturer and partly by Philips. This enables us to take advantage of globally unique parameters such as EUI-64 as well as flexibly configure Nodes with local parameters such as NodeID. Figure 6-6 shows the parameter configuration.

Table 6.7: Predeployment configuration parameters (partial)

Parameter	Meaning
NodeID	Unique Identifier of a Node (unique with in the PAN)
Ordered list of channel	Prioritized list of channels to scan through
Operating frequency band	the frequency band at which the PAN is operating
MAC address*	Layer two address of the Node (May be given by manufacturer)
EUI-64*	a 64 bit Extended Unique Identifier
Join key*	A one time join key for securing requests
Token identification program	a program that can recognize proprietary tokens

(*)Parametters that can be configured by a Node manufacturer

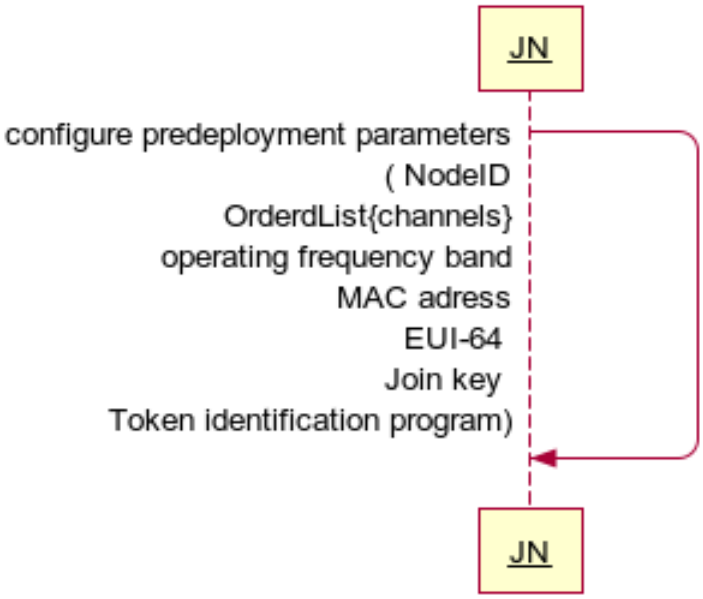


Figure 6-6: Pre-deployment Configuration

Network discovery: Once the Node is configured and deployed, it has to scan through the list of channels in the frequency band it is configured to operate. The scanning is optionally done in passive manner, i.e., listening to any packet activity on the channel. Because a passive scan is limited to looking at existing traffic, it does not show the complete and accurate information about the existence of routers. Thus, it should only run for a short period of time which is less than or equal to the super frame structure of the MAC layer. If a passive scan fails to collect beacon frames, an active scan should run, i.e., broadcasting a beacon request MAC message to all one-hop reachable Nodes and receive 802.15.4 MAC frame from all one-hop reachable routers. An IEEE 802.15.4 MAC command frame and MAC beacon frame is used to solicit and deliver beacon frames respectively. This task is diagrammatically represented in Figure 6.7.
The frame structures shown in Figure 6-8 is an IEEE 802.15.4 MAC Command frame that is used for soliciting beacon frames from neighboring routers. A MAC frame is at most 127 byte long. The size of the header (including addressing fields) as shown in the figure, is a maximum of 16 byte. That means we can accommodate a command which is 111 byte long.
Figure 6-9 demonstrates the contents of the beacon frame. The contents are explained in Table 6.8.

Network selection: As mentioned at the beginning of this section, network discovery is based on the content of the beacon payload. That means, a Node chooses the PAN it wants to join based on CID, CPI and BS control options (optionally proprietary token, if it is available).

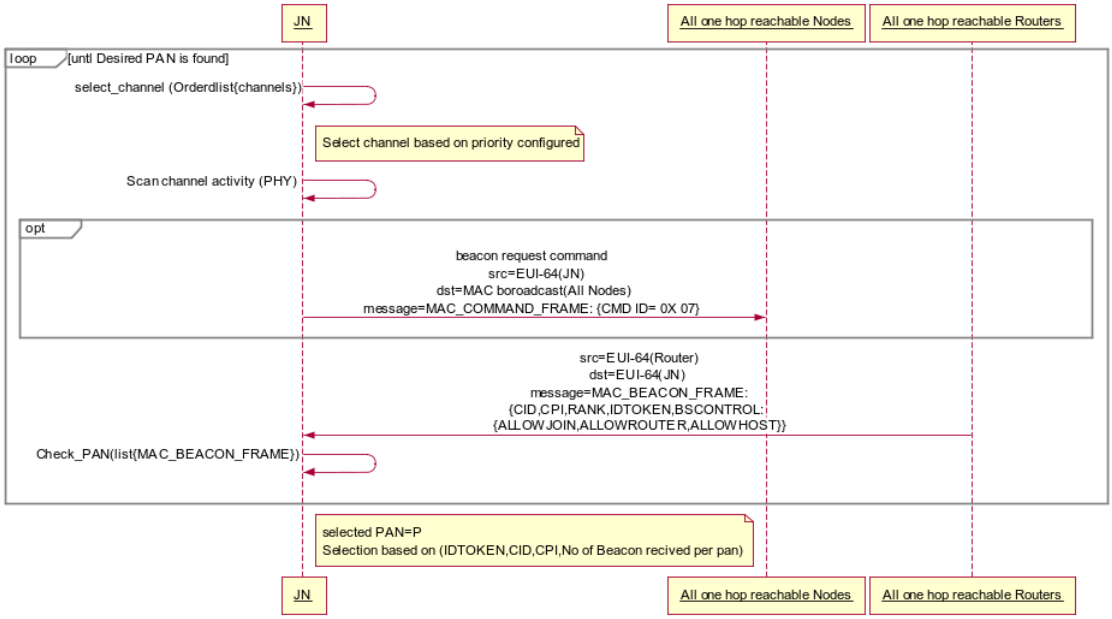


Figure 6-7: Network Discovery

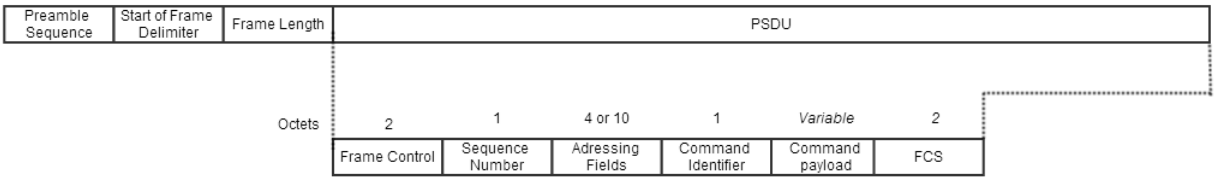


Figure 6-8: 802.15.4 MAC Command Frame

Table 6.8: Specification of beacon payload

Data	Meaning	Type
CID	A Human readable company identifier text	string
CPI	Company Protocol Identifier, the identifier of the protocol in the company	hexadecimal number
Rank	Value that tells the position of the Node with respect to the Gateway	integer
BS Control	Bootstrapping control information	
Allow Join	A flag that tells if the Node is allowing other Nodes to join	boolean
Allow Router	A flag that tells if the Node is allowing other routers to join	boolean
Allow Host	A flag that tells if the Node is allowing other host Nodes to join	boolean
Optional	An optional space in the beacon payload (can be used to store proprietary tokens)	

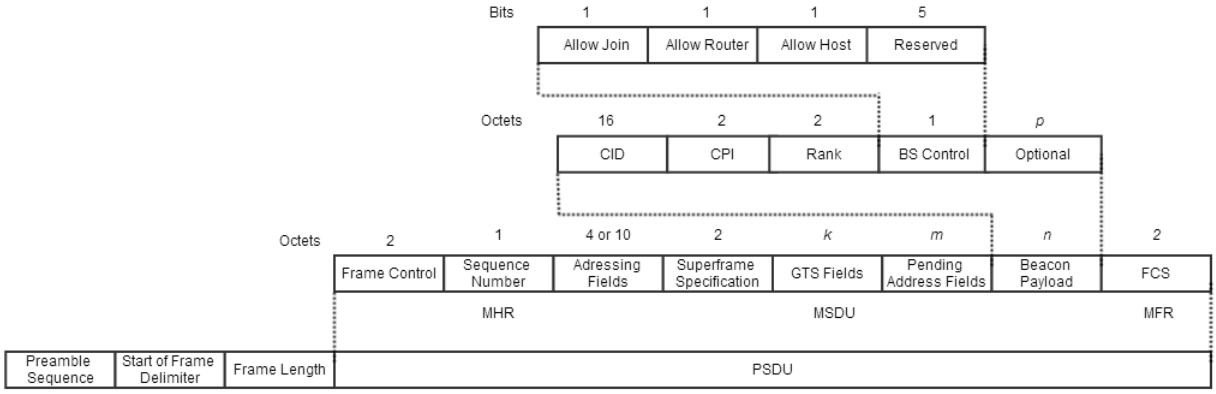


Figure 6-9: 802.15.4 Beacon Frame and Payload

Parent selection: Since there could be more than one router from the chosen PAN, a JN has to select one router to serve as a bootstrapping agent. This selection is also done by comparing the beacon frames received from the routers. Specifically, the Rank value in the BS control and the LQI from MLE (MLE is used at this stage to establish and secure links between JN and neighboring Nodes) are used to compute the best parent. The rank value only shows how close a router is from Gateway and the LQI only indicates how strong the link is between the router and the joining Node. Choosing a parent based on one of these value compromises the other. For example if we choose a parent with the best rank and that parent has a bad link quality, the success of message exchange with this Node is very low due to the weak signal. On the other hand if we choose a parent with best link quality, we do not know anything about its position in the mesh, i.e., it may have a very bad rank so that messages to Gateway has to be routed in a large number of hops; thus, the probability of packet loss is high. An efficient way of selecting a parent is a selection based on both parameters, for example selecting the best rank which has at least a minimum acceptable link quality (LQI_{min}.)

Assume there are x one-hop reachable router Nodes in the selected PAN that broadcast a beacon frame during network discovery, i.e., N = $\{\mathrm{N}_1, N_2.....N_x\}$ where N is a set of one-hop reachable router Nodes. Every Node has a rank and link quality attribute that tells about its position in the mesh and its link quality respectively, i.e., $\forall N, N.rank \in [0, 2^{16}]$ and N.lqi $\in [0, 1]$. Note that rank attribute has a 2 octet space, meaning the maximum rank that a Node can have is 2^{16} where a Node with a rank 0 is a Gateway Node. Link quality is a number between 0 to 1 where 0 is no link and 1 is a link with no packet loss. The idea is to check every Node in N starting from minimum (best) rank to the maximum (worst) rank incrementally to determine if it has at least a minimum acceptable link quality LQI_{min}. To do so we first set LQI_{min}=1, i.e., the minimum acceptable link quality is a link with no packet loss. Since this is mostly not

true, we decrement the value of LQI_{min} by some value 0 <D <1 after checking every Node in N.

Algorithm 1 Parent Selection

```
1:  procedure selectBestParent(N,D)          ▷ take set of Nodes and decrement
2:      LQI_min ← 1                          ▷ initialize LQI_min
3:      N' ← sortByRank(N)                   ▷ N' is a set of Nodes sorted by Rank
4:      while LQI_min >= 0 do
5:          for (KinN') do                   ▷ for every Node in the sorted set
6:              if K.lqi >= LQI_min then     ▷ K has a minimum acceptable LQI ?
7:                  return K                 ▷ if so select the Node as a best parent
8:              end if
9:          end for
10:         LQI_min ← LQI_min − D            ▷ Decrement the minimum acceptable LQI
11:     end while
12:     return null                          ▷ No Parent is available
```

Authentication: Authentication of JN to the PAN aswell as to the back-end server is achieved by DTLS relay. It is used by the joining device to establish a secure DTLS connection with a DTLS server that may or may not be located in the PAN. The details of DTLS relay is discussed in section 6.3.1 of this document.

Bootstrapping request: Having selected a BA and getting security credentials, a JN now can request network resources from a Back-end Server. At this stage a JN does not have a globally routable IPv6 address.Thus, the request has to be made by using its Link-Local IPv6 address. The request is made to the one-hop reachable BA and forwarded to Back-end Server via a multi-hop route. To do so the BA has to provide a service that receives a bootstrapping request from JN and relays it to the Back-end Server. This is done through DTLS relay [14] (explained in detail in section 6.3.1).

Joining routing tree: As described in Design Options, RPL is used to form and maintain the destination oriented directed acyclic routing graph (DODAG). MLE may run to secure and configure links with the RPL parent. RPL control messages (DIS,DIO,DIO-Ack) are exchanged between Nodes to allow joining of new Nodes as shown in Figures 6-10 - 6-12.

6.3.1 Parameter distribution using DTLS relay

Assuming a joining node has been authenticated using DTLS relay handshake protocol (i.e., key agreement and secured channel already established between a joining node and a back-end server.), the DTLS record layer can be used to transport the application layer packets in a relay manner. Since CoAP is the most preferred application layer protocol in constrained

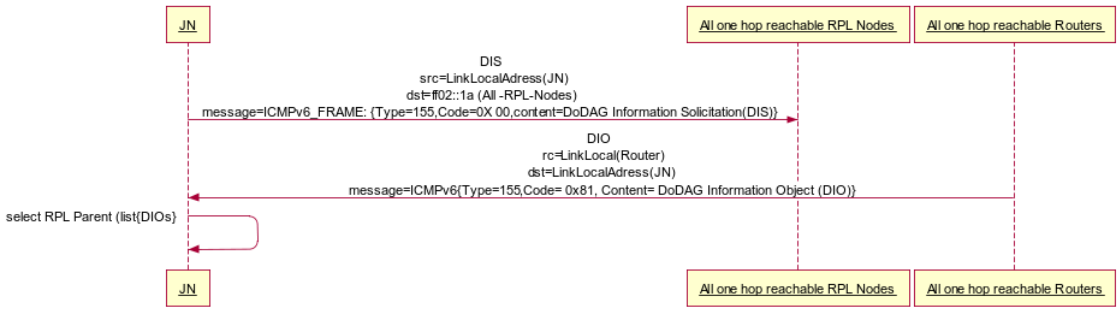


Figure 6-10: Joining RPL DODAG

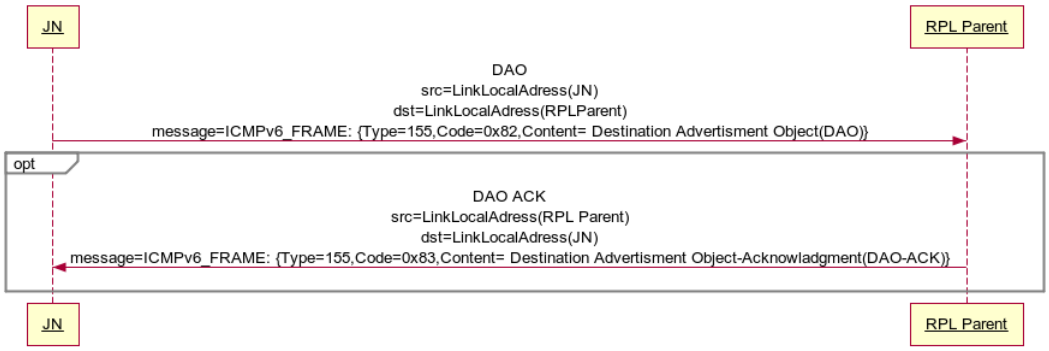


Figure 6-11: Destination Advertisement in Storing Mode

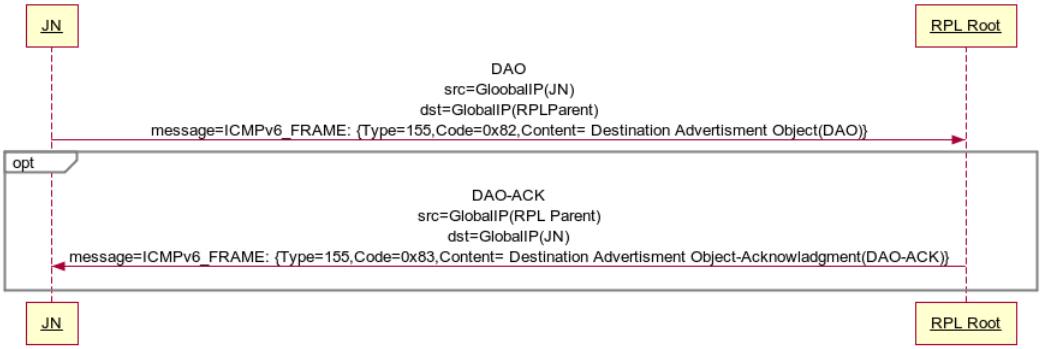
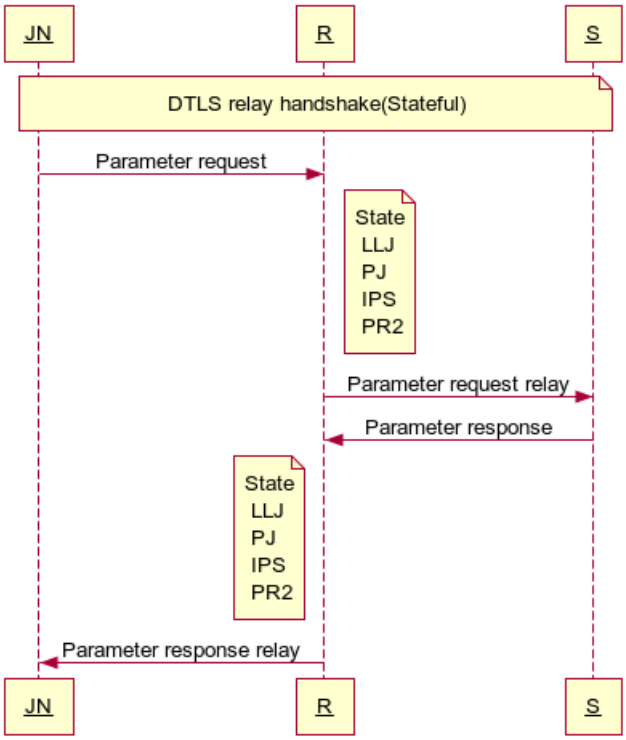


Figure 6-12: Destination Advertisement in Non-Storing Mode

environment, the solution can be described simply as the transmission of CoAP messages using a DTLS relay for parameter distribution. The process starts with a DTLS relay handshake as described by the DTLS relay draft with the DTLS server defined by the relay node. This scenario is suitable for POLAR bootstrapping usecase because the joining node does not know the address of the DTLS server. This may be done in either a stateless or stateful fashion.

Parameter request/response (Stateful)

The DTLS authenticated joining node makes a DTLS secured CoAP request to the relay device. (This is because the joining device does not know the IP address of the back-end server) the DTLS payload is encrypted using a pairwise symmetric key agreed between joining node and back-end server during the DTLS relay handshake phase. Thus, the relay device can neither decrypt the secured request nor does it have the service to accept the parameter request and respond. Therefore, it relays the request to a back-end server by changing the address and port of the Joining node to its own IPv6 address and a randomly generated port. Additionally, the relay device keeps a tuple of the joining node's link local address, the joining node's port, its randomly chosen port, and the IPv6 address of the back-end server to forward the response back to a joining node. Normally, a server would decrypt the DTLS payload using the relay-server key.Since this is a relayed message that originates from a joining node, it can only be decrypted using the joining node-server key. For the server to successfully decrypt the request, it must understand where the request originateds. Therefore, upon receiving the request, the server uses the DTLS header to use the right key for decryption. This is followed by a corresponding response to the relay. On receiving the response, the relay device consults its locally stored tuple to select the node to which it should relay the response and forwards it accordingly. The joining node receives the response and can decrypt the payload to configure its interface. The main goal of these proposed solutions (stateless and stateful) is to apply the DTLS relay concepts for parameter distribution including the joining node's IPv6 address.



Name	meaning
JN	Joining node
R	Relay device
LLJ	Link local address of JN
LLR	Link local address of R
PJ	port of JN
PR1	Port of R for communication with JN
PR2	Randomly generated port number of R for relay
IPR	Globally routable IP address of R
IPS	Globally routable IP address of S
PS	Port of S

Parameter request

IP		UDP		DTLS		DTLS Payload
Src	LL_J	Src	P_J	Type	T_1	COAP Request
Dst	LL_R	Dst	P_{R1}	Version	V_1	
...		...		Epoch	E_1	
				Sequence	S_1	
				Length	L_1	

Parameter request relay

IP		UDP		DTLS		DTLS Payload
Src	IP_R	Src	P_{R2}	Type	T_1	COAP Request
Dst	IP_S	Dst	P_S	Version	V_1	
...		...		Epoch	E_1	
				Sequence	S_1	
				Length	L_1	

Parameter response

IP		UDP		DTLS		DTLS Payload
Src	IP_S	Src	P_S	Type	T_1	COAP Response
Dst	IP_R	Dst	P_{R2}	Version	V_1	
...		...		Epoch	E_1	
				Sequence	S_1	
				Length	L_1	

Parameter response relay

IP		UDP		DTLS		DTLS Payload
Src	LL_R	Src	P_{R1}	Type	T_1	COAP Response
Dst	LL_J	Dst	P_J	Version	V_1	
...		...		Epoch	E_1	
				Sequence	S_1	
				Length	L_1	

Figure 6-13: Parameter Distribution Using DTLS Relay(Stateful)

Table 6.9: Flow of events (Stateful)

Number	Flow of event
1	Joining node: makes a DTLS secured CoAP request to a relay device
2	Relay device: receives DTLS secured CoAP request (relay device cannot read the payload since it is encrypted for the server)
3	Relay device: stores a state consists of Joining node's link local address, joining node's port, server's IP address and a randomly generated port number by which it will relay the request to the server
4	Relay device: relays the DTLS message to the server using its globally routable IP address and the randomly generated port
5	Server: receives the relayed DTLS message
6	Server: consults the DTLS header and decrypts the message
7	CoAP server: processes the request and generate response accordingly
8	Server: encrypts the response for the joining node
9	Server: sends the DTLS encrypted CoAP response to the relay device
10	Relay Device: receives the DTLS encrypted CoAP response from server
11	Relay Device: consults the stored state and relays the message to joining device by changing the IP address and port of the server to its link local address and port
12	Joining node: receives the relayed message
13	Joining node: consults the DTLS header and decrypts the message to configure its interface

Parameter request/response (Stateless)

When using the stateless mode of DTLS relay for parameter distribution, the request as well as the response must be modified by a relay device to add an additional DTLS relay (DRY) header. Yet, the DTLS payload is unreadable by the relay due to the fact that it is encrypted using a joining node-server key. Thus, the relay device must add the following DRY header to the original DTLS payload. Subsequently, the DRY message is the encapsulation of DRY header and the original DTLS message from the joining node. DRY Message = DRY header + original DTLS message. Following this, the relay device relays the DRY message to the server using its globally routable IP address and the randomly generated port. Up on receiving the DRY message, the server decrypts the original DTLS message, processes the request, then generates the corresponding response. To reply the response back to the joining node, the server

Table 6.10: DRY header

Number	DRY header	Size (bit)
1	Link local Address of joining node	64
2	Port of joining node	16

first appends the DRY header to DTLS message (response) and sends it to the relay device. The relay device then uses the DRY header to relay the DTLS message to the joining node. Finally the joining node receives the DTLS message, decrypts the payload, and configures its interfaces.

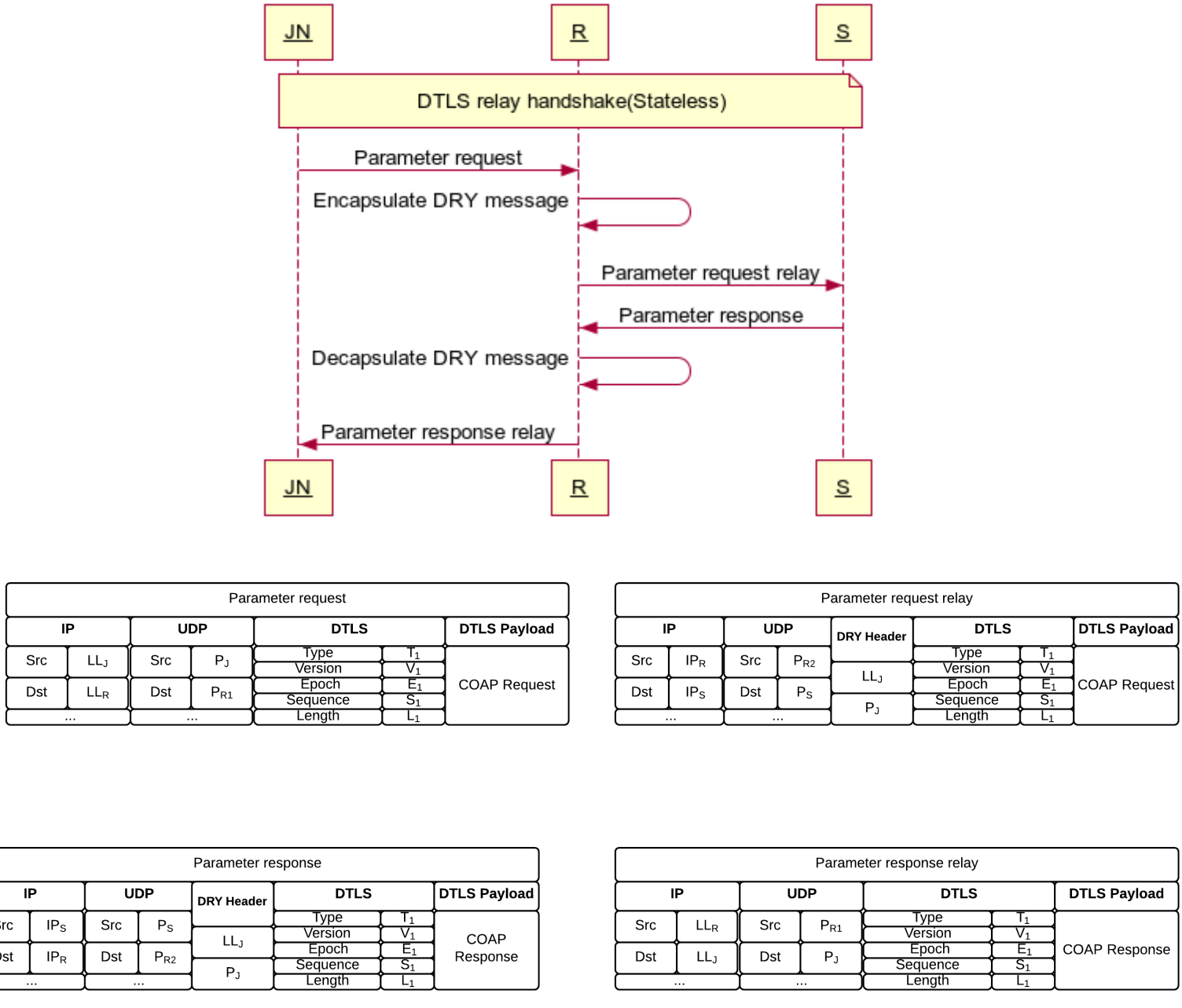


Figure 6-14: Parameter Distribution Using DTLS Relay(Stateless)

Table 6.11: Flow of events (Stateless)

Number	Flow of event
1	Joining node: makes a DTLS secured CoAP request to relay device
2	Relay device: receives DTLS secured CoAP request (relay device cannot read the payload since it is encrypted for the server)
3	Relay device: appends DTLS relay header (DRY header) to the original DTLS message
4	Relay device: relays the DRY message to the server using its IP address and port
5	Server receives the relayed DRY message
6	Server consults the DTLS header and decrypts the DTLS payload (CoAP request is now readable by server)
7	CoAP server: processes the request and generate response accordingly
8	Server: encrypts the response for the joining node
9	Server: appends the DRY header to the DTLS encrypted CoAP response and sends it to the relay device
10	Relay Device: receives the DRY message from server
11	Relay Device: consults DRY header and relays the message to joining device by changing the IP address and port of the server to its link local address and port
12	Joining node: receives the relayed message
13	Joining node: consults the DTLS header and decrypts the message to configure its interface

Packet size consideration

Like any OSI model, the packet is constructed by appending lower layer protocol headers and a payload. Therefore, for a CoAP request and response using DTLS security the packet contains the following headers.In stateless mode of operation, since there is no state kept in the relay device, an additional DRY header is needed to transport the joining node's link local address and port. This leads to 10 octets extra per packet.In contrast stateful mode of operation, a state has to be saved at the relay device. This state consists of

- JN's link local address : 8 octet
- JN's Port: 2 octet

- IPv6 saddress of DTLS server: 16 octet
- Randomly generated Port of Relay device: 2 octet
- Total : 28 octet per Joining node

Although DTLS relay have two modes of operation, namely, stateless and stateful. The stateful mode of operation is chosen to demonstrate due to the fact that it does not modify the DTLS payload being transmitted. Thus, it makes it possible to use an ordinary DTLS server.

Chapter 7

Implementation

7.1 Overview

As part of the POLAR project, a demonstrator has already been implemented to display the wireless outdoor lighting connectivity features such as connected sensing. I investigated the bootstrapping scenario of this demonstrator and created a design model using state machines. Furthermore, I modified the bootstrapping procedure to include scenarios where the joining Node does not have access to the back-end server (bootstrapping via agent). For parameter distribution, I implemented a DTLS relay demonstration software that serves one or more joining Nodes to request and receive initial parameters from a dedicated server.

7.2 Bootstrapping in the POLAR demonstrator

The POLAR demonstrator creates a wireless sensor network of Nodes. A single board computer (SBC) is used for a Node with Redwire econotag platform as a radio interface. Architecturally, the demonstrator follows a client-server approach, where the server runs a CoAP server component to responds to bootstrapping requests and gather usage reports, a MYSQL database to store bootstrapping and application informations, server applications and a web service. Clients run one or more applications, a bootstrapping module to make an initial parameter request using CoAP Client, a message dispatcher to direct IPv6 packets to Ethernet or 802.15.4 interfaces, and different drivers. The econotags are connected to SBC via USB and slip radio program from Contiki stack. RPL and mesh APIs of Contiki stack are also running on the client machine. The demo architecture with sensor application is depicted in figure 7-1.

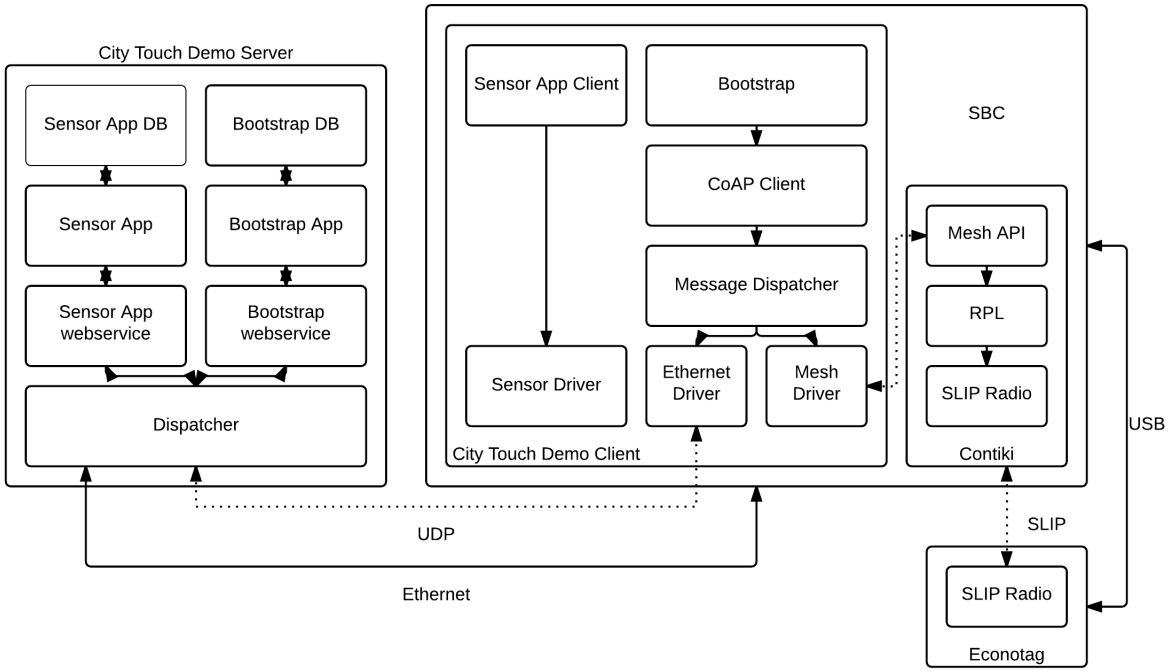


Figure 7-1: POLAR Demo Architecture

7.2.1 Modification

The POLAR demo setup only supports bootstrapping of Nodes by directly requesting parameters from the back-end server. However, the main goal of this project is to bring a mesh bootstrapping feature to the POLAR architecture. To support this feature, I modified the bootstrapping module according to the following guideline:

- A gateway Node bootstraps as discussed in section 7.2.1.
- Joining Node makes bootstrapping (parameter) request to the gateway Node using its link local IPv6 address.
- The gateway Node have a CoAP server running with a local bootstrapping database.
- Mesh only Nodes do not have Ethernet interface (Only Mesh).
- Mesh only Nodes receives parameter from gateway node and configure their interfaces accordingly.
- A joined Node stays connected with gateway Node since update also comes from it. Thus, the modification supports only a one-hop (from the gateway Node)join.

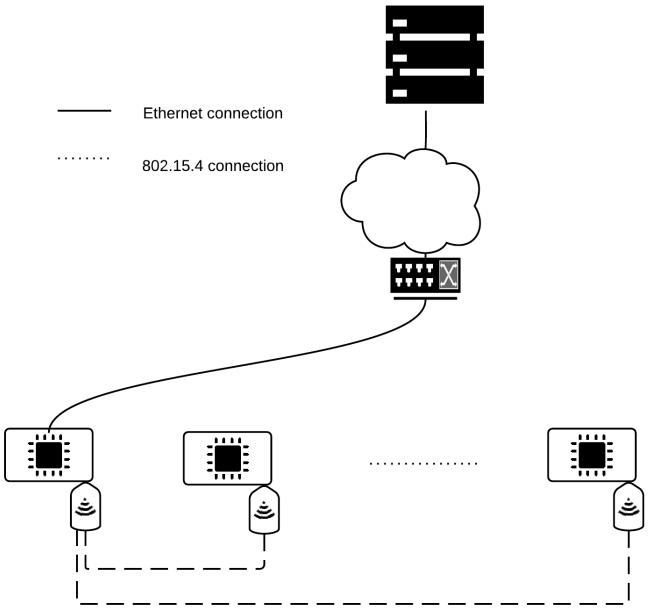


Figure 7-2: Mesh bootstrapping-POLAR demo

The modification enables Nodes to make parameter request and receive parameters using their mesh interface. Figure 7-2 demonstrates the bootstrapping sequence of gateway Node and joining Nodes. Chronologically, the solid line (parameter request via Ethernet) always precedes the dashed line (parameter request via 802.15.4)

7.3 DTLS relay demonstrator

The DTLS relay demonstrator is an implementation of the solution discussed in section 6.3.1. The implementation supports one or more DTLS clients to perform DTLS handshake as well as transmit/receive DTLS record protocol messages with a DTLS server via a relay device. The client(s) does not necessarily have to know the address of the server to which they are communicating. This make it useful for POLAR nodes to collect initial parameters. As described in Section 6.3.1. stateless mode of operation is chosen to demonstrate.

7.3.1 Setup and architecture

The demonstrator uses two SBCs running a DTLS client echo program, a windows machine with JVM running DTLS relay program , and a virtual linux machine running a DTLS server echo program as shown in Figure 7-3. I used a TinyDTLS library on both the client and server side with a simple echo application to establish DTLS handshake and later the server prints

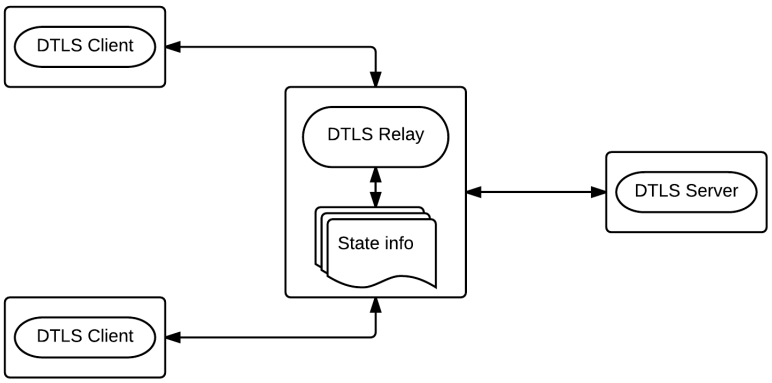


Figure 7-3: DTLS relay demo

out any text that is passed from the client.

The relay program listens to any UDP packet forwarded to it on a specific port. It then forwards it to the predefined (hard coded)DTLS server by keeping state information locally about the source of the packet and the port used to forward it to the server. It also receives UDP packets from the server and uses the locally kept state information to forward it to the specific client.

Figure 7-4 shows the class design of the DTLS relay program. The relaying functionality is implemented as an extension to a UDP relay program (from jCoAP library). It assigns a separate datagram channels for client and server connections. The relay program also uses a connection manager to keep track of clients who are connecting to the relay device. The connection manager in turn make use of state information in order to forward packets to the intended receivers.

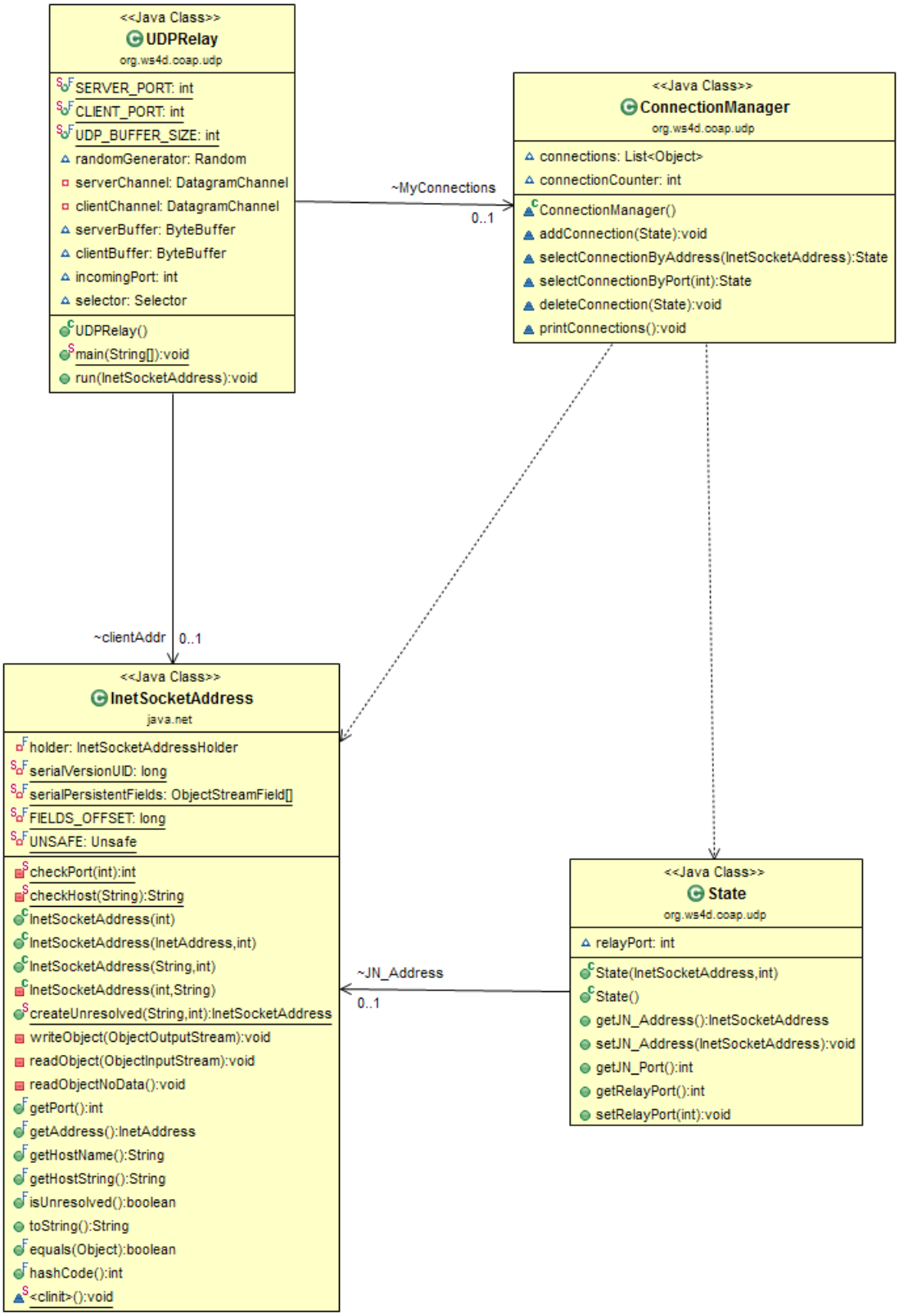


Figure 7-4: DTLS relay class diagram

Chapter 8

Validation and Verification

8.1 Overview

In this chapter I discuss about the reasons why the design and implementation presented in chapter six and seven of this document are considered a valid solution for the problems and requirements listed in chapter three and five.
The set of solutions and concepts crafted in this project are not suitable for formal verification methods. I.e there is no globally accepted technique to prove or disprove the solutions that they are valid. Thus, I followed an informal approach to confirm the validity and applicability of the I solutions proposed. This approach includes wide range of views to the solutions including experts opinion from stakeholders, alignment with standards, proof of concept demonstrations and packet sniffing to compare the contents of actual traffic from the intended one.

8.2 Design Verification

As discussed in chapter six, the design is a set of methods, protocols and procedures that are necessary for a successful mesh bootstrapping in the POLAR domain. It is a sequence of eight steps, each solving a particular problem. Thus, the verification can also be presented per step as each of them are verified using different method(s). Table 8.1. depicts the steps and their corresponding verification method and result.

Table 8.1: Design Verification

Steps	Verification		
	Compliance	Expert opinion	Demonstrator
Pre-deployment configuration	802.15.4 specification Zigbee-IP Zigbee-NAN	Approved by POLAR	POLAR demonstrator
Network discovery	802.15.4 specification Zigbee-IP Zigbee-NAN	Approved by POLAR	POLAR demonstrator
Network selection	802.15.4 specification Zigbee-IP	Approved by POLAR	-
Parent selection	MLE protocol RPL	Approved by POLAR	-
Authentication	DTLS protocol DTLS relay draft	Approved by POLAR	DTLS relay demonstrator
Bootstrapping request	IPv6 standard CoAP DTLS relay draft	Approved by POLAR	DTLS relay demonstrator
Data representation	TLV	Approved by POLAR	POLAR demonstrator
Joining routing tree	RPL	Approved by POLAR	POLAR demonstrator (Contiki RPL)

8.3 Test result

To test the validity of the POLAR mesh bootstrapping I run the bootstrapping module while observing its behavior from Wireshark packet sniffer and from the POLAR demo web application.

8.3.1 POLAR demonstrator

To confirm the success of the bootstrapping module of the POLAR demonstrator, the joining node must successfully connect to a gateway Node and transmit sensor information to back-end server. The POLAR web application visualizes the bootstrapping of Nodes in web browser as shown in figure 8-1 using solid lines. The figure visualizes a gateway Node(presented with red circle) connected to a back-end server (presented with white cloud) and a joining Node (presented in a green circle) successfully bootstrapped and connected to the gateway Node using mesh bootstrapping. Wireshark packet sniffer makes it possible to display an already joined Node reporting dummy sensor information to a back-end server via a gateway Node. Figure 8-2 shows a real time snapshot of Wireshark when an alrady joined node (Address:

Figure 8-1: Mesh bootstrapping visualization

dddd:205:c2a:8cf7:e1d5) periodically reports sensor information using CoAP to a back-end server (Address: bbbb::bbbb). In the meantime, the gateway Node (Link Local Address: fe80 205:c2a:8c4e:b630) broadcasts RPL DODAG Information Object (DIO) message.

8.3.2 DTLS relay demonstrator

Two test are conducted to examine the credibility of the demonstrator. The first one is setting up the demonstrator as it is shown in Figure 7-5 with two clients (address : bbbb::2 and bbbb::4) and observing the log output from the relay device which shows the messages being forwarded and the state kept. The snapshot of result of this experiment can be shown in Figure 8-3.
The other experiment is conducted by running the DTLS relay demonstrator in three different orders and observe the packet traffic if it behave as it was intended.

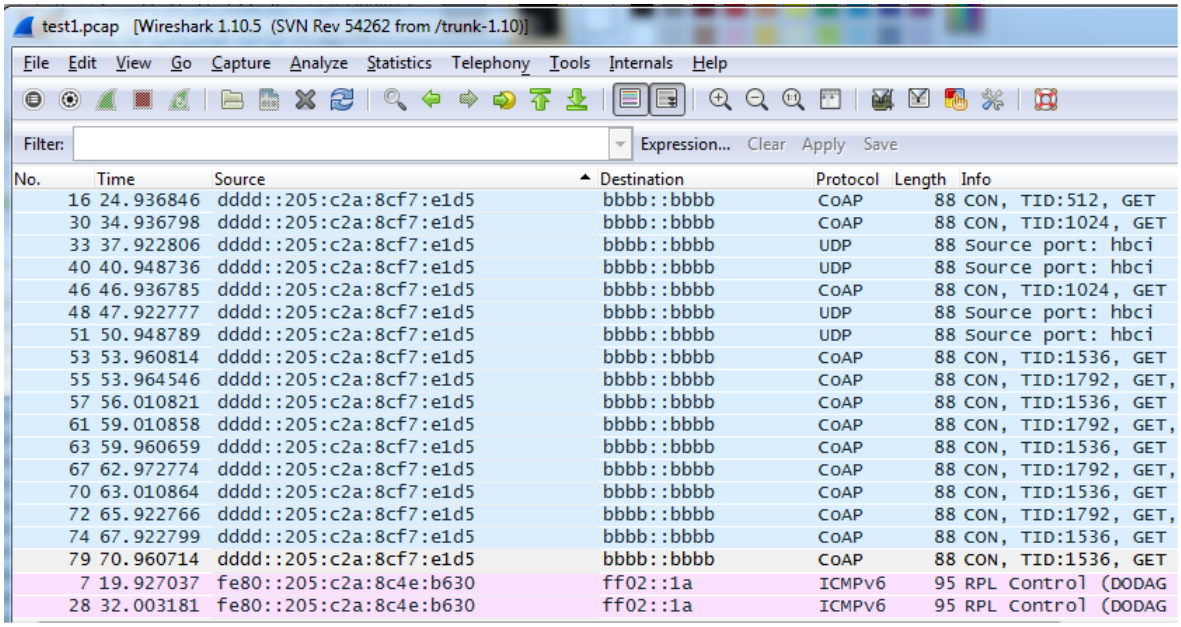


Figure 8-2: Packet traffic in POLAR demonstrator

Table 8.2: Testing order

Order 1	
Node	Task
bbbb::2	HS
bbbb::2	APP
bbbb::4	HS
bbbb::4	APP

Order 1	
Node	Task
bbbb::2	HS
bbbb::4	HS
bbbb::2	APP
bbbb::4	APP

Order 3	
Node	Task
bbbb::2 and bbbb::4	HS
bbbb::4	APP

HS: Handshake APP: Aplication data

Based on the orders of tasks shown in table 8.2, I performed a test by running the DTLS relay program and observed the actual packet traffic shown in Figure 8-4 to 8-6. The order and content of the actual traffic reflects the test order. Thus, we can claim that the Demonstrator behaves as it is intended.

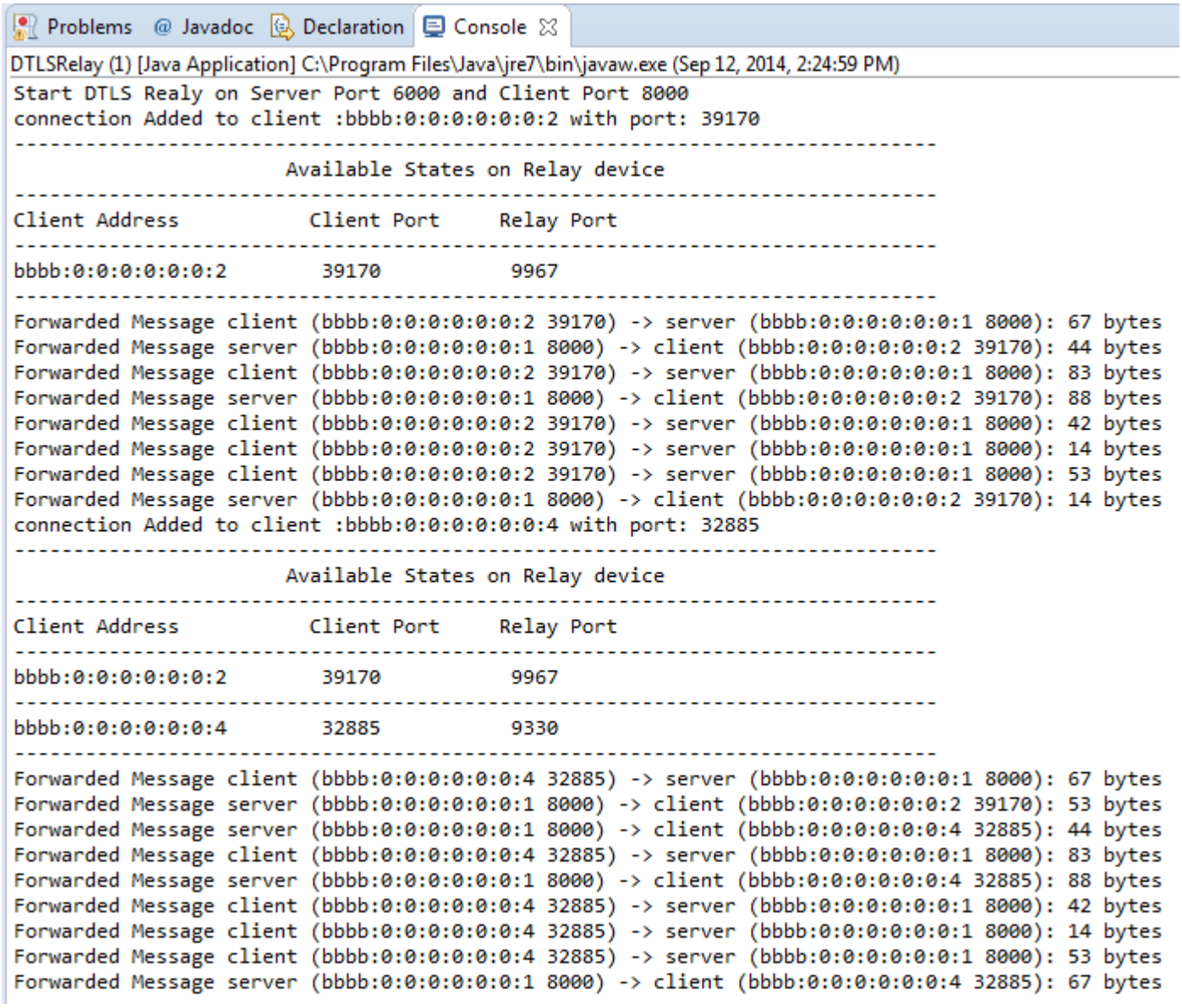
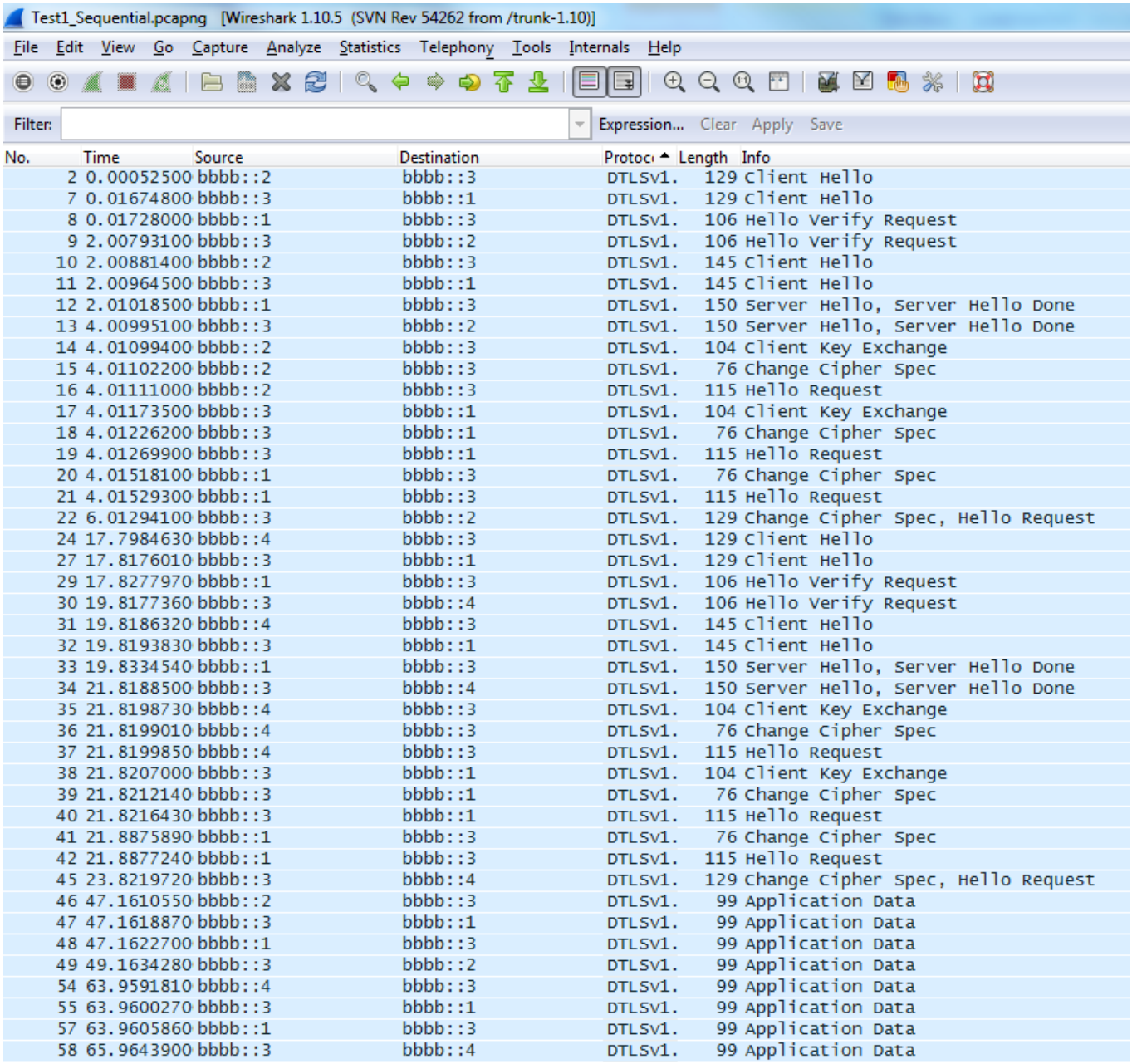
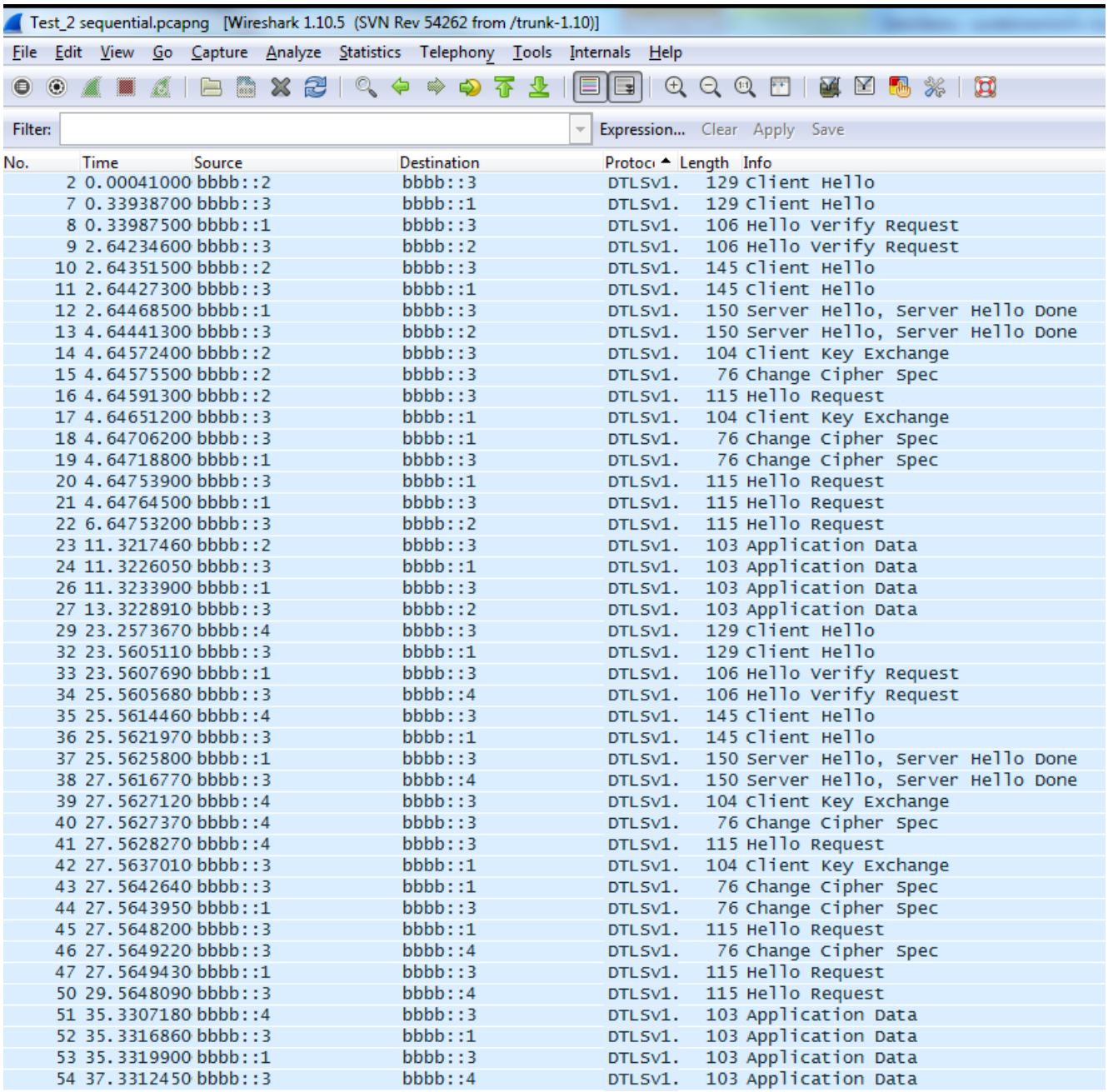


Figure 8-3: DTLS relay program output log



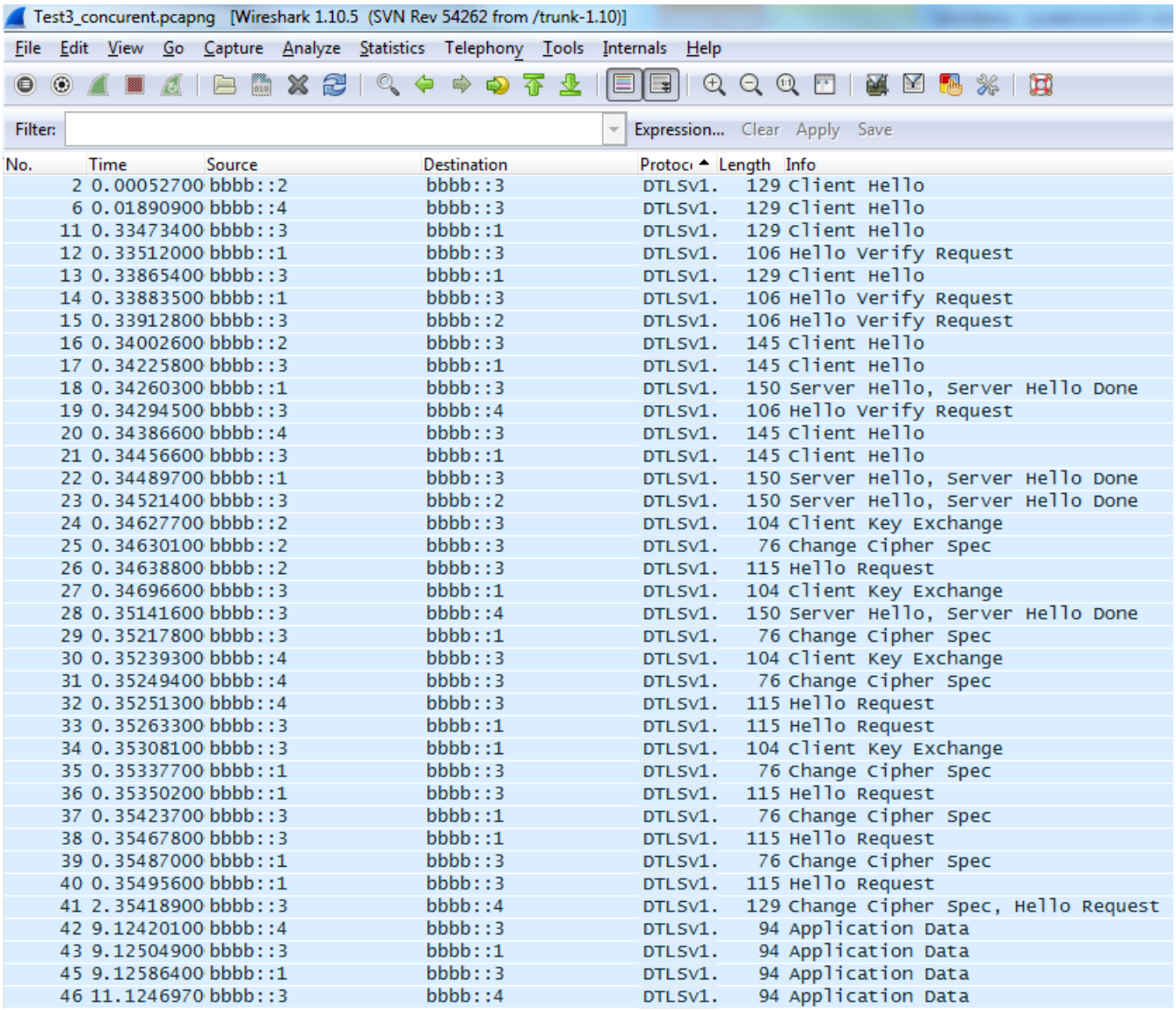
No.	Time	Source	Destination	Protoc	Length	Info
2	0.00052500	bbbb::2	bbbb::3	DTLSv1.	129	Client Hello
7	0.01674800	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
8	0.01728000	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
9	2.00793100	bbbb::3	bbbb::2	DTLSv1.	106	Hello Verify Request
10	2.00881400	bbbb::2	bbbb::3	DTLSv1.	145	Client Hello
11	2.00964500	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
12	2.01018500	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
13	4.00995100	bbbb::3	bbbb::2	DTLSv1.	150	Server Hello, Server Hello Done
14	4.01099400	bbbb::2	bbbb::3	DTLSv1.	104	Client Key Exchange
15	4.01102200	bbbb::2	bbbb::3	DTLSv1.	76	Change Cipher Spec
16	4.01111000	bbbb::2	bbbb::3	DTLSv1.	115	Hello Request
17	4.01173500	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
18	4.01226200	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
19	4.01269900	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
20	4.01518100	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
21	4.01529300	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
22	6.01294100	bbbb::3	bbbb::2	DTLSv1.	129	Change Cipher Spec, Hello Request
24	17.7984630	bbbb::4	bbbb::3	DTLSv1.	129	Client Hello
27	17.8176010	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
29	17.8277970	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
30	19.8177360	bbbb::3	bbbb::4	DTLSv1.	106	Hello Verify Request
31	19.8186320	bbbb::4	bbbb::3	DTLSv1.	145	Client Hello
32	19.8193830	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
33	19.8334540	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
34	21.8188500	bbbb::3	bbbb::4	DTLSv1.	150	Server Hello, Server Hello Done
35	21.8198730	bbbb::4	bbbb::3	DTLSv1.	104	Client Key Exchange
36	21.8199010	bbbb::4	bbbb::3	DTLSv1.	76	Change Cipher Spec
37	21.8199850	bbbb::4	bbbb::3	DTLSv1.	115	Hello Request
38	21.8207000	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
39	21.8212140	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
40	21.8216430	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
41	21.8875890	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
42	21.8877240	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
45	23.8219720	bbbb::3	bbbb::4	DTLSv1.	129	Change Cipher Spec, Hello Request
46	47.1610550	bbbb::2	bbbb::3	DTLSv1.	99	Application Data
47	47.1618870	bbbb::3	bbbb::1	DTLSv1.	99	Application Data
48	47.1622700	bbbb::1	bbbb::3	DTLSv1.	99	Application Data
49	49.1634280	bbbb::3	bbbb::2	DTLSv1.	99	Application Data
54	63.9591810	bbbb::4	bbbb::3	DTLSv1.	99	Application Data
55	63.9600270	bbbb::3	bbbb::1	DTLSv1.	99	Application Data
57	63.9605860	bbbb::1	bbbb::3	DTLSv1.	99	Application Data
58	65.9643900	bbbb::3	bbbb::4	DTLSv1.	99	Application Data

Figure 8-4: Packet traffic in order 1



No.	Time	Source	Destination	Protoc	Length	Info
2	0.00041000	bbbb::2	bbbb::3	DTLSv1.	129	Client Hello
7	0.33938700	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
8	0.33987500	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
9	2.64234600	bbbb::3	bbbb::2	DTLSv1.	106	Hello Verify Request
10	2.64351500	bbbb::2	bbbb::3	DTLSv1.	145	Client Hello
11	2.64427300	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
12	2.64468500	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
13	4.64441300	bbbb::3	bbbb::2	DTLSv1.	150	Server Hello, Server Hello Done
14	4.64572400	bbbb::2	bbbb::3	DTLSv1.	104	Client Key Exchange
15	4.64575500	bbbb::2	bbbb::3	DTLSv1.	76	Change Cipher Spec
16	4.64591300	bbbb::2	bbbb::3	DTLSv1.	115	Hello Request
17	4.64651200	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
18	4.64706200	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
19	4.64718800	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
20	4.64753900	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
21	4.64764500	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
22	6.64753200	bbbb::3	bbbb::2	DTLSv1.	115	Hello Request
23	11.3217460	bbbb::2	bbbb::3	DTLSv1.	103	Application Data
24	11.3226050	bbbb::3	bbbb::1	DTLSv1.	103	Application Data
26	11.3233900	bbbb::1	bbbb::3	DTLSv1.	103	Application Data
27	13.3228910	bbbb::3	bbbb::2	DTLSv1.	103	Application Data
29	23.2573670	bbbb::4	bbbb::3	DTLSv1.	129	Client Hello
32	23.5605110	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
33	23.5607690	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
34	25.5605680	bbbb::3	bbbb::4	DTLSv1.	106	Hello Verify Request
35	25.5614460	bbbb::4	bbbb::3	DTLSv1.	145	Client Hello
36	25.5621970	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
37	25.5625800	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
38	27.5616770	bbbb::3	bbbb::4	DTLSv1.	150	Server Hello, Server Hello Done
39	27.5627120	bbbb::4	bbbb::3	DTLSv1.	104	Client Key Exchange
40	27.5627370	bbbb::4	bbbb::3	DTLSv1.	76	Change Cipher Spec
41	27.5628270	bbbb::4	bbbb::3	DTLSv1.	115	Hello Request
42	27.5637010	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
43	27.5642640	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
44	27.5643950	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
45	27.5648200	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
46	27.5649220	bbbb::3	bbbb::4	DTLSv1.	76	Change Cipher Spec
47	27.5649430	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
50	29.5648090	bbbb::3	bbbb::4	DTLSv1.	115	Hello Request
51	35.3307180	bbbb::4	bbbb::3	DTLSv1.	103	Application Data
52	35.3316860	bbbb::3	bbbb::1	DTLSv1.	103	Application Data
53	35.3319900	bbbb::1	bbbb::3	DTLSv1.	103	Application Data
54	37.3312450	bbbb::3	bbbb::4	DTLSv1.	103	Application Data

Figure 8-5: Packet traffic in order 2



No.	Time	Source	Destination	Protoc	Length	Info
2	0.00052700	bbbb::2	bbbb::3	DTLSv1.	129	Client Hello
6	0.01890900	bbbb::4	bbbb::3	DTLSv1.	129	Client Hello
11	0.33473400	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
12	0.33512000	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
13	0.33865400	bbbb::3	bbbb::1	DTLSv1.	129	Client Hello
14	0.33883500	bbbb::1	bbbb::3	DTLSv1.	106	Hello Verify Request
15	0.33912800	bbbb::3	bbbb::2	DTLSv1.	106	Hello Verify Request
16	0.34002600	bbbb::2	bbbb::3	DTLSv1.	145	Client Hello
17	0.34225800	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
18	0.34260300	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
19	0.34294500	bbbb::3	bbbb::4	DTLSv1.	106	Hello Verify Request
20	0.34386600	bbbb::4	bbbb::3	DTLSv1.	145	Client Hello
21	0.34456600	bbbb::3	bbbb::1	DTLSv1.	145	Client Hello
22	0.34489700	bbbb::1	bbbb::3	DTLSv1.	150	Server Hello, Server Hello Done
23	0.34521400	bbbb::3	bbbb::2	DTLSv1.	150	Server Hello, Server Hello Done
24	0.34627700	bbbb::2	bbbb::3	DTLSv1.	104	Client Key Exchange
25	0.34630100	bbbb::2	bbbb::3	DTLSv1.	76	Change Cipher Spec
26	0.34638800	bbbb::2	bbbb::3	DTLSv1.	115	Hello Request
27	0.34696600	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
28	0.35141600	bbbb::3	bbbb::4	DTLSv1.	150	Server Hello, Server Hello Done
29	0.35217800	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
30	0.35239300	bbbb::4	bbbb::3	DTLSv1.	104	Client Key Exchange
31	0.35249400	bbbb::4	bbbb::3	DTLSv1.	76	Change Cipher Spec
32	0.35251300	bbbb::4	bbbb::3	DTLSv1.	115	Hello Request
33	0.35263300	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
34	0.35308100	bbbb::3	bbbb::1	DTLSv1.	104	Client Key Exchange
35	0.35337700	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
36	0.35350200	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
37	0.35423700	bbbb::3	bbbb::1	DTLSv1.	76	Change Cipher Spec
38	0.35467800	bbbb::3	bbbb::1	DTLSv1.	115	Hello Request
39	0.35487000	bbbb::1	bbbb::3	DTLSv1.	76	Change Cipher Spec
40	0.35495600	bbbb::1	bbbb::3	DTLSv1.	115	Hello Request
41	2.35418900	bbbb::3	bbbb::4	DTLSv1.	129	Change Cipher Spec, Hello Request
42	9.12420100	bbbb::4	bbbb::3	DTLSv1.	94	Application Data
43	9.12504900	bbbb::3	bbbb::1	DTLSv1.	94	Application Data
45	9.12586400	bbbb::1	bbbb::3	DTLSv1.	94	Application Data
46	11.1246970	bbbb::3	bbbb::4	DTLSv1.	94	Application Data

Figure 8-6: Packet traffic in order 3

Chapter 9

Conclusion

9.1 Overview

In this chapter, I present a condensed summary of the proposed design for a bootstrapping procedure in 6LoWPAN based outdoor lighting connectivity networks which is specifically adopted to Philips's City Touch architecture. The design is supported by a proof of concept demonstration which is expected to be integrated as a part of the prototype system of the POLAR solution. I also discussed the delivered artifacts, the limitations and my recommendation for future work.

9.2 Deliverables

The main deliverable of this project are:

- A survey of bootstrapping protocols with a comparison as well as analysis of their relevance to the POLAR architecture.
- A set of design choices that are investigated to be efficient in the project context.
- A reference implementation that shows a secured parameter distribution using DTLS technology.

In order to conclude that the deliverables have served the purpose of the project, I made a check list of bootstrapping requirements(shown in Figure 5-3) and conform that they are met at the end. Table 9.1 shows the list of bootstrapping requirements and how they are realized.

Table 9.1: Requirement checklist

Requirement	Design	Demonstrator		Discussed in
		POLAR Demo	DTLS Demo	
BOOT01	✔			Sections 6.2.2, 6.2.3, 6.2.4
BOOT02	✔	✔	✔	Sections 6.2.2, 6.2.3, 6.2.4
BOOT03	✔			Section 6.2.3
IN-CON01	✔	✔		Sections 6.2.2, 6.2.3, 6.2.4
IN-CON02	✔	✔		Section 6.2.3
IN-CON03	✔	✔	✔	Section 6.3
IN-CON04	✔			Section 6.2.3
COM01	✔	✔	✔	Section 6.2.8
COM03	✔			Section 6.2.8
COM07	✔	✔	✔	Section 6.3
RELI01	✔	✔	✔	Section 6.2.6
RELI02	✔			Section 6.2.6
RELI05	✔			Section 6.2.3
ARCH02	✔	✔		Section 6.2.8
Optional				
SECU01	✔		✔	Section 6.3.1
SECU03	✔		✔	Sections 6.2.5, 6.3.1
SECU04	✔		✔	Section 6.3.1
SECU07	✔			Section 6.3.1
SECU08	✔		✔	Section 6.2.5, 6.3.1

9.3 Limitations

Due to the fact that the technologies required for a mesh bootstrapping in 6LoWPAN are relatively new to the computing community, there are very few implementations of fully functional

IPv6 stacks available in the open source community. This makes it challenging for researchers to freely access an already available stacks and develop their prototype. Furthermore, the companies who are working in this domain does not always disclose their specification, product details, and prototypes with out commitment (e.g. payment,partnership, e.t.c). This again puts an obstacle for accessing already existing solutions.

9.4 Recomendation

After closely analyzing the problem domain and the available solutions in the market, I found it clear that there is no one off-the-shelf solution that can fully address the problems listed in this document. I believe that the survey, design, and implementation I presented here, will give an extensive overview of the design space for the problem. However, it is my strongest recommendation that Philips selects one of the already available stacks (summarized in table 4.2) and improve the coverage of supported features according to the design guidelines presented in this document.

Bibliography

[1] Bernard Aboba, Larry Blunk, John Vollbrecht, James Carlson, Henrik Levkowetz, et al. Extensible authentication protocol (eap). Technical report, 2004.

[2] ZigBee Alliance. Zigbee ip specification. *ZigBee Public Document 13-002r00*, 2013.

[3] Dan Forsberg, Yoshihiro Ohba, Basavaraj Patil, Hannes Tschofenig, and Alper Yegin. Protocol for carrying authentication for network access (pana). *RFC5191*, 2008.

[4] Olivier Hersent, David Boswarthick, and Omar Elloumi. *The Internet of things: Key applications and protocols*. John Wiley & Sons, 2011.

[5] IEEE. Wireless medium access control and physical layer specifications for low-rate wireless personal area networks. *IEEE standard*, 2003.

[6] IEEE. Low-rate wireless personal area networks (lr-wpans). *IEEE standard*, 2011.

[7] IEEE. Low-rate wireless personal area networks (lr-wpans). amendment 3: Physical layer (phy) specifications for low- data-rate, wireless, smart metering utility networks. *IEEE standard*, 2012.

[8] R.K. Kelsey. Mesh link establishment protocol (mle), draft-ietf-core-coap-13. *The Internet Engineering Task Force–IETF, Feb*, 2013.

[9] K Kim, S Yoo, S Daniel, J Lee, and G Mulligan. Commisioning in 6lowpan. *draft-6Iowpan commisioning-02*, 2008.

[10] Nandakishore Kushalnagar, Gabriel Montenegro, C Schumacher, et al. Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals. *RFC4919, August*, 10, 2007.

[11] David McGrew and Eric Rescorla. Datagram transport layer security (dtls) extension to establish keys for the secure real-time transport protocol (srtp). *RFC4347*, 2010.

[12] ZigBee Standards Organization. Zigbee specification. *ZigBee Document 053474r17 Jan*, 2008.

[13] Shahid Raza, Daniele Trabalza, and Thiemo Voigt. 6lowpan compressed dtls for coap. In *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*, pages 287–289. IEEE, 2012.

[14] O. Garcia-Morchon S. Kumar, S. Keoh. Dtls relay for constrained environments. *IETF draft, April*, 2014.

[15] Z Shelby, K Hartke, C Bormann, and B Frank. Constrained application protocol (coap), draft-ietf-core-coap-13. *Orlando: The Internet Engineering Task Force–IETF, Dec*, 2012.

[16] Z Shelby, P Thubert, J Hui, S Chakrabarti, and E Nordmark. Neighbor discovery for 6lowpan. draft-ietf-6lowpan-nd-02 (work in progress). *RFC6775*, 2009.

[17] William Stallings. Ipv6: the new internet protocol. *Communications Magazine, IEEE*, 34(7):96–108, 1996.

[18] Tim Winter. Rpl: Ipv6 routing protocol for low-power and lossy networks. *RFC6550*, 2012.

3TU.School for Technological Design, Stan Ackermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the three technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology and University of Twente. For more information please visit: www.3tu.nl/sai.