Secure service discovery in building automation and control systems

Ünlü, M.

Award date:
2012

Secure Service Discovery in Building Automation and Control Systems

MASTER'S THESIS

Mehmet Ünlü
Department of Mathematics and Computer Science

Supervisors:
Prof. Dr. Milan Petković (TU/e)
Dr. Sandeep S. Kumar (Philips Research)

October 16, 2012

Abstract

Building Automation and Control Systems (BACS) are intelligent networks of distributed sensors and actuators, which enable monitoring and control of heating, ventilation, and air conditioning (HVAC), lighting, and safety systems deployed in commercial buildings. These sensors and actuators can be seen as resources of the corresponding nodes interacting with each other, and with users providing various device services. For easy commissioning and cost-effective operation of BACS, service discovery operations are essential.

BACS is very critical part of a building infrastructure requiring that device and services are accessed and controlled in a secure way. Consequently, security and access control must be incorporated as integral part of the methods for service discovery in BACS. Furthermore, any security solution needs to take into account the domain specific requirements such as constraints on computation power, limited memory of nodes, and communication bandwidth, among others.

The objective of this thesis is to identify design requirements of a secure service discovery protocol in BACS. Further based on these requirements, a design solution is presented and analyzed based on the constraints of the system. Functionality of the proposed design is implemented as a proof-of-concept and the system is evaluated.

Keywords. building automation, security, service discovery, sensor networks, constrained devices

Acknowledgements

This thesis report has been written as a part of my Master's study at the Department of Mathematics and Computer Science at Eindhoven University of Technology, in cooperation with Philips Research, Eindhoven.

Firstly, I would like to thank Sandeep Kumar, Sye Loong Keoh, and Oscar Garcia Morchon from Philips Research for providing me the opportunity to work on this project and for providing constant feedback on my work. I heartily express my thanks and gratitude to Sandeep for his supervision, and for providing great advice and support at every step during this project. I sincerely thank my supervisor Milan Petković from Eindhoven University of Technology for his valuable guidance and constructive comments.

I would also thank the members of my assessment committee, Milan Petković, Sandeep Kumar and Tanır Özçelebi, for the time they invested in evaluating my thesis.

Besides other invaluable friends in Eindhoven, I also thank all my colleagues at Philips Research for the pleasant working environment and for the nice moments together. A special thanks goes to my office mate Roge for the nice discussions, the exchange of ideas, and for daily help and support.

Last but absolutely not least, I wish to extend my deepest and most sincere thanks and gratitude to my family for their love, pray and support throughout my study years, and to mein Schatz, Nur, for being and bearing with me through all difficult moments that existed while working on this project.

Eindhoven, October 2012
Mehmet Ünlü

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks 12
ACE Access Control Engine 35
BACS Building Automation and Control Systems 7
CoAP Constrained Application Protocol 22
CoRE Constrained RESTful Environments 11
CT Commissioning Tool 46
DHCP Dynamic Host Configuration Protocol 15
DNS Domain Name System 8
DNS-SD DNS-Based Service Discovery 14
DPWS Devices Profile for Web Services 14
DTLS Datagram Transport Layer Security 22
IETF Internet Engineering Task Force 14
MAC Message Authentication Code 24
mDNS Multicast DNS 23
RD Resource Directory 21
RR Resource Record 23
RRset Resource Record Set 24
SLP Service Location Protocol 14
SSDP Simple Service Discovery Protocol 17
TSIG Transaction Signature 24
UPnP Universal Plug and Play 14
WSA Web Services Architecture 18

Chapter 1

Introduction

Building Automation and Control Systems (BACS) are an important part of modern commercial buildings. They provide automated control of indoor conditions by improving interaction with and between devices in a building environment. The driving factor of building automation and control is to provide more comfortable environments for inhabitants, while reducing building energy and maintenance costs. Although traditionally only the core domains which are lighting and heating, ventilation, and air conditioning (HVAC) were targeted, today other typical building services, such as safety, security, and lifting systems, are also considered for an integration. Diversity of devices which are integrated into BACS are increasing due to relatively decreasing costs of processing and storage capabilities of embedded systems. Moreover, the trend towards the IP-based communication and recent advances in wireless communication network technologies, such as ZigBee and 6LoWPAN, lead to a tighter integration of building subsystems. New functionalities, heating or cooling adapting to window blinds, for instance, become possible with this integration.

With growing number of devices interacting and operating together in buildings, additional requirements have emerged for a more efficient interoperation. BACS are mainly networks of distributed sensors and actuators. These sensors and actuators can be seen as resources of the corresponding nodes interacting with each other, and with users providing various device services. For easy commissioning and cost-effective operation of BACS, service discovery operations are essential. The ability to locate other devices and their services within the building and being able to communicate with them without the need for reconfiguration each time, is fundamental to capabilities of devices in BACS. Service discovery is the technology that has been developed using a number of approaches to achieve this.

Since BACS also constitute a critical infrastructure of building, it is required that device services are accessed and controlled in a secure way. This requires secure and access controlled methods for service discovery and

control in the BACS. Furthermore, any security solution needs to take into account the domain specific requirements such as constraints on computation power, limited memory of nodes, and communication bandwidth, among others.

1.1 Problem domain and context

This study concerns itself with the device and service discovery problem in the domain of BACS. Although there have been various service discovery mechanisms and protocols proposed in the IT world so far, applicability of those in a BACS environment still remains as an issue to be studied.

Any device and service discovery solution for BACS should first identify the functional, technical, and security requirements of the domain. It should consider the trending technologies in device communication, while keeping an eye on the propriety nature of the legacy systems. In addition, a possible design should target seamless integration with less overhead and minimal human intervention.

The design solution proposed in this study is supported with a functional prototype since the work is based on the real world problems. With the analysis and further validation of the design, some of the remaining challenges will be identified, as this is an initial attempt to solve the problem.

1.2 Objective

The objective of this thesis is to identify design requirements of a secure service discovery protocol in BACS. Based on these requirements and existing service discovery approaches, a design solution is presented and analysed while keeping the constraints of the system in mind. Finally, a prototype of the proposed service discovery solution is implemented on a wireless sensor network and the system is evaluated.

1.3 Intended audience

The intended audience of this document includes graduate students, professionals, and researchers in the field of security for building automation and service discovery. The reader is assumed to be familiar with the basics of data and communication security. A basic knowledge of wireless sensor networks and Domain Name System (DNS) is also useful.

1.4 Structure of the thesis

The thesis is organized as follows:

- Chapter 2 covers the relevant background information related to the problem domain. We first give an overview of building automation and control systems. We describe architectural details and different phases of BACS lifecycle. Next, we discuss various service discovery solutions with their capabilities and security features.

- In Chapter 3, we present a use case scenario for service discovery in BACS. After discussing possible threats and misuses, we enumerate functional, technical, and security requirements for the service discovery system in the last section of this chapter.

- In Chapter 4, we propose a design solution satisfying the system requirements while keeping the BACS lifecycle in mind.

- Chapter 5 contains the details of the prototype implementation and evaluates the proposed solution based on message sizes, memory requirements and time based efficiency. Furthermore, a validation of the proposed design is given in terms of functional and security perspectives.

- The achievements of the thesis are summarized in Chapter 6. The thesis is concluded with final remarks and potential directions for future work.

Chapter 2

Background and related work

This chapter provides the relevant background information related to the problem domain. The first section gives an overview of building automation and control systems while describing architectural details and different phases of BACS lifecycle. Various service discovery solutions with their capabilities and security features are discussed in the remaining parts.

2.1 Building automation and control systems

Core functionality of Building Automation and Control Systems (BACS) is to maintain comfortable indoor conditions via the integration of various systems, such as heating, ventilation, and air conditioning (HVAC), lighting, and safety systems, while minimizing the energy used. The field of BACS has been continuously evolving for the last years, resulting in increased complexity as new services and requirements are added. With more powerful embedded systems, building automation devices are more capable of processing data and communicating with other information systems.

Before designing any solution for BACS, it is necessary to have a closer look at the architectural details, enabling technologies, and main processes in BACS lifecycle.

2.1.1 Architecture of BACS

A typical architecture of BACS is presented in Figure 2.1. The classical model is basically a three-layer hierarchical structure, in which each layer has a particular set of functions [20].

The functional level (or field level) mainly consists of sensors and actuators, which provide collection of real-time status data and real-time control. Occupancy sensors, temperature and flow meters, light switches and dimming controllers, and valves in a heating circuit are examples of sensors and actuators in the field level. The control in BACS is realized using various

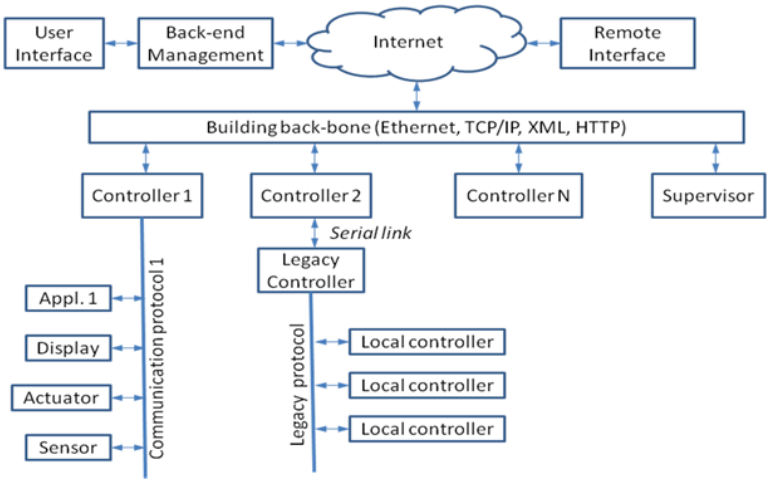


Figure 2.1: Typical architecture of BACS

controllers connected to the network of these sensors and actuators. It is also likely to have different subsystems with their own communication architectures and protocols in the functional level. Users can read the status from displays and change the settings according to their preferences using switches. A supervisor unit attached to the building network configures and manages the BACS, providing precise definitions of the behaviour of the system under various circumstances. Additionally, the overall management of the system can be done from a back-end management server or through remote interfaces like a web browser.

2.1.2 IP-based networking in BACS

In traditional BACS deployments, different subsystems in the functional level are loosely coupled within a building, and maintained by different personnel. However, with the advent of wireless mesh open standards (like ZigBee on top of IEEE 802.15.4), the physical silos between the various systems are largely removed creating a more closely integrated system. Another pushing factor of this integration is the technologies shared from IT world, such as standardization efforts of Constrained RESTful Environments (CoRE) working group at the IETF, appearing in building control networks.

Although IP was originally designed to be used for the communication between personal computers through the Internet, today a variety of devices are shifting to IP-based communication in the general Internet-of-Things trend, including those used in building automation and control.

Using IP networks as a backbone for BACS is not a new concept [12]. Possibility of using already available LAN cabling and easier integration with the rest of the IT world have been major benefits of this approach. The trend

resulted in new high level protocols for system integration, including those that employ well established Web technologies, such as XML, SOAP and RESTful Web Services. For example, BACnet, one of the most commonly used communications protocol for BACS, has been extended to support generic Web services interface recently [1].

With an IP-based backbone, several subsystems of the BACS are connected by gateways to that common backbone network. Different field level protocols might be used for communication between devices inside each subsystem. For example, lighting system of the building may employ DALI protocol, while another subsystem of wireless temperature sensors communicate over ZigBee. In addition to using IP networks as a backbone, current direction is to use it to communicate directly with the end devices. Although it was previously too expensive to implement the whole IP stack for resource constrained devices, increasingly many devices are able to connect to the Internet as microcontrollers become more powerful and cheaper. With the advent of the IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) standard, even battery-powered wireless sensor nodes can use an IP interface for communication [18].

Integration in the functional level and accessibility of data from all subsystems lead to new functionalities, such as heating or cooling adapting to window blinds. It also makes it possible to have end-to-end security between devices, ease of configuration without middle boxes like gateways, and interoperability between devices from different manufacturers. Moreover, it becomes much easier to adapt standards-based technologies such as existing device and service discovery protocols. Utilizing these technologies in BACS can improve the functionality of integrated subsystems to provide more comfortable environments for occupants, and to support and ease the work of system operators. A common communication platform could enable building systems to operate together rather than in separate modes. In addition, such an intelligent integrated system is able to perform more efficiently.

While understanding that IP-centric communication is the direction of BACS for the future, it is important to note that there are new security challenges that result from this convergence. The tighter integration of formerly dedicated stand-alone systems together with remote management facilities requires the underlying control system to be reliable and robust against malicious attempts. Easier and remote access to subsystems and to even field level devices gives rise to new threats in the BACS domain. Misuse of compromised management devices to gain access to sensors, actuators, and controllers, or directly accessing to field level devices to manipulate the behaviour of the hosted control applications are only some examples. This is especially true for the integration of more sensitive systems like access control and security alarm systems. Nevertheless, protection is also desirable for other service types as well, considering undesirable economic impact of a company-wide attack on the lighting system, for example.

2.1.3 Installation, commissioning and operational phases

There are three main phases after the design stage of BACS. These are namely installation, commissioning, and operational phases. Different personnel with different expertise perform these roles typically.

Installation process covers setting up endpoint devices in pre-determined places in the building. Sensors, actuators, displays and controllers are installed according to a floor plan which shows physical locations of devices to be installed. After providing power connections, installation is completed by running a verification procedure that shows proper connectivity and proper local operation of the devices.

Commissioning is the process of ensuring that all building systems perform interactively according to the design intent and operational needs. It covers operations such as distributing network and security parameters and operational configuration to the devices, establishing relationships between them, enabling them to communicate over an application level protocol, and similar bootstrapping. This process integrates and enhances the previously separate system functions by high level configuration and verification. Commissioning is largely done after the installation of devices, controllers and communication network, and it can be applied throughout the life of the building.

After installation and initial commissioning, all systems in the building are configured and calibrated to operate optimally. Status of the building environment gets collected by sensors continuously during operation, and the systems adjust their behaviour accordingly in order to provide satisfactory indoor environmental conditions.

2.2 Service discovery

With the number and diversity of networked devices increasing, automatic discovery of those devices and their services is getting more important in many network scenarios, including home and building automation systems. With service discovery, devices may automatically be aware of the services of other devices that they can interact with, and advertise their own services together with their properties.

In BACS, service discovery process begins after the installation of field level devices and network infrastructure. With the necessary configuration being done in the commissioning phase, devices can advertise their services and search for other required services on the network using the underlying discovery protocol.

As a considerable amount of work has been done in this field, there are various mechanisms and protocols which have been proposed and/or implemented already. Some of those protocols are designed for particular sets of objectives, while more general purpose approaches also exist.

This section overviews some well-known service discovery protocols, namely Service Location Protocol (SLP), Universal Plug and Play (UPnP), Devices Profile for Web Services (DPWS), DNS-Based Service Discovery (DNS-SD), and a more recent protocol, CoRE Resource Directory, with the focus on their suitability for BACS.

2.2.1 Service Location Protocol (SLP)

The SLP [16] is an Internet Engineering Task Force (IETF) standard for service discovery that enables the discovery of services by computers and other devices in a network without prior configuration. SLP is designed for TCP/IP networks, and it is scalable up to large enterprise networks, as it combines several techniques useful in such networks, such as optional directory services and fine-grained search.

The SLP architecture consists of three main components:

- User Agents (UA) are devices that perform service discovery, on behalf of the client.
- Service Agents (SA) are devices that advertise services, with their location and characteristics.
- Directory Agents (DA) are devices that collect service information received from SAs in their cache and respond to queries from UAs.

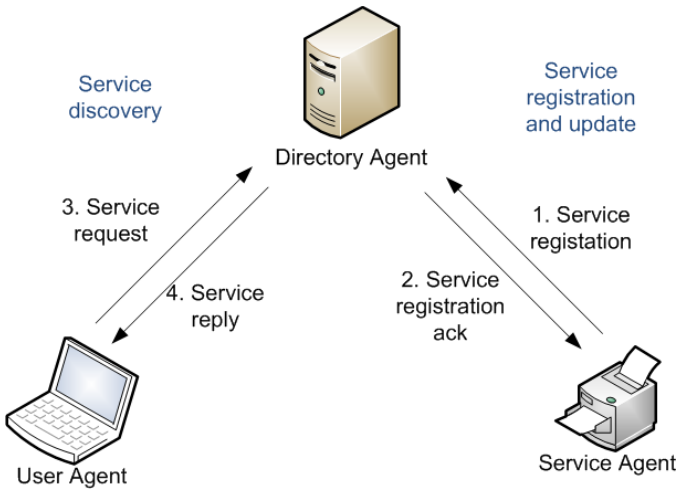


Figure 2.2: SLP agents and transactions

Interactions between the SLP agents are shown in Figure 2.2. When a new device providing a service joins a network, it registers its service to DA. Later on, a user can query this service from the DA. The user can finally access the service using the address and attributes fetched from the DA.

Before using the DA, clients (UA or SA) must first discover the existence of the DA. This can be done in three ways: static, active, and passive. In static method, Dynamic Host Configuration Protocol (DHCP) is used to distribute the addresses of DAs to SLP agents that request them, using the DHCP options defined in [27]. With active discovery, agents send multicast queries to SLP multicast group address for DA discovery. A DA responds directly to the requesting agent via unicast eventually. As the third method, DAs multicast periodically for their services in case of passive discovery.

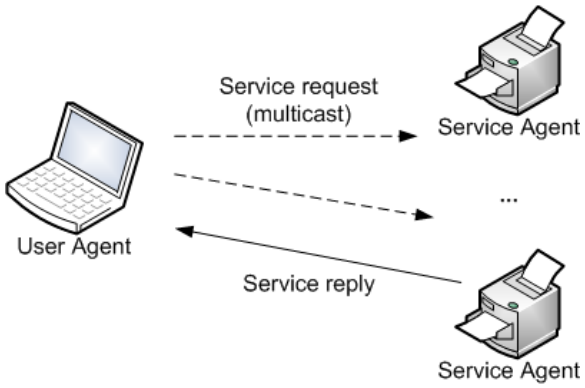


Figure 2.3: SLP service discovery without DA

It should be noticed that the existence of DAs in a network is not mandatory. They are used especially in large networks with many services in order to reduce the amount of traffic, categorize services into different groups, and allow SLP to scale. Therefore, SLP supports two operational modes for different administration requirements and network scales, depending on whether a DA is present or not. For smaller networks such as home networks, it is more effective to deploy SLP without a DA. If there is no DA, UAs send their service requests to the SLP multicast address (see Figure 2.3). SAs listening for these multicast requests will send unicast responses to the UA if they have the requested service. However, if a DA exists on the network, UAs and SAs are required to use it instead of communicating directly.

A service is defined with a *URL* and additional unlimited number of *attributes*. The URL contains the host address, the port number, and the path of the service. Service attributes are the name/value pairs that characterize the service. Service URL and service attributes with their default values are specified by *service templates*, which are defined in [15].

Most packets in SLP are transmitted using UDP, while it is also possible to use TCP to transmit longer packets. Since UDP is not reliable, SLP agents repeats multicast transmissions several times until they receive an answer.

A public-key cryptography based security mechanism is defined for SLP

in order to provide the authenticity and integrity of service announcements by signing them. However, the deployment of SLP with this mechanism may be inconvenient, because public keys of every SA must be installed on every UA, defeating the original purpose of locating services without prior configuration.

SLP is a vendor-independent, open-source protocol, and has been adopted widely. It has been implemented for many devices already [2], especially for LAN-enabled printers that are discoverable out of the box.

2.2.2 Universal Plug and Play (UPnP)

UPnP [28] is a combination of network protocols defined by a consortium of vendors from the industry, including Microsoft, Intel, Sony, and Samsung. It is designed to provide simple peer-to-peer connectivity between devices and PCs in a rather small network without enterprise class devices. Main concept of UPnP is to extend the idea of Plug and Play (PnP), which allows adding new peripherals to a PC easily, throughout the network. Seamless discovery and control for the devices of different vendors with zero configuration is targeted. UPnP aims that compatible devices can join a network, obtain IP addresses, announce themselves, query other devices and services, and disconnect dynamically.

UPnP utilizes many existing standard protocols including IP, TCP, UDP, HTTP, XML, and SOAP. It is designed to be independent of operating systems, programming languages, or physical media.

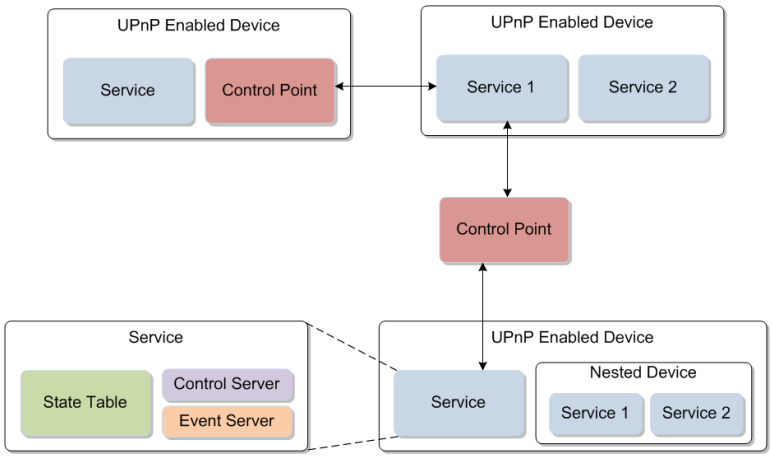


Figure 2.4: Basic building blocks of a UPnP network

There are three basic components of a UPnP network, namely devices, services, and control points as shown in Figure 2.4. A device can contain services and other nested devices. An XML-based document is hosted by each UPnP device, which describes the set of services and properties of the

device. Services are the smallest units of control in UPnP. They are defined in the device description document, together with a pointer to the service description. A service consists of a state table, a control server, and an event server. The state table represents the current service state through state variables. The control server updates the state table as new action requests received, and those changes are sent to interested subscribers by the event server. A control point is a controller which is capable of discovering and controlling other devices. After discovery, it can retrieve device and service descriptions, invoke actions to control the service, and subscribe to services' events. For a full peer-to-peer networking, devices are expected to implement control point functionality.

Device addressing, discovery, description, control, event notification, and HTML-based presentation of device services are included in the complete UPnP protocol. Simple Service Discovery Protocol (SSDP) is the protocol that enables advertisement and discovery of services in a UPnP network. It was described as an IETF Internet draft [13] in 1999. Although this proposal has since expired, SSDP stayed as a part of the UPnP protocol stack. It is a text-based protocol based on HTTPU, an extension of HTTP using UDP as the data transport protocol. A control point sends multicast search requests to discover devices and services. On the other hand, devices listening on this multicast port send unicast replies if the search criteria match. Similarly, SSDP allows devices to advertise their services to control points on the network.

UPnP does not provide any authentication and authorization for devices and applications by default. Device implementations therefore must have their own mechanisms, or implement Device Security Service [10] which is specified by UPnP Security Group. UPnP Security concerns with the control protocol, SOAP, between devices and control points. It provides control message security for authentication, authorization, integrity, and confidentiality. It also specifies a security procedure to discover and take ownership of secure devices on a network [37]. Every secure component must have a public and private key pair according to this specification.

UPnP-UP (UPnP - User Profile) [30] is a non-standard extension proposed to allow authentication and authorization mechanisms for UPnP devices and applications. It employs a new network entity, *UPnP User Profile Server*, which is responsible for storing user profile information such as fullname, username, and password, and providing authentication and access control methods based on these credentials.

2.2.3 Devices Profile for Web Services (DPWS)

Web Services provide standard solutions for interoperability between different software components which are running on various platforms and/or frameworks [4]. W3C Web Service Standards (WS-*), which are based on

the SOAP protocol, are widely used for this purpose. DPWS [19] is an OASIS standard [7] that defines a subset of WS-* specifications for secure messaging, discovery, description, and event notification on resource-constrained devices.

Although DPWS has similar objectives to those of UPnP, it is more aligned with Web Services technology including many extension points that allow the integration of device services in more general enterprise scenarios. The specification was first published in 2004, and DPWS version 1.1 was approved as OASIS standard together with WS-Discovery 1.1, and SOAP-over-UDP 1.1 in 2009.

There are two types of services defined in the DPWS specification: *hosting services* and *hosted services*. Hosting services are special services attached to a device, and has the main role in device discovery. Hosted services are the services that define the device functionality. Discovery of hosted services depends on their hosting device. DPWS devices and services are depicted in Figure 2.5. Beside these hosted services, there are also built-in services for service advertisement and discovery, service metadata exchange, and publish/subscribe event notifications.

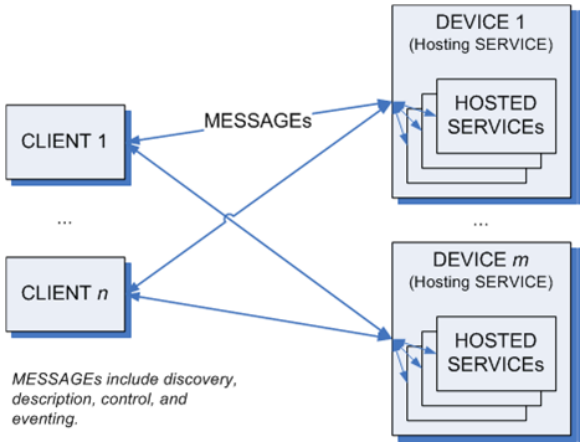


Figure 2.5: DPWS clients (controllers) and devices [7]

DPWS is based on core Web Services standards and uses further specifications as it is shown in Figure 2.6. Most of the specifications are part of the Web Services Architecture (WSA) [4] and are not explained in this chapter. SOAP and WS-Addressing are used for all messaging. WS-Policy is used to express declarations about capabilities, requirements and characteristics of service implementations. WS-Security provides optional mechanisms for end-to-end message integrity, confidentiality, and authentication. WS-MetadataExchange and WS-Transfer are used for retrieval of metadata that describes what other endpoints need to know to interact with a specific endpoint. WS-Eventing defines a protocol for managing the Web Services

based publish/subscribe mechanism.

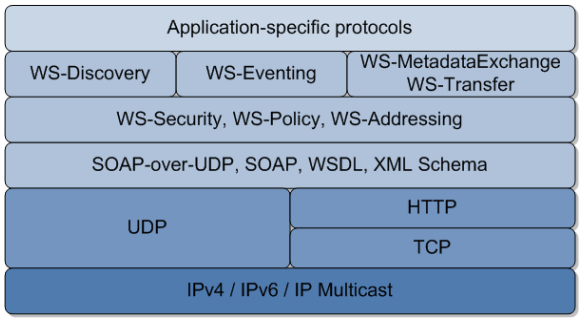
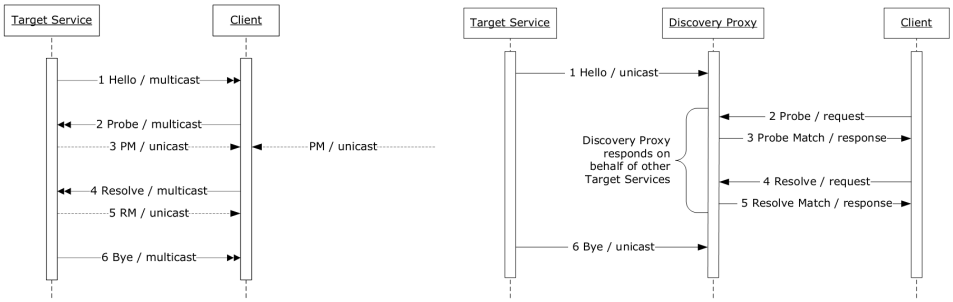


Figure 2.6: DPWS protocol stack

WS-Discovery, a discovery protocol based on SOAP-over-UDP, is leveraged by DPWS for automatic service discovery. It defines three different endpoint types: *target service*, *client*, and *discovery proxy*. Clients search for target services that offer themselves to the network, and discover them dynamically. The search starts with a *Probe* message; devices that match the search send a unicast *ProbeMatch* response directly to the client. Then, a client sends a *Resolve* message to locate a device by name, and the matching device sends a *ResolveMatch* response to the client. Figure 2.7 shows an overview of the messaging in discovery.



(a) In ad-hoc mode (b) In managed mode

Figure 2.7: DPWS discovery messages [24]

A discovery proxy is used in order to scale the discovery to enterprise-wide scenarios, since multicast discovery is limited to local subnets. Another function of a discovery proxy is to suppress multicast messages to reduce network traffic. Therefore, a client may work in an ad-hoc or a managed mode of operation depending on the existence of a discovery proxy on the network (see Figure 2.8).

Although DPWS does not require that endpoints be secure, it defines a recommended baseline for interoperable security between devices and clients. In order to mitigate message alteration, denial of service, replay, and spoof-

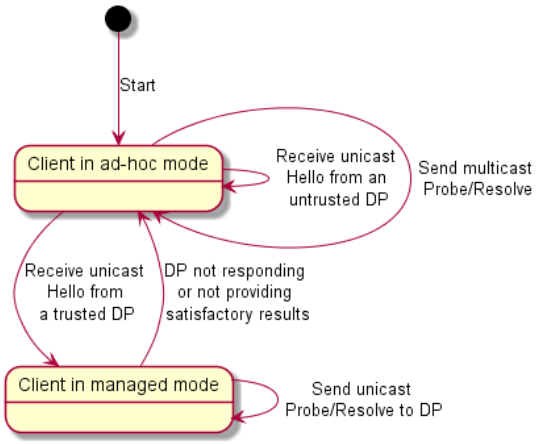


Figure 2.8: Client operation modes

ing attacks regarding discovery process, *compact signatures* are used as an evidence of authenticity and integrity of discovery messages. A sender creates the compact signature from a full XML Signature for optimized transmission, and the receiver expands the compact signature to a full XML Signature for verification. Discovery messages are not considered as confidential. TLS/SSL is used to create an end-to-end secure channel for metadata exchange and control traffic. It enables each participant to authenticate the identity of the other, to verify the integrity of the received messages, whilst also providing confidentiality for all messages between devices and clients. X.509 certificates are used as credential by devices. Signing WS-Discovery messages and establishing TLS/SSL connections are done by using those certificates. X.509 certificates or username/password credentials through HTTP Authentication can be used for client authentication.

As DPWS relies on Web Services, it is programming language independent, and has generic mechanisms of abstraction. It provides a complete set of functionalities for device integration, reducing interdependencies between components. On the other hand, despite its limited constraint functionalities targeting small devices, it still has limitations because of complex descriptions and verbosity of XML Web Services. Embedded devices rarely have enough memory and processing power to process complex XML messages. Furthermore, acceptance from the home and building automation industry and manufacturing partners is not very high yet.

Within the European R&D ITEA project SIRENA (Service Infrastructure for Real-time Embedded Networked Applications) [3], some of the first DPWS implementations have been produced for embedded devices.

2.2.4 CoRE Resource Directory

Constrained RESTful Environments (CoRE) is an IETF working group whose purpose is to overlay the REST architecture to resource constrained devices and networks in a suitable form. Machine-to-machine (M2M) applications such as smart energy and building automation are mainly targeted by CoRE. In such environments, where no humans are included in the process and statically created interfaces result in fragility, discovery of resources offered by devices has a big importance. CoRE Resource Directory [32] is a specification proposed by this group that describes how to discover resources from constrained servers that host them.

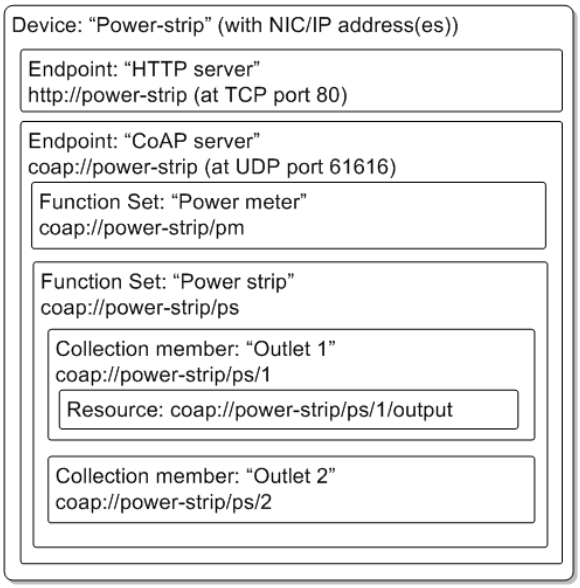


Figure 2.9: Device with endpoints, function sets and resources

Direct discovery of resources using multicast is not very practical in many M2M settings because of large and dispersed networks or sleeping nodes. CoRE Resource Directory specification aims to solve this problem by utilizing a Resource Directory (RD), which allows other servers to register their resources and query them later on. It defines the interfaces which are required to discover a resource directory, and to register, maintain, lookup and remove resource descriptions.

Figure 2.10 depicts the resource directory architecture. An endpoint (EP) is a web server that registers its resources to the RD. It is defined as a web server with an associated port number, thus a physical node may have more than one endpoints (see Figure 2.9). The RD provides a set of REST interfaces to be used by other endpoints to register and maintain resource directory entries. Those entries are expressed in CoRE Link Format [31]. Furthermore, the RD implements other interfaces to validate entries, and

for clients to lookup resources from the RD.

Resource directory entries on the RD are soft state and should be periodically updated by the endpoints. Also, an RD can proactively discover the resources from endpoints, or validate existing entries in the same manner.

REST interfaces between an RD and endpoints are called *Resource Directory Function Set*. Notice that all those interfaces can be realized using Constrained Application Protocol (CoAP), an HTTP-like application layer RESTful protocol for constrained environments, or using HTTP. Discovery of an RD can be done by sending multicast or unicast requests to a well-known path. Some alternative methods are using a default location such as edge router, assigning an anycast address to RD, or using DHCP.

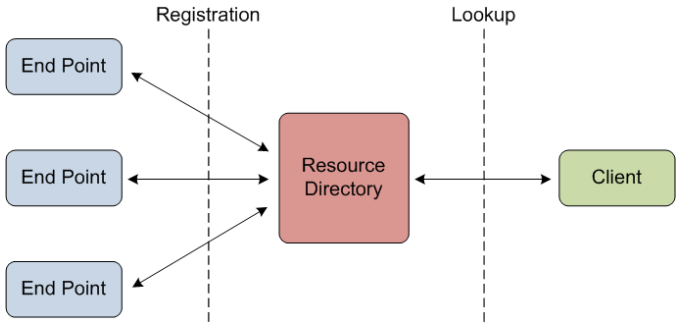


Figure 2.10: The resource directory architecture

After discovering the directory, an endpoint may register its resources by sending a POST request containing the list of resources to the registration interface. The update interface is used to update or refresh previously registered entries. An RD can check the validity and freshness of an endpoint's registered resources by using the validation interface. This is done by sending a special GET request to the endpoint. All RD entries have a timeout, and are removed after their lifetime, however they should be removed by endpoints if they are no longer available. This is performed using the removal interface of the RD.

The lookup interface allows lookups for registered resources. Filters can be applied to return only entries with specified attribute values when querying the RD. The result is a list of links corresponding to the lookup request.

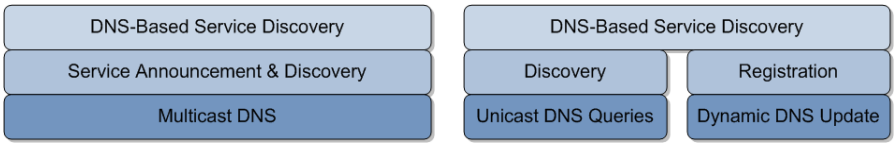
Security for CoRE Resource Directory is provided by recommended security techniques for CoAP. CoAP may be used with Datagram Transport Layer Security (DTLS), a protocol that provides communications security for unreliable datagram traffic [29].

2.2.5 DNS-Based Service Discovery (DNS-SD)

DNS-Based Service Discovery is a method of using the existing DNS architecture and records to locate devices and their services [5]. Although DNS

is known exclusively as mapping host names to IP addresses, it is a more general hierarchical database that can store almost any kind of data for different purposes, as indicated in the specification [11]. DNS-SD was originally proposed by Apple, and was implemented under the name Bonjour (previously Rendezvous) [1] with a combination of Multicast DNS (mDNS). mDNS is another specification which enables devices to use DNS functionality without a conventional DNS server in a local network environment [6]. It is often implemented together with DNS-SD.

DNS Resource Records (RRs) are used in DNS-SD to provide information about services. SRV RRs (RFC 2782) are used to describe a service with a service name, port number, and the corresponding hostname. Mapping between hostnames and IP addresses are defined with A (RFC 1035) or AAAA (RFC 3596) records, in case of IPv4 and IPv6 addresses respectively. A TXT record (RFC 1035) additional to SRV record can hold user-defined service attributes, for instance, queue names which are often used with LPR printers. Finally, PTR records (RFC 1035) are useful for assigning service instances to a service. For example, an SRV record with the name `_http._tcp.example.org` lists all `_http._tcp` services (i.e. Web servers) for the domain `example.org`. An example for PTR records can be a record with the name `MyPrinter._ipp._tcp.example.org`. It defines a printer instance, `MyPrinter`, with the Internet Printer Protocol (IPP) that is accessible via TCP on the domain `example.org`.



(a) Local LAN (b) Wide Area

Figure 2.11: DNS-SD architecture

Although major DNS-SD implementations make use of mDNS in order to operate on the local link, it can also be deployed with a conventional unicast DNS server. In this way, integration of resource constrained devices into the Internet infrastructure would be more feasible, following the *Internet of Things* vision. DNS-SD architectures for both cases are shown in Figure 2.11.

DNS-SD security

DNS-SD is only a specification for how to use the existing DNS for service discovery purposes. It defines how to name and use records to utilize DNS

[1] Apple Inc. mDNSResponder [Online] `http://www.opensource.apple.com/tarballs/mDNSResponder/`

as a service discovery solution. It does not specify any additional security mechanisms to those that already exist for DNS security [5].

DNSSEC [21] is recommended to be used where the authenticity of the information is important. Also, in order to control which clients have permission to update DNS records, secure updates [36] is recommended.

DNS Security Extensions (DNSSEC) is an extension specification for DNS that allows clients to validate the authenticity and the integrity of a DNS response through the definition of additional RRs. The main purpose of DNSSEC is to protect DNS clients from believing forged DNS data.

Public key cryptography is used by DNSSEC to meet its requirements. A Resource Record Set (RRset) is a set of RR within a zone that share the same name, class and type. Every RRset is digitally signed by the authoritative owner, and published along with the corresponding public key in the DNS itself. It does not make use of digital certificates or any other form of external credentials. The DNS system itself is used for storage and distribution of all the necessary security credentials.

In order to check a DNS response, a client retrieves the related digital signature RRs, calculates the hash value of the RRset, and verifies the signature using the published originator public key. Then the client can validate the originator public key using a hierarchical signature path that leads to a point of trust. After all these checks are successfully done, the client has some confidence that the DNS response was authentic and complete.

In secure DNS, Transaction Signature (TSIG) and SIG(0) mechanisms are used to protect the exchange between two particular entities.

In TSIG, a Message Authentication Code (MAC) is calculated through a shared secret between a pair of hosts. These hosts can exchange DNS information together with a signature based on that secret. Since only these two hosts hold the secret, no third party can create the signature. The mechanism is lightweight and useful between paired hosts that have a long-term relationship such as primary and secondary name servers for a domain. It also has a key distribution scheme by means of the TKEY extension [9].

SIG(0) mechanism is based on public-key cryptography. The message is signed by the host's private key while the public key is stored in DNS. The generated MAC/signature is included in the message as a SIG(0) RR. Authentication using SIG(0) is more scalable while calculation and verification of MAC in TSIG is computationally inexpensive.

DNS dynamic updates allow users to update (add or delete) their RRs and RRsets. Dynamic updates should be authenticated in order to protect DNS data from malicious parties. TSIG or SIG(0) can be used for those authentications.

Chapter 3

Use cases and requirements

In this chapter, we contemplate over possible use cases for service discovery in BACS. After discussing possible security threats, the last section enumerates functional, technical, and security requirements for the service discovery system.

3.1 Use cases

This section describes few of the service discovery related use cases in each phase of the BACS. We use a lighting system scenario in the use cases. The scenario includes a light sensor, lamps, and window blinds, which are planned to operate in a specific room cooperatively. The light sensor controls the lamps and window blinds according to the amount of light in the room automatically. However, it is supposed to control only those lamps and window blinds which are in the same room as the sensor. Following use cases illustrate the scenario starting from the installation phase, and explains the usage of discovery protocol during the whole life cycle.

3.1.1 Device installation

Device installation scenario covers setting up a device in a pre-determined place in the building, providing necessary connections, and verifying the local functionality eventually. An installer, a low-voltage electrician usually, arrives on-site during the construction of the building. He installs the sensor, window blinds, and lamps according to a floor plan which shows physical locations of devices to be installed. Notice that the installer does not have to know operational details of the devices necessarily. After fixing the devices and providing the power connection, he checks if the devices can be powered on and off, using a central power switch for instance.

3.1.2 Pre-commissioning

After the devices have been installed and powered on, they become part of a wireless network due to an *easy network setup* mechanism. This assumes that an IP network has been installed and is operable already. Being part of the network means having an IP address, and being able to send and receive messages over IP. Getting an IP address can be done using DHCP or using an address auto-configuration method, for instance. As soon as the devices is able to communicate within the network, they start to announce/register their functionalities, i.e. the services they provide. For example, a lamp can have a *lamp* service that can be turned on, turned off, or dimmed. In order that registration to be done, a device first needs to locate the registration point on the network, if any exists. This can be achieved by sending multicast queries (active discovery), or waiting for an advertisement message from the registration point (passive discovery). Service registration/announcement messages need to be authenticated and confidential if it is stated as a requirement.

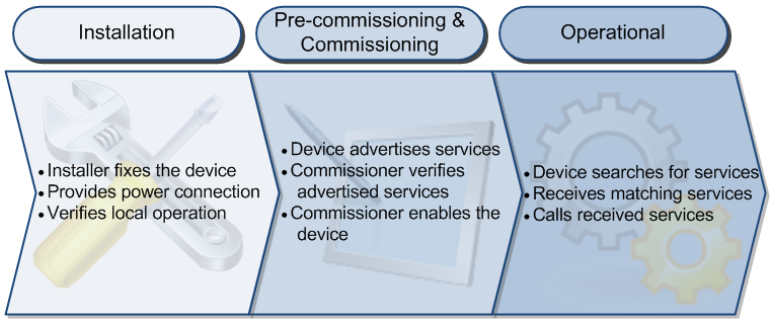


Figure 3.1: Use case scenarios in different phases

3.1.3 Commissioning

Commissioning of the installed devices is done by a commissioner using a commissioning tool. Commissioning tool is a device such as a tablet computer that can be *trusted*. This is typically achieved through a certificate issued by a trusted authority. The commissioner goes to each device using a floor plan. IP addresses of the devices are not known by the commissioning tool at that stage.

Every device has a unique identifier (UID) that can be read by the commissioner, e.g. using a barcode reader. After reading the UID of a specific device, the commissioner queries for the device and service information that has been registered/advertised by the device itself. He verifies that the registration data matches with intended type of the device, and that data is authentic. The assumption here is that the commissioning tool is able to

authenticate the source of this information (the device itself), for example by holding a factory default pre-shared key associated with the UID read. After the verification, the commissioner authorizes the device to be part of the system, so that the device can query other devices and their services via the discovery protocol. Finally, further configuration of the device is performed by adding the operational instructions to the device and/or to the controller, e.g. which devices and services will be queried by a commissioned device during operation. In addition, if there is an access control mechanism for the devices and services, necessary access control policies consistent with the operational instructions are set by the commissioner.

3.1.4 Operational

The light sensor is programmed to control only the lamps and window blinds in the same room according to the current amount of light. After the initial commissioning of the devices has been performed, the sensor first needs to find the lamps and window blinds in the room. It uses the discovery protocol to get a list of required devices and their services. Then it gets the information that describes how to access those services again using the discovery protocol. It should be possible to authenticate the sender of these discovery query and response messages. Furthermore, those messages are considered confidential, and should be encrypted appropriately against eavesdropping. Preferably, the sensor caches the discovery results for future use for a defined lifetime. After discovering the lamps and window blinds, the light sensor sends service requests to those devices in order to control their current state according to the light intensity measured. Target devices process the requests and adjust their state accordingly, after applying the relevant access control policies eventually.

The complete process divided into different phases is summarized in Figure 3.1.

3.2 Security threat analysis

We need to consider the main assets, potential attackers, and security threats in the system before determining the requirements. It is important to note that the analysis in this section is done from the service discovery point of view.

Assets in the system which should be protected from illegitimate access or modification are device and service information transmitted with the discovery protocol, and commissioning material including configuration settings, keys and security parameters which enables secure service discovery.

Some potential attackers by whom the mentioned assets could be threatened:

- *Hackers* may be with mischievous or malicious intention. They may be college kids playing with building systems (e.g., turning lights off) or criminals who try denial-of-service (DoS) attacks, theft, destruction of property, etc.

- *Employees* who are disgruntled may use their knowledge of networks, computers, and protocols, or using physical access to those to perform unauthorized actions.

- *Criminals* may be thieves, terrorists, competitors, employees, etc. Typical scenarios include simply gathering information about the building to break in, and DoS attacks with a variety of purposes such as making a political statement, interfering with business, etc.

- *Competitors* monitoring the network to gather information about resource usage patterns, for example, can use this information for corporate research.

- *Human errors* which can affect system implementation, network administration, flaws in software, etc. Those are the main reasons for commissioning.

The following is a list of threats that could be directed at the components of the discovery system during different phases [14, 17]:

1. *Unauthorized registration point:* An unauthorized registration point may be installed on the network with the intention of collecting device and service information, or sending incorrect information to devices. Such a malicious agent could disrupt the operation of discovery protocol, by returning unauthorized information, or by simply not returning any results at all.

2. *Unauthorized device:* An unauthorized device may be introduced into the system with the intention of registering/announcing many bogus services, or accessing and collecting information registered at a registration point.

3. *Unauthorized commissioning tool:* An attacker could access and modify device and service information on the registration point using an illegitimate commissioning tool. Moreover, configuration settings and security parameters of devices may also be compromised by such an attack.

4. *Eavesdropping attack:* An adversary may intercept the transmitted data during the commissioning and operational phases. This is even easier in case of a wireless medium. Device and service information, security parameters, or configuration settings may be susceptible to

eavesdropping. Obtained data then might be used to compromise the authenticity and confidentiality of the communication channel.

5. *Man-in-the-middle attack:* Commissioning and operational phases may be vulnerable to man-in-the-middle attacks. Data transmitted between two system components over the IP backbone may be controlled by an adversary sitting in between the entities, and the data can be collected, modified, or simply dropped thereafter.

6. *Packet insertion:* Bogus packets may be sent to devices or other entities in the system by inserting fabricated messages into the network or replaying previously recorded messages.

7. *Privacy threat:* Privacy of the occupants who are using a device might be compromised by tracking the device's location and usage. Gathered information may allow an attacker to deduce behavioral patterns of users without their consent. Such information can then be sold to interested parties for marketing purposes, for example.

8. *Denial-of-service attack:* Since devices have scarce memory and computation resources, they can be easily affected from resource exhaustion attacks. A single aggregation point of service information could also be target of a DoS attack to make it unresponsive or unreachable. Furthermore, network availability can be compromised by jamming the communication channel, or by flooding the network with a large number of packets.

3.3 Requirements

Use cases discussed above, features of the existing service discovery solutions, and related studies in [23, 25, 35] guide us to categorize the requirements for our service discovery architecture in following types:

3.3.1 Functional requirements

1. **Discovery of the registration point:** If there exists a central registration unit on the network, devices should be able to locate it before discovering other devices and services because they first need to register their services to the registration point in this case.

2. **Service registration/announcement:** A device should be able to register or advertise the information describing the device and the services it hosts in order other devices to be able to discover it.

3. **Activation by the commissioner:** A device becomes part of the system only after the cross-check and confirmation by the commissioner. This assures that the device is functioning as it is intended and interacting with other components of the system properly.

4. **Finding service instance:** Service instances should be returned to a device after the discovery request. The response will be processed by the device to get further descriptions of services and to invoke them eventually.

5. **Resolving service instance:** Instance names can be resolved to service invocation information (e.g. IP address, port and path) which makes it possible to prepare and send service requests.

6. **Device/service grouping:** It should be possible to create virtual groups of devices and services, including groups by physical location, in order to answer queries like "all luminaries within the second floor".

7. **Filtered device/service discovery:** Searches can be performed using criteria such as group and service type so as to filter out required services.

3.3.2 Technical requirements

1. **Operational efficiency:** Due to the limited resources on the devices and the real-time nature of the BACS, service discovery process should be efficient in terms of operational time, memory footprint, and network overhead.

2. **Suitability with the BACS architecture:** Service discovery system should be such that it is easy to apply it to the underlying operational architecture of BACS.

3. **Scalability:** Since BACS usually consist of thousands of manifold devices in different subsystems, the solution should be extendable to a large network of devices.

4. **Minimal configuration and maintenance:** The system should be able to set up and operate with minimal human intervention for the sake of efficiency of the overall BACS lifecycle.

5. **Support for legacy systems:** The discovery system should be operable with legacy systems in order to make the discovery of legacy devices possible.

6. **Interoperability with different vendors/systems:** Devices from various vendors might be operating together in the system. The discovery system thus should be designed such a way that it assures interoperability between those devices.

3.3.3 Security and privacy requirements

Following security and privacy requirements are defined in order to eliminate the identified threats or reduce them to an acceptable level:

1. **Authentication and authorization:** System components must be identifiable and authorized during the interactions they are part of:
   - *Registration point:* A registration point must be authenticated during service registration, commissioning, and discovery processes in order to prevent collecting services, and returning illegitimate information by an unauthorized registration point.
   - *Devices:* Only authorized devices must be able to register and query device and service information.
   - *Commissioning tool:* Commissioning device must be authenticated while performing verification and configuration of devices and the registration point.
2. **Message integrity:** Messages containing device and service information, configuration settings, and security parameters must be protected against malicious modifications while in transit. Such unauthorized changes must be at least detectable to prevent using illegitimate data.
3. **Message confidentiality:** Configuration settings and security parameters should not be sent in plain text. In certain installations, messages containing device and service information must be kept secret against eavesdropping so that an attacker cannot perform an inventory of available devices and services.
4. **Inference reduction:** An adversary should not be able to infer about users and the building by collecting service information communicated.
5. **DoS protection:** Service discovery system should help protect system components from the denial-of-service attacks.

Table 3.1 shows which security threats can be prevented or mitigated by implementing those security requirements. *ST1* to *ST8* stand for the security threats explained in Section 3.2, while *SR1* to *SR5* are the security requirements.

	ST1	**ST2**	**ST3**	**ST4**	**ST5**	**ST6**	**ST7**	**ST8**
SR1	×	×	×					
SR2					×	×		
SR3				×	×		×	
SR4							×	
SR5								×

Table 3.1: Security requirements and the threats they are related

Chapter 4

Design

In this chapter, we propose a design solution satisfying the system requirements while keeping the BACS lifecycle in mind. It starts with an overview and motivation for the proposed design. Section 4.2 presents architecture and basic components of the system. Details of the protocol specifications are provided in Section 4.3. Finally, some of the remaining considerations and challenges are identified in the last section.

4.1 Overview

With the further integration of subsystems in BACS and higher interoperability through more standard communication protocols, discovery requirements of building automation and control applications are demanding. All previously discussed service discovery solutions have their own design goals, specific features, and scopes. In addition to taking those solutions into consideration, our approach should be to find a discovery solution that establishes a good balance between functional, technical and security related requirements. Furthermore, the solution must consider the technology trends and specific constraints in the BACS domain.

Our proposed design is a solution based on DNS-Based Service Discovery (DNS-SD) in the broadest sense. Here, the aim is to design a DNS-SD based solution with minimal modifications and additions to meet the requirements of our system. Our method is following the installation, commissioning, and operational phases of the BACS lifecycle throughout the design.

4.1.1 DNS-based service discovery for BACS

Although many protocols, including those mentioned in Section 2.2, have been proposed to do network service discovery with IP, none of them has succeeded to achieve ubiquity in the marketplace yet. At least, none has achieved anything close to the popularity of the deployment of DNS tech-

nology and infrastructure today. Making use of those existing protocols, implementations, infrastructure, and expertise is the major advantage of using DNS as the basis for service discovery.

As it is stated previously (see Section 2.2.5), DNS-SD running over mDNS is a generally used solution that provides zero-configuration and ad hoc service discovery for small networks, e.g., home networks. However, in larger networks like a commercial building network, a high volume of IP multicast traffic may not be desirable. Moreover, reachability of a multicast discovery in such a large network is also problematic. Thus, a credible service discovery protocol intended for a BACS network has to provide a registration and lookup mechanism using a central server (or servers) instead of exclusively using multicast. Given that most of the companies with an IP network already operates and maintains a DNS server, it makes sense to make use of this expertise instead of having to introduce, operate and maintain a complete different system and server. In addition, DNS already has a dynamic registration protocol [34, 36].

In the BACS architecture discussed in Section 2.1, we have different types of devices and one or more services hosted by them in the functional level. A service has properties such as type, port, and path. As it is described in the specification [5], DNS-SD uses A or AAAA, SRV, TXT, AND PTR Resource Records (RR) for discovery purposes. An SRV RR specifies an endpoint, i.e. a hostname and port number. An associated (identically named) TXT RR can contain a URI path of the service on that host. Therefore, the SRV/TXT pair can specify a service on a device. An A or AAAA RR binds a device name to an IP address, while the PTR RR binds a service type to an endpoint or directly to a service instance on a device.

DNS-SD specification does not introduce additional security mechanisms apart from standard methods which have been developed for DNS. DNSSEC, TSIG, and SIG(0) are those mechanisms that can be utilized to provide authentication and integrity of DNS data (see Section 2.2.5). A standard mechanism to ensure the confidentiality of DNS messages at application level does not exist yet. As the discovery protocol in our system will be working on a network that mostly consists of resource constrained devices, it cannot rely on heavyweight public-key operations like in DNSSEC and SIG(0). Thus, we need to combine symmetric key based standard DNS security mechanisms and additional traffic protection techniques to meet the security requirements of the system.

In the next section, we describe the overall architecture and main components of the proposed design. Section 4.3 provides detailed protocol specification step by step. Finally, this chapter concludes with other considerations and open challenges regarding our design.

4.2 System architecture

Overall architecture of the design is presented in Figure 4.1. It depicts basic components of the discovery system along with an optional access control mechanism, which is assumed to already exist in the BACS architecture, and interactions between different components.

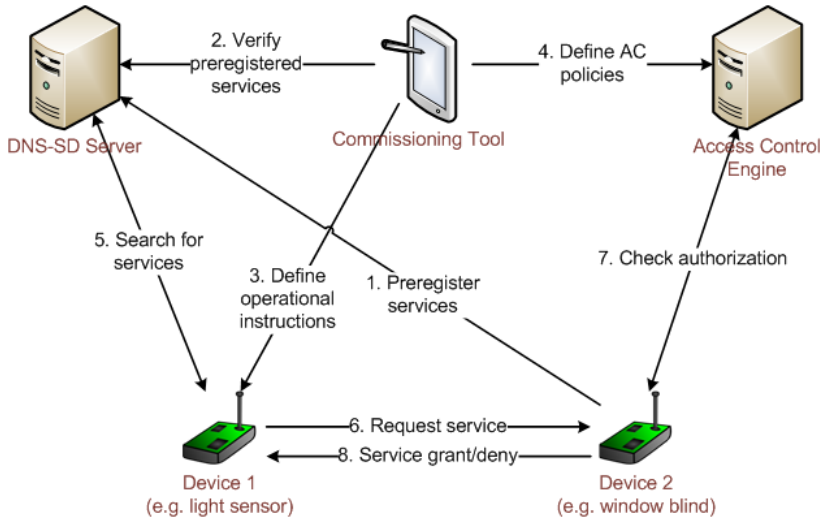


Figure 4.1: Overview of the system components and interactions

DNS-SD Server is the central registration and lookup point for service discovery. *Devices* and the *Commissioning Tool (CT)* interact with the server to pre-register, verify, and query device and service information. Commissioning Tool is a resourceful mobile device in the form of a laptop or a tablet PC, for example. One of the devices (e.g., a window blind) is used to illustrate the service pre-registration and handling of service requests, whilst the other device (e.g., a light sensor) does the discovery and invocation of a service. *Access Control Engine (ACE)* is not part of the actual discovery process, but it is presented on the architecture in order to show the possible integration and interaction with such a system.

Interactions enumerated in the figure are defined as follows:

1. *Pre-register services.* Initial registration of device and service information by a device itself. The preregistered data is not available to be queried until it is verified by the CT. This step assures that a device has started operating as expected and no illegitimate data exists on the DNS-SD server during the operational phase.

2. *Verify preregistered services.* Verification of preregistered data by the CT. This process includes cross-checking the device and service information preregistered by a device, and enabling them to be discoverable

by other devices.

3. *Define operational instructions.* Setting the instructions that a device executes during its operation, including which devices and services to search for and to request. These instructions are provided by the CT to devices, or to controllers that run automated processes. The format of these instructions is out of the scope, and is not specified in this work.

4. *Define access control policies.* Defining access control policies which are applied by devices when they receive requests to access their services. This step is performed if there is an access control mechanism in the system and is out-of-scope of this thesis.

5. *Search for services.* Discovery of devices and services with specific types and properties via the *DNS-SD Server*.

6. *Request service.* Sending a service request to a device.

7. *Check authorization.* Authorization check done by the device that has received a service request.

8. *Service grant/deny.* Result of the service request.

We use a centralized approach in order to fulfill the performance and scalability related technical requirements. Aggregation of device and service information eliminates the usage of excessive multicasting on the network. Moreover, using the DNS caching mechanism helps reducing the number of redundant queries generated every time when searching for a service. Every DNS record has a Time-to-Live (TTL) value, and devices can cache the recently queried information until they reach their TTL values. Since DNS has a hierarchical infrastructure by design, the same can be applied here. Subdomains of the whole discovery domain can be delegated to different servers running on different physical pieces of hardware.

The network topology considered while designing the system is shown in Figure 4.2. A LAN infrastructure constitutes the backbone of the network. Devices are wireless nodes communicating over 6LoWPAN. An edge router is responsible for routing between 6LowPAN and LAN. It performs the IP header compression, although this is transparent for the endpoints. A Commissioning Tool is a mobile device which uses Wi-Fi connection in order to interact with other components on the network. All service discovery traffic runs on IP. Although DNS also supports TCP, UDP is used in the transport layer due to its lower overhead.

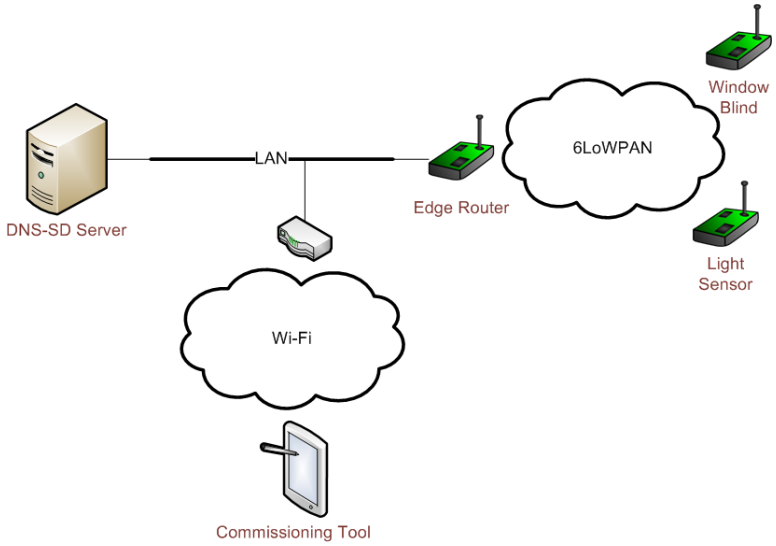


Figure 4.2: Network topology of the system

4.2.1 DNS-SD security layer

A *DNS-SD Security Layer* is introduced in our solution as a proxy between the conventional DNS server and other parties in the system (see Figure 4.3). It behaves like a firewall intercepting all DNS-SD traffic. It is responsible for determining what domain a request is associated with, whether the client is allowed access to that domain, and returning the appropriate information back to the requestor. If it is allowed to access, the security layer forwards the request to the DNS server, and returns the result to the requestor.

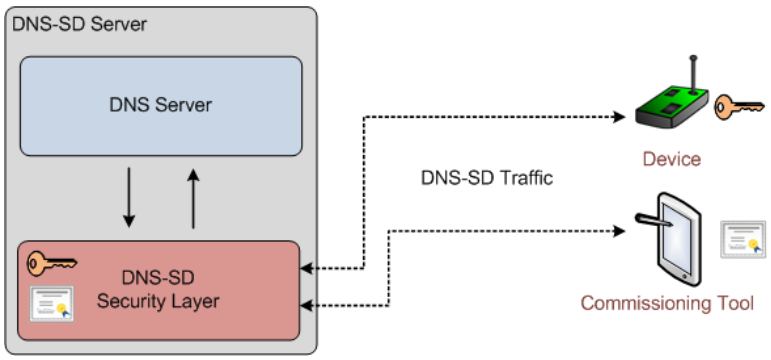


Figure 4.3: DNS-SD Server components

Most of the DNS server implementations also have simple access control settings. The Berkeley Internet Name Daemon (BIND), the most widely used DNS software on the Internet, for example, provides a mechanism to specify access control lists based on network addresses and TSIG keys,

ensuring that updates of DNS zone data can only be performed by authorized parties. However, the functionality is primitive, e.g., defining different TSIG keys for different clients is not possible. Moreover, depending solely on the access control mechanism of the DNS server implementation has a negative effect on modularity of the system, since those mechanisms may not be provided on every server implementation in the same way.

In our design, a pre-shared symmetric key is used between a device and the server in order to create signatures or to encrypt the traffic. On the other hand, the traffic between the server and the Commissioning Tool is secured using public key cryptography. Both parties hold certificates signed by a trusted party, and use those as their credentials. The security layer is the point on the server side that implements these functionalities.

The *DNS-SD Security Layer* also improves the flexibility of the system in terms of security. In case that resource constrained devices in the system make use of some lightweight algorithms for encryption or hashing, which are not specified in the DNS security standards, those algorithms can be implemented as a part of the security layer.

CT and devices are not aware of the location of the DNS-SD Server at first, and they need to discover it before using it. Since the security layer represents the DNS-SD Server for the CT and devices, server advertising is also one of the functionalities of the security layer. It is responsible for sending unicast or multicast messages that specify its IP address. Advertising messages are sent in the clear and do not have a particular security protection.

The DNS Server can be deployed on the same physical server with the security layer, on some other host on the network, or even somewhere else on the Internet. If it does not reside on the same host, then the traffic must be secured by some standard means, using an encrypted SSL/TLS connection, for instance.

4.3 Protocol specifications

In this section, details of the discovery protocol are described step-by-step using UML sequence diagrams. The interactions completed on the architecture figure is also presented for each step.

The protocol is explained using two types of devices for illustration. The first one is a window blind which provides a service to adjust the amount of light coming through the window. The other device is a light sensor which detects the current amount of light in the room it is placed in. The sensor can dynamically discover the blind in the same room, and queries the service provided by the blind in order to adjust the light in the room automatically. In the sequence diagrams, the window blind registers its services, then the sensor queries and requests the service that is provided by the window blind.

It is worth to be noted that only service discovery related interactions are explained and shown in detail in the sequence diagrams. For example, definition of the access control rules on the ACE is not detailed as it is not part of the discovery protocol.

We use a three-level design for the security of the system. Different levels of security can be achieved using this approach. Here, the security level is a system-wide decision, and should be determined according to the actual needs. Presenting the design with different security levels also allows better understanding of the functionality in each step. The security levels are the following:

- *Security level 0:* No security measures are employed for the protection of the device and service information.
- *Security level 1:* Authentication and integrity of the device and service information is achieved using message signatures generated using symmetric keys.
- *Security level 2:* Confidentiality of the sensitive data is achieved by encryption in addition to authentication and integrity.

Discovery process is explained in three parts: pre-commissioning, commissioning, and operational. Pre-commissioning is the phase after the installation of devices and before the commissioning is performed. Commissioning and operational phases are part of the BACS lifecyle as they are described in the corresponding background section. In each of these phases, the protocol is explained for the three different security levels.

4.3.1 Pre-commissioning

Preregistration of the hosted services is performed in this phase (see Figure 4.4).

After all devices have been installed and the network setup is complete, devices have their IPv6 addresses and they are part of a 6LoWPAN network. A *DNS-SD Server* has been installed (e.g., at a controller) and functional, but it has not been located by the devices yet. An ACE is also installed and accessible by the devices for e.g. by querying the DNS-SD server during the operational phase.

Server discovery

Finding location of the DNS-SD Server is the first step of the discovery process. This can be done in two different ways: *active* or *passive* discovery. In active discovery, a device sends a multicast query, and the server replies with a unicast response which contains its IP address (see Figure 4.5). In

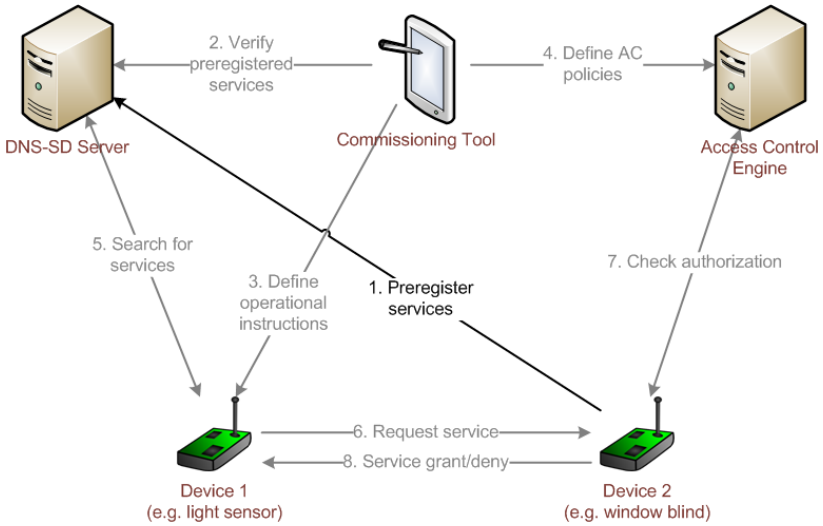


Figure 4.4: Steps completed in pre-commissioning phase

passive discovery, the server advertises its IP address periodically (see Figure 4.6). Both modes are supported by the designed protocol.

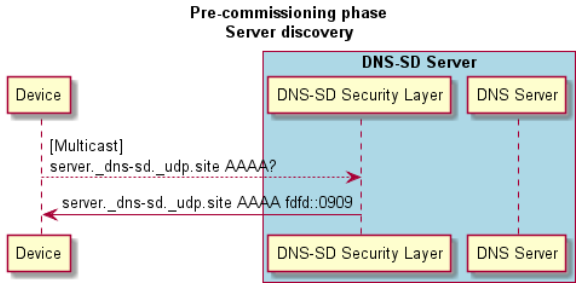


Figure 4.5: Discovery of the server by devices (active discovery)

In server discovery process, multicast DNS messages can be sent to multicast IP addresses defined by Multicast DNS (mDNS) [6] or Extended Multicast DNS (xmDNS) [22]. Those specifications define how to perform DNS-like operations on the local link and on a multi-hop LAN without a conventional unicast DNS server. It should be noted that both specifications have been published as Internet Drafts by IETF, and they have not been standardized yet. However, since the Security Layer is performing this functionality, we can use a standard DNS server.

An A or AAAA DNS RR is used in query and response messages in server discovery. The name of the record is defined as `server._dns-sd._udp.site`. Devices do not have the discovery domain information at the first place as that configuration is done by the commissioner later. `.site` top-level domain name is borrowed from xmDNS specification to indicate that the query

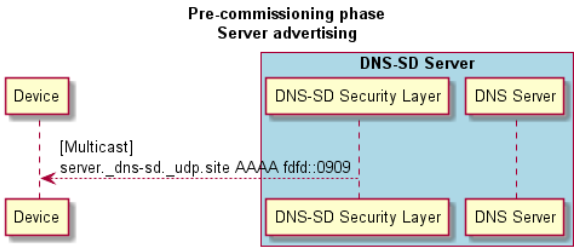


Figure 4.6: Server address advertising (passive discovery)

or response is valid within the site local, i.e. building network. The first part of `_dns-sd._udp` denotes the application protocol name, while `_udp` stands for UDP as the transport protocol. Those naming conventions are defined in the DNS-SD specification [5]. Finally, `server` part indicates that the IP address queried or sent is of the *DNS-SD Server*.

DNS-SD Security Layer is responsible for handling server discovery messages or multicasting its location on the server side. *DNS Server* does not take part in this process.

Security level 0: no security

After discovering the server, devices register the following information:

- *Unique Identifier (UID)* of the device, possibly based on the EUI-64 value of the network interface. UID of a device is also put on the device in a visually readable form (e.g., via a barcode).
- *IP address* of the device.
- *Service information*, i.e. for each hosted service:
  - *Service type*
  - *Port number*
  - *URI path*

This initial registration of device and service information is called *preregistration* process, since the information registered by a device at that stage is not yet activated for use. It has to be checked by the commissioner before it becomes accessible for discovery. Preregistration process is depicted in Figure 4.7 and is explained step-by-step below.

Basically, the device creates and sends *Dynamic DNS Update* messages to the server in order to register the device and service information. It first sends an update message with an A/AAAA record containing its IP address. Then it registers its services one by one using SRV and TXT records. It creates PTR records pointing to each registered service as the last step. It

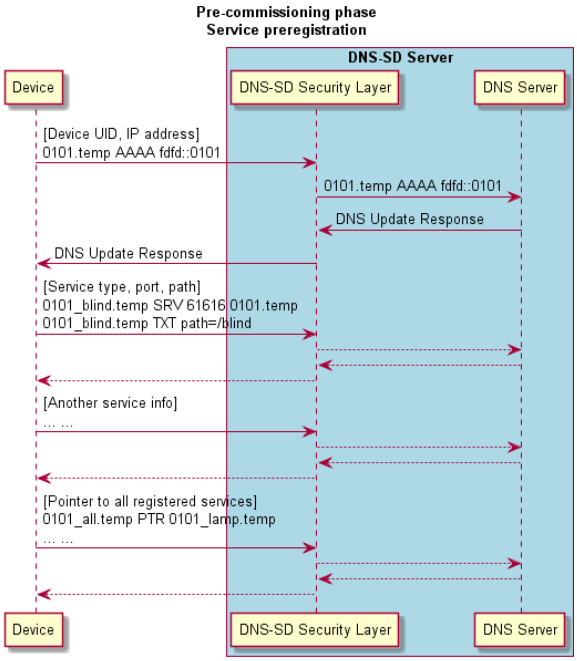


Figure 4.7: Service preregistration

is also possible to send all those records using fewer DNS update messages by putting more records in a single packet. Here, a separate packet per piece of information to be registered is used for clarity.

The device address is registered with the name `0101.temp` in the sequence diagram. `0101` denotes the UID of the device. Although here a hexadecimal notation of two-byte identifier is used for the device UID, a scheme should be defined for generating and denoting them in real-life systems. `.temp` top-level domain is used for all pre-registration requests. Notice that neither the server is configured, nor the devices are aware of the actual discovery domain yet. The server is running out-of-box, and accepts registration requests only for the `.temp` domain.

Each service instance is registered with an SRV/TXT record pair. Both records share the same name, which is generated automatically by the device. Auto generation of the name `0101_blind` is done by concatenating the device UID with the service type, i.e. `_blind`. Service types are typically defined by a Standards Development Organization (SDO), and is ultimately registered with Internet Assigned Numbers Authority (IANA) [5]. If there happens to exist more than one service instances with the same type on a device, an additional sequence number can be appended to that name. Port number (`61616`) and hostname (`0101.temp`) are set by the SRV record, while the service path (`/blind`) is specified with the TXT record. In case of additional service properties, they are stored with the same TXT record. Data

syntax and general format rules for DNS-SD TXT records are laid out in the DNS-SD specification [5]. Another SRV/TXT pair is used for each service hosted by the device.

A PTR record with the same name (`0101_all.temp`) is created for each registered service. These records is generated in order to allow the *Commissioning Tool* to get the list of all services that belong to a specific device during the verification process. The name of the PTR record is generated by appending the fixed `_all` suffix to the device UID.

DNS-SD Security Layer behaves transparently during the whole preregistration process. It just forwards the DNS update messages to the DNS server, and returns the results to the requesting devices. As the underlying transport protocol (UDP) is not reliable, devices may retry sending a DNS update packet after a random delay period in case of no response from the server within a pre-defined time.

It should be noticed that preregistration is done without any authentication. For the authenticity, integrity, and privacy of the preregistered data, countermeasures are defined in other security levels. However, neither devices nor the server authenticates the other party during the process. As there is no shared secret between devices and the server yet, there is no way to prevent registering bogus information or introducing a fraudulent server into the network at discovery protocol layer under the current assumptions. Other measures should be taken to prevent denial-of-service attempts, e.g., by monitoring the network, or flood detection for the discovery server and can be implemented in the security layer.

Security level 1: authentication and integrity

In this security level, origin authentication and integrity of the preregistered information is achieved. Message Authentication Codes (MACs) of the preregistration data are calculated by devices for this purpose. This calculation is based on the assumption that all devices have initial factory keys installed. In addition, those device factory keys are only available to the *Commissioning Tool* in the system. Hence, generated MACs can be verified by the *Commissioning Tool* in the commissioning phase. Figure 4.8 shows how service pre-registration process is performed using MACs.

The generated MACs cover the following data:

- device UID and IP address;
- service type, port number, and path properties for each service;
- the list of service names

A separate MAC is calculated for A/AAAA record with device UID and IP address, for each SRV/TXT record pair with service information, and for all

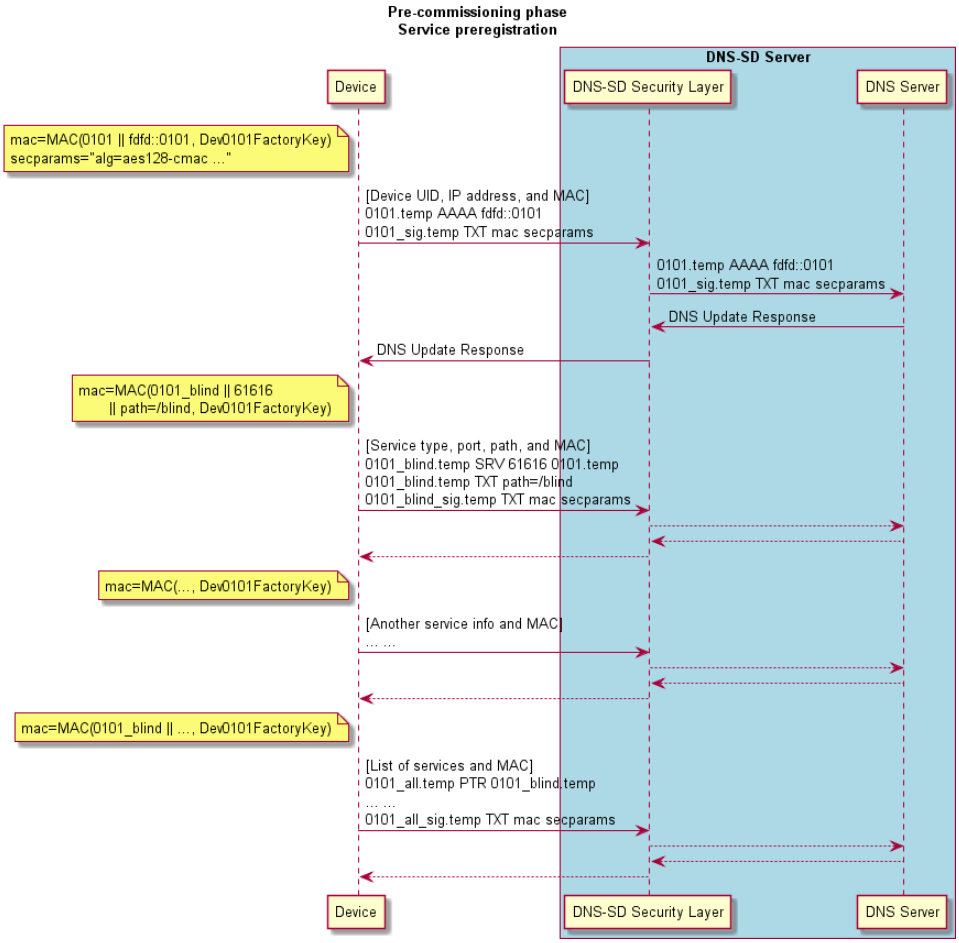


Figure 4.8: Service preregistration in security level 1

PTR records. Those MACs along with the corresponding security parameters which are used in calculation (e.g., algorithm used) are stored using separate TXT records. Names of the TXT records with MACs consist of the name of the corresponding record set plus `_sig` suffix, `0101_blind_sig` for the SRV/TXT record set with the name `0101_blind`, for instance. This information is registered with DNS-SD, and will be verified later during commissioning.

Security level 2: confidentiality

In order to provide confidentiality of the device and service information, symmetric encryption is utilized. Device and service information is encrypted using device's factory key. Although we mention encryption, the cryptographic algorithm used should be an authenticated encryption algorithm that provides both authentication and confidentiality of the message.

A major difference from the other security levels is that only TXT records are used in the preregistration process as it can be seen in Figure 4.9.

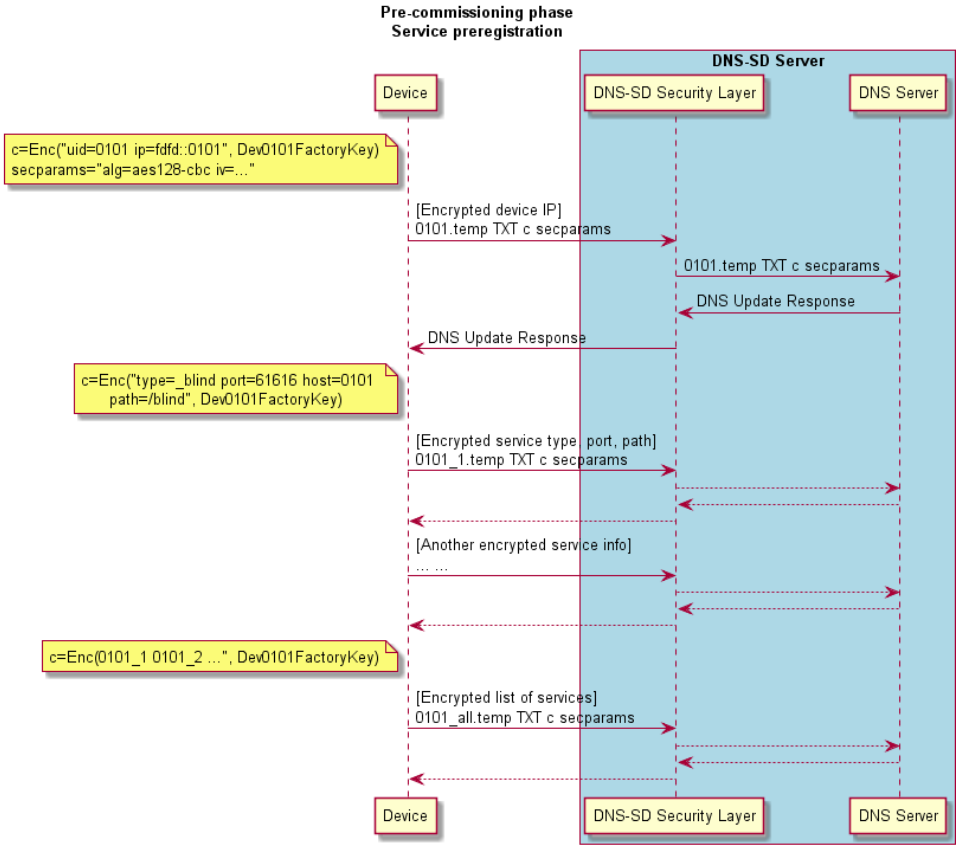


Figure 4.9: Service preregistration in security level 2

Data to be encrypted is constructed as *name=value* pairs with the same format as in the data field of DNS-SD TXT records. Device UID and IP address, each service information, and the list of services are encrypted and stored separately. Security parameters are also stored along with the encrypted data. Algorithm used, and initialization vector (IV) are some examples that can be included in security parameters. Device UID is not considered confidential, albeit it is included in the encrypted data to bind the information to the device UID. It is a part of every TXT record name that is to be registered as it is used by the commissioner to query those data later on. TXT record name of each service instance is labeled with a sequence number instead of service type, appended to the device UID. The sequence number is used for creating names for different service instances in an automatic way. Service type is a part of confidential data, and encrypted together with service port, host UID, and service path. Notice that, *DNS-SD Security Layer* is transparent during pre-registration as in the other

security levels since it does not possess the device factory keys perform any verification.

4.3.2 Commissioning

In the commissioning phase, verification of preregistered services, and definition of operational instructions and relevant access control policies are performed by the *Commissioning Tool (CT)* (see Figure 4.10).

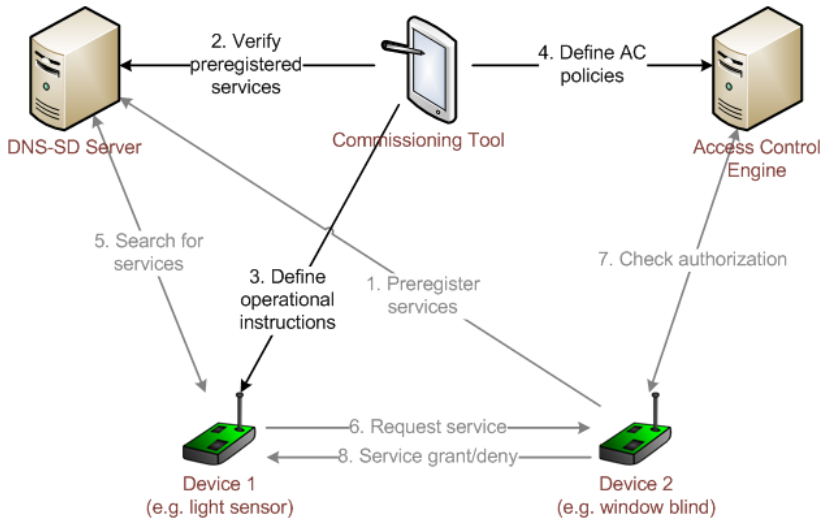


Figure 4.10: Steps completed in the commissioning phase

Commissioner uses a floorplan and goes to each device to perform the specified tasks. CT can access the server, however it can not communicate with the devices since it does not have their IP addresses yet.

Main assumption in this phase is that both CT and the security layer on the server side have public and private key pairs, and certificates signed by a common trusted party. The certificates can be signed with the private key of the owner organisation or company, for instance. A secure channel is created between the CT and the server using those certificates beforehand, and all messages which are part of the discovery protocol are transmitted over this channel. The traffic can be secured via DTLS as it consists of datagrams, while SSL/TLS can also be used since DNS messages can be sent over a TCP connection as well. Communication between CT and server is always encrypted in all security levels. However, verification of preregistered device and service information varies depending on the security level.

Security level 0: no security

Commissioning of a device begins with fetching the list of all preregistered services of that device. Figure 4.11 presents the sequence diagram of this

process and explained in detail below.

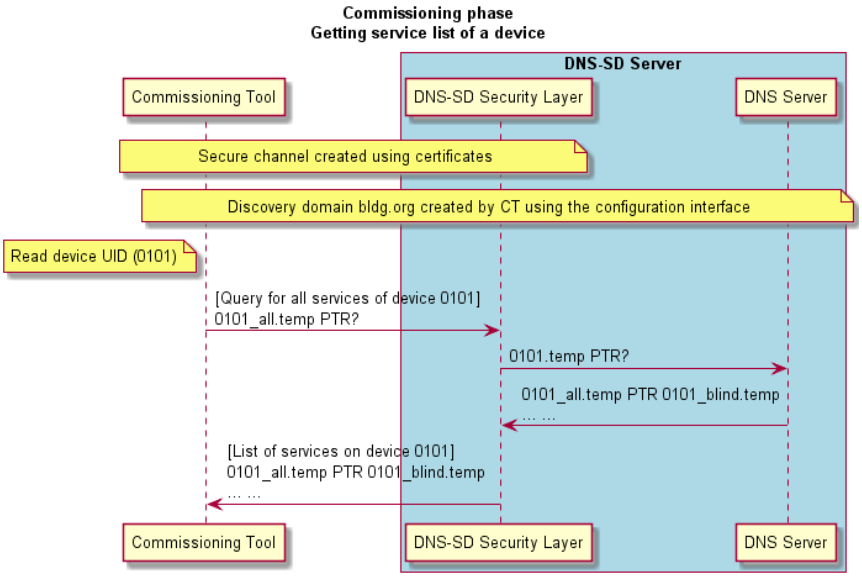


Figure 4.11: Querying all services preregistered by a device

Before starting commissioning, another task of the commissioner is to make necessary configurations on the DNS server. Creating the discovery domain on the server is one of them. This domain name will be used in all DNS-SD queries and responses during the operational phase. CT connects to DNS server, e.g., via its configuration interface, and creates the discovery domain (`bldg.org` in Figure 4.11) as the first task in this phase.

After the creation of a secure channel between CT and the server, and configuration of the actual discovery domain, CT reads UID of the device that is to be commissioned, for example, by scanning a barcode. Then it queries the server for all services preregistered by that device. This is done by creating and sending a standard DNS query with a specially named PTR record (`0101_all.temp`). As the requested PTR records have been created on the `.temp` domain by the device during pre-registration, server returns all PTR records with that name, each with a service name as value.

CT then queries for the information for each of the received services in order to verify. As it is shown in Figure 4.12, this is performed by querying *ANY* records with the name of service (`0101_blind`). Corresponding SRV and TXT records are returned as response. Returned service information is checked by the commissioner to verify if it matches with intended type of device. Service type, port number, service path, and hosting device are checked if they are consistent. This verification ensures that the device has discovered the DNS-SD server and pre-registered its services successfully. After checking, CT prepares a DNS update message to register this verified

information to the actual discovery domain (`bldg.org`). This process is called *enabling* a service as the information registered to this domain is now available for discovery. If CT receives no pre-registered data for a device UID, or the received data does not match with the intended type of device, either there is a problem with the installation of device or pre-registered data is corrupted.

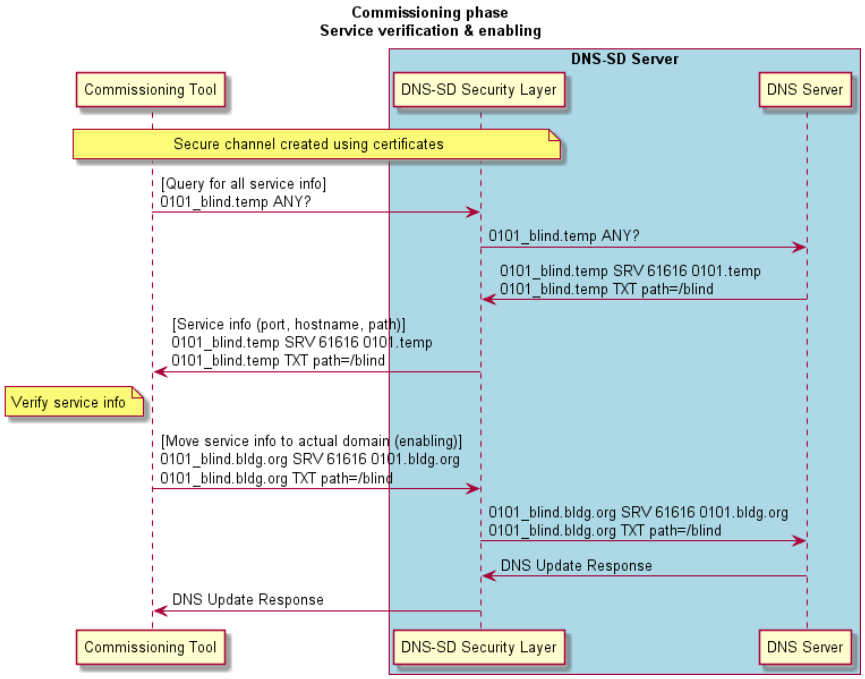


Figure 4.12: Service verification and enabling by CT

Pre-registered device IP address is queried and registered at the discovery domain in the same way. CT sends a query with UID of the device, and with type A/AAAA, then it creates a record on the discovery domain (`bldg.org`) with the returned data. Verification of the returned IP address is not possible since CT does not have this information before. So, only verification is that the device has been able to preregister its IP address successfully. After this step, CT knows the IP address of the device and it can communicate with it directly if required.

During the verification and enabling, the device and service information, CT also sets Time-to-Live (TTL) values for each resource record it registers at the discovery domain. Every DNS resource record has a TTL value which specifies the maximum time period that the corresponding record can be cached by the clients who use it. Therefore, this value is used by devices to cache the queried device and service information.

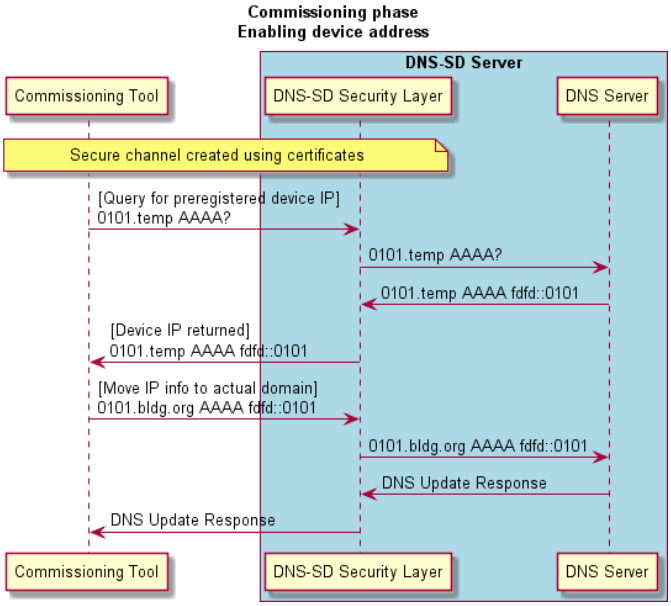


Figure 4.13: Getting and enabling device IP address

Security level 1: authentication and integrity

Verification of preregistered device and service information is slightly different in security level 1. Basically, CT also needs to get and verify the MACs which have been computed by the devices. As an example, verification of a service information is shown in Figure 4.14.

Only difference from service verification in *security level 0* is that CT queries for the TXT record that contains corresponding MAC (`0101_blind_sig.temp`) along with the SRV/TXT records (`0101_blind.temp`). Upon receiving the service information and the MAC value, it verifies the MAC with the corresponding symmetric key, i.e. factory key of the device, in order to ensure the authenticity and integrity of the received information. After verifying the MAC and cross-checking the service information, CT enables the service by registering it with the actual discovery domain.

It is to be noted that any shared keys between the server and the devices, which can be used in the operational phase for authentication and encryption purposes, do not exist yet. In order to address this issue, an *operational key* has to be created for each device by CT. This key then has to be sent both to the server and to the device. This step is performed if authentication or confidentiality is aimed during operation, i.e. in security levels 1 and 2. Sequence diagram of generating an operational key is presented in Figure 4.15.

CT randomly generates an operational key for the device (`Dev0101OpKey`). Then it can send the generated key in a TXT record (`0101_key_server`) to

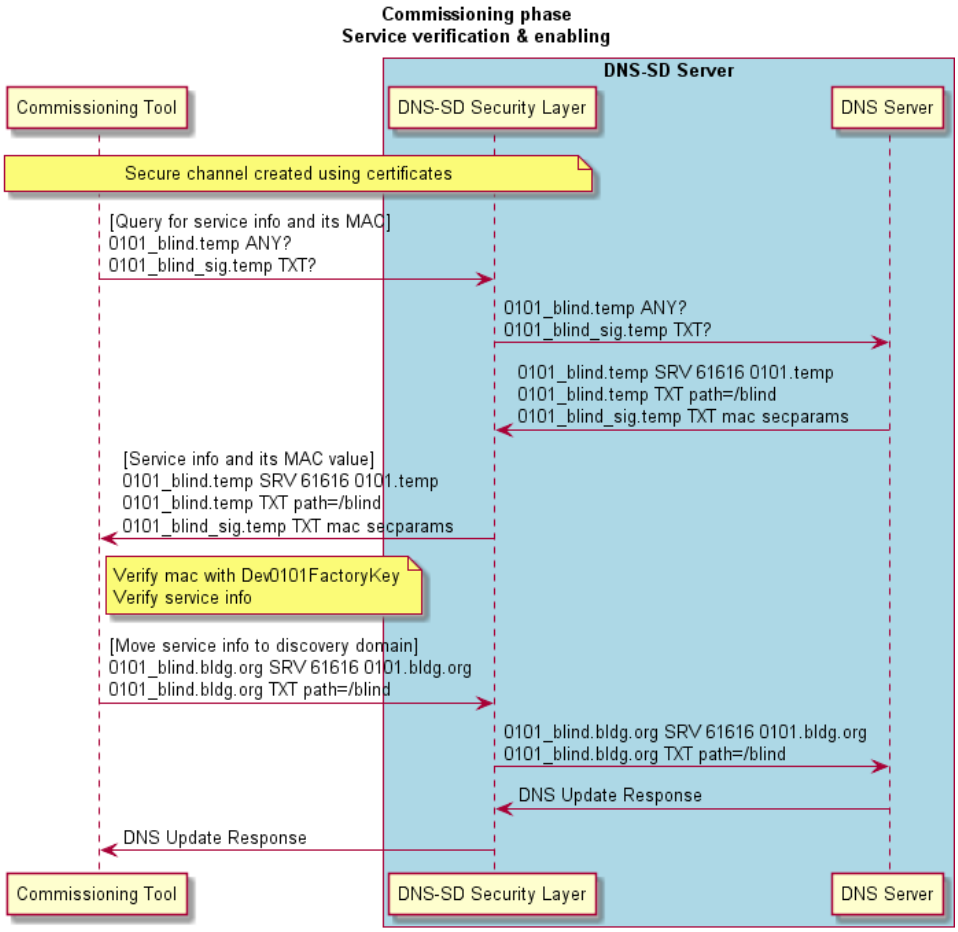


Figure 4.14: Service verification and enabling in security level 1

the server directly, since the channel in between is already secure. *DNS-SD Security Layer* processes this record and stores the operational key, associating it with the device UID `0101`. In addition, CT encrypts the generated key using the device's factory key (`Dev0101FactoryKey`) and sends it to the server to be stored at the DNS server. This encrypted version of the operational key can be fetched by the device, and decrypted using the device's factory key. Encrypted operational key on the server can be used as a backup mechanism, since CT is also able to send it directly to the device during commissioning.

Security level 2: confidentiality

In this security level, all device and service information preregistered is stored encrypted on the DNS server. Thus, CT needs to query those TXT records which contain encrypted preregistered data, and decrypt them be-

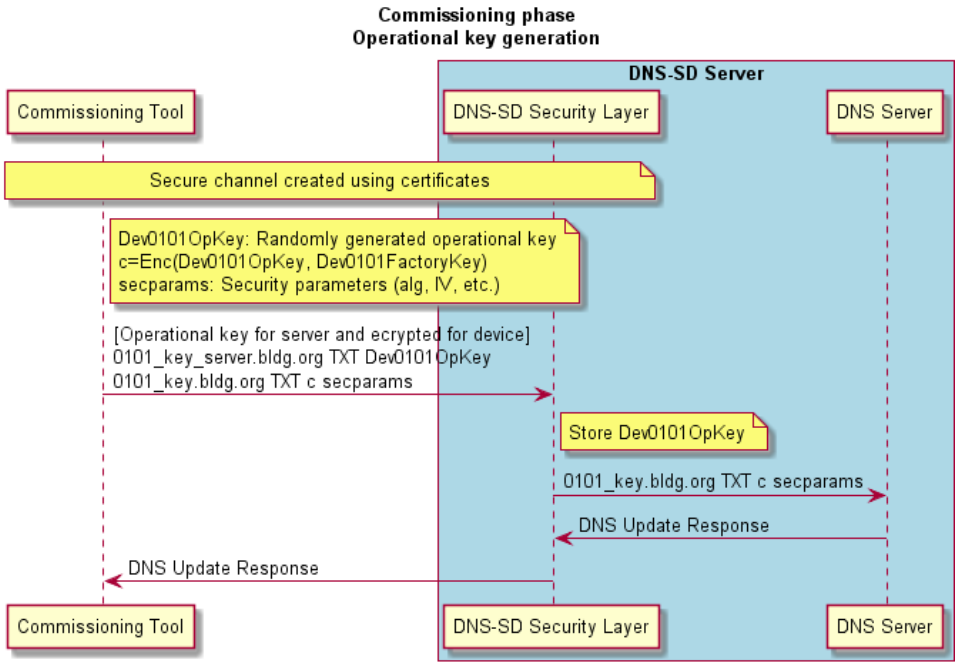


Figure 4.15: Operational key generation by CT

fore performing further operations.

We can consider the verification of a service information, for instance. As it is presented in Figure 4.16, after getting all service names preregistered by a device, CT first queries the TXT record with the sequenced service name (`0101_1.temp`), because devices pre-register their services with sequence numbers, and encrypt the corresponding service types as those are considered as sensitive data in this security level. Returned data is decrypted using the device's factory key, and decrypted service information is checked and moved to the discovery domain as in previous security levels.

As the last step of commissioning, CT connects to the device via a predefined configuration port. Device still lacks some configurations that it will need during its operation. These configurations include which operational instructions to execute (i.e. which services to discover and call), which domain to use while sending discovery queries, and what key to use for creating and verifying message signatures or encrypting the traffic. Sending operational configuration is shown in Figure 4.17

Service grouping

Groups can be used to express a set of devices supporting a specific service (e.g., HVAC equipment controlled by the closest temperature sensor). It is also necessary when a set of devices should react synchronously to a

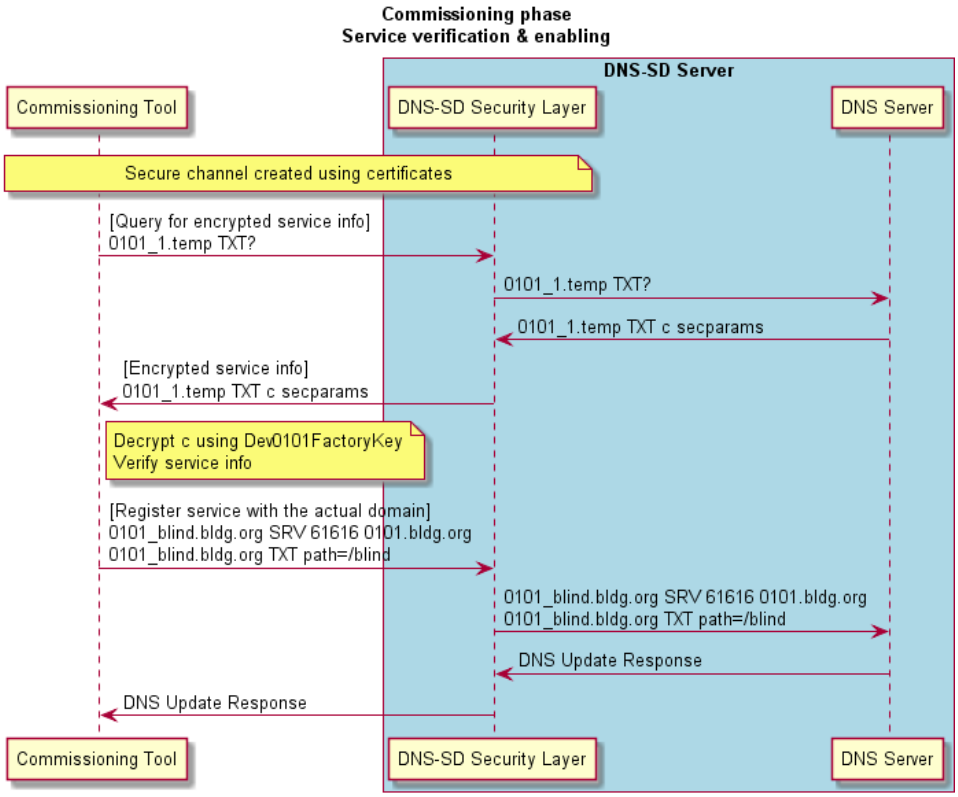


Figure 4.16: Service verification and enabling in security level 2

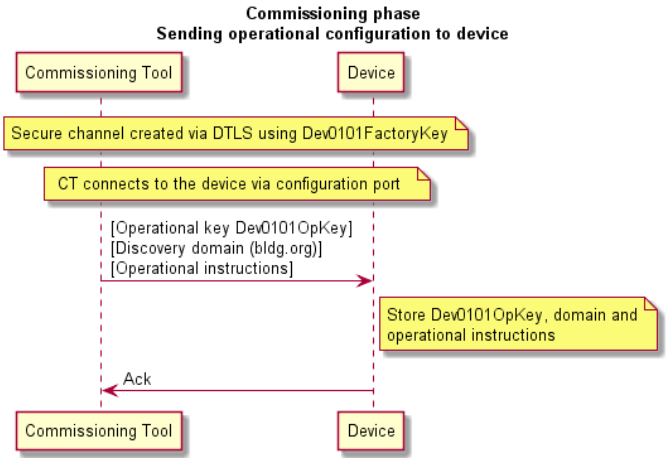


Figure 4.17: Sending operational configuration to device

sequence of commands. A common example is that a subset of lights in the building are dimmed to the same level, set to the same color, or switched off simultaneously.

Addressing a group can be done in two ways:

1. *Addressing each group member individually (i.e. serial access):* Additional PTR records can be used to specify the possible service discovery queries. For example, to support queries like "all window blinds in floor 2, room 5", "all lamps in floor 3", or "all services within `bldg.org`", PTR records with the name of the service types can be created by CT. Values of these PTR records refer to the names of the services to be filtered. Commissioner should determine what kind of filtering is needed, before creating PTR records for filtering services.

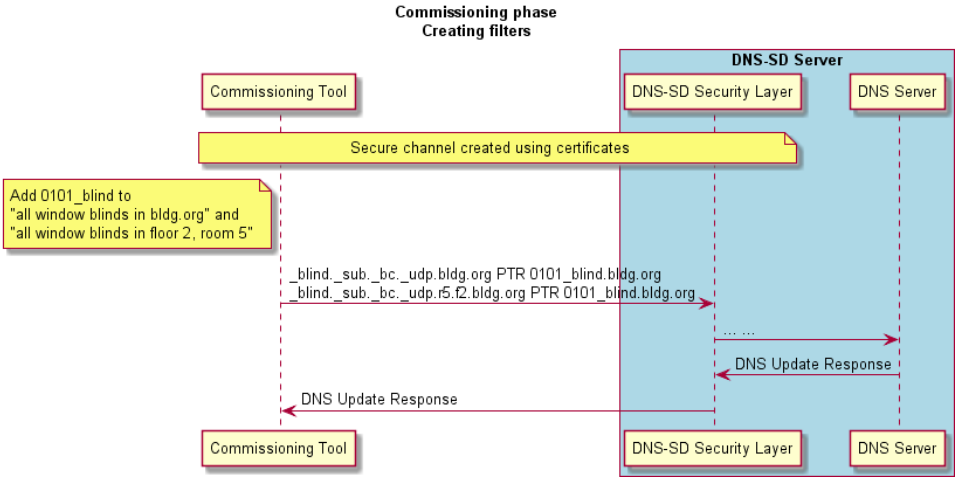


Figure 4.18: Creating filters with PTR records

In Figure 4.18, the service `0101_blind` is associated with two filters: "all window blinds in `bldg.org`" and "all window blinds in floor 2, room 5". In the second one, for instance, `_blind._sub._bc._udp` specifies the service type as subtype (`_blind`), type (`_bc`), and protocol (`_udp`). Naming of the service types are defined in the DNS-SD specification. `r5.f2` is the subdomain for "floor 2, room 5", and can be created using any naming convention. So, the second PTR record is for filtering on all window blinds within domain `r5.f2.bldg.org`.

2. *Defining a multicast address for a multicast group:* In this case, each member device must enable reception of messages sent to the specified group multicast address. Furthermore, every member must have identical port number and path, since the requests are specified in a single multicast message.

   As an example, Figure 4.19 shows defining a multicast group of lamps on the same floor. Creating a multicast group is similar to creating a device with a service. An AAAA record is created to enable resolution from group name to multicast address, and an SRV/TXT pair is

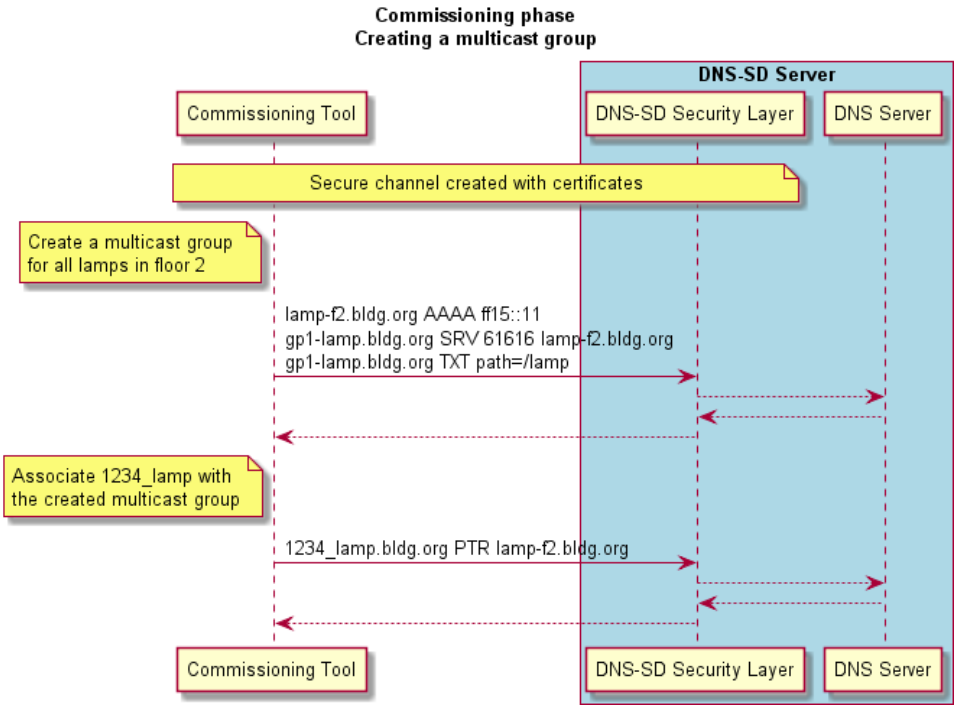


Figure 4.19: Creating a multicast group

used to specify the path and port. There is no specific convention for group naming. However, it can be automated using service types and sequence numbers for distinguishability. After creating the multicast group, a service instance on a device (`1234_lamp`) is associated with the group.

4.3.3 Operational

After completion of the commissioning steps, a device starts operating as it has been set by the operational instructions. Device's services have been verified and enabled, and the device has been authorized to discover and request other services on the BACS network. Authorization of a device is done in security levels 1 and 2 by sending the security credential to the device, i.e. the operational key.

When a device needs to call a service or a set of services, it searches for the required service(s) using the discovery protocol first. After receiving the service invocation information from the DNS-SD server, it can send requests to discovered services. Service request is granted or denied by a target device based on the access control policy, if an access control mechanism exists. These actions are depicted in Figure 4.20. Again, it should be noted that access-control related interactions are not detailed in the protocol

specifications.

All of the actions numbered from 5 to 8 on Figure 4.20 are performed only if a device queries for the required service(s) for the first time. Devices can cache the results of discovery and access control queries, and they can directly send service requests or process those requests thereafter.

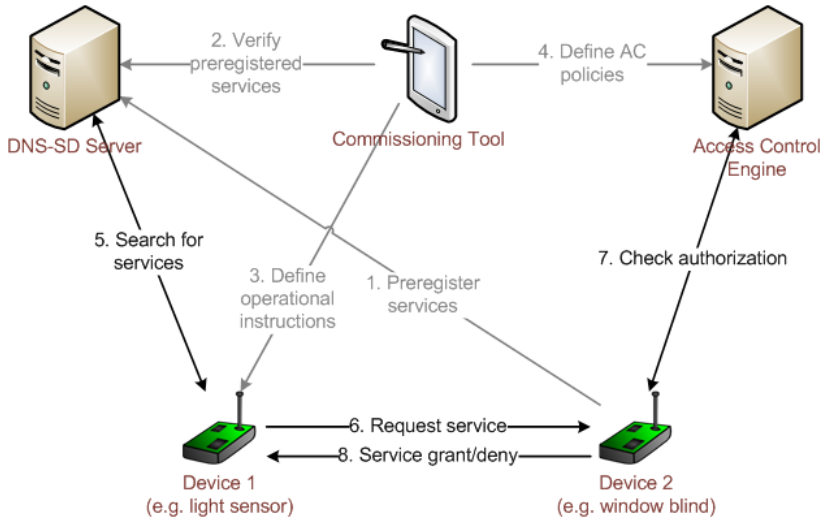


Figure 4.20: Actions performed in the operational phase

Design details of the discovery protocol is discussed for different security levels in the following subsections. Protocol is explained with a sample scenario. A device with a light sensor searches all window blinds in a specific room, and then sends request to a discovered service.

Security level 0: no security

The light sensor executes the operational instructions, when a triggering occurs, e.g., when a predefined threshold for the amount of light is reached. The sensor first sends a query to the discovery server in order to get the list of all required services, however after checking if that information does not already exist in cache (see Figure 4.21). This is performed with a PTR query which includes required service type (`_blind._sub._bc._udp`) and subdomain information (`r5.f2.bldg.org`). This PTR query is a part of the operational instructions set by CT during commissioning.

Upon receiving the list of matching `_blind` services, the light sensor queries for the service invocation details of each matching service. In Figure 4.21, port number, hosting device name, and service path of a matching service (`0101_blind`) are queried as an example. Then the IP address of the hosting device is resolved using an AAAA query. After caching the received results if it is not done yet, the sensor sends a service request command to

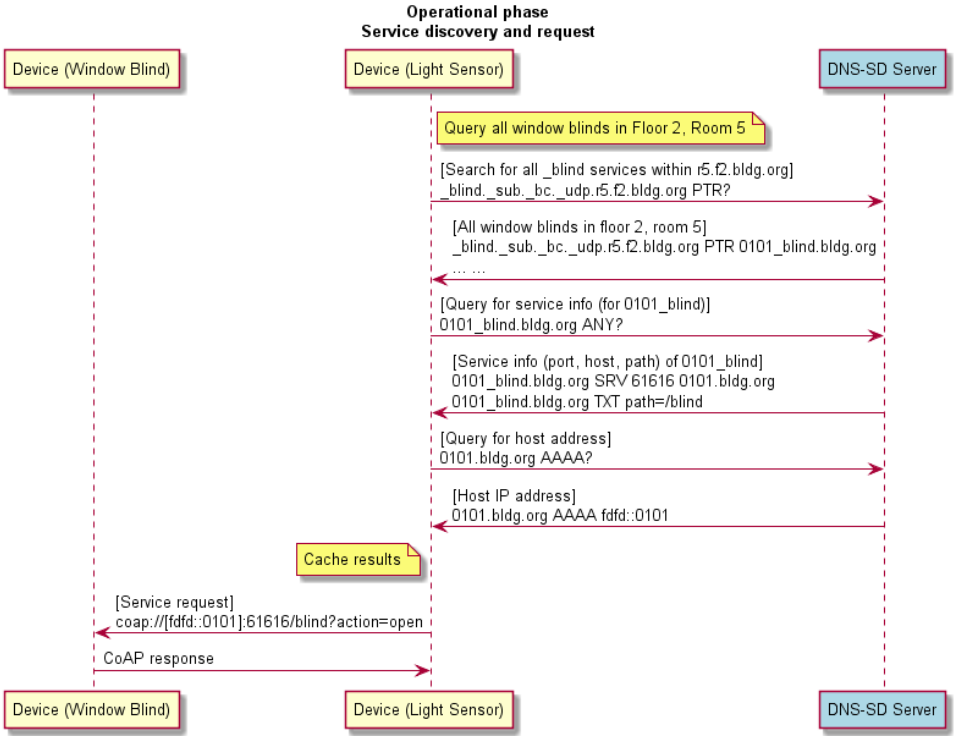


Figure 4.21: Service discovery and request

the discovered window blind, in this case using the CoAP protocol. Target device applies the access control policies if it exists, and then processes the service request. Possible access control interactions are not shown for clarity on the sequence diagram.

In this security level, neither sender authentication nor traffic encryption is performed.

Security level 1: authentication and integrity

In order to protect the discovery request and response messages, TSIG records are employed in this security level. As it is stated in Section 2.2.5, TSIG is a standard mechanism to provide authenticity and integrity of the exchange between two DNS entities based on a shared secret. After commissioning, each device shares an operational key with the *DNS-SD Security Layer*, and this key can be used to calculate and verify TSIG MACs. Figure 4.22 shows the discovery process with transaction signatures.

Message sequence is basically the same as in the *security level 0*, except each message includes a TSIG structure as the last DNS record in the packet. Both parties verify the included message signature first, before processing the message content. *DNS-SD Security Layer* determines which key to use

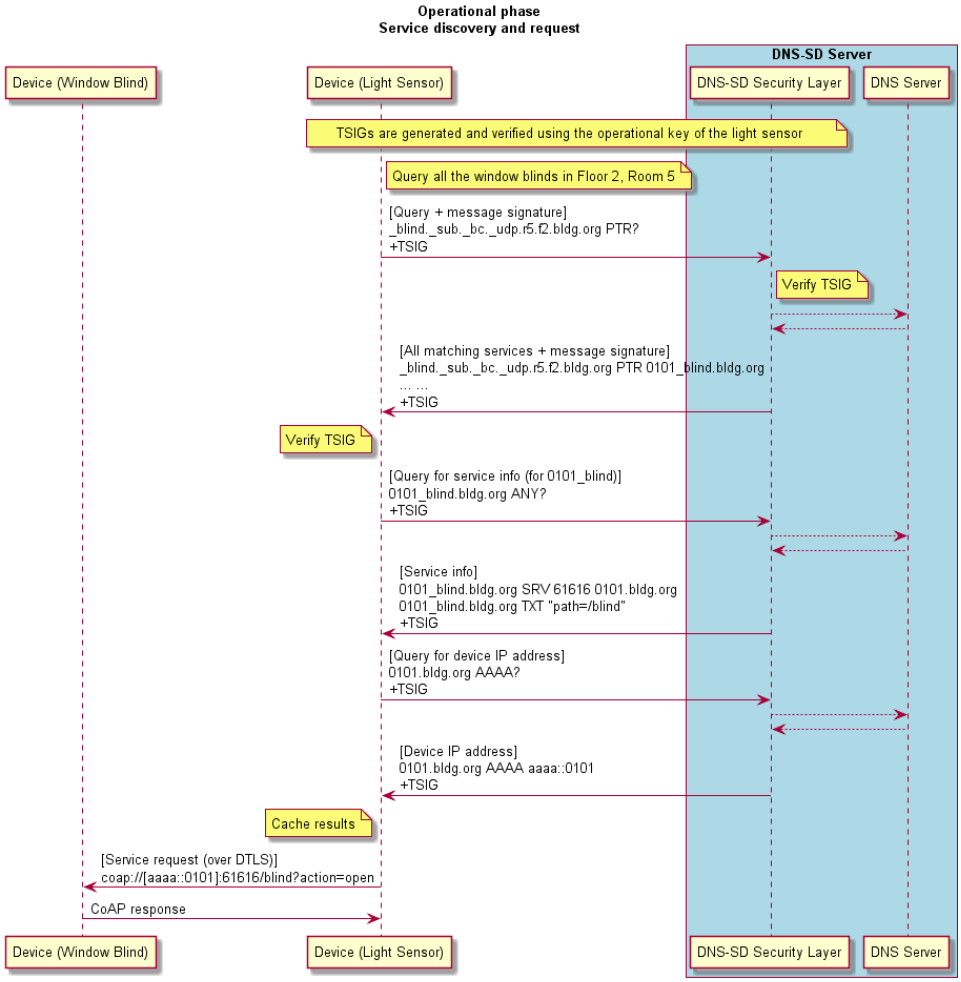


Figure 4.22: Service discovery and request with transaction signatures

to verify a TSIG, using the *Name* field of the TSIG record. It uses the same key to generate the signature for the corresponding response message. After getting the invocation information for a service via discovery protocol, it is sensible to use DTLS to further secure the service requests in both security levels 1 and 2.

Security level 2: confidentiality

Confidentiality of the discovery messages is achieved by encrypting the packets between a device and the server in this security level. Encryption of the packets can be performed through a secure channel created using DTLS. Device's operational key is employed to create this channel. There is no additional signatures used per message, as authenticity of the sender is also achieved by DTLS. It should be stated that, the traffic between devices while

calling discovered services must be protected by some means as well, using DTLS, for example. Securing the service invocation traffic is considered out of the discovery process, however an equivalent level of security must be maintained there also. Figure 4.23 presents the discovery message sequence through a DTLS channel.

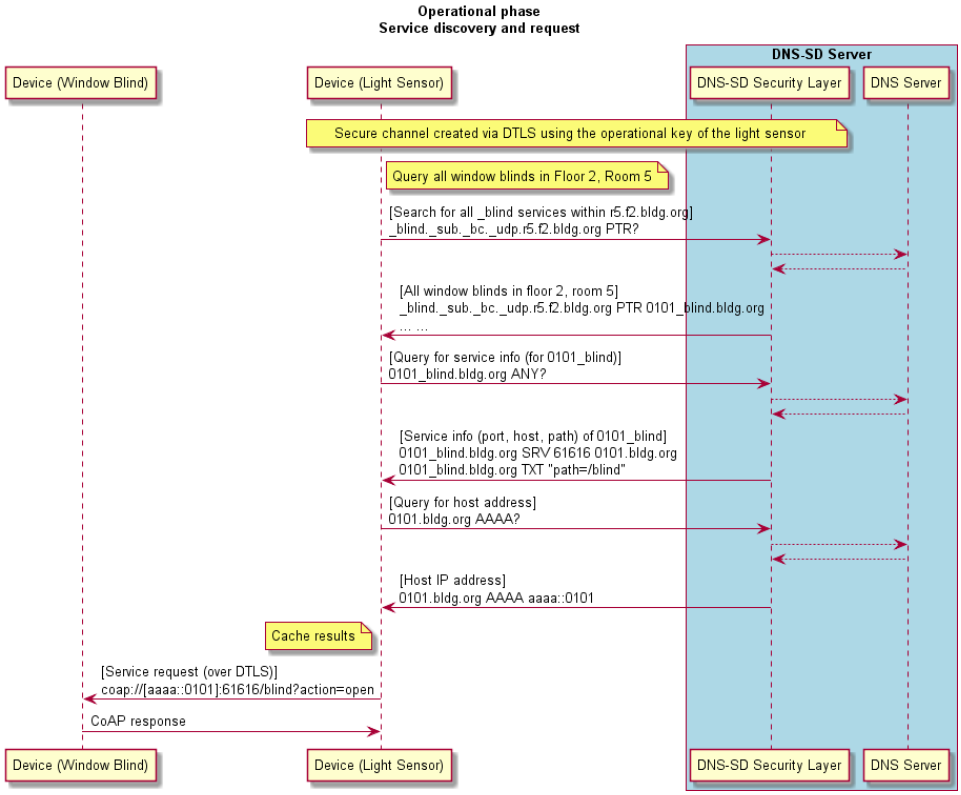


Figure 4.23: Service discovery and request over DTLS

4.4 Other considerations and open challenges

An architectural framework and a DNS-SD based service discovery protocol were proposed in this chapter. Our approach focuses on developing a discovery solution for commercial building environments while providing required security measures. Although proposed solution is detailed following the BACS lifecycle in previous sections, there are other considerations regarding the overall design:

- *Discovery of legacy systems.* It is possible to have some subsystems with devices communicating via legacy protocols. These subsystems are supposed to connect to the IP network through gateways. Such a

gateway presents a list of services within the subsystem on behalf of the legacy devices. Then, discovery of the gateway is done as for other devices communicating over IP.

- *Battery-powered sleeping devices.* Discovery of sleeping devices is not very different from discovery of legacy gateways. It is expected that a proxy will handle communications for the sleeping device. The proxy will take over all the services of the sleeping device during the set-up process. Those services then can be registered by the proxy and consequently discovered like any other device.

- *Naming conventions.* Names which are used to define service types, groups, and paths are expected to be standardized by the standardization organizations in the BACS field.

- *Removing preregistration data on the* `.temp` *domain.* Removal of stale data on DNS server can be performed by the commissioner using the configuration interface. Some DNS server implementations can also be configured to delete stale records, such as in Microsoft DNS Server.

- *Unreliable messaging.* As UDP is used for transporting discovery messages, system components need to implement a re-transmission mechanism against packet loss.

Lastly, we discuss open challenges remaining in this work and possible future directions:

- *Denial-of-service during preregistration.* With the given assumptions, our design does not propose a specific mechanism against denial-of-service attempts during service preregistration. Mutual authentication of devices and the DNS-SD server can help but is not possible out-of-the box. Therefore other mechanisms based on intrusion detection may be required to prevent this.

- *Distribution of operational keys.* Distribution of operational keys is done by CT using DNS messages and sending directly to devices thorough a configuration port. A more dedicated protocol can be integrated into the design for bootstrapping and key distribution purposes.

- *A more distributed design.* Current design is based on a central server as an aggregation point for service information in order to support operating on large networks. The solution can be extended or modified to become more hybrid instead of centralized, making use of (x)mDNS, for example. Single points of failure can be avoided or minimized against DoS attacks in this way.

- *Event notification.* Additional capability of subscribing and publishing service notifications can be included, like publish/subscribe mechanisms in UPnP and DPWS.

Chapter 5

Design validation and evaluation

This chapter contains the details of the prototype implementation and validation of the proposed design. We evaluate the prototype based on message sizes, memory requirements and time based efficiency. Furthermore, a validation of the proposed design is given in terms of functional and security perspectives.

5.1 Implementation

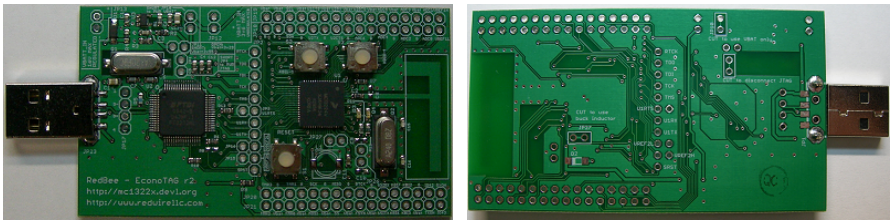
A model based on DNS-SD is proposed as a design solution for service discovery in BACS in the design chapter. Our design requires resource constrained devices to be able to create and parse various DNS messages. Furthermore, those devices need to generate and verify message signatures (TSIGs). These requirements constitute the bottleneck for the whole system. Therefore, a reference implementation of the client-side as a proof-of-concept is most important to show that the design is realizable for real world use. Implementation of the server-side is kept as simple as possible. In this section, we first provide the details of the hardware and the operating systems used in the setup. Then we present the details of the actual implementation, and supported scenarios in our implementation.

5.1.1 Reference implementation environment

Redbee Econotags [1] were used as wireless sensor nodes running the client-side implementation. Figure 5.1 shows the model hardware.

The hardware specification of Redbee Econotag are as follows:

[1] MC1322x Hardware Guide [Online] `http://mc1322x.devl.org/hardware.html`



(a) Top (b) Bottom

Figure 5.1: Econotag hardware

- ARM7 32-bit, 24 MHz microcontroller
- IEEE 802.15.4 2.4 GHz radio
- 96KB SRAM
- 128KB on-board flash memory
- 128-bit AES hardware encryption engine

We run the Contiki version 2.5 [8] on the Econotag nodes. Contiki is an open source operating system for embedded hardware with constrained memory and computing resources. It provides IP connectivity with the integrated *uIP* stack, which supports IPv4, IPv6, UDP, and TCP protocols. The core system is based on an event-driven kernel which effectively shares system resources between processes by treating each process as an event handler that run till completion and return back to the kernel when finished.

A standard PC hardware with Linux operating system is used for DNS-SD server counterpart.

5.1.2 Implementation details

Following two core functionalities are realized in this reference implementation:

1. Registration of hosted services by a device
2. Discovery and invocation of specific services by a device

The client implementation running on Contiki is built with the following capabilities:

1. Register the hosted services by creating and sending DNS Update packets
2. Register device IP address

3. Search for services with a specific type and subdomain by sending DNS queries to the server

4. Parse and cache the discovery response messages coming from the server

5. Generate and verify transaction signatures (TSIGs) for discovery messages

DNS-SD Client process is implemented as a service waiting for events coming from other processes running on the node. Basically, *uIP DNS resolver* implementation is used as a template for the general structure of the client. This is a service that can create and send DNS queries with type A records to a server in order to resolve IPv4 addresses. We extended it to handle AAAA, SRV, TXT, PTR, and TSIG DNS records. To achieve that, we implemented a lightweight DNS library for Contiki. This library was used to create and parse DNS packets with the mentioned specific records.

Two nodes are deployed as clients with the Econotag hardware. One of them provides a service which is accessible with a CoAP interface. The service provided is an emulation that simply toggles an LED on the node. The other device is programmed to search for a specific type of service which is provided by the first node. After getting the required information, the second device sends a CoAP request to call the service. Deployed components are shown in Figure 5.2 with all intercommunication numbered.

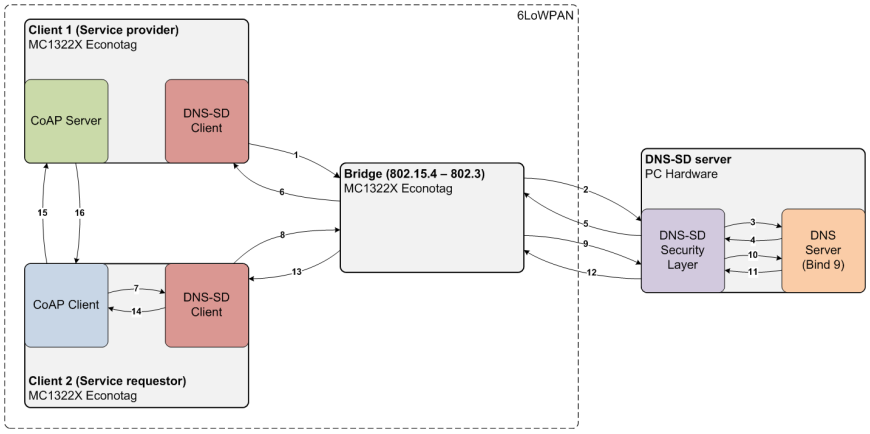


Figure 5.2: Implementation of service registration and discovery

Both client devices are enabled to use IPv6. They first automatically configure an IPv6 address, which is created based on their Ethernet MAC addresses, after the boot-up process. Another Econotag node is used as a bridge between the client nodes and the server. It uses Serial Line Internet Protocol (SLIP) to bridge the wireless 6LoWPAN network onto a PC via USB interface.

Devices can calculate and include message signatures in the form of TSIG records optionally. If this option is enabled, they append a TSIG record to DNS messages that they send to the server, and verify the TSIGs included in the received messages.

Currently, there are three HMAC algorithms standardized to be used with TSIGs: HMACs with MD5, SHA-1, and SHA-256. However, we decided to use a Cipher-based MAC (CMAC) with AES, since Redbee Econotag nodes already have a built-in AES encryption engine, and hash function implementations would consume more system resources than those used in AES-CMAC calculation. AES-128 with CBC mode is used for this purpose, as it is described in the AES-CMAC specification [33].

As the commissioning step is bypassed in the experiments, devices are preconfigured with the necessary operational parameters. Devices are assumed to have operational keys, discovery domain, and server address information during the tests.

DNS-SD Security Layer is deployed with minimal functionality. It waits for registration and discovery messages coming from devices, and forwards them to the DNS server after verifying TSIG MAC. *ldns* [26] library is used for handling DNS packets on the server side. AES-CMAC support is implemented in this part additionally.

We deployed *DNS-SD Security Layer* and the DNS server on the same PC hardware. *Bind 9* is used as the DNS server software. A discovery domain, `bldg.org` is created and preconfigured together with the `.temp` domain on the DNS server. *DNS-SD Security Layer* is configured to accept packets on port 53 (standard DNS port), while the DNS server runs on a non-standard port.

After Client 1 (see Figure 5.2) registers its service information and IP address, following records are created on the DNS server:

```
249B.temp.       IN AAAA aaaa::0250:c2a8:ca77:249b

249B_lamp.temp. IN SRV  0 0 61616 249B.temp
249B_lamp.temp. IN TXT  path=/lamp
```

The first record keeps the relation between device name and IP address. `249B` is the 2-byte UID of the device, and it is associated with the device's IPv6 address. The `SRV` record represents a `_lamp` service instance with port `61616` on the device. Service path `/lamp` is stored via a `TXT` record with the same name.

Before Client 2 sends the discovery query, necessary records have been copied to the domain `bldg.org` which would happen during the commissioning step:

```
249B.bldg.org.       IN AAAA aaaa::0250:c2a8:ca77:249b
```

```
249B_lamp.bldg.org. IN SRV  0 0 61616 249B.bldg.org
249B_lamp.bldg.org. IN TXT  path=/lamp

_lamp._sub._bc._udp.bldg.org. IN PTR 249B_lamp.bldg.org
```

Client 2 first queries the `PTR` record to get the list of all `_lamp` services. Then it resolves the service details and the device address with the returned value. Finally, it prepares and sends a CoAP request to invoke the service on Client 1. Client 1 toggles the LED after receiving the CoAP message, and sends back an acknowledgement.

Both the security layer on the server side and Client 2 include a TSIG record created with the pre-shared key while sending the discovery queries and responses. Generated TSIG records contain the following information:

- *Name:* The name of the key used in the MAC calculation.
- *Algorithm Name:* Name of the MAC algorithm, which is "AES-CMAC" in our case.
- *Time Signed:* Time of signature creation in seconds. A preset value has been used assuming that the device is aware of the current time, via a time server on the network, for example.
- *Fudge:* Seconds of error permitted. A value of 300 seconds has been used.
- *MAC Size:* Number of bytes in MAC. This is 16 bytes in case of AES-CMAC.
- *MAC:* Signature calculated. It covers the whole DNS message including the other TSIG fields.

5.2 Discovery validation

This section describes how the functional requirements are met with the proposed design.

1. *Discovery of the registration point.* Server discovery is performed in the pre-commissioning phase. It basically requires usage of multicast messages and can be done in two different ways (*active* or *passive* discovery).
2. *Service registration/announcement.* Device and service information is preregistered by each device in the pre-commissioning phase using DNS Update messages with A/AAAA, SRV, TXT, and PTR records.

3. *Activation by the commissioner.* Device and service information pre-registered by the devices is stored at the `.temp` domain on the DNS server. It is not accessible by other devices until it is checked and enabled, i.e. moved to the actual discovery domain, by the commissioner.

4. *Finding service instance.* Service discovery is done by sending standard DNS queries to the DNS-SD server. Required services are queried with PTR records, and names of all matching services are returned as response.

5. *Resolving service instance.* SRV and TXT records are queried with a service name for service invocation information. Service port, hosting device name, and service path are received with this query. Hostname is resolved into an IP address with an A/AAAA query.

6. *Device/service grouping.* Grouping are done with additional PTR records as it is described in Section 4.3.2. After verifying and enabling the services, they can be grouped into different subdomains. In addition, a multicast group can be created with the same records which are used to describe a device and service.

7. *Filtered device/service discovery.* Services can be queried with a specific type and subdomain. This is achieved with the help of created filters. For example, a PTR query with the name
`_lamp._sub._bc._udp.r5.f2.bldg.org`
can be used to filter all lamps in floor 2, room 5 within the domain `bldg.org`.

5.3 Performance

In this section, we present an evaluation of the performance of our solution in relation to the functions implemented in terms of message size, memory footprint, and response time. We monitor DNS packets with Wireshark [2] in order to verify the correctness of generated DNS records from the devices, to measure the DNS message sizes, and to monitor the interaction between the server and devices. The response time was measured as the difference between times that a device sends a DNS message to the server and receives the corresponding response.

5.3.1 Message size

Since routing IP packets with Contiki's uIP stack relies on the lower layer fragmentation skills and radio transceiver of the device, supported IP packet

[2] Wireshark Network Protocol Analyzer [Online] `http://www.wireshark.org/`

size is limited depending on the sensor device type. For example, maximum available payload size of an IPv6 packet is 1300 bytes for AVR Raven and Redbee Econotag, as those can handle the lower layer fragmentation better, while it is only 240 byes for Tmote Sky, and 140 bytes for Zolertia Z1 devices. In order to avoid the IP packet reassembly, each message of the discovery protocol must fit into a single IP packet.

The length of resource records which are included in DNS messages depend on the length of the device UID, service name, domain name, and service information in the TXT record. We implemented domain name compression to reduce the size of DNS messages for efficiency. With this method, repetitions of common parts of the domain names are represented with a 2-byte pointer to the first appearance of the name in the message.

Registration		
Message	**RRs Included**	**Size (B) (Min/Sample)**
Device Address	AAAA	44/55
Service Information	SRV+TXT	47/80

Table 5.1: Size of DNS messages for registration

Sizes of messages used in registration are shown in Table 5.1. Those messages are DNS Update messages which consist of a 12-byte DNS header, the domain in which updates are performed, and the resource records to be updated. The table shows both minimum and sample message sizes. Minimum sizes are computed using domain names and TXT information with minimum length. Sample sizes are the sizes of the messages observed in the experiments with the messages stated in the *Implementation* section. Sample parameters which affect the message sizes can be listed as follows:

- Temporary domain: `temp`
- Device UID: `249B`
- Service type: `_lamp`
- Service path: `path=/lamp`

An update response message with 22 bytes is returned for each of the registration message.

Table 5.2 summarizes the size of the discovery query and response messages. Following values are used for the calculation of sample message sizes:

- Discovery domain: `bldg.org`

Discovery			
Message	**RRs Included**	**Size (B) (Min/Sample)**	
		Request	**Response**
Service List	PTR	17/46	29/70
Service Information	SRV+TXT	17/36	46/92
Device Address	AAAA	17/31	44/59

Table 5.2: Size of DNS messages for discovery

- Search filter: `_lamp._sub._bc._udp`
- Device UID: `249B`
- Service type: `_lamp`
- Service path: `path=/lamp`

While querying the services with a search filter, the size of the response depends on the number of matching services. In the experimental setup used, there is one service returned with a 9-byte name (`249B_lamp`). Another 15 bytes plus the length of the service name must be added to the message size for each extra matching service in these settings. It should be noted that records in the question section of a DNS query message are also included in the corresponding response message. So, this is another factor affecting the size of discovery response messages.

As it is stated in the implementation details, a TSIG record can also be included in the discovery query and response messages optionally. Additional overhead for including TSIG in a discovery message is 59 bytes with an 16-byte AES-CMAC and a 5-byte key name.

5.3.2 Memory footprint

We focus on the client side implementation in this part since memory constraints of the client devices constitute the main limitation. Because of the limited memory resources, reducing the code size and the number of variables and buffers is crucial. The experimental client-side implementation consists of 1406 lines of code. It is only 861 lines if the TSIG-related code is not included. A large part (about 50%) of the source code is for creating and parsing DNS messages. In order to optimize the memory usage in this part, handling of DNS messages is performed directly inside the uIP buffer of Contiki. This method eliminates the need for additional buffers for creating and parsing DNS messages, therefore provides memory efficiency.

The code is compiled with *arm-none-eabi-gcc* (GCC) 4.3.2 for Redbee Econotag. The reference implementation with one service requires 55.19 kB

Application	Memory Requirement (kB)
DNS-SD (with uIPv6)	55.19
DNS-SD w/o TSIG (with uIPv6)	53.80
Contiki "UDP Client" Example with uIPv6	51.28
Contiki "Hello World" Example	27.76

Table 5.3: Memory requirements of the implementation and example applications

of RAM together with Contiki and the uIPv6 stack. Without the code for handling TSIG and AES-CMAC, memory requirement is 53.80 kB. Each additional service to be registered costs 0.1 kB of RAM. Table 5.3 shows the memory requirements of the implementation and some example applications for comparison. Notice that the uIP stack consumes the most significant amount of memory in the implementation.

5.3.3 Response time

We define response time as the length of time a client has to wait from the moment it sends a request to the moment it receives the response from the server. It should be noted that these are the best timings and actual real-world response times are affected by network congestion and other factors. The average response time for the messages without TSIG is 35 *ms*. The value is 50 *ms* with TSIG verification enabled. It should be noted that a 6LowPAN border router is deployed between the server and devices in our experiments. Packets therefore are always delayed via one hop. Multi-hop routing is handled by the uIP stack of Contiki, and it depends on the performance of the lower layers used.

5.4 Security

Device and service information, configuration settings, and security parameters are considered as data assets to be protected in the system. Device and service information includes device IP address, service type, service port number, service path, and the relation between devices and services. The proposed solution provides not only the required discovery functionality with the *security level 0*, we define security levels *1* and *2* to meet the security requirements in a gradual manner. Several security mechanisms are employed in *pre-commissioning*, *commissioning*, and *operational* phases in order to fulfill the requirements, and to avoid the relevant security threats or minimize the negative consequences in the given BACS scenario.

In the pre-commissioning phase, there is no preshared secret between

any device and the discovery server. Moreover, devices are not able to perform public-key cryptographic operations due to their constrained resources. Therefore, it is not possible to create a trust relationship between devices and the server in pre-commissioning phase with the current assumptions. It should be added that this might lead to some threats such as denial-of-service attacks in this phase. A rogue party may announce itself as server and collect service information, for example. In such a case, disclosure of device and service information can be prevented by encrypting data as in the security level 2. In another case, an attacker may send bogus preregistration data continuously to the server threatening the server availability. As the proposed discovery solution does not provide explicit mechanisms for protection from DoS attacks in the pre-commissioning phase, other countermeasures are needed to alleviate the effects of such attacks, such as monitoring the network activity and using intrusion detection systems. Also noteworthy is the fact that all preregistered device and service information is temporary and not enabled to be used as discovery data before it is verified by the Commissioning Tool.

For origin authentication and integrity of preregistered data, devices create and register additional TXT records containing MACs of the device and service information preregistered in the security level 1. Those MACs are created using the factory key of the device. Thus, any party that has the factory key of the device can authenticate the preregistered information and can verify its integrity. With encrypting preregistered data in the security level 2, confidentiality is also achieved in addition to data origin authentication and integrity. Moreover, privacy requirements are met since it is not possible for an unauthorized party to decrypt and observe the device and service information preregistered.

All communication between CT and server is secured regardless of the security level in the commissioning phase. A secure channel is created with SSL/TLS or DTLS using the mutually trusted certificates of both parties. This assures that only CT with a trusted certificate can have access to preregistered data. CT can verify the authenticity and integrity of device and service information by verifying the associated MACs or decrypting the data with the corresponding device key in security levels 1 and 2. This is based on the assumption that CT holds initial factory keys of the devices. Eavesdropping the traffic between CT and server is not profitable since the channel is encrypted.

Data, which is verified and enabled by CT, becomes available to authorized devices in the operational phase. Authorization of devices is done with the operational keys generated and distributed by CT. The operational key which is shared between a device and the server is used to create a security context during the discovery process. In security level 1, authenticity and integrity of data exchanged is ensured by the TSIG mechanism. Both device and server create and append a TSIG record, which contains the

MAC of the transmitted data, to all outgoing packets. The other party verifies the MAC with the operational key used between the pair. TSIG mechanism also ensures that only authorized parties can access the discovery information on the server. Security level 2 requires a DTLS connection in order to provide message confidentiality. All packets are encrypted with a secure channel established via DTLS, using the operational keys. With this encrypted channel, it is not possible for an adversary to infer further information by observing the discovery traffic.

Table 5.4 provides a summary of proposed security mechanisms in our solution. It also shows which security requirements are met with a given mechanism in each phase. *SL1* and *SL2* are the security levels 1 and 2 respectively. *SR1* to *SR5* represent the security requirements, which are defined in Section 3.3.3.

Security Mechanisms	**SR1**	**SR2**	**SR3**	**SR4**	**SR5**
Pre-commissioning					
SL1: Additional TXT records with MAC	+	+	-	-	-
SL2: Preregistered data encrypted with device's factory key	+	+	+	+	-
Commissioning					
Secure channel established between *CT* and server via SSL/TLS/DTLS	+	+	+	+	∼
Operational					
SL1: TSIG mechanism used	+	+	-	-	∼
SL2: DTLS channel created between device and server using the operational key	+	+	+	+	∼
∼: mitigates the risk					

Table 5.4: Security mechanisms and requirements met

Although the countermeasures in the pre-commissioning phase do not provide a mutual authentication between a device and server, they allow authorized parties (i.e., CT in the commissioning phase) to authenticate the origin of the preregistered data later on. *SR1*, authentication and authorization requirement, is therefore shown as met in the pre-commissioning phase in Table 5.4. Data integrity and confidentiality is achieved by computing MACs and encrypting the preregistered data in this phase. Protecting the system availability especially in the pre-commissioning phase is hard to achieve by solely the service discovery system. Hence, the issue should be addressed in a separate layer by some other means as mentioned before. Authentication, data integrity and confidentiality requirements are met via

the secure channel established between CT and server in the commissioning phase. Securing the communication also reduces the risk of someone learning service usage patterns by observing the traffic. Attempts to access to server for commissioning without a valid certificate are simply refused, therefore constituting a measure against denial-of-service attacks. TSIG mechanism provides message authentication and integrity at transaction level during operation. Furthermore, access control to device and service information is done by verifying TSIG record in a request. DTLS, which is required in security level 2, is used if privacy of the transmitted data is also required.

Chapter 6

Conclusion and final remarks

We proposed a device and service discovery system based on DNS-SD in this work. As it is stated in the background chapter, there are many existing service discovery mechanisms and protocols for specific or general purposes in the IT domain. We presented and analyzed some of them that we think most promising to be utilized in BACS. These solutions are getting more realizable in BACS domain, as BACS are more increasingly using IP-based communication in all levels and devices have become more powerful in terms of processing power and storage. Nevertheless, they are still not directly applicable in a commercial building environment mainly because of two reasons. First, the existing service discovery solutions, which are based on protocols and technologies like XML, SOAP and Web Services, are still too heavyweight for embedded devices. Secondly, most of the discovery solutions are either not designed with built-in security mechanisms, or implement public key cryptography which is again not truly achievable with resource constrained devices yet. Based on the analysis of possible threats, as we discussed in Chapter 3, a service discovery solution should provide authentication, integrity, and confidentiality mechanisms in BACS domain.

The main contribution of this study is to analyze functional, technical, and security requirements of a device and service discovery solution for BACS, and to provide a realizable design while taking into account the installation, commissioning, and operational phases of the BACS lifecycle. A special effort has been put on making use of the already existing techniques when possible. This is also the primary reason for us to choose DNS-SD as the basis of the designed protocols. As DNS-SD promotes, it makes more sense to use the existing software, infrastructure, and expertise that every network needs already, instead of deploying an entire parallel system just for service discovery.

Since DNS-SD does not specify any additional security mechanisms to those that already exist for DNS security, the proposed design employs several measures to meet the security requirements in different phases. Those

measures have been defined gradually within three security levels. For instance, we defined a symmetric key based message signature scheme for the authenticity and integrity of device and service information, as the recommended DNSSEC mechanism does not fit the constraints of the system because it is based on public key cryptography.

It is important to note that while this thesis constitutes an attempt to solve the device and service discovery problem in BACS, there are still remaining challenges and possible improvements to the proposed design solution. Some of these challenges have already been posed in the last section of Chapter 4. To summarize, the model can be extended to become more distributed and avoid single points of failure. A special key distribution mechanism can be integrated for the management of operational keys. Additional capabilities, such as publish/subscribe mechanisms in UPnP and DPWS, can be included to make the proposed design a more complete device services solution. Finally, although a functional prototype has been implemented on a wireless sensor network, we believe that with further optimizations it is possible to reduce the network and memory overhead.

Bibliography

[1] BACnet - A data communication protocol for building automation and control network. ANSI/ASHRAE Std. 135-2008, 2009.

[2] Bettstetter, C., and Renner, C. A comparison of service discovery protocols and implementation of the service location protocol. *Proc. EUNICE Open European Summer School, Twente, Netherlands* (2000).

[3] Bohn, H., Bobek, A., and Golatowski, F. SIRENA - service infrastructure for real-time embedded networked devices: A service oriented framework for different domains. In *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on* (2006), IEEE, pp. 43–43.

[4] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. Web services architecture. *World Wide Web Consortium* (2004).

[5] Cheshire, S., and Krochmal, M. DNS-based service discovery. Internet Draft, December 2011.

[6] Cheshire, S., and Krochmal, M. Multicast DNS. Internet Draft, December 2011.

[7] Driscoll, D., and Mensch, A. Devices profile for web services version 1.1. *OASIS Standard* (June 2009).

[8] Dunkels, A., Gronvall, B., and Voigt, T. Contiki - A lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on* (2004), IEEE, pp. 455–462.

[9] Eastlake, D. Secret key establishment for DNS (TKEY rr). RFC 2930, September 2000.

[10] Ellison, C. UPnP security ceremonies 1.0. UPnP Security Working Group.

[11] Elz, R., and Bush, R. Clarifications to the DNS specification. RFC 6347, July 1997.

[12] Finch, E. Is IP everywhere the way ahead for building automation? *Facilities 19*, 11/12 (2001), 396–403.

[13] Goland, Y., Cai, T., Gu, Y., and Albright, S. Simple service discovery protocol v1.0. Internet Draft, June 1999.

[14] Granzer, W., Praus, F., and Kastner, W. Security in building automation systems. *Industrial Electronics, IEEE Transactions on 57*, 11 (2010), 3622–3630.

[15] Guttman, E., Perkins, C., and Kempf, J. Service templates and service: Schemes. RFC 2609, June 1999.

[16] Guttman, E., and Veizades, J. Service location protocol, version 2. RFC 2608, June 1999.

[17] Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S., Kumar, S., and Wehrle, K. Security challenges in the IP-based internet of things. *Wireless Personal Communications* (2011), 1–16.

[18] Hui, J., and Culler, D. Extending IP to low-power, wireless personal area networks. *Internet Computing, IEEE 12*, 4 (2008), 37–45.

[19] Jammes, F., Mensch, A., and Smit, H. Service-oriented device communications using the devices profile for web services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing* (2005), ACM, pp. 1–8.

[20] Kastner, W., Neugschwandtner, G., Soucek, S., and Newmann, H. Communication systems for building automation and control. *Proceedings of the IEEE 93*, 6 (2005), 1178–1203.

[21] Larson, M., Massey, D., Rose, S., Arends, R., and Austein, R. DNS security introduction and requirements. RFC 4033, March 2005.

[22] Lynn, K., and Sturek, D. Extended multicast DNS. Internet Draft, March 2012.

[23] Martocci, J., Schoofs, A., and van der Stok, P. Commercial building applications requirements. Internet Draft, July 2010.

[24] Modi, V., and Kemp, D. Web services dynamic discovery (WS-discovery) version 1.1. *OASIS Standard* (June 2009).

[25] Munir, S., Dongliang, X., Canfeng, C., and Ma, J. Service discovery in wireless sensor networks: Protocols & classifications. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on* (2009), vol. 2, IEEE, pp. 1007–1011.

[26] NLnet Labs. ldns library. http://www.nlnetlabs.nl/projects/ldns/ Accessed: 20 Sep 2012.

[27] Perkins, C., and Guttman, E. DHCP options for service location protocol. RFC 2610, June 1999.

[28] Presser, A., Farrell, L., Kemp, D., and Lupton, W. UPnP device architecture 1.1. UPnP Forum.

[29] Rescorla, E., and Modadugu, N. Datagram transport layer security version 1.2. RFC 6347, January 2012.

[30] Sales, T., Sales, L., Pereira, M., Almeida, H., Perkusich, A., Gorgônio, K., and de Sales, M. Towards the UPnP-UP: Enabling user profile to support customized services in UPnP networks. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM'08. The Second International Conference on* (2008), IEEE, pp. 206–211.

[31] Shelby, Z. Constrained restful environments (CoRE) link format. RFC 6690, August 2012.

[32] Shelby, Z., Krco, S., and Bormann, C. CoRE resource directory. Internet Draft, July 2012.

[33] Song, J., Poovendran, R., Lee, J., and Iwata, T. The AES-CMAC algorithm. RFC 4493, June 2006.

[34] Vixie, P., Thomson, S., Rekhter, Y., and Bound, J. Dynamic updates in the domain name system (DNS update). RFC 2136, April 1997.

[35] Weber, R. Internet of things - new security and privacy challenges. *Computer law & security review 26*, 1 (2010), 23–30.

[36] Wellington, B. Secure domain name system (DNS) dynamic update. RFC 3007, November 2000.

[37] Zhu, F., Mutka, M., and Ni, L. Service discovery in pervasive computing environments. *Pervasive Computing, IEEE 4*, 4 (2005), 81–90.