Network Management with Semantic Descriptions for Interoperability on the Internet of Things

---

Dissertation

eingereicht von

Diplom-Informatikerin (Fachhochschule) Kristina Sahlmann

vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

zur Erlangung des Akademischen Grades

Doktorin der Ingenieurwissenschaften

- Dr.-Ing. -

in der Wissenschaftsdisziplin "Betriebssysteme und Verteilte Systeme”

angefertigt am

Institut für Informatik und Computational Science
der Universität Potsdam

Professur Betriebssysteme und Verteilte Systeme

Betreuung Prof. Dr. Bettina Schnor, University of Potsdam
Prof. Dr.-Ing. Thomas Schwotzer, HTW Berlin

1. Gutachter Prof. Dr. Mesut Güneş
Institut für Intelligente Kooperierende Systeme
Otto-von-Guericke- Universität Magdeburg

2. Gutachter Prof. Dr.-Ing. Jochen Schiller
Institut für Informatik
Freie Universität Berlin

Potsdam, den 20. März 2021

Diplom-Informatikerin (Fachhochschule) Kristina Sahlmann
Network Management with Semantic Descriptions for Interoperability on the Internet of Things
Dissertation, den 20. März 2021
First Supervisor: Prof. Dr. Bettina Schnor, University of Potsdam
Second Supervisor: Prof. Dr.-Ing. Thomas Schwotzer, HTW Berlin
Reviewers: Prof. Dr. Mesut Güneş and Prof. Dr.-Ing. Jochen Schiller

Universität Potsdam
Professur Betriebssysteme und Verteilte Systeme
Institut für Informatik und Computational Science
Mathematisch-Naturwissenschaftliche Fakultät

Abstract

The Internet of Things (IoT) is "a system of physical objects that can be discovered, monitored, controlled, or interacted with by electronic devices that communicate over various networking interfaces and eventually can be connected to the wider Internet", [Guinard and Trifa, 2016]. IoT devices are equipped with sensors and/or actuators and may be constrained in terms of memory, computational power, network bandwidth, and energy. Interoperability can help to manage such heterogeneous devices. Interoperability is the ability of different types of systems to work together smoothly. There are four levels of interoperability: physical, network and transport, integration, and data. The data interoperability is subdivided into syntactic and semantic data. Semantic data describes the meaning of data and the common understanding of vocabulary e.g. with the help of dictionaries, taxonomies, ontologies. To achieve interoperability, semantic interoperability is necessary.

Many organizations and companies are working on standards and solutions for interoperability in the IoT. However, the commercial solutions produce a vendor lock-in. They focus on centralized approaches such as cloud-based solutions. This thesis proposes a decentralized approach namely Edge Computing. Edge Computing is based on the concepts of mesh networking and distributed processing. This approach has an advantage that information collection and processing are placed closer to the sources of this information. The goals are to reduce traffic, latency, and to be robust against a lossy or failed Internet connection.

We see management of IoT devices from the network configuration management perspective. This thesis proposes a framework for network configuration management of heterogeneous, constrained IoT devices by using semantic descriptions for interoperability. The MYNO framework is an acronym for MQTT, YANG, NETCONF and Ontology. The NETCONF protocol is the IETF standard for network configuration management. The MQTT protocol is the de-facto standard in the IoT. We picked up the idea of the NETCONF-MQTT bridge, originally proposed by Scheffler and Bonneß[2017], and extended it with semantic device descriptions. These device descriptions provide a description of the device capabilities. They are based on the oneM2M Base ontology and formalized by the Semantic Web Standards.

The novel approach is using a ontology-based device description directly on a constrained device in combination with the MQTT protocol. The bridge was extended in order to query such descriptions. Using a semantic annotation, we achieved that the device capabilities are self-descriptive, machine readable and re-usable.

The concept of a Virtual Device was introduced and implemented, based on semantic device descriptions. A Virtual Device aggregates the capabilities of all devices at

the edge network and contributes therefore to the scalability. Thus, it is possible to control all devices via a single RPC call.

The model-driven NETCONF Web-Client is generated automatically from this YANG model which is generated by the bridge based on the semantic device description. The Web-Client provides a user-friendly interface, offers RPC calls and displays sensor values. We demonstrate the feasibility of this approach in different use cases: sensor and actuator scenarios, as well as event configuration and triggering.

The semantic approach results in increased memory overhead. Therefore, we evaluated CBOR and RDF HDT for optimization of ontology-based device descriptions for use on constrained devices. The evaluation shows that CBOR is not suitable for long strings and RDF HDT is a promising candidate but is still a W3C Member Submission. Finally, we used an optimized JSON-LD format for the syntax of the device descriptions.

One of the security tasks of network management is the distribution of firmware updates. The MYNO Update Protocol (MUP) was developed and evaluated on constrained devices CC2538dk and 6LoWPAN. The MYNO update process is focused on freshness and authenticity of the firmware. The evaluation shows that it is challenging but feasible to bring the firmware updates to constrained devices using MQTT. As a new requirement for the next MQTT version, we propose to add a slicing feature for the better support of constrained devices. The MQTT broker should slice data to the maximum packet size specified by the device and transfer it slice-by-slice.

For the performance and scalability evaluation of MYNO framework, we setup the High Precision Agriculture demonstrator with 10 ESP-32 NodeMCU boards at the edge of the network. The ESP-32 NodeMCU boards, connected by WLAN, were equipped with six sensors and two actuators. The performance evaluation shows that the processing of ontology-based descriptions on a Raspberry Pi 3B with the RDFLib is a challenging task regarding computational power. Nevertheless, it is feasible because it must be done only once per device during the discovery process.

The MYNO framework was tested with heterogeneous devices such as CC2538dk from Texas Instruments, Arduino Yún Rev 3, and ESP-32 NodeMCU, and IP-based networks such as 6LoWPAN and WLAN.

Summarizing, with the MYNO framework we could show that the semantic approach on constrained devices is feasible in the IoT.

Acknowledgement

Many people accompanied me on the long way to doctoral thesis. First and foremost, I am extremely grateful to my supervisor, Bettina Schnor, at the University of Potsdam. Her invaluable advice, continuous support and immense knowledge, encouraged me throughout and enabled me to finish this doctor thesis. But let's take one thing at a time and how it all began.

I would like to thank Reingard Jundt, a laboratory engineer from the Beuth Hochschule, who encouraged me to take this path and who sent me an invitation to a meeting event at the Technical Museum of Berlin called "How to become a female professor at a university of applied sciences" in 2014.

I would like to express gratitude to Juliane Siegeris, professor at the HTW Berlin (Hochschule für Technik und Wirtschaft) who I met on this event. She told me that she is looking for a Ph.D. student. I applied for this job and was accepted. This job was funded by the Federal Ministry of Education and Research under grant number Professors Program II which supports women who strive to become professors at universities of applied sciences. As my mentor, Juliane helped me in word and deed at any time. At the beginning, I had only a rough idea about my research topic, semantics, because I worked on the content-management-systems before and was fascinated of data and knowledge.

I would like to thank Thomas Schwotzer, professor at the HTW Berlin, who was researching on semantics. Juliane introduced us, and he agreed to supervise my doctor thesis as the second supervisor. Then, I discovered the Internet of Things (IoT) as an application area of semantics as a new research field. My research topic was narrowed. We published a poster about MOCAP (Micro-Ontology Context-Aware Protocol) on a semantic conference about how ontologies can be used in MQTT and CoAP protocols.

An IoT device has its origin in a sensor board known from wireless sensor networks (WSN). I was looking for a university professor as my first supervisor. My very much appreciated and highly regarded mentor Bettina Schnor, professor at the University of Potsdam, to whom Thomas Schwotzer had introduced me, teaches distributed systems and is interested in wireless sensor networks (WSN). She agreed to supervise my doctor thesis as the first supervisor. But before, I had to manage exams in three subjects: Wireless Sensor Networks, Semantic Web, and theoretical Computer Science. All this effort was required, since I had a university of applied sciences degree. That was a setback to me.

I would like to thank Thomas Scheffler, at that time professor at the Beuth Hochschule. He published a paper about network management in the IoT with NETCONF and MQTT protocols. Thomas Scheffler is a former Ph.D. student of Bettina Schnor, and he asked me, if I know an ontology for description of IoT devices. I had an idea for his question, and we published a paper together. He also has provided the CC2538dk

sensorboards from Texas Instruments. My research topic sharpened up to a whole framework, called later MYNO. Further publications on conferences followed, among others about Virtual Device with Thomas Schwotzer.

My appreciation also goes out to all students who supported me on the implementation and evaluation of the framework: Alexander Lindemann, Christian Hoffmann, Philip Ullrich, Alexander Nowak, Michael Nowak, Vera Clemens, Ekaterina Kapranova, Wolf-Jörgen Stange, Uchralt Temuulen, Florian Mikolajczak, and all other students taking part of my IoT seminars.

I would like to thank the research colloquia and laboratory engineers at the HTW Berlin and University of Potsdam, reviewers of my papers and conference attendees who provided me with feedback, support and inspiration. Thanks the funding for young researchers at the Faculty of Science of the University of Potsdam, I could finish my thesis after all.

Finally, I want to thank my family for supporting me in the past 6 years through this path of trial and tribulation.

Contents

1 Introduction

> *In protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away.*
>
> **— RFC 1925, Nr. 12**
> (The Twelve Networking Truths)

In the last decades, sensors and actuators were embedded in physical things, linked through Wireless Sensor Network (WSN) to the Internet and the term *Internet of Things (IoT)* was established (see Definition 1). Another synonym is *Cyber-Physical Systems (CPS)* [289]. *Things*, also called *smart objects*[402, p. 3], are electronic devices which can be found in application domains like Home Automation, Smart City, Smart Agriculture, Industrial IoT (IIoT) (or Industry 4.0 in Germany). Hundreds of such IoT devices (further referred to devices) can be connected and build a complex infrastructure. The term *smart* describes that such systems have intelligent algorithms to process collected sensor data (e.g. for predictive maintenance or optimization of processes).

Definition 1 (Internet of Things (IoT)).
The Internet of Things (IoT) is a system of physical objects that can be discovered, monitored, controlled, or interacted with by electronic devices that communicate over various networking interfaces and eventually can be connected to the wider Internet. [131] Such electronic devices are equipped with sensors and/or actuators.

IoT devices can have constrained resources in terms of memory, computational power, network bandwidth and energy (e.g. devices for environment sensing or controlling) (see Definition 2). Some IoT devices are so called *power-affluent* in terms of energy (e.g. refrigerators, washing machines, air-conditioner, lightbulbs).

Definition 2 (Constrained devices).
Constrained devices are cyber-physical objects with limited resources of computational power, memory, network bandwidth, and energy. RFC 7228 [40].

The RFC 7228 [40] provides the core terminology for constrained nodes and networks and defines three classes of constrained devices. The classification is based on the combination of two constraints: data size (RAM) and code size (ROM/Flash), and provides rough indications of these parameters in Table 1.1.

According to the RFC 7228 [40], the three device classes have following characteristics:

Tab. 1.1: Classes of Constrained Devices, RFC 7228 [40]

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

- Class 0 devices are "very constrained sensor-like motes" which are pre-configured with a small function set and require gateways to participate in Internet communications.
- Class 1 devices are "capable enough to use a protocol stack specifically designed for constrained nodes" (MQTT, CoAP) and can provide support for the security functions on the network.
- Class 2 devices are more capable but they still "benefit from lightweight and energy-efficient protocols and from consuming less bandwidth" to leave more resources for applications.

The constrained devices of the Class 1 and higher are in focus of the IETF CoRE working group and of this thesis. Several commercially available microcontroller boards were used in this thesis. They are classified in Table 1.2. In border cases, the RAM size is crucial. Constrained devices beyond the Class 2 are not further classified. These can be single-board computers with an operation system like Raspberry Pi with Raspbian OS. However, the boundaries of these classes will move over time because of technological advancement and reduction of costs.

Tab. 1.2: Examples of constrained Devices used in this thesis

Name	Example Devices (RAM, ROM/Flash)
Class 0	–
Class 1	CC2538dk (32 KB, 512 KB) , Arduino Nano 33 IoT (32 KB, 256 KB)
Class 2	ESP-32 NodeMCU (520 KB, 16 MB)
Beyond Class 2	Arduino Yún Rev 2 (16 MB, 64 MB), Raspberry Pi 3B (1 GB, 16 GB), Raspberry Pi Zero w (512 MB, 16 GB)

Constrained devices (also called *nodes*) are usually connected by *constrained networks*. Constrained networks may have constraints such as: low bit rate/throughput, high packet loss, link-layer fragmentation for larger packets, limits on duty cycle (periodical *wake up*), asymmetric link characteristics. *Constrained-node networks* are constrained networks with mainly constrained nodes. The RFC 7228 introduced a class of a constrained network called LoWPAN or 6LoWPAN when used for IPv6. *6LoWPAN* stands for "Low-Power Wireless Personal Area Network" over IPv6 and is defined in the RFC 4944 [237]. This network contains devices which implement the IEEE 802.15.4 standard [160]. LoWPANs have been proposed "for urban monitoring, control of large buildings, and industrial control applications" [40]. This thesis used 6LoWPAN for the implementation with devices of the Class 1 and WLAN with devices of the Class 2 and beyond.

Meanwhile, billions of IoT devices are connected to the Internet (see Figure 1.1) and the number of IoT devices is expected to grow in the coming years. This forecast

is reasonable because the research methodology of this report includes 50 insights on the current market which are the global players in the IoT (i.e. IBM, Amazon, Microsoft, etc.). The IoT devices are connected in different kinds: most of them are wireless connected by a Wireless Personal Area Network (WPAN), followed by a Wireless Local Area Network (WLAN) and a Low-power Wide Area Networks (LPWAN). WPAN includes Bluetooth, Zigbee, Z-Wave or similar technologies. WLAN network is already an established and widespread IP-based network technology [412][1]. Many devices reuse the existing network infrastructure. Examples of LPWAN are NB-IoT, LTE-M, LoRa, Sigfox. This diagram shows only devices or gateways that connect to the Internet. 6LoWPAN or IEEE 802.15.4 are not considered on this diagram because they have their origins in WSN and are usually behind a gateway. However, these technologies are worth considering.

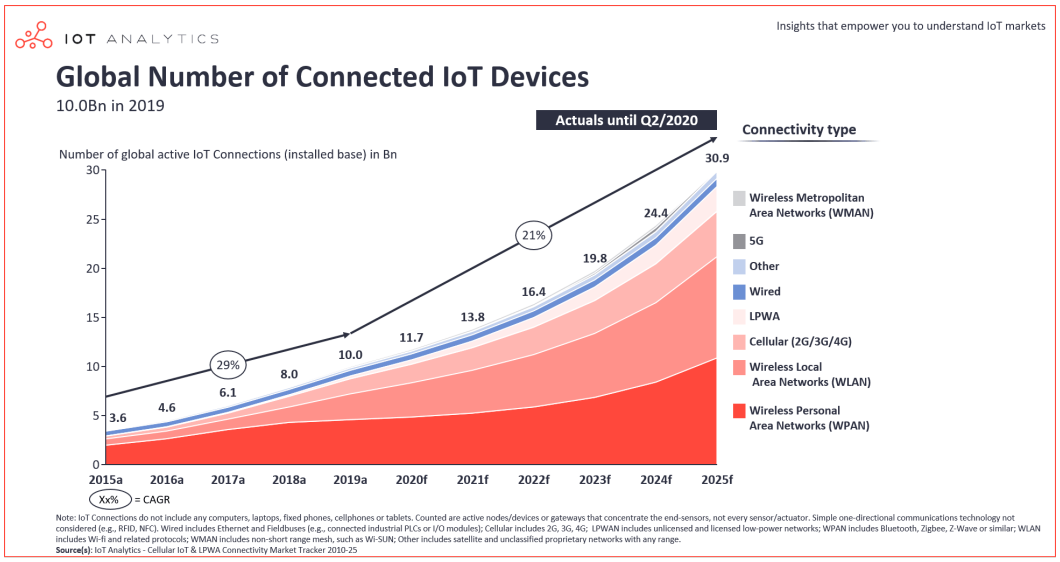


Fig. 1.1: Number of connected IoT devices in billions [384]

An IoT network might consist of hundreds of devices which can join and leave this network at any time (for example because of the limits on *duty cycle*). Network management tools are required to maintain such a network. Network management has a long history and starts with the Open Systems Interconnection (OSI) Management Framework in 1989, also known as ISO/IEC 7498-4 [169] standard. The framework defines terminology and provides a structure for OSI Management, as well as describes its activities.

The *OSI Management* is defined as "facilities to control, coordinate and monitor the resources which allow communication to take place in the OSI environment" [169]. Further, a *managed object* is defined as a resource within the OSI environment which is managed by OSI Management protocols. Such *managed object* is defined "in terms of attributes it possesses, operations that may be performed upon it, notifications that it may issue and its relationships with other managed objects" [169]. A set of these *managed objects* with their properties in a system is saved in a *Management Information Base (MIB)* which is defined as a "conceptual repository of management information". *System management* provides mechanisms for the monitoring, control

[1] Citations of websites are marked by the @ character.

and coordination of managed objects through the System Management Application Entity (SMAE) as the *managing entity*.

OSI Management categorizes a number of functional areas, known as *FCAPS* management (fault, configuration, accounting, performance, and security) [169]:

- *Fault management* includes functions like fault detection, maintenance and examination of error logs, faults correction.
- *Configuration management* provides functions to identify and control managed objects, "collect information about the current condition of the system, obtain notifications about changes", change the configuration of the system.
- *Accounting management* allows to establish charges for the use of managed objects and to inform users of incurred costs.
- *Performance management* includes the evaluation of behavior of managed objects and effectiveness of communication activities, gathering and examination of statistical information, altering the system modes of operation to improve performance.
- *Security management* provides functions to support the application of security policies by "creation, deletion and control of security services and mechanisms, distribution of security relevant information and reporting of security-relevant events".

Definition 3 (Network Configuration Management (NCM)).
Network configuration management identifies, exercises control over, collects data from and provides data to open systems for the purpose of preparing for, initializing, starting, providing for the continuous operation of, and terminating interconnection services. [169]

Looking above on these definitions, we state that the managed objects are configurations of IoT devices, and the device management in the IoT can be seen from the perspective of the network configuration management. Hence, this thesis is focusing on the configuration management (the C in the FCAPS) accompanied by configuration related security aspects (e.g. firmware update distribution), see Figure 1.2.

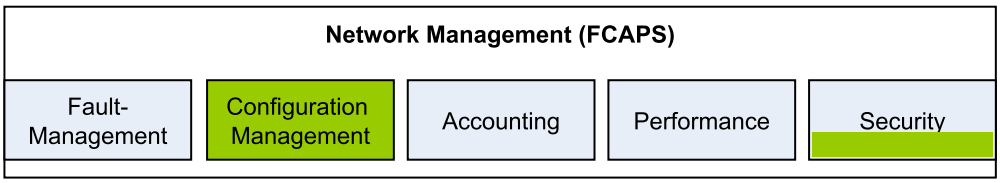


Fig. 1.2: Network Management (FCAPS)

IoT devices are often *heterogeneous* in their capabilities: constrained and non-constrained, equipped with various sensors and actuators, with varied energy supply, support for different protocols. To manage such heterogeneity, *interoperability* is required. In the definition of interoperability [34, p. 218-219], Bill differs between two system levels, conceptual and technical.

Definition 4 (Interoperability).
Interoperability refers to the possibility of integrating different types of systems and data in a single workflow. This requires that the syntax and semantics of the data and systems are provided in a uniform way. ... At concept level, interoperability means that there is a common understanding of the facts between the participants (common world view). At the system level, interoperability requires that different software applications communicate with each other directly and smoothly. (translated from [34])

Vasseur and Dunkels [402, p. 19] notice multiple facets for interoperability for IoT devices. On the physical layer, the physical frequencies, the type of modulation and the transfer rate must be agreed upon. At the network level, devices must agree on the format of information, nodes addressing and message transport. At the application or integration level, IoT devices "must share a common view on how data should be entered or extracted" from their network and how it can be reached from outside systems. The authors remark that interoperability goes with standardization.

Guedria, Panetto et al. [125, 272] surveyed and compared interoperability maturity models. Among them is the Levels of Conceptual Interoperability Model (LCIM) [387, 410] which is focusing on the data to be interchanged. "While covering semantic barriers, it also deals with technological ones" [125]. This model is a candidate for applying to the IoT. However, the naming of the levels in LCIM does not exactly match the common understanding of the interoperability in the IoT.

We propose an adjusted interoperability model for the IoT (see Figure 1.3). The four levels of interoperability are: (1) the physical level which refers to the connectivity protocols like IEEE 802.15.4 or WLAN; (2) the network and transport level addresses communication protocols like TPC/IPv6, UDP/IPv6, 6LoWPAN; (3) the integration interoperability includes application protocols like MQTT, CoAP, HTTP; (4) data interoperability is subdivided into syntactic data which describes the format and structure like XML and JSON; and semantic data which describes the meaning of data and the common understanding of vocabulary, e.g. with the help of dictionaries, taxonomies, ontologies and formalization method for sharing meaning with a model language, e.g. OWL [269], Domain Specific Language (DSL) [92].

Levels	Description	Examples
Level 4 Data Interoperability	Semantic data	OWL, DSL
	Syntactic data	XML, JSON
Level 3 Integration Interoperability	Application protocols	MQTT, COAP, HTTP
Level 2 Network and Transport Interoperability	Communication protocols	TCP/IPv6, UDP/IPv6, 6LoWPAN
Level 1 Physical Interoperability	Connectivity protocols	IEEE 802.15.4, WLAN

Fig. 1.3: Interoperability Model for the IoT

Obviously, to achieve a certain level of interoperability in an IoT system, communication and application protocols are not sufficient. The meaning and the context of

exchanged data must be defined. Semantic annotation of data can provide "machine-interpretable descriptions on what the data represents, where it originates from, how it relates to its surroundings, who is providing it, and what the quality, the technical, and the non-technical attributes are" [23]. An example for *a device provides data* with semantic annotation could be: (what kind of device?) a sensor (which functionality?) provides measuring functionality (which kind of data?) for temperature data (in which units?) in degree Celsius or Fahrenheit.

Definition 5 (Semantic interoperability).
Semantic interoperability means providing data with unambiguous, shared meaning which can be processed and interpreted by machines.

Data-centricity is one of the central aspects in WSN [186] and IoT [3]. Al-Fuqaha et al. [5] considers semantics as one of the building blocks of the IoT because semantics "refers to the ability to extract knowledge smartly by different machines to provide required services". Moreover, the semantic annotations support the reuse of the same information by multiple applications. Hence, semantics enables support for interaction with heterogeneous devices in the IoT.

Summarizing the definitions, a network in the IoT contains often hundreds of constrained devices with sensors and actuators. Network configuration management tools are required to maintain such a network. The IoT devices might be heterogeneous. Interoperability can be achieved by using standards. However, semantic interoperability is necessary to achieve full interoperability.

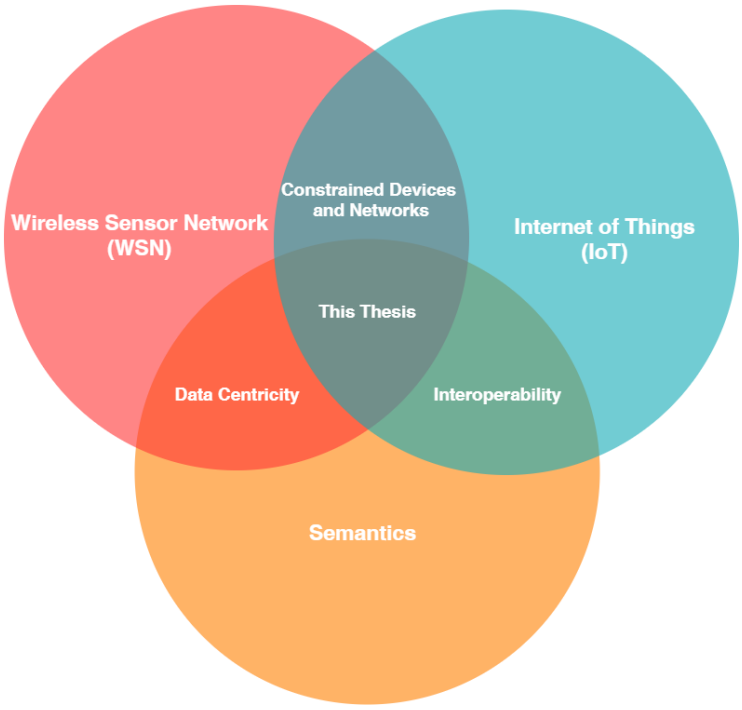


Fig. 1.4: Fields and intersections in this research

This thesis deals with the intersection of three research fields (Wireless Sensor Network (WSN), IoT and Semantics), as shown in Figure 1.4. The focus is on

particular aspects in these three fields: publish-subscribe paradigm in WSN, network management in the IoT, Semantic Web Standards as formalization in Semantics. Each research field has many facets but these three fields have also some aspects in common, the intersections: Data Centricity, Constrained Devices and Networks, and Interoperability. Altogether, these are the core research emphases in this thesis.

Thus, the **motivation** for this thesis was to develop a framework for automatized network management of heterogeneous and constrained devices while using open standards. Before continuing with research details, here is a brief introduction to some use cases for such application in the IoT: building automation and environment monitoring for precision agriculture. Further use cases for management of networks with constrained devices are described in the RFC 7548 [95].

For example, in the building automation, the distributed systems are connected to the same communication infrastructure (e.g. IP-based) and involve wired or wireless communication networks. A large number (up to 100,000) of sensors and controllers with different functionality are required to deploy in the building. The density of nodes is high and the distance between neighboring nodes may vary from 1 to 20 meters. Consequently, a network management solution must be able to discover devices in the network and manage a large number of devices. This network might be segmented, for example using constrained protocols. Also, a network segment can cover a floor or an area on the floor. Examples for such functions are measurement and controlling performed by sensors and actuators for "regulating the quality, humidity, and temperature of the air inside the building as well as regulating the lighting" [95]. A network management system must be able to identify and to locate devices (e.g. address and name for maintaining) and to group them (e.g. for group management).

In the environment monitoring for precision agriculture, a number of sensors is deployed in a greenhouse or on the field to monitor environment conditions (e.g. air temperature and soil moisture). Additionally, a number of actuators can be deployed to precisely control such conditions (e.g. watering pumps for constant soil moisture). Such deployments can help to achieve better harvest. In a greenhouse, the environment is more protected from external conditions (e.g. from tampering). On a field, sensors can be deployed over a large area, must tolerate a number of failures, and must be able to establish connectivity (so-called auto-configuration). The sensing interval can be lower than in the building automation.

1.1 Research Questions

Many vendor platforms (e.g. Amazon [15], IBM [159]) claim to be interoperable. They provide vertical solutions for IoT systems: the IoT devices, the gateway, and the cloud-based services come from a single source. This approach has an advantage because there are no incompatibility issues to expect. However, the disadvantage is the dependency on the vendor for enhancements or upgrades, so-called *vendor lock-in*. Connecting two vendor solutions is a near impossible mission.

Horizontal solutions are preferable. The idea is to decouple single IoT components like devices, gateways and cloud resources from each other and use common standards for communication and information exchange. There are some initiatives and single standards [363, 67] for network management in the IoT but still many

open issues (i.e. interoperability, scalability, security, energy saving) and no common comprehensive solution.

Moreover, the most vendor platforms are cloud-based. If the connection to the cloud is disrupted, local devices might not operate properly. The solution is *Edge Computing*. A system deployed at the edge network has many advantages: managed by yourself in your local enterprise network, accessible by devices even if the Internet connection is disrupted, latency can be decreased and speed increased, network bandwidth can be saved because the amount of data can be reduced through pre-processing.

The management of a network consisting of many constrained and heterogeneous devices is a particular challenge. Another challenge is to reduce or even to avoid a human intervention for configuration of new devices in such a dynamic network and to automatize this process. Configuring every device manually is a tedious and error-prone task which should be omitted.

We propose a framework for managing dynamic IoT networks which is based on standards and open-source implementations and acts on the edge of the network. The MQTT protocol is the common IoT protocol and the basis for the framework. Further, interoperability must be provided by the framework to support heterogeneous devices. The idea for the solution is to use the network configuration protocol NETCONF to bridge the MQTT protocol [327] and Semantic Web Standards for semantic interoperability. The proposed MYNO framework is based on the following technologies: MQTT, YANG, NETCONF and Ontology, which build the acronym.

The main research questions are:

1. How does the NETCONF protocol suit for the network configuration management task in the IoT?

2. How do the Semantic Web Standards (i.e. ontology used for device descriptions) enhance interoperability of this architecture approach?

3. Which security issues must be considered by the design of the MYNO framework?

Applying the Semantic Web Standards brings an overhead on IoT devices. Therefore, further sub-questions are arising:

- How to implement semantic device descriptions on the constrained devices?

- Is this effort justifiable? Advantages vs. disadvantages should be considered.

- Which ontology should be used for device descriptions?

- How can ontology-based data be compressed for constrained devices, without loosing the meaning of data?

- How can the logic formalism in Semantic Web Standards be used for inference?

1.2 Research Contributions

This thesis tackles interoperability in the MQTT-based IoT framework while regarding this issue as a network configuration management task. The main scientific contributions of this thesis are the following:

1. The MYNO framework for network configuration management in the IoT was designed, implemented and evaluated. The NETCONF-MQTT bridge [327] was used instead of installing NETCONF server on IoT devices. Semantic device descriptions were introduced for description of the device capabilities. The novel approach is using a ontology-based device description directly on a constrained device with MQTT protocol. These descriptions are based on an ontology and Semantic Web Standards. The bridge was extended in order to query such descriptions. This approach was evaluated and work on edge-devices such as Raspberry Pi 3B and constrained IoT devices. Using a semantic annotation, we achieved that the device capabilities are self-descriptive, machine readable and reusable.

2. A comprehensive study of the IoT protocols and approaches for interoperability, especially semantic interoperability, was performed. Semantic Web Standards for device descriptions are used to support semantic interoperability in the IoT. Many ontologies were surveyed and one was re-used and extended for device descriptions, namely the oneM2M Base ontology.

3. A discovery process with semantic device descriptions and a bootstrap process for the MYNO framework were developed and tested. These processes make the framework robust and fulfill the requirement for scalability.

4. The implementation and evaluation of MYNO shows that the proposed framework is feasible and interoperable in terms of the four interoperability levels: physical, network and transport, integration, data (syntactic and semantic). The MYNO framework supports heterogeneous and constrained devices and IP-based networks such as 6LoWPAN and WLAN.

5. A model-driven Web-Client was developed as a NETCONF client. Instead of developing a mobile app for network configuration, we use the YANG data model generated on basis of device descriptions, as an input for the graphical user interface. The RPC calls are generated automatically based in this model.

6. CBOR and RDF HDT were evaluated for optimization of ontology-based device descriptions for use on constrained devices, without loosing the meaning of data and retaining compatibility to Semantic Web Standards.

7. The concept of a Virtual Device was introduced and implemented, based on semantic device descriptions. A Virtual Device aggregates the device capabilities at the edge network and contributes therefore to the scalability.

8. The role of the logic formalism in Semantic Web Standards was evaluated in context of the IoT and constrained devices. The Python library, RDFLib, with SPARQL query language was used for implementation of the Virtual Device and the bridge parser, and evaluated on the Raspberry Pi 3B.

9. The MYNO Update Protocol (MUP) for firmware update was developed and evaluated on constrained devices of the Class 1 and 6LoWPAN. The MYNO update process is focused on freshness and authenticity of the firmware. Further security aspects of the framework were discussed.

1.3 Publications

The most research contributions were reviewed and published in the following papers. All papers, except the journal paper, were presented at the according conferences and workshops.

International

`MUP: Simplifying Secure Over-The-Air Update with MQTT for Constrained IoT Devices`
Kristina Sahlmann, Vera Clemens, Michael Nowak and Bettina Schnor
Trusted IoT Ecosystem, special issue of Sensors 2021, 21, 10.
December 2020
`https://www.mdpi.com/1424-8220/21/1/10`

`Ontology-based Virtual IoT Devices for Edge Computing`
Kristina Sahlmann and Thomas Schwotzer
8th International Conference on the Internet of Things (IoT)
Santa Barbara, California, USA, October 2018

`Ontology-driven Device Descriptions for IoT Network Management`
Kristina Sahlmann, Thomas Scheffler and Bettina Schnor
IEEE Global Internet of Things Summit (GIoTS) Proceedings,
3rd Workshop on Interoperability and Open-Source Solutions for the Internet of Things (InterOSS-IoT),
Bilbao, Spain, June 2018

`MOCAP: Towards the Semantic Web of Things (Poster)`
Kristina Sahlmann and Thomas Schwotzer
Semantics, 11th International Conference on Semantic Systems
Vienna, Austria, September 2015

National

`Binary Representation of Device Descriptions: CBOR versus RDF HDT`
Kristina Sahlmann, Alexander Lindemann and Bettina Schnor
17th GI/ITG KuVS Fachgespräch: Drahtlose Sensornetze
Braunschweig, Germany, September 2018

`An Ontology-based NETCONF-MQTT Bridge for Sensor Devices in the IoT (Demo)`
Kristina Sahlmann, Alexander Lindemann, Thomas Scheffler, Bettina Schnor
17th GI/ITG KuVS Fachgespräch: Drahtlose Sensornetze
Braunschweig, Germany, September 2018

`Managing IoT device capabilities based on oneM2M ontology descriptions`
Kristina Sahlmann, Thomas Scheffler and Bettina Schnor
16th GI/ITG KuVS Fachgespräch: Drahtlose Sensornetze
Hamburg, Germany, September 2017

The Ad hoc Semantic Internet Protocol (ASIP) for Constrained Devices
Kristina Sahlmann, Thomas Schwotzer and Bettina Schnor
15th GI/ITG KuVS Fachgespräch: Drahtlose Sensornetze
Augsburg, Germany, September 2016

1.4 Scope

The scope of this thesis is defined as following:

- only IP-based networks restricted to 6LoWPAN and WLAN
- focus on constrained devices of Class 1 and Class 2 as defined in Section 1
- no SDN approach
- no mobile networks or mobile devices
- no agent systems
- no ICN/NDN networks
- only horizontal domain, no vertical domains (e.g. Industry 4.0)

Overall, the scope of this thesis is based on four scenarios derived from the following use cases: device discovery, controlling and measuring services, aggregation of services, event triggering.

1.5 Terminology and Conventions

For simplicity, the following name conventions for terms are used in this thesis, when nothing else is noted. We use the term **device** in place of *IoT device* which is equipped with sensors and/or actuators, has communication ability and can be constrained.

We use the term **device description** which means our ontology-based *device self-description* which describes the device and its capabilities in terms what it can do and how others can communicate with that device.

We use the term **capabilities** to describe the operations or functions which are supported by a device. We use this term also because it is used in the network management protocol NETCONF where it has a similar meaning:

> Capabilities augment the base operations of the device, describing both additional operations and the content allowed inside operations. The client can discover the server's capabilities and use any additional operations, parameters, and content defined by those capabilities. [93]

Manufacturers understand device capabilities similar to us, as having sensors (gyroscope, magnetometer, accelerometer) or hardware (WiFi, Bluetooth-LE) enabling certain functions, see Apple[2] or Samsung[3].

In the IoT literature, the term **resources** is often used with the meaning of services provided by a device. Thus, one single device can have several resources: for

[2] https://developer.apple.com/support/required-device-capabilities/
[3] https://docs.smartthings.com/en/latest/capabilities-reference.html

example, one resource for temperature sensor, another resource for humidity sensor, and third resource for controlling a lamp.

Although, the MQTT specification calls its central component as *a server*, we use the term **broker** for three reasons: better description of the role; the most implementations use this term; as distinction from the traditional Client-Server architecture.

We use the term **bridge** as a short form for the NETCONF-MQTT bridge of the MYNO framework.

1.6 Thesis Outline

After the introduction in Chapter 1, the thesis is structured as follows:

Chapter 2 provides the background about the used technologies in this thesis. This chapter describes the development from WSN to IoT research area. The application protocols MQTT and CoAP are introduced and compared. The main functionality of the network protocols 6LoWPAN and WLAN is described. The protocols for network management, SNMP and NETCONF, are outlined. The Semantic Web Standards are introduced.

Chapter 3 outlines the related work considering interoperability in the IoT. Standardization efforts and open-source implementations are surveyed as well as vendor approaches.

Chapter 4 introduces the proposed MYNO framework. The overall architecture, its components and concepts of the framework are described. Related work about device and network management in the IoT is discussed.

Chapter 5 studies the semantic interoperability in the IoT. Related work and IoT ontologies are surveyed. A concept for the MYNO device description is introduced and the chosen ontology is extended.

Chapter 6 introduces the concept of a Virtual Device and its role in the MYNO framework. The related work about virtual objects and digital twins is discussed in this chapter.

Chapter 7 analyzes the security aspects of the MYNO framework and introduces the concept of the MUP, the MYNO Update Protocol. Also, the related work about security and firmware updates in the IoT is discussed.

Chapter 8 provides the prototype implementation details of the MYNO framework, the heterogeneous devices, the processing of the semantic device descriptions and the MUP protocol.

Chapter 9 shows the evaluation results. There is the proof of concept and feasibility of the MYNO framework according to selected criteria, evaluation of the semantic device descriptions, as well as performance evaluation of the MUP protocol and the MYNO framework with 10 IoT devices. Additionally, the evaluation on the compression of device descriptions and an TLS benchmark are provided.

Chapter 10 concludes with a discussion of the results and the future work.

Background

2

> *It is more complicated than you think.*
>
> **— RFC 1925, Nr. 8**
> (The Twelve Networking Truths)

In this chapter some fundamentals are explained which are necessary for understanding of this thesis. First of all, Wireless Sensor Network (WSN) is introduced as the predecessor and enabler of the Internet of Things (IoT). Challenges of WSN are similar to challenges of constrained devices and networks in the IoT.

Further, an overview of protocols in the IoT is given over the layers of the Open Systems Interconnection (OSI) model. Particular, the MQTT and CoAP application protocols are introduced for comparison. The 6LoWPAN and WLAN protocols are used for network communication. Further, the NETCONF protocol for network configuration and management and its data modeling language YANG are described and compared with their predecessor SNMP protocol. Finally, the Semantic Web standards as the formal representation technologies for ontology and query language are introduced.

2.1 Wireless Sensor Networks (WSN)

By now, there are sensors available for many kinds of physical parameters: humidity, temperature, light, pressure, acoustic, vibration, infrared, magnetic, chemical sensors etc. Additionally, actuators (like an LED lamp, a relais or similar) can be controlled by a node. On the basis of nodes equipped with sensors and/or actuators in combination with communication abilities and computation power, many kinds of application are possible [186, p. 3-5]. Some of these applications are already in use, for example in a seismic zone or in disaster area after a wildfire, sensor nodes can be spread around in the area and collect information about the environment conditions without human risk. Chemical pollution and environmental control of outlands are further applications for WSN.

Further use cases are intelligent buildings and facility management. Depending on the condition of the building, sensor nodes can be retrofitted into existing or incorporated into new buildings. The sensors can be used for efficient Heating, Ventilation, Air Conditioning (HVAC), fire and earthquake monitoring or keyless entry and intrusion detection. Sensor nodes operate wireless and therefore can be retrofitted on industry machines for monitoring and preventive maintenance. They can also be placed on farm land or attached to livestock, such as cows or pigs, enabling precise agriculture. Wireless sensors can be used in the medical field and health care to track patient condition. Passive sensors use readouts of data (for example Radio Frequency Identifier (RFID) tags) and can be used for tracking goods in logistics application. Sensors embedded on the streets or roadsides for monitoring traffic conditions are examples of telematics applications.

Smart Dust project in Berkeley explored the limits on size and power consumption in autonomous sensor nodes [183]. They developed a complete sensor/communication system which fits into a cubic millimeter. Gross was inspired by Smart Dust and wrote about sensors in his article:

> In the next century, planet earth will don an electronic skin. It will use the Internet as a scaffold to support and transmit its sensations. [...] It consists of millions of embedded electronic measuring devices. [...] These will probe and monitor cities and endangered species, the atmosphere, our ships, highways and fleets of trucks, our conversations, our bodies – even our dreams. [123]

The term Wireless Sensor Network appeared around the year of 2000 in the geospatial research (see e.g. for observing a seismic zone [278]). Recent developments on micro-electro-mechanical systems (MEMS) technology and wireless communication made a widely use of sensor networks feasible and affordable. The main characteristics of a Wireless Sensor Network are:

- A sensor network consists of a "large number of tiny sensor nodes that are densely deployed" and collaborate with each other to fulfill their task because a single node is not capable to do so.
- Sensor nodes use wireless communication, usually on short distances, to perform collaboration.
- Sensor network protocols need self-organizing capabilities as nodes are random positioned e.g. in a seismic zone.
- Sensor nodes (also called *motes* or *sensor dust* [183]) are tiny micro-controller boards with low-computing and low-power resources.
- These nodes "interact with their environment by sensing or controlling physical parameters" [278]. Therefore, every node has a sensor or an actuator on-board.
- Every WSN has a *sink* where sensor data is delivered.

WSN Interaction Patterns
Some interaction patterns are typical for WSN applications [186, p. 6]. These interactions can refer time span or certain space or even both:

- *Event detection*: Sensor nodes report an event (fire or motion) only on their detection. The most of time they are quite passive. A special cases are *function approximation and edge detection*. Physical values like temperature or humidity can change inside of the surveillance area. Such changes can be approximated in a function of this location. Detection of some edge values (e.g. temperature over 40 degrees of Celsius) is a similar application.
- *Periodic measurements*: Sensor nodes periodically send measured values. This can also be triggered by an event.
- *Tracking*: The source of an event can move, and sensor nodes can detect the position of the event source. For example, several motion sensors work in collaboration.

Sensor Node Hardware
A WSN node consists of basic components [196, p. 2ff], as Figure 2.1 shows:

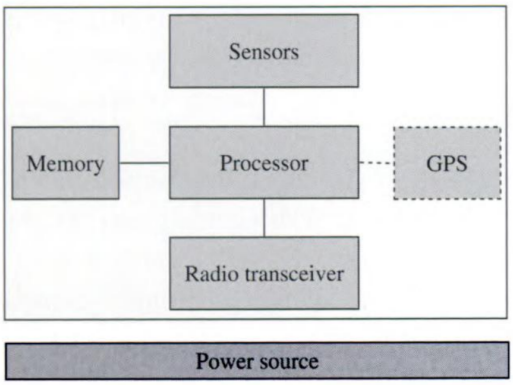


Fig. 2.1: Schematic of a basic wireless sensor network node [196]

1. *Low-power embedded processor* usually performs two main computation tasks: to process the local sensing information and to communicate with other nodes. Computational power is very limited (e.g. an ARM Cortex M3 is a 32-bit 32-MHz processor).
2. *Memory/storage* includes program memory and data memory with limited quantity for economical reasons (e.g. Texas Instruments CC2538dk [381] boards have 32 KB of RAM and 512 KB Flash memory).
3. *Radio transceivers* have low-rate, short-range wireless radio capabilities (10-100 kbps, <100m). Radio communication is often the most power-consuming operation on sensor node.
4. *Sensors* or *actuators* are installed on-board of WSN nodes. They perform low-data-rate sensing or controlling tasks. Several sensors can be installed for measurement of different physical parameter.
5. *Geopositioning system* is an optional feature if the exact location is required, mostly in the outdoor applications. Otherwise the location should be pre-configured or dynamically computed by localization algorithms.
6. *Power source* on WSN nodes is usually a battery (e.g. CC2538dk needs 20-24 mA voltage or two LiMH AAA batteries) while some of nodes may be wired on power source or use energy harvesting techniques.

WSN Challenges

Constraints on WSN and sensor nodes bring new challenges for research [196], among them:

1. **Extended lifetime**: WSN nodes are typically powered by batteries are energy constrained. Long lifetimes are also desired because monitor and replace batteries for a large network is expensive and nearly not feasible. In the practice such nodes should operate several years. Improvements on hardware, energy harvesting techniques but also protocols designs are designed for energy efficiency in WSN.
2. **Responsiveness**: Extending network lifetime is possible through duty-cycled life with periodic sleep and wake-up phases. The challenge is to synchronize such periods and to reduce latency to ensure rapidly reported events.

3. **Robustness**: High density and large scale are possible due to low cost but such nodes are prone to failure. The global performance of a WSN system is more important as individual node failures. Protocol designs have to be robust and provide built-in mechanisms to compensate such failures.

4. **Scalability**: Fine-granularity sensing and large coverage area can easily demand thousand of sensors in a large scale. Protocols should be distributed, hierarchical, use localized communication in order to provide such scalability.

5. **Heterogeneity**: Heterogeneous nodes in terms of capabilities (computation, communication, sensing) are realistic settings. This has some design challenges in network architecture.

6. **Self-configuration**: WSNs are *unattended* distributed systems because of their scale and nature. Autonomous operation of the network is the key challenge. From the start, nodes in WSN have to be able to configure their own network topology.

7. **Privacy and security**: Ensuring privacy and security in WSN is a particular task [197].

Akyildiz et al. summarized the challenges for WSN design in their survey as following:

> A sensor network design is influenced by many factors, which include fault tolerance; scalability; production costs; operating environment; sensor network topology; hardware constraints; transmission media; and power consumption. [4]

Obviously, WSN constraints have consequences on a special operation mode. Factually, one of the consequences is that communication consumes more power than computation. For example, Hill et al. [146] showed in their experiments that the RFM TR1000 radio transceiver needs $1\,\mu\mathrm{J}$ to transmit a single bit and $0.5\,\mu\mathrm{J}$ to receive a bit. During this time, the processor can execute 208 cycles (which are about 100 instructions) and can consume up to $0.8\,\mu\mathrm{J}$ which results to $8\,\mathrm{nJ}$ per instruction. However, energy required for computational tasks should not be ignored and heavy computational tasks should be avoided on the sensor nodes. This results in the ratio of about 190 communication costs over computation. This leads to a consequence: Radio transceivers have to work at a low *duty cycle* choosing from operational states at the physical layer: transmit, receive, idle (ready to receive), sleep [186, p. 25-26].

Wireless channel and constraints of sensor nodes lead to further implications: data rates are low (few tens of kilobits per second) and data packets can get lost. There are several reasons for lost packets on the physical layer: obstacles on the way, hidden-terminal problem, noise, waveform modulation because of collision, overhearing, reflection, diffraction, scattering. These problems are addressed by MAC protocols, contention-based protocols like slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) (used in IEEE 802.15.4) and Request To Send / Clear To Send handshake (RTS/CTS) (used in IEEE 802.11).

WSN Solution Approaches

WSN solution approaches has to follow some central design principles [186, p. 67-81] to manage the challenges mentioned before:

- **Self-organization** (or distributed organization) or auto-configuration of WSN must be independent and tolerates failing nodes and integrates new nodes.
- **In-network processing and aggregation** not only saves communication costs but also saves energy by doing computation within the network. In some applications, a single node is not able to detect an event or make a decision but several nodes can collaborate and join their data providing enough information for decision. This collaboration can be completed either hop-by-hop (in-networking processing) or aggregation at the edge of the network.
- **Data centricity**: addressing data not nodes results in the data-centric networking. Traditional networks work address-centric because data is transferred from one node to another that have network addresses. This is different in WSN since nodes are typically deployed redundant and the identity of the particular node is irrelevant. The information itself is important and therefore switching to data-centric paradigm is promising.

The data-centric approach is of interest for this thesis and therefore introduced more in detail in the next section.

2.1.1 Data-centric Networking

Nodes in a WSN are usually deployed with high density or are redundant to compensate the failure of some nodes. Not the individual node is the center of interest but the data reported about the environment. Therefore such networking is called *data-centric networking* [186, p. 71-73]. In traditional identity-centric networks, every node is identified by an address (e.g. IP address). To find out some information, the requesting node has to know the address of another node which has the capability to respond the required information. For example, the information about an average temperature which is measured by all nodes in the room. Compared to identity-centric networks, data-centric networking allows in-network processing like data fusion and aggregation. The relationship in communication is implicitly defined. Decoupling in time is also supported by data-centric networking because a request does not have to wait for a response. This feature is useful for event-detection applications.

There are several implementations of data-centric networking. Among them are overlay networks like peer-to-peer applications for file sharing and distributed hash tables (DHT), publish/subscribe approach, WSN as a dynamic database. We consider the last two approaches. The publish/subscribe paradigm was surveyed in [97]: "any node interested in a given kind of data can *subscribe* to it, and any node can *publish* data, along with information about its kind". In this way full *decoupling of the communicating nodes in time, space, and synchronization* shown in Figure 2.2 is achieved. Publishers and subscribers do not know each other. If a subscriber is no longer interested in data, it can *unsubscribe* from it. In WSN, sensor nodes are obviously publishers, and sink is a subscriber. Also relevant is the naming of such data subscriptions for matching publishers and subscribers. A first idea is using keywords or subjects as names. Then these subjects can be organized hierarchically with sub-subjects. The other question is how to manage these subscriptions: in a central way by one entity or in a distributed way? A distributed solution is more preferable for WSN but also more complicated. The publish/subscribe paradigm is a

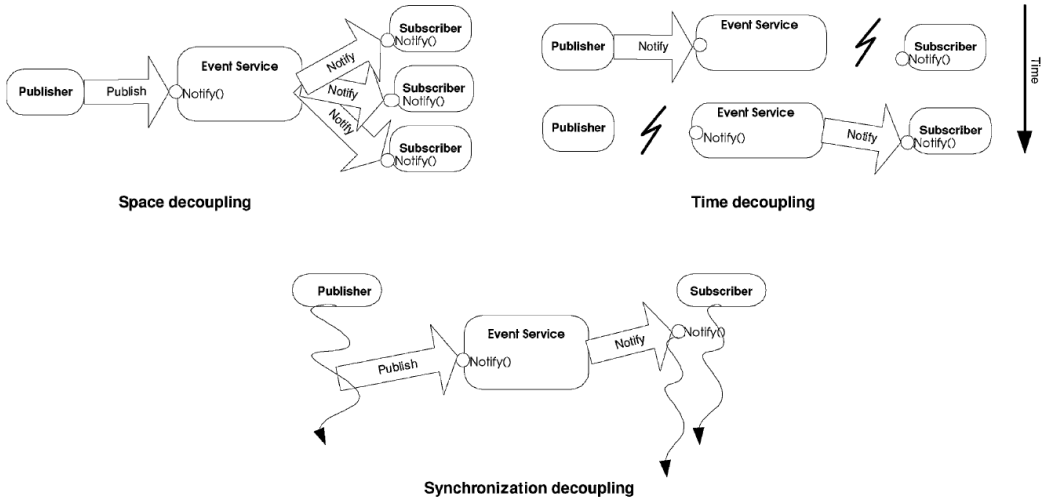


Fig. 2.2: Space, time, and synchronization decoupling with the publish/subscribe paradigm [97]

very popular approach in WSN and many protocols implement this principle [186, p. 331ff], [97]:

- directed diffusion routing [163, 162];
- content-based publish/subscribe [143];
- SPIN (Sensor Protocols for Information via Negotiation) [144, 199].

In directed diffusion [163, 162] an *interest message* will be send by a sink node and disseminated in the network. The interest message describes the desired data. Sensor nodes matching the interest generates a *data message* and upstream this to the sink.

The addressing in data-centric networking can be also defined by some attributes applied to (group of) nodes. This approach is referred as *data-centric addressing* or *content-based addressing*. Heidemann et al. [143] implemented a low-level naming mechanism based on Attribute Value Operation (AVO) tuples. This mechanism was integrated in *directed diffusion* routing. The AVO tuples are built as following: attribute is a subject (e.g. temperature), values can be concrete (e.g. 25°C) or a placeholder (ALL or ANY), operators express a match (EQ for equal, LT for lower than, etc.). The computational effort for matching the interest and data should be as low as possible. Examples for interest and data messages are shown in Listings 2.1 and 2.2.

```
1 <type,temperature,EQ>
2 <threshold-from-below,20,IS>
3 <x-coordinate,20,LE>
4 <x-coordinate,0,GE>
5 <y-coordinate,20,LE>
6 <y-coordinate,0,GE>
7 <interval,0.05,IS>
8 <duration,10,IS>
9 <class,interest,IS>
```

Listing 2.1: AVO example interest message [186]

```
<type,temperature,IS>
<x-coordinate,10,IS>
<y-coordinate,10,IS>
<temperature,20.01,IS>
<class,data,IS>
```

Listing 2.2: AVO example data message [186]

The SPIN (Sensor Protocols for Information via Negotiation) [144] protocols family uses so-called *metadata* for data-centric routing. These metadata describes the data, a node can produce, and is used in the advertising (ADV) message. ADV messages are disseminated by sensors in the network. Then, interested nodes can query that data by request (REQ) message. This approach is different from directed diffusion because the sink queries the sensor nodes fist.

Another data-centric networking implementation is to consider a WSN as a dynamic database and to formulate queries on certain aspects of the physical environment [217]. Sensors are like a "virtual table to which relational operators can be applied". A query can be written in SQL language, see Listing 2.3.

```
SELECT AVG (temperature)
FROM sensors
WHERE location = "Room 123"
```

Listing 2.3: SQL-based request for sensor readings [217]

The impact of the data-centric approach in the IoT will be seen in the publish/subscribe protocol MQTT (Section 2.2.1) which was developed to save energy on the IoT devices. The query language from Madden et al. [217] has some similarity with the SPARQL language in the Semantic Web (Section 2.5.4).

2.1.2 From MANET to WSN and IoT

The WSN was enabled by the hardware development. The devices has decreased in size and networks became more flexible. First, notebooks were mobile and connected to a WLAN access point. Then, mobile phones and tablets connected to mobile networks (e.g. GSM, UMTS, LTE). Finally, tiny sensor nodes were able to communicate in a network on a short distance. Thus, the communication has developed from wired to wireless, mobile, ad-hoc, distributed, P2P [328].

> Ad-hoc network allows mobile computer users with (compatible) wireless communication devices to set up a possibly short-lived network just for the communication needs of the moment. [275, p. 2]

WSN and traditional Mobile Ad Hoc Network (MANET) have some in common but there are also many differences [186, p. 11-12], [275, chapter 1.2.6]

- *Scalability*: The number of sensor nodes in a WSN can be much larger than the number of nodes in an ad-hoc network.
- *Dependability and QoS*: In MANET, each peer has to be reliable. The opposite is in sensor networks where nodes are prone to failures. This requires new concepts of Quality of Service.

- *Data Centricity*: Sensor nodes are redundantly deployed, which makes it attractive to data-centric protocols. In MANET, data centric is irrelevant.
- *Mobility*: The topology of a network may change in MANET and WSN but for different reasons. The peers in MANET are mobile. In WSN, sensor nodes remain stationary, once placed; or can be mobile (e.g. attached to cattle); the sink of information can be mobile or an observed event; or sensor nodes are in sleep mode.
- *Equipment and Energy*: Sensor nodes have limited power, computational capacities, and memory. In MANET a peer can be quite powerful (a laptop or a PDA). Furthermore, MANET applications usually involve a human in the loop.
- *Self configurability*: In this aspect WSN and MANET have the most similarity. Both networks require to be self configured but the limited resources in sensor networks require more efficient solutions.
- *Interaction*: MANET supports conventional applications (like Web, voice, etc.) with well understood traffic characteristics. WSN can have very different traffic depending on the application: very bursty traffic when an event happens or regular traffic of sensor observations.
- *Simplicity*: Due to resource scarceness, sensor networks require simpler solutions for network layering. Heavy-weight routing protocols used in MANET are not applicable to sensor nodes. An overview of such protocol stacks is given in [100].
- *Communication*: Sensor nodes "mainly use broadcast communication paradigm whereas most MANETs are based on point-to-point communications".
- *Global ID*: Sensor nodes "may not have global identification (ID) because of the large amount of overhead and large number of sensors".

From the user point of view, further terms were established in the IoT besides mobile computing. Mark Weisser, as the pioneer of the IoT, adopted the term *ubiquitous computing* [413] in 1991 in his paper "'The Computer for the 21st Century'". In his concept, computers in forms of notebooks, tablets, personal digital assistants (PDAs) are overall present. They communicate with each other and assist the people. The term *pervasive computing* was shaped by the industry and means ubiquitous computing and sensors are pervading business processes and everyday life.

One of the remarking WSN development was the standardization of wireless protocols for low power, low data rate and low-cost wireless sensor communication, as the survey of Rawat et al. [291] in 2014 shows. All these protocols are now used in the IoT: IEEE 802.15.4 (ZigBee, 6LoWPAN, WirelessHART, ISA100.11a), IEEE 802.15.4a - Ultra Wideband, Bluetooth and Bluetooth Low Energy (BLE), Z-wave, ANT, Wavenis, Dash7, EnOcean. These standards are compared in Table 2.1. However, these standards differ in areas of application. Dash7, ISA100 and WirelessHART will be found in traditional manufacturing environments. Technologies like Bluetooth, ZigBee, Wi-Fi and EnOcean are more applicable in end-consumer applications like healthcare tracker, automotives, smart buildings, smartphones. Wavenis protocol is used in Honeywell smartmetering modules.

	IEEE 802.15.4 (ZigBee)	UWB IEEE 802.15.4a	Blue-tooth	BLE	Z-wave	ANT	Wavenis	Dash7	EnOcean
Frequency (ISM)	868/915 MHz; 2.4 GHz	3.1-10.6 GHz	2.4 GHz	2.4 GHz	sub-1 GHz	2.4 GHz	868, 915, 433 MHz	433 MHz	868;315 MHz
Max Data rate	250 kbps	110 Mbps	3 Mbps	1 Mbps	40 kbps	1 Mbps	100 kbps	200 kbps	125 kbps
Range	100 m	10 m	10-100 m	200 m	30 m		1-4 km	2 km	300 m
Battery life	Days-years	Multi-year		Months-years	Multi-year	Year	Multi-year	Multi-year	Battery-less
Network topology	Star, P2P, Mesh		P2P	P2P	Mesh	Star, P2P Tree Mesh	P2P		
Power consu-mption	Low	Low	Low	Ultra-low	Low	Ultra-low	Ultra-low	Low	Ultra-low
Open	✓	✓	✓	✓	✓	X	✓	✓	✓
IPv6	✓			✓	✓		✓		
Target Market/ Applic-ation	Smart-meter, Smart grid devices	Real-time monitor and track location (Indoor)	Consumer electronics	Health fitness, Smart devices	Home autom-ation, security, consumer electronics	Health Fitness Heart-rate monitor, Speed sensors	M2M, Smart meter, Telemetry, Home automation	Mobile payments, Smart meter, Supply chain,	Building, Industrial Automation self-powered sensors, switches

Tab. 2.1: WSN standards and technologies [291]

2.1.3 Agent-based approaches

According to Russel and Norvig [305, p. 60], *agents* (also called intelligent agents) have sensors for perception of environment and actuators for accomplishment of actions towards environment, see Figure 2.3. They also have an internal program which implements artificial intelligence algorithms. This internal program analyzes the sensor input and delivers an appropriate action output. All agents can learn and improve their action behavior. As an example for agents, they mention mainly roboters (e.g. for packaging, controller or deciding systems). Several agents can cooperate and build a multi-agent-system.

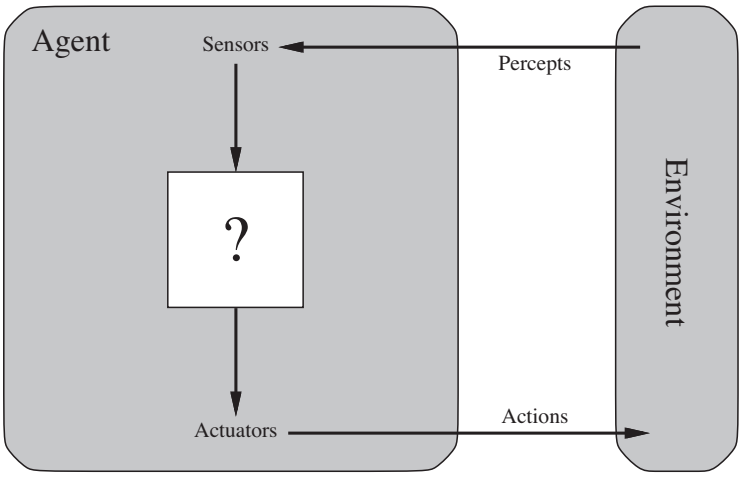


Fig. 2.3: Agents interact with environments through actuators and sensors [305]

In common with WSN and devices in the IoT, agents have sensors and actuators. They also perceive and interact with the environment. In contrast of the WSN and constrained devices in the IoT, agents run only on non-constrained devices and their programs need more computational power.

Further, one category of agents are *knowledge-based agents* [305, p. 291]. They use knowledge base (KB) and logic. Knowledge base consists of data sets and is described with a knowledge representation language. There must be a possibility to add data sets (TELL) and to query over the data sets (ASK). Both tasks can include an *inference* that is new data sets can be derived from existing ones. Using knowledge base and logic are the common aspects of agents and the IoT.

Kaefer and Harth in [182] developed user agents for the IoT. They adopt the idea of operational agent-oriented programming [362]. Abstract State Machines (ASMs) provide operational semantics based on rules. Using them, they were able to specify the so called user agent behavior.

In [1], negotiation approaches for agents were applied to WSNs and analyzed. They were surveyed in three contexts: Common Settings, Service Provision/Acquisition, Wireless Networks. Indeed, agents and IoT devices are similar in some aspects, and IoT devices can learn from the agents.

2.2 Application Protocols for IoT

Communication protocols in the IoT are classified into application layer, transport layer, network layer and data link (also called physical) layer protocols, see also [100, 131, 353]. OSI reference model can be matched but some layers are missing in Machine-to-Machine (M2M) communication and can be ignored (i.e. presentation and session layers). However, some protocols define a whole stack over all layers (e.g. Zigbee, Bluetooth, EnOcean). An overview of the common IoT protocols and layers is given in Figure 2.4. Based on the surveys of the protocols in the IoT [353, 5, 321], the IoT protocols are classified in three categories: data link, network and application layers, see Table 2.2.

<table>
<tr><td>7. Application</td><td>HTTP, MQTT, MQTT-SN, CoAP, XMPP, WebSocket</td><td rowspan="8">Zigbee Stack</td><td rowspan="8">Bluetooth Stack</td></tr>
<tr><td>6. Presentation</td><td>SSL, TLS, DTLS</td></tr>
<tr><td>5. Session</td><td></td></tr>
<tr><td>4. Transport</td><td>TCP, UDP</td></tr>
<tr><td>3. Network</td><td>IPv4, IPv6, RPL</td></tr>
<tr><td rowspan="2">2. Data Link</td><td>6LoWPAN</td></tr>
<tr><td>IEEE 802.11 MAC, IEEE 802.15.4 MAC</td></tr>
<tr><td>1. Physical</td><td>IEEE 802.11, IEEE 802.15.4</td></tr>
</table>

Fig. 2.4: OSI Model and IoT protocols according to [131, p. 114]

Further, following abbreviations are common in the IoT communication: Machine-to-Machine (M2M), Device-to-Device (D2D), Device-to-Service (D2S), Service-to-Service (S2S). When machines or devices communicate with each other, the terms M2M or D2D are used (e.g. DDS as a fast bus for integrating intelligent machines considered as D2D protocol). When sensor data is collected by a device and will be sent to a server, the term D2S is used (e.g. MQTT as a protocol for collecting data from devices and distributing it to clients; XMPP as a protocol for connecting devices to people, since people are connected to the servers, a special case of the

Tab. 2.2: IoT Protocols grouped by layers

Application Layer
MQTT, CoAP, HTTP REST, XMPP, AMQP, DDS
Network Layer
IPv6, RPL, 6LoWPAN, Thread
Data Link Layer
Bluetooth Low-Energy (BLE), ZigBee, Z-Wave, DECT/ULE, LoRa WAN, Sigfox, Dash7, ANT+, Cellular GSM/3G/4G/5G, NB-IoT, LTE-A, IEEE 802.15.4, IEEE 802.11 (WLAN)

D2S pattern). When the server infrastructure has to share data: the S2S is used (e.g. AMQP as a queuing system designed to connect servers to each other).

Communication technologies such as Near Field Communication (NFC) and Radio Frequency Identifier (RFID) [344] are also considered in the IoT but they are passive (read only) and do not build a network in common sense.

We introduce and compare the two common IoT protocols: MQTT(-SN) and CoAP in the next sections. Their protocol stack is shown in Figure 2.5. Both protocols are IP-based, either on the Ethernet, WLAN or 6LoWPAN. The OSI model layer 4 differs, either TCP for MQTT or UDP for MQTT-SN and CoAP.

<table>
<tr><td>7. Application</td><td>MQTT</td><td>MQTT-SN</td><td>CoAP</td></tr>
<tr><td>6. Presentation</td><td>TLS</td><td></td><td>DTLS</td></tr>
<tr><td>5. Session</td><td></td><td></td><td></td></tr>
<tr><td>4. Transport</td><td>TCP</td><td>UDP</td><td>UDP</td></tr>
<tr><td>3. Network</td><td>IP</td><td>not specified</td><td>IPv6</td></tr>
<tr><td rowspan="2">2. Data Link</td><td rowspan="3">not specified</td><td rowspan="3">not specified</td><td>6LoWPAN</td></tr>
<tr><td>IEEE 802.15.4 MAC</td></tr>
<tr><td>1. Physical</td><td>IEEE 802.15.4</td></tr>
</table>

Fig. 2.5: Typical protocol stack for MQTT, MQTT-SN and CoAP according to [131, p. 133]

2.2.1 MQTT

The Message Queue Telemetry Transport (MQTT) is one of the oldest application protocols in the IoT. The MQTT protocol was developed by IBM in 1999 and 15 years later standardized in Version 3.1.1 by Oasis [242] and by ISO/IEC [168] in 2016. MQTT-SN is a lightweight version for Sensor Networks which was specified by IBM in Version 1.2 in 2013 [370] but never standardized. Finally, MQTT Version 5.0 was finalized by Oasis in 2019 [243] and optimized for use in constrained environments.

The most popular implementations of the MQTT protocol are: an open-source Mosquitto broker from Eclipse Foundation [90], commercial broker HiveMQ [149], open-source MQTT-SN client software from Eclipse Paho [271]. Big player like

IBM [159] and Amazon [15] use MQTT as the default protocol for the IoT applications.

First, the basic function of the MQTT protocol will be described. Afterwards, the differences of the MQTT-SN will be outlined. Finally, the new features in the version 5.0 will be introduced. The MQTT protocol is defined by the specification as following:

> MQTT is a Client Server *publish/subscribe* messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium. [242]

The MQTT server is often referred to as a *broker*. Also, for distinction to traditional Client-Server architecture, we use the term *broker* in the following. The protocol runs over TCP/IP or WebSocket [104] connection. It supports Transport Layer Security (TLS) for secure transport. Additionally, *User and Password* flags can be used for authentication.

Fig. 2.6: MQTT Topic Structures [149]

MQTT Topics

Messages reference so-called *Topics* of interest. The *topic name* is an UTF-8 string (case-sensitive!) and consists of one or more levels which are separated by a forward slash (so-called topic level separator). The topic name will be created on the fly in the broker, as soon as a publish or subscribe message arrives. Figure 2.6 shows some examples of the topic structures. Besides the topic name, there are *topic filters* with wildcards possible. They can only be used on a subscription. The single-level wildcard is represented by a plus and replaces one topic level which can contain an arbitrary string. The multi-level wildcard is represented by a hash symbol and must be placed as the last character and preceded by a forward slash. If a client subscribes to such a topic, it will receive all messages of topics which start with

the string before the wildcard. Topic names beginning with $SYS/ are reserved for internal use of the broker.

Publish/Subscribe Paradigm
The MQTT broker manages the topics, publications and subscriptions. The broker notifies the clients about new messages. Figure 2.7 shows how the publish/subscribe messaging works. Clients are publishers and subscribers. The client on the left side is a temperature sensor, which publishes measured sensor values. The clients on the right side are subscribers which subscribed to this topic. This topic must be known by both, publishers and subscribers and is usually determined before.

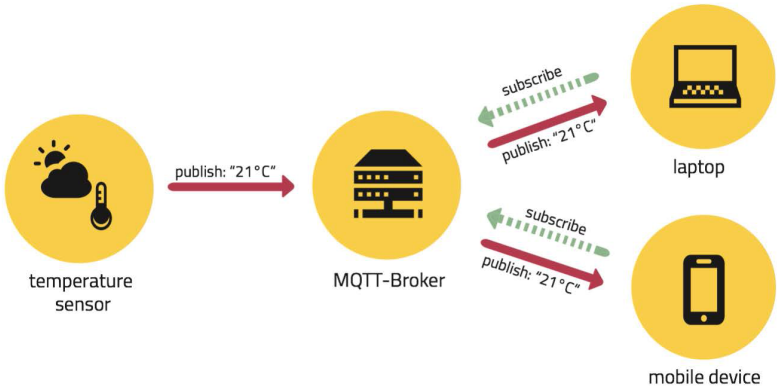


Fig. 2.7: Publish/Subscribe Messaging in MQTT [149]

Quality of Services
There are three qualities of service (QoS) for message delivery defined up on a publish:

- *QoS 0: At most once delivery* Messages will be delivered to the best efforts of the operating environments. Messages can be lost. When many sensor data will be send periodically, it doesn't matter if one of the records is lost because the next one will be published soon.
- *QoS 1: At least once delivery* Messages guarantee to be delivered but duplication can occur.
- *QoS 2: Exactly once delivery* Message guarantee to be delivered exactly once. This level can be used in sensitive data environments where duplicate or lost messages could lead to incorrect data.

Control Packets
The MQTT protocol works by exchanging several MQTT *Control Packets* in a defined way. There are 15 control packet types specified in the version 3.1.1, as shown in Table 2.3. Every control packet includes a fixed header but only some packets include a variable header and a payload. The control packets CONNECT, CONNACK and DISCONNECT are responsible for establishing connection before publish or subscribe and also for disconnection after message transport. The control packets PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP will be exchanged upon a publish with appropriate QoS. The control packets SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK will be used on subscription and unsubscription of topics. The control

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Tab. 2.3: Control packet types in MQTT 3.1.1 [242]

topics PINGREQ and PINGRESP are used by clients for PING request and response of the broker.

There are some flags defined in the CONNECT control packet. They define the robustness of communication. If the *Will Flag* is set, the broker must store a *Will Message* (also called Last Will) which will be sent upon a DISCONNECT packet. The *Will Retain* flag specifies if the Will Message should be retained when it is published. Each client that subscribes to a topic pattern which matches the topic of the retained message will receive the retained message immediately after they subscribe. The broker stores only one retained message per topic.

MQTT-SN

The first version of MQTT-SN was the subject of WSN research in [158] (yet called MQTT-S). Design goals for this protocol were: (i) as close as possible to MQTT; (ii) optimized for tiny sensor/actuator devices; (iii) consideration of constrained network with high link failure rates, low bandwidth, and short message payload; (iv) network independent: point-to-point and one-hop broadcast data transfer services must be possible. Based on these goals, there are some differences in MQTT-SN:

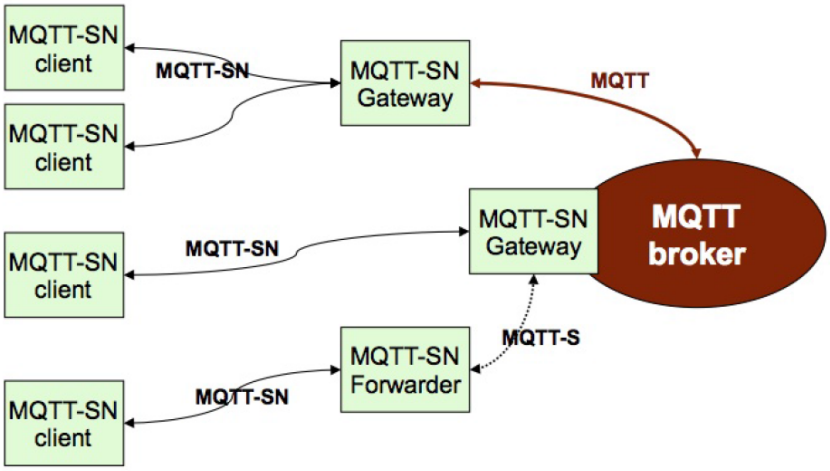


Fig. 2.8: MQTT-SN Architecture [370]

- User Datagram Protocol (UDP) is used as the underlying network protocol.
- *Gateways* and *forwarders*, see Figure 2.8 were introduced to support a discovery procedure. Furthermore, gateways can be transparent or aggregating.
- Messaging optimization, namely shortened topic as topic-id, pre-defined topic ids and short topic names, address short message length and the limited transmission bandwidth in WSN.
- The CONNECT message is split into three messages. The two additional messages are optional and can used to transfer the Will topic and the Will message to the broker.
- *Sleeping* clients are supported by the new keep-alive procedure. During the sleeping time all messages will be buffered by the broker/gateway and delivered on the wake-up.
- The discovery procedure includes following control packets: ADVERTISE (broadcasted periodically by a gateway), SEARCHGW (broadcasted by a client when it searches for a gateway), GWINFO (response to a SEARCHGW message).

Related research shows some weaknesses of the MQTT-SN protocol. Roy et al. [127] proposed and evaluated so called smart gateway selection method because to find the next right gateway turns as a challenging task in a network with several gateways. The researcher in [247] compared the performance of MQTT-SN with MQTT and CoAP. They conclude, that MQTT-SN implementation is not mature for the real world.

MQTT 5.0

The main motivation for the new version [243] was to improve performance and support for constrained devices, to enhance security and scalability for large systems and to meet the developer's requirements for more robustness.

Constrained devices
Support for constrained clients is assured by a range of following features. *Topic Alias*

allows the topic name to be abbreviated to a small integer. Similar, a *Subscription Identifier* allows a numeric identifier to be specified on a SUBSCRIBE. This allows the Client to determine which subscription or subscriptions caused the message to be delivered. *ClientID* identifies the Client to the broker and can be generated by the client or assigned by the broker. Constrained clients can save resources when they set Receive Maximum (limit the number of QoS 1 and QoS 2 publications that they are willing to process concurrently), Maximum Packet Size (specify the maximum packet size they are willing to accept), Topic Alias Maximum (determine how many topic aliases they allow). Therefore, the constrained clients have Flow Control over the message exchange.

The following features are especially interesting for constrained clients who are not often connected with a broker. The Clean Session flag in the CONNECT variable Header is replaced through the new *Clean Start* flag in the CONNECT variable Header and the new *Session Expiry Interval* in the CONNECT Properties. The *Message Expiry Interval* defines how long a published message should be stored.

Robustness
Two big issues were formalized for software developers in the new protocol version. *Properties and User Properties* as well as *Reason Code and Reason String* were introduced in almost all packets. Properties are part of the control packets (Variable Header and Payload) and determine the underlying conditions for messaging transport. Every control packet has another subset of properties. *Reason codes and Reason string* are included in all acknowledge control packets returned by the broker. They indicate the result of an operation whether the request succeeded or not.

The Request/Response pattern is now supported by properties *Response Topic*, *Correlation Data*, *Response Information*, *Request Response Information*. They allow response messages "to be routed back to the publisher of a request".

In the previous version of MQTT the payload was sent as binary data. Now, the payload format can be set by the new *Payload Format Indicator* and the new *Content Type* which uses the MIME content type.

Scalability and Security
Shared Subscription Topic Filter with pattern $share/ShareName/filter was introduced for load balanced subscribers. A subscription option *No Local* can be used to implement bridging applications. This features allow to scale the system around MQTT.

There is only one new Control Packet with the value 15 which is an AUTH control packet and used for authentication exchange besides the existing user and password. The client can include an Authentication Method and Authentication Data in the CONNECT packet and AUTH Properties in the AUTH Variable Header. This mechanism should allow SASL (Simple Authentication and Security Layer) style authentication to be used if supported by both client and broker.

Summarizing, MQTT protocol with its publish/subscribe pattern could assert itself since 18 years in the IoT world. The new version 5.0 reflected developer's issues and address constrained environments as well as optimized for large scale. MQTT-SN is designed for WSN but was not standardized and is not further developed.

2.2.2 CoAP

The Constrained Application Protocol (CoAP) was specified by the Internet Engineering Task Force (IETF) Working Group named Constrained RESTful Environments (core). It was published as RFC 7252 [356] in 2014 and therefore it is quite a recent protocol. CoAP is a "specialized web transfer protocol for use with constrained nodes and constrained networks". It is designed for Machine-to-Machine (M2M) applications such as building automation, smart energy, etc.. The main considerations of the protocol were: easily interface with HTTP for integration with the Web, small overhead, multicast support (RFC 7390 [287]), discovery via service and resource discovery (CoRE Resource Directory [358]), asynchronous message exchange (over UDP). In this section, the basic concepts and functions of CoAP protocol will be introduced.

CoAP is a datagram-based protocol and runs on top of the User Datagram Protocol (UDP) protocol. It supports the Datagram Transport Layer Security (DTLS) for security. The HTTP and CoAP protocols are directly compared in the Figure 2.9. The main differences are TCP and UDP on the transport layer, WLAN and 6LoWPAN on the data link and physical layers. Of course, the CoAP protocol is working on IPv4 and WLAN as well.

Web Stack

Layer	Protocol
7. Application	HTTP
6. Presentation	HTML
5. Session	SSL
4. Transport	TCP
3. Network	IP (IPv4)
2. Data link	IEEE 802.11 (Wi-Fi)
1. Physical	

Constrained Environment

Layer	Protocol
7. Application	CoAP
6. Presentation	
5. Session	
4. Transport	UDP
3. Network	IP (IPv6)
2. Data link	IEEE 802.15.4 (LoWPAN)
1. Physical	

Fig. 2.9: HTTP and CoAP Protocol Comparison based on OSI model

CoAP is based on Representational State Transfer (REST)ful architecture [105] which is based on request/response interaction. The resources on the endpoints are identified by an URI with coap-prefix, e.g. `coap://server_address/sensors/temperature`. The methods are defined in similar manner to HTTP: GET, POST, PUT, DELETE.

CoAP Messages

The message transport over UDP is unreliable: "messages may arrive out of order, appear duplicated, or go missing without notice"[356]. This leads to the asynchronous message exchange. For this reason, CoAP implements a lightweight mechanism to achieve reliability. There are four types of messages: *Confirmable, Non-confirmable, Acknowledgement, Reset*. The combination how the message types can be used within request and response is provided by the Figure 2.10. Reading this table, a confirmable request require a confirmable response, and a non-confirmable request expects a non-confirmable response, if any available. "*" is used to provide a "CoAP ping" on help of the Reset message.

The CoAP message consists of a fixed-size 4-byte header, a token for correlation of messages, options and a payload. The message size is constrained and according to

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Fig. 2.10: Usage of CoAP Message Types [356]

the RFC 7252 should "fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously needs to fit within a single IP datagram". Therefore, the message size results with 1280 Byte in the IPv6 network and 576 Byte in IPv4 network. The second one is the datagram size in UDP. The specification recommends the absolute minimum value of the IP Maximum Transmission Unit (MTU) for IPv4 as 68 bytes.

Reliable messaging is provided by the Confirmable (CON) message: "the recipient sends an Acknowledgement message (ACK) with the same Message ID (for example, 0x7d34) from the corresponding endpoint", see Figure 2.11. If a message does not require reliable transmission (for example, sending sensor data), then it can be sent as a Non-confirmable message (NON). There is no acknowledgement but a Message ID for duplicate detection (for example, 0x01a0).

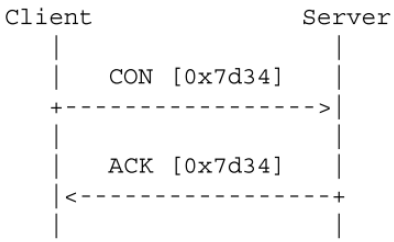


Fig. 2.11: Reliable Message Transmission [356]

If the request is carried in a confirmable message (CON) and the response is immediately available, there is no need to send the acknowledgement (ACK) separately from the response. Such response is called *piggybacked response* and is shown in the Figure 2.12.

CoAP Proxying

For the interface with HTTP, the cross-protocol proxying between CoAP and HTTP was specified. Because the CoAP protocol supports only a limited subset of HTTP functionality, an intermediary is required. For CoAP-HTTP Proxying, CoAP clients are enabled to access resources on HTTP servers through an intermediary: "This is initiated by including the Proxy-Uri or Proxy-Scheme Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy". For HTTP-CoAP Proxying, HTTP clients are enabled to access resources on CoAP servers through an intermediary: "This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy". The guidelines for mapping implementations for HTTP-CoAP Proxying are specified meanwhile in the RFC 8075 [58].

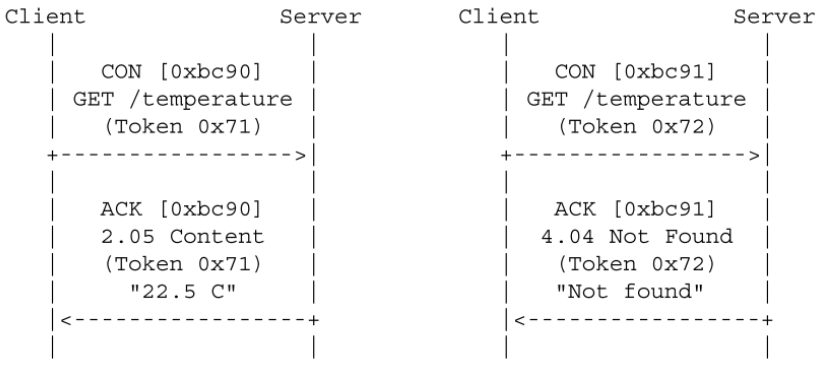


Fig. 2.12: GET Requests with Piggybacked Responses [356]

CoAP Resource Directory

In order to discover the services offered by a CoAP server, a client must know the endpoint (a URI) used by a server. The server is should be reachable at the default port number 5683 (5684 for DTLS). The URI contains `.well-known/core`, for example like this `coap://example.net/.well-known/core`. The client makes a GET request on this URI. As a response, a list of available resources or services will be received. The *if* attribute provides the interface description. For example:
$</sensors/temp>;if = "sensor"$,
$</sensors/light>;if = "sensor"$

Querying all connected devices can be a tedious task. More efficient is to use a *CoRE Resource Directory* [358]) which provides information about resources hosted by other devices. The single device has to register at the resource directory once and its services are available for all clients. Alternatively, clients can use multicast CoAP to find CoAP servers. This approach is described in the next section.

Multicast and Group Communication

The RFC 7390 [287] about CoAP group communication includes mainly use cases and their protocol flows as well as deployment guidelines. The term "group communication" means the one-to-many communication between CoAP endpoints. All destination nodes belong to this group. The source node may be a part of the group or outside the group, compare the nodes A and C in the Figure 2.13. The groups may be preconfigured or dynamically formed. The group communication runs on constrained network such as a low-power, lossy network. Further on, the UDP/IP multicast for the requests and the unicast UDP/IP for the responses are the underlying technologies.

IP multicast is the real-time communication which uses network layer in OSI reference model. It uses specially reserved IP multicast address blocks in IPv4 and IPv6. The basic principle is as following: the source sends the packet only once, and the nodes in the network replicate this packet. This builds a multicast distribution tree. The big advantage of the IP multicast is the high scalability, because it does not require knowledge about the receiver identity or the number of receivers. Further considerations on IP multicast are routing and forwarding capabilities and reliable group communication. A complete IP multicast solution may include support for

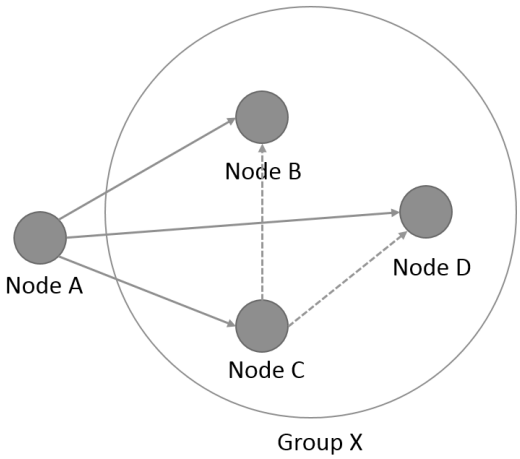


Fig. 2.13: CoAP Group Communication

managing group memberships and IP multicast routing/forwarding, e.g. (RPL) in RFC 6550 [415].

Member Discovery and Configuration
A resource directory (RD) can be used for lookup of CoAP groups and memberships, but is not required. The configuration of the members in a group can be made in one of the following ways:

- Membership can be pre-configured before deployment. For this purpose IP Multicast address or hostname Fully Qualified Domain Name(FQDN) is needed.
- A Node uses specific service directory. This is a programming way, where techniques such as DNS-based Service Discovery (DNS-SD) and RD are used.
- A Node is configured by another node. The configuring node can be i.e. a commissioning device.

The methods POST, DELETE, PUT, GET are used to create a new multicast group membership, to delete, read or update it. The group membership contains key/value pairs. The "n" stands for "name" and identifies the group with a hostname. The "a" key/value pair specifies the IP multicast address of a group. It contains an IPv4 address or an IPv6 address. An example for such membership is the following:
"*n*": "*All-Devices.floor1.west.bldg6.example.com*",
"*a*": "*[ff15::4200:f7fe:ed37:abcd]:4567*"

As shown in Figure 2.14, the network topology of a large room (Room-A) includes three lights controlled by a light switch. The lights are organized into two subnets. The two routers are connected to an IPv6 network backbone which is multicast enabled. A CoAP RD and a controller (CoAP client) are connected to the network backbone. The DNS server is optional.

Ongoing Development of CoAP
CoAP requires the necessity to run a server on a device enabling REST-based requests. This works fine for devices which have constant power supply but are problematic for constrained devices powered by battery.

Therefore the CoAP protocol was further developed and improved over the last years. The observation pattern for CoAP was introduced by RFC 7641 [140]. In case that

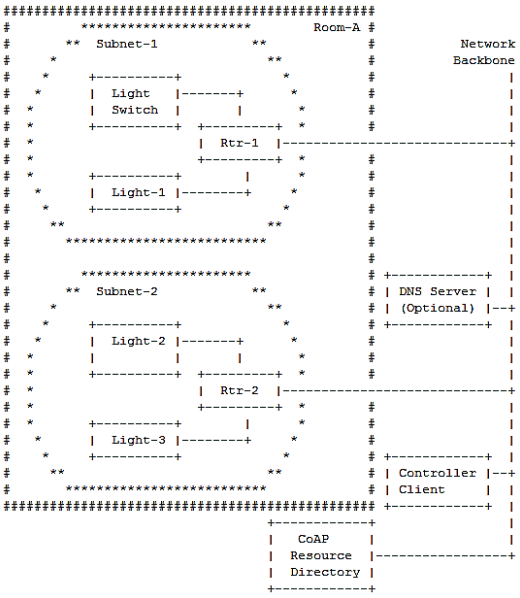


Fig. 2.14: Use Case: Network Topology with CoAP protocol [287]

the data doesn't fit in a single IP packet, the block-wise transfers were introduced in CoAP by RFC 7959 [44]. Because UDP is an unreliable protocol, a specification for CoAP over TCP, TLS, and WebSockets was defined by the RFC 8323 [43].

Even though the CoAP protocol is implemented by several frameworks and programming languages[4], the protocol is still not supported by big platforms like Amazon. The CoRE Resource Directory is still a draft, and the RFC about group communication is experimental. Thus, CoAP is not mature for the real world.

CBOR Encoding

The common message format for RESTful application is JSON [75], as well as for CoAP protocol. JavaScript Object Notation (JSON) is already a compact format compared to XML. However, another IETF working group made an effort to develop a data format which should reduce the size of JSON data by binary encoding. The Concise Binary Object Representation (CBOR) was originally defined in RFC 7049 [41] in 2013. This RFC is made obsolete by the new version RFC 8949 [42] while staying compatible with the format of the RFC 7049. No new version of the format will be created. Basically, the new RFC provides editorial improvements, new details, and errata fixes.

CBOR Encoding was discussed in [314]. The goal of the CBOR is a standardized format for binary representation of structured data. The conversion from JSON to CBOR and vice versa were defined as part of the specification. Objectives of CBOR are the representation of basic data types and structures of JSON using binary encoding, a compact encoder and decoder code supporting constrained devices, and self-describing data so that a generic decoder can be written. With CBOR encoded data can be decoded without a schema description. CBOR defines 8 major types, see Table 2.4.

[4] `http://coap.technology/impls.html`

Tab. 2.4: CBOR Major Data Types [314]

	Major Type
0	unsigned integer
1	negative integer
2	byte string
3	text string
4	array of data items
5	map of pairs of data items
6	optional semantic tagging of other major types
7	floating-point numbers and simple data types

Each byte is encoded as a major type (the high-order 3 bits) and additional information (the low-order 5 bits), see Table 2.5. The resulting integer in all additional information values is interpreted depending on the major type. For example, it represents the integer value itself for integer type if the value of additional information is less than 24. The integer value 20 is encoded as *0x14* or binary *000 10100* with major type 0 and additional information 20.

Tab. 2.5: Initial byte of each data item in CBOR [314]

X X X	X X X X X
major type	additional information

The integer value 128 is encoded as *0x1880* or binary *000 11000 10000000* with major type 0, additional information 24 and value 128. The additional information 24 signals that additional bytes for an integer immediately follow. The byte string "*CBOR*" is encoded as *0x6443424F52* or binary *011 00100 01000011 01000010 01001111 01010010* with major type 3, additional information 4 (string length) and the string value in bytes. An implementation of a CBOR decoder can use a jump table given in the RFC with all 256 defined values for the initial byte.

Meanwhile, CBOR has developed to an encoding method for constrained devices for use with CoAP. For example, CBOR Object Signing and Encryption (COSE) is specified in the RFC 8152 [325] using CBOR for serialization of signatures, message authentication codes, and encryption.

2.3 Network and Data Link Layers Protocols

In this thesis the IP-based communication for IoT devices is considered. The advantage of using IP-Based protocols for the IoT is that the existing IP-based transport protocols such as well-established TCP and UDP can be used. It is assumed, that the reader is familiar with the well-known IP, TCP and UDP protocols and they will not be described in detail in this thesis. The underlying data link layer (layer 2 in the OSI model) is either WLAN or IEEE 802.15.4, as shown in Figure 2.15. The 6LoWPAN protocol stands for "IPv6 over IEEE 802.15.4 networks" [237]. There are

also other, mostly commercial, protocols layered over IEEE 802.15.4 (e.g. Zigbee[5], Thread[6], WirelessHART[7], ISA100.11a[8]) but they are not IP-based.

The IEEE 802.15.4 hardware is supported by several operating systems such as Contiki and Contiki-NG, TinyOS, RIOT, Mbed OS, Zephyr, OpenWSN, Unison RTOS. However, only a few are still actively maintained (e.g. Contiki-NG, RIOT, Mbed OS, Zephyr, Unison RTOS).

WLAN is a long-existing and proven technology. WLAN networks are widely deployed. Microcontroller boards are often supplied with a low-power WLAN chip, for example EPS32 [96] chip module.

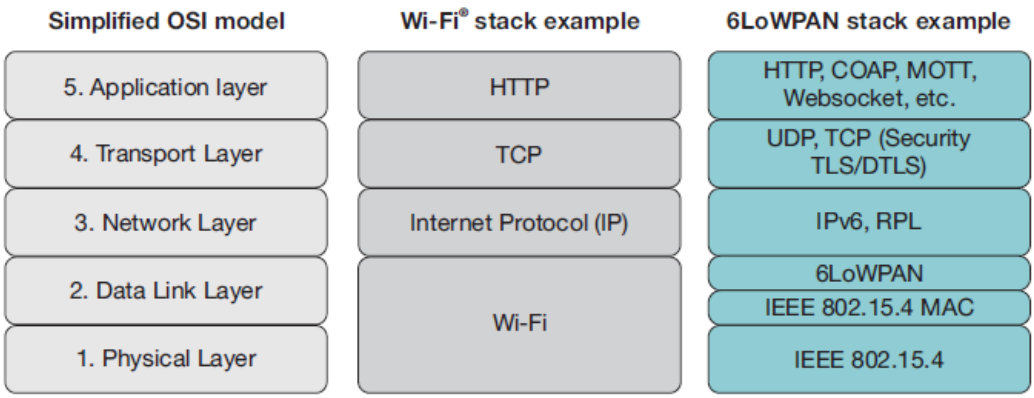


Fig. 2.15: OSI Model with Wi-Fi Stack and 6LoWPAN Stack in Comparison [264, p. 4]

2.3.1 IPv6 over Low Power Networks

IETF has following workgroups for IPv6 over Low Power Networks and of Resource-constrained Nodes: IPv6 over Low power WPAN (6lowpan)[9], 6lo (IPv6 over Networks of Resource-constrained Nodes)[10], lpwan (IPv6 over Low Power Wide-Area Networks)[11]. While focusing on the work of the 6lowpan and 6lo Working Groups, we do not consider the lpwan group because they work on IPv6 over LPWAN networks such as LoRaWAN, Sigfox, NB-IoT, Wi-SUN Alliance Field Area Network (FAN).

The 6lo Working Group specified also the RFC 7668 for IPv6 over Bluetooth Low Energy. The MTU is 23 bytes (for BLE 4.0/4.1) [145], 251 bytes (for BLE 4.2, PDU is 257 bytes). Although, Bluetooth Low-Energy (BLE) is also broadly used in the IoT, it is not sufficient to build a network with hundreds devices. However, BLE is often used for configuration of single devices.

[5] https://zigbeealliance.org/
[6] https://www.threadgroup.org/
[7] https://fieldcommgroup.org/technologies/hart/hart-technology
[8] https://www.isa.org/isa100/
[9] https://datatracker.ietf.org/wg/6lowpan/documents/
[10] https://datatracker.ietf.org/wg/6lo/documents/
[11] https://datatracker.ietf.org/wg/lpwan/documents/

IEEE 802.15.4

IEEE 802.15.4 defines the operation of Low-Rate Wireless Personal Area Networks (LR-WPANs). The protocol targets devices using low-data-rate, low-power, and low-complexity short-range radio frequency (RF) transmissions in a wireless personal area network (WPAN). The current version is IEEE 802.15.4-2015 [160].

The protocol specifies the Physical layer (PHY) and the Data Link Layer for LR-WPANs. IEEE 802 splits the OSI Data Link Layer into two sub-layers: Logical Link Control (LLC) Sublayer and Media Access Control (MAC) Sublayer. LLC Sublayer defines multiplexing protocols transmitted over the MAC layer (when transmitting) and decoding them (when receiving).

Devices are organized into Personal Area Network (PAN)s. IEEE 802.15.4 distinguishes two device types [160]: a Full Function Device (FFD) and a Reduced Function Device (RFD). The FFD is capable of serving as a PAN coordinator or a coordinator. The WPAN includes at least one FFD, which operates as the PAN coordinator. The RFD is not capable of serving as either a PAN coordinator or a coordinator. It provides simple applications such as a light switch or a passive infrared sensor. It does not need to send large amounts of data. The RFD is only associates with a single FFD at a time and using minimal resources and memory capacity.

IEEE 802.15.4 defines two addressing modes: 64-bit globally unique device address and 16-bit addresses unique within the PAN. The 16-bit address is assigned by the PAN coordinator function during an association event. There is also a broadcast address which enables to address all nodes in a PAN. This is a default short address 0xffff.

Each independent PAN selects a unique identifier a 16-bit number. The PAN identifier allows communication between devices within a network using short addresses [160]. It enables transmissions between devices across independent networks and can be pre-determined or scanned for at coordinator start-up time. The PAN coordinator is usually an FFD device. Each PAN has only one PAN coordinator which is used to initiate, terminate, or route communication around the network and assign short addresses to devices.

The devices in a IEEE 802.15.4 network can build a star topology or a peer-to-peer topology, see Figure 2.16. The Star topology has a PAN coordinator as a single central controller. Devices use either the extended address or the short address for direct communication within the PAN. The PAN coordinator will often be mains powered. Devices are mostly battery powered. Typical applications are home automation, personal computer (PC) peripherals, and personal health care.

Peer-to-peer topology has one PAN coordinator, but any device is able to communicate with any other device. Mesh networking topology is possible. This allows multiple hops to route messages from any device to any other device on the network. Typical applications are industrial control and monitoring, wireless sensor networks, asset and inventory tracking, intelligent agriculture, and security [160].

It is also possible to build a Cluster Tree Network, see Figure 2.17. Then, the most devices are FFDs. RFD connects as a leaf device at the end of a branch. Any FFD is able to act as a coordinator. Only one of these coordinators is the overall PAN coordinator.

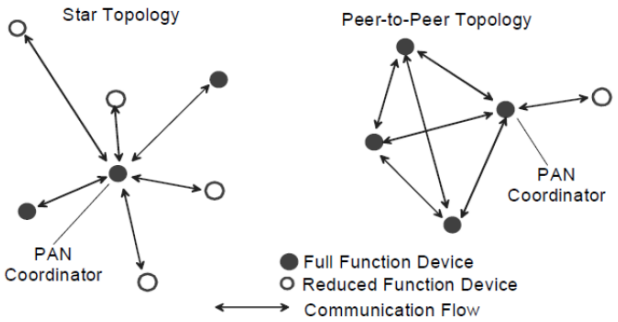


Fig. 2.16: Star topology and peer-to-peer topology in IEEE 802.15.4 [160]

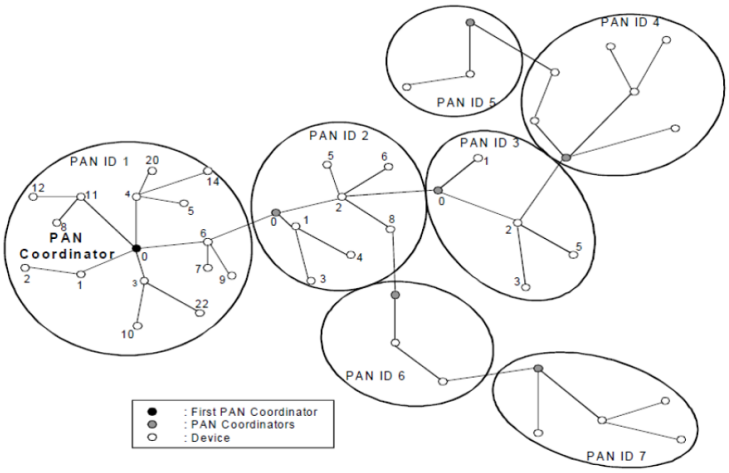


Fig. 2.17: Cluster Tree Network in IEEE 802.15.4 [160]

There are three types of data transfer transactions: to a coordinator, from a coordinator, and between two peer devices. In a star topology, data is exchanged only between the coordinator and devices. In a peer-to-peer topology, all three transactions are used in this topology.

Superframes and Beacons

For communication during the Contention Access Period (CAP) between two beacons, any device competes with other devices using a slotted CSMA-CA or ALOHA mechanism [160], as appropriate. The PAN coordinator reserves portions of the active superframe to that application. These portions are called Guaranteed Time Slots (GTS) and form the Contention-Free Period (CFP), which appears at the end of the active superframe and starts at a slot boundary immediately following the CAP, see Figure 2.18. GTSs are allocated by the PAN coordinator for low-latency applications. The size of CFP depends on total lengths of GTSs (up to seven GTSs). The GTS can occupy more than one slot period. Before the CFP begins, all contention-based transactions are completed. Additionally, each device transmitting in a GTS has to ensure that its transaction is complete before the time of the next GTS or the end of the CFP.

The superframe is bounded by network beacons which are sent by the coordinator, see Figure 2.19. It is divided into 16 slots of equal duration and can have an active and an inactive portion. The format of the superframe is defined by the coordinator.

Beacons are used "to synchronize the attached devices, to identify the PAN, and to describe the structure of the superframes" [160].

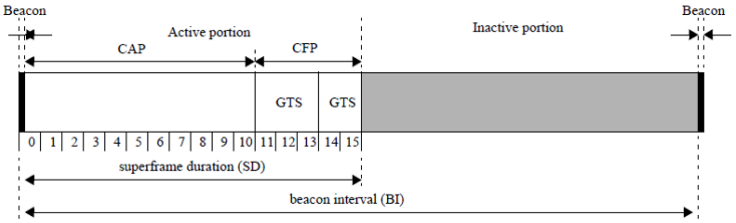


Fig. 2.18: Superframe with beacons [160]

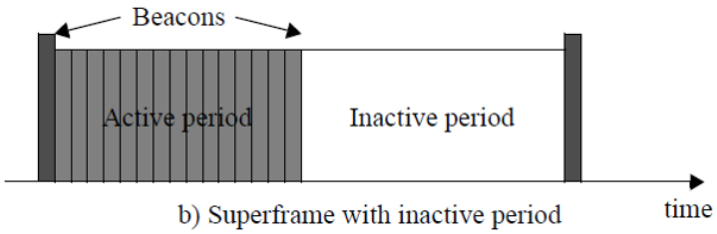


Fig. 2.19: Superframe with inactive period [160]

ALOHA and CSMA-CA

Device communication during the Contention Access Period (CAP) has to use a slotted CSMA-CA or ALOHA mechanism. ALOHA mechanism works as the following: "a device transmits without sensing the medium or waiting for a specific time slot" [160]. ALOHA is appropriated for lightly loaded networks because the probability of collision is quite small when the probability of clear channel is large enough. Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) is an algorithm. There are paths for Time-Slotted Channel Hopping (TSCH) mode (special case of IEEE 802.15.4), slotted and unslotted CSMA-CA. The basic idea of this algorithm is to perform the clear channel assessment (CCA) (i.e. sense the medium) before transmitting and therefore to avoid collision. Three variables have to be maintained by a device: *NB, CW and BE*. *NB* is the number of times the CSMA-CA algorithm was required to back off while attempting the current transmission. *CW* is the contention window length, defining the number of backoff periods that need to be clear of channel activity before the transmission can begin. *BE* is the backoff exponent which indicates "how many backoff periods a device shall wait before attempting to assess a channel" [160].

The TSCH mode is an amendment (IEEE 802.15.4e) to the Medium Access Control (MAC) of the IEEE 802.15.4 standard. In short, TSCH introduced a channel hopping approach while changing time and frequencies for every transmission. This approach makes the transmissions in low-power and lossy networks (LLNs) more reliable. 6TiSCH [405] is an IETF Working Group focused on IPv6 over TSCH. The TSCH mode is supported by several implementations and can be enabled via configuration.

IEEE 802.15.4 characteristics

The physical layer has several frequency bands, modulation and data rates. The most microcontroller boards use the 2.4-GHz frequency band because this frequency is license-free. This ISM band is also used by IEEE 802.15.4 and Bluetooth. Thus,

interference may occur. IEEE 802.15.4-2006 specifies 16 channels within the 2.4-GHz band. These channels are numbered 11 through 26 and are 5 MHz apart [381]. The basic framework has a communication range of 10 meters and a transfer rate of 250 kbit/s.

The Maximum Transmission Unit (MTU) of IEEE 802.15.4 in the physical layer is of 127 octets (aMaxPHYPacketSize). The maximum frame overhead is of 25 (aMaxFrameOverhead). The resultant maximum frame size at the media access control layer is 102 octets. The Link-layer security in the maximum case has: 21 octets of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively. This leaves only 81 octets available for payload.

There are two security support mechanisms by MAC and higher layers: encryption and replay protection. MAC supports Advanced Encryption Standard (AES)-CCM* mode of operation. Data confidentiality is "assurance that transmitted information is only disclosed to parties for which it is intended". Data authenticity is "assurance of the source of transmitted information (and, hereby, that information was not modified in transit)". Replay protection is "assurance that duplicate information is detected". The *macFrameCounter* attribute of the originator is used. Additionally, data verification is ensured by Cyclic Redundancy Check (CRC) and is used to detect errors in every PHY service data unit (PSDU).

The authors in [250] compared Bluetooth Low Energy (BLE) and IEEE 802.15.4 (CSMA, Low-power listening, and TSCH), summarizing "both technologies will likely continue to co-exist, making interoperability a central concern". Concluding, IEEE 802.15.4 is a mature standard with high complexity because besides the main specification a lot of amendments exist.

6LoWPAN

The 6LoWPAN [357, 246] specifies the transmission of IPv6 Packets over IEEE 802.15.4 Networks. The 6lowpan Working Group has specified a row of RFCs around 6LoWPAN, and the 6lo Working Group continues this work. The central specification is the RFC 4944 [237]. An example of the 6LoWPAN architecture is shown in Figure 2.20. It consists of *edge router* (also called *border router*), router and hosts. The edge routers are connecting the 6LoWPAN network with the IP-based network (e.g. local network or Internet). The edge routers carry several important tasks for 6LoWPAN: neighbor discovery, header compression, fragmentation. Nodes in 6LoWPAN may be take the role of router or host. Routers are hosts with routing capabilities and enable to build a multihop mesh topology.

Fragmentation
IPv6 packets must be carried on data frames. The maximum physical layer packet size of 127 Bytes (aMaxPHYPacketSize) and a maximum frame overhead of 25 (aMaxFrameOverhead). This results in the maximum frame size at the media access control layer is 102 Bytes. Link-layer security produces further overhead, which in the maximum case (21 Bytes of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively) leaves only 81 Bytes available. Furthermore, since the IPv6 header is 40 Bytes long, this leaves only 41 Bytes for upper-layer protocols such as UDP. The UDP uses 8 octets in the header and leaves only 33 Bytes for application data. The MTU size for IPv6 packets over IEEE 802.15.4

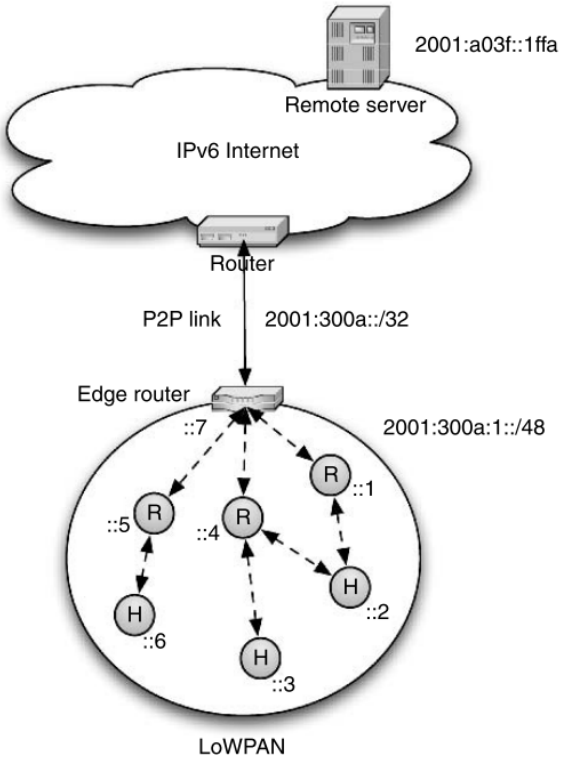


Fig. 2.20: A 6LoWPAN architecture example [357]

is 1280 Bytes. Therefore, a fragmention and reassembly adaptation layer is provided by 6LoWPAN.

Header Compression
Assumption in some cases: using small payload, datagram will fit the packet into a single IEEE 502.15.4 frame. For all other cases a header compression is provided. The RFC 6282 [156] introduced $LOWPAN_IPHC$ and $LOWPAN_NHC$ encoding formats. $LOWPAN_IPHC$ provides a header compression of Unique Local, Global, and multicast IPv6 Addresses. $LOWPAN_NHC$ provides encoding format for next header compression, namely UDP Header Compression. An example for $LOWPAN_IPHC$ header compression is provided in Figure 2.21.

6LoWPAN Security
There are three security objectives for 6LoWPAN which are identified in [357, p. 84]: (i) confidentiality; (ii) integrity and authentication; (iii) availability.

Confidentiality means data remains secret except for the authorized participants. This can be achieved by cryptographic encryption. Integrity means data cannot be modified by unauthorized participants. Authentication means message is originating from the source that claims to be. Both, integrity and authentication, can be achieved by adding cryptographic integrity checks to messages. Availability: DoS (e.g. through jamming) attacks must be prevented.

The threat model for 6LoWPAN is not much different from the general threat model assumed for Internet security protocols in RFC 3552 [296]. More relevant are the device constraints like small code size, low power operation, low complexity, and

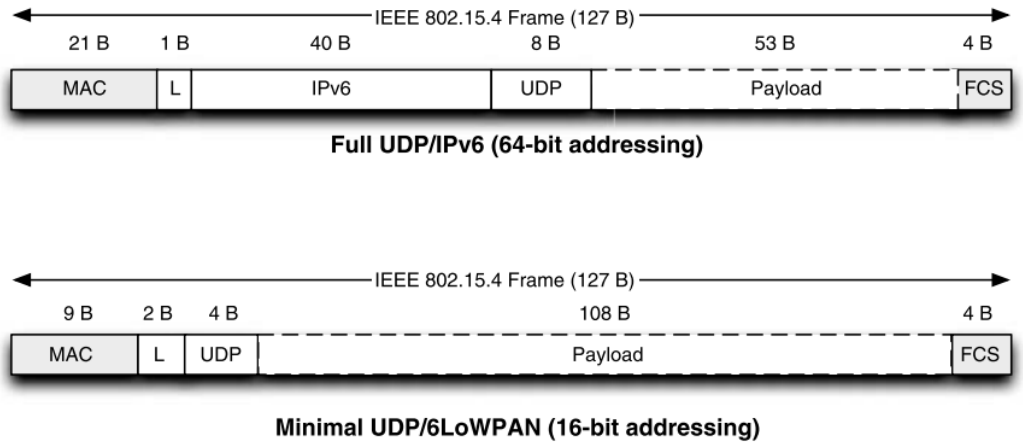


Fig. 2.21: 6LoWPAN header compression example (L = LoWPAN header) [357]

small bandwidth requirements which decide about the security concept for 6LoWPAN networks.

There are some security mechanisms on Layer 2 and 3. For link layer (layer 2), most IEEE 802.15.4 devices have support for AES link-layer security [203]. AES is a "block cipher operating on blocks of fixed length". To encrypt longer messages, several modes of operation may be used. The CCM mode (counter with CBC-MAC) have been designed to ensure both confidentiality and message integrity.

For network layer (layer 3) security, two models are applicable [203]: end-to-end security (e.g. using IPsec (RFC 4301) transport mode) or security that is limited to the wireless portion of the network (e.g. using a security gateway and IPsec tunnel mode). The disadvantage of the latter is the larger header size, which is significant at the 6LoWPAN frame MTUs. A device can use its AES/CCM cryptographic hardware chip also for Encapsulating Security Payload (ESP) encryption [357, p. 87-89] in IPsec and reduce the overhead.

Neighbor Discovery and Routing
6LoWPAN neighbor discovery [355, 386] is optimized for low-power wireless networks. It uses a simplified IPv6 stateless address autoconfiguration [385].

Routing and forwarding in 6LoWPAN can happen both below the IP layer (L2 forwarding "Mesh-Under") and on the IP layer (L3 routing "Route-Over"). 6LoWPAN has a Mesh Address Header which supports routing of packets in a mesh network, see Figure 2.22, but leaves the details of routing to the link layer. In such cases, Full Function Devices (FFDs) run an ad hoc or mesh routing protocol to populate their routing tables. Two devices do not require direct reachability in order to communicate. The sender is known as the "Originator", and the receiver is known as the "Final Destination".

Multicast is not supported natively in IEEE 802.15.4, but by the IPv6. IEEE 802.15.4 supports broadcast. IPv6 level multicast packets must be carried as link-layer broadcast frames in IEEE 802.15.4 networks. This must be done because the broadcast frames are only considered by devices within the specific PAN of the link. 6LowPAN provides Broadcast with $LOWPAN_BC0$ dispatch in the Broadcast Header.

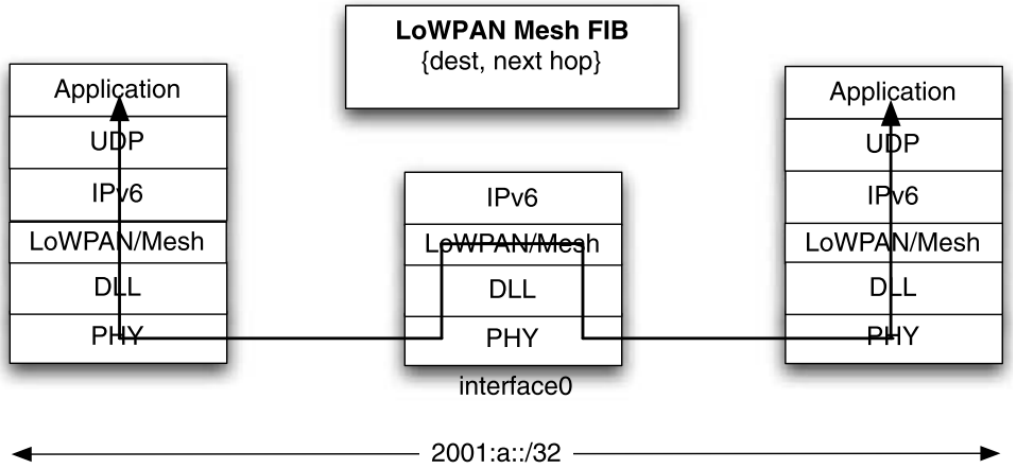


Fig. 2.22: LoWPAN adaptation layer mesh forwarding [357]

Further routing capabilities are not in scope of RFC 4944 [237]. RPL routing has established itself in the implementations of a border router.

RPL Routing

For routing in 6LoWPAN, the roll Working Group specified several RFCs[12]. The RFC 6550 [415] is the central one for IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL). RPL was developed for Low-Power and Lossy Networks (LLNs). This protocol supports "multipoint-to-point traffic from devices inside the LLN towards a central control point as well as point-to-multipoint traffic from the central control point to the devices inside the LLN are supported". Support for point-to-point traffic is available, too.

First, some terms will be introduced. Directed Acyclic Graph (DAG) is a directed graph shows that all edges are oriented in such a way that no cycles exist. All edges are contained in paths oriented toward and terminating at one or more root nodes. Destination-Oriented DAG (DODAG) have a single DAG root (the DODAG root) as a destination. A DAG root has no outgoing edges. The DODAG root may act as a border router for the DODAG. An example of a DAG and a DODAG graph is shown in Figure 2.23.

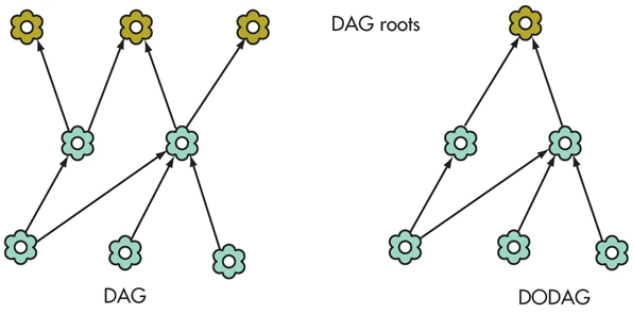


Fig. 2.23: RPL DODAG and DAG Graphs [377]

[12] https://datatracker.ietf.org/wg/roll/documents/

The distributed algorithm of the DODAG construction in RPL is organized as follows [415]: "Some nodes are configured to be DODAG roots with associated DODAG configurations. Nodes advertise their presence, affiliation with a DODAG, routing cost, and related metrics by sending link-local multicast DODAG Information Object (DIO) messages to all-RPL-nodes. Nodes listen for DIOs and use their information to join a new DODAG (thus, selecting DODAG parents), or to maintain an existing DODAG, according to the specified Objective Function and Rank of their neighbors. Nodes provision routing table entries, for the destinations specified by the DIO message, via their DODAG parents in the DODAG Version. Nodes that decide to join a DODAG can provision one or more DODAG parents as the next hop for the default route and a number of other external routes for the associated instance."

According to the RFC 6550, a node's *Rank* defines the node's individual position relative to other nodes with respect to a DODAG root. *Rank* strictly increases in the Down direction and strictly decreases in the Up direction. The exact way *Rank* is computed depends on the DAG's Objective Function (OF). An OF defines how routing metrics, optimization objectives, and related functions are used to compute *Rank*. Furthermore, the OF dictates how parents in the DODAG are selected and, thus, the DODAG formation. An Objective Code Point (OCP) is an identifier that indicates which Objective Function the DODAG uses.

The RPL Control Message consists of an ICMPv6 header followed by a message body. The RPL header consists of type, code, checksum, base and options. The RPL control message is an ICMPv6 information message with a Type of 155. There are three main codes for the following RPL control message types are defined [377]:

- DODAG information solicitation (DIS): The DIS solicits a DODAG information object (DIO) from an RPL node.
- DODAG information object (DIO): The DIO carries information that allows a node to discover a RPL Instance, learn its configuration parameters, select a DODAG parent set, and maintain the DODAG.
- Destination advertisement object (DAO): The DAO is used to propagate destination information upward along the DODAG.

To construct the DODAG topology, nodes may use a DIS message to request a DIO, or they may periodically send link-local multicast DIO messages. Nodes then listen for DIOs and use their information to join a new DODAG or to maintain an existing DODAG. Based on information in the DIOs, the node chooses parents that minimize the path cost to the DODAG root.

Routing in 6LoWPAN networks was surveyed [200] and evaluated [392]. Depending on the routing algorithm, there are differences in metrics like energy consumption, memory uses, and scalability. However, it was shown in [392] that RPL routing provided optimal path choice and can handle trade-offs and scalability.

2.3.2 WLAN

Many IoT devices and developer boards use WLAN or WiFi because this network standard is successfully used in mobile communication and the WLAN networks already exist. The following WLAN standards are used on the IoT devices: IEEE as 802.11b/g/n and the 2,4-GHz frequency range. The ESP32 modules can achieve rates up to 150 Mbps in 802.11n.

In a simple case, WLAN builds a star topology with an access point (AP) which is constantly listening and the IoT devices attempt to transmit data at any time. Every AP is identified by a Service Set Identifier (SSID). The MAC protocol of IEEE 802.11 is based on CSMA/CA. WLAN access can be secured by user/password or more advanced mechanisms WEP, WPA, WPA2 encryptions. However, not all microcontroller boards support these encryptions.

Long time, WLAN was not appropriated for WSN because of high power consumption but recent hardware development enables to produce low power WLAN network controller chips, for example ATWINC1500[13] on the Arduino MKR1000 WIFI[14].

2.4 Network Management

Besides the FCAPS definition for network management, another acronym *Operations, Administration, and Maintenance (OAM)* can be found in the network management. However, this term is not very clearly defined. Therefore, IETF publishes guidelines for their use of the "OAM" acronym [10] and defines OAM(&P) as follows:

- Operation activities are undertaken to keep the network and the provided services up and running.
- Administration activities involve keeping track of resources in the network and how they are used.
- Maintenance activities are focused on facilitating repairs and upgrades – for example, when equipment must be replaced, when a router needs a patch for an operating system image, or when a new switch is added to a network.
- Provisioning activities involve configuring resources in the network to support the offered services.

Summarizing, OAM is a general term that refers to a toolset that can be used for fault detection and monitoring, as well as network configuration and maintenance, which are some aspects of the FCAPS management.

Parallel to OSI Management [169] standards, the IETF worked on the Simple Networking Management Protocol (SNMP) [57, 60] protocol for use in the TCP/IP environment. The SNMP protocol found a broad community [202, p.817], [333]. For years, the network management was focused on the SNMP protocol. The managed devices are typically network components like cable modems, routers, switches, servers, workstations, printers, etc.

However, the IETF network management community recognized the need for automated network management[15] because of the number and diversity of devices. IETF worked on the requirements for network and configuration management of IP-based networks [324, 331, 333]. Based on these requirements in year 2011, the Network Configuration Protocol (NETCONF) in RFC 6241 [93] was specified. The YANG data modeling language is used to model configuration and state data.

The requirements from the RFC 3535 [331] were summarized by Schönwälder [333, 335]. A configuration management protocol:

[13] `https://www.microchip.com/wwwproducts/en/ATwinc1500`
[14] `https://store.arduino.cc/arduino-mkr1000-wifi`
[15] `https://www.ietf.org/topics/netmgmt/`

- R1 - must be able to distinguish between configuration state and operational state.
- R2 - must provide primitives to prevent errors due to concurrent configuration changes.
- R3 - must provide primitives to apply configuration changes to a set of network elements in a robust and transaction-oriented way.
- R4 - must be able to distinguish between several configurations and devices should be able to hold multiple configurations.
- R5 - must distinguish between the distribution of configurations and the activation of a certain configuration.
- R6 - must be clear about the persistence of configuration changes.
- R7 - must be able to report configuration change events.
- R8 - must support a full configuration dump and a full configuration restore operations.
- R9 - must represent configuration state and operational state in a form enabling the use of existing tools for comparison, conversion, and versioning.

Tab. 2.6: Requirements for network configuration protocol according to RFC 3535 [335]

No	Description	SNMP	NETCONF
R1	config vs. oper state	-	+
R2	concurrency support	o	+
R3	config transactions	-	[+]
R4	multiple configs	-	[+]
R5	distribution vs. activation	-	[+]
R6	persistence of config state	o	+
R7	config change notifications	-	+
R8	config dump and restore	-	+
R9	support of standard tools	-	+

These requirements were evaluated with SNMP and NETCONF protocols and the results are outlined in Table 2.6. Considering these results, NETCONF protocol fulfill the requirements for network configuration protocol where SNMP only partly does. For many years the SNMP and CLI scripting was used to configure network devices. But these technologies also have disadvantages since network configuration maybe a complex task. The CLI has the lack of transaction management and lack of structured error management. The changing structure and syntax of CLI commands makes CLI scripts fragile and costly to maintain. The main differences between SNMP and NETCONF are listed in Table 2.7.

A managed device has three types of data: configuration, operational state and statistics. The data is described by SMIvy in SNMP and YANG in NETCONF. However, there is no separation of configuration and operational state in SNMP. The comparison between these two languages is shown in Table 2.8 and Figure 2.24. To show the functionality of the SNMP, a brief review of the protocol is outlined in the following.

Tab. 2.7: Comparison between SNMP and NETCONF protocols [33]

Layer	SNMP/SMIv2	NETCONF/YANG
Content: Device Data, Notification Data	VARBIND lists (OBJECT-TYPE, NOTIFICATION-TYPE)	XML subtrees (data-def-stmt, notification-stmt)
Operations	RFC-defined (Set, Get, GetNext, GetBulk)	YANG-defined (rpc-stmt, (edit-config, copy-config))
Messages: Device Data, Notification Data	PDU: Set, Get, Get-Next, GetBulk PDUs, Trap PDU, Inform PDU, Report PDU	RPC: <rpc>, <rpc-reply>, <notification>
Transport	Message-based: UDP	Session-based: SSH/TLS over TCP

Tab. 2.8: Comparison between SMIv2 and YANG modules, see [33]

Layer	SMIv2	YANG
Hierarchy	depth = 1	depth = n
Additional building blocks	-	choice and case, leaf and container
Data Naming	OBJECT IDENTIFIER Naming (OID)	XPath Absolute Path Naming (URI)
Data Augment	each AUGMENT in own OID subtree	all augmenting data within the augmented subtree
Data Translation	SMIv2 tables as flat lists or combined as nested lists in YANG	standard read-only translation of SMIv2 data with NETCONF

2.4.1 SNMP

The Simple Networking Management Protocol (SNMP) exists meanwhile in version 3 as SNMPv3, RFC 3410 [56] to 3418 [281]) which is more secure than the others. The protocol was developed in 2002 in a modular way consisting of the Management Information Base (MIB), the data definition language Structure of Management Information (SMI) and the SNMP. These three modules are protocol independent because originally SNMP was developed as a temporary framework which should use MIB developed by the ISO and ease the migration [202, p.819] which never took place. Security and management capabilities came with the Version SNMPv3.

SNMP is an application protocol which is based on UDP/IP. SNMP uses MIB modules which describe a set of managed objects and mirror the current state. The data in the MIB modules is defined with the SMI/SMIv2 [227] language. It has a hierarchical structure but with a particular syntax. A device (switch or router) has an *SNMP agent* running. This agent gathers information about the device and stores it in a MIB database. The MIB is "a hierarchical, pre-defined structure that stores information that can be queried or set". The Network Management Station (NMS) has an *SNMP manager* running and is the monitoring software which communicates with

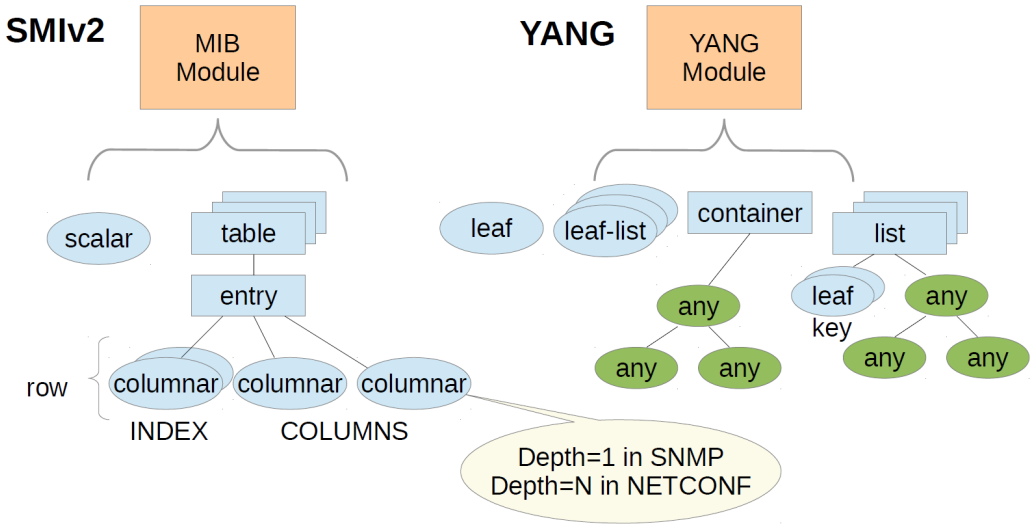


Fig. 2.24: Basic Data Building Blocks in SMIv2 and YANG [33]

the agents via request/response (e.g. Get, Set and Response message types) and asynchronous notification (so-called Traps and Inform messages) paradigms.

In the version SNMPv3, security was enhanced. A user's authentication can be set as one of these models [56]

- NoAuthNoPriv: Users connecting with this level have no authentication in place and no privacy of the messages they send and receive.
- AuthNoPriv: Connections using this model must authenticate, but messages are sent without any encryption.
- AuthPriv: Authentication is required and messages are encrypted.

An access control mechanism was implemented to provide granular control over MIB data. Version 3 has the ability to leverage the security provided by the transport protocols, such as SSH or TLS.

SNMP was developed to be used for both: monitoring and management. However, SNMP is mostly used for fault and performance monitoring and not for configuration changes. The main reason lies in the complexity of SNMP. For example, the lack of a standard automatic discovery process that finds the (correct) MIB modules that the device is using. Thus, the discovery work must be done by the users.

2.4.2 NETCONF

The current version of Network Configuration Protocol (NETCONF) is specified by RFC 6241 [93] in 2011. NETCONF provides "mechanisms to install, manipulate, and delete the configuration of network devices". The YANG data modeling language, used by NETCONF, specifies data models and protocol operations. The NETCONF protocol operations are realized as remote procedure calls (RPCs). The XML-based data encoding is used for protocol messages and for the configuration data. The *configuration data* is the set of writable data which is required to transform a system from its initial default state into its current state. The *state data* is the additional data on a system which is not configuration data like read-only status information and

collected statistics. Usually, a NETCONF server is installed on the network devices such as routers or switches which have many resources. An amendment of NETCONF is RESTCONF [32] which offers a REST-based interface with YANG model in JSON format.

NETCONF conceptually consists of four layers as shown in Figure 2.25. Layer 1 provides a secure transport between server and client. NETCONF can be layered over any transport protocol that provides the required set of functionality like support for RPC, sessions, authentication. For secure NETCONF transport the following is specified, among others: NETCONF over SSH (TCP-based) [411], Transport Layer Security (TLS) [20], TLS with Mutual X.509 Authentication [21]. Layer 2 provides messaging mechanism for RPCs and notifications. Notification is a "server-initiated message indicating that a certain event has been recognized by the server" (e.g. if a client needs to be aware of changes in NETCONF server). Layer 3 defines a set of base protocol operations called as RPC methods with XML-encoded parameters. Layer 4 is the content layer for data models. The YANG data modeling language [38] specifies NETCONF data models and protocol operations for the operations and the content layers. Moreover, the RFC 6536 [31] specifies the NETCONF Access Control Model for restricted protocol access for particular users to a pre-configured subset of operations and content.

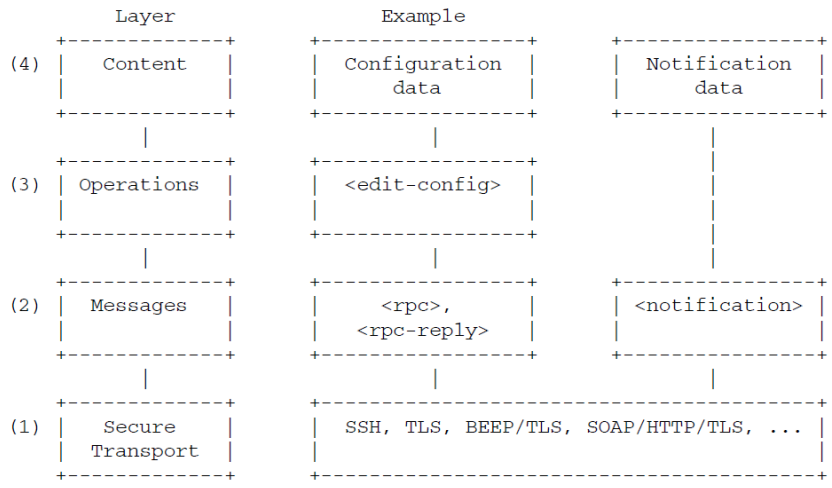


Fig. 2.25: NETCONF Protocol Layers [93]

The NETCONF protocol provides a set of low-level operations to manage device configurations in a datastore and request device state information [93]:

- <get> operation retrieves configuration and state data.
- <get-config> operation retrieves configuration data only.
- <edit-config> operation allows to change a configuration or part of it.
- <copy-config> operation replaces an entire configuration.
- <delete-config> operation Deletes a configuration.
- <lock> operation allows the client to lock the entire configuration of a device, e.g. to avoid changes for a short time.
- <unlock> operation is used to release a configuration lock.

- <close-session> is a request for termination of a NETCONF session.
- <kill-session> forces the termination of a NETCONF session.

One of the advantages of NETCONF is that the management protocol can closely mirror the native functionality of the device. This allows timely access to new features. Such device functionality can be defined by so called *capabilities*. *Capabilities* are additional operations besides the base operations. They are advertised by the NETCONF server during session establishment, and a client discovers the set of protocol extensions supported by a server. These capabilities allow the client to adjust its behavior to take advantage of the features exposed by the device.

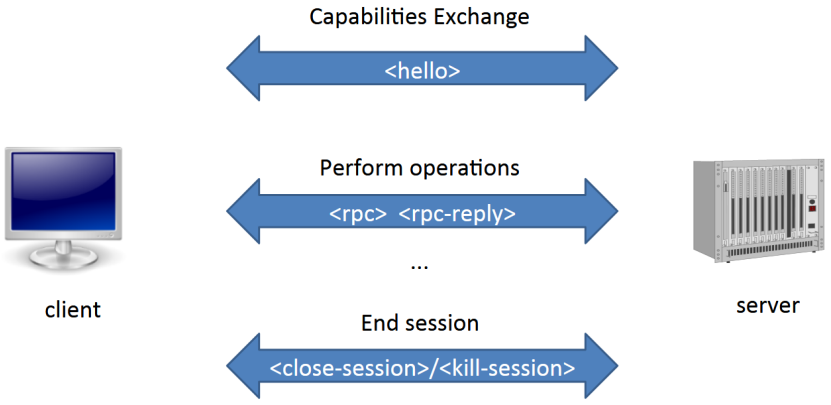


Fig. 2.26: Basic NETCONF session [190]

A NETCONF capability is identified with a URI. The base capabilities are defined using URNs [229]. Capabilities in NETCONF have the following format where name is the name of the capability.: urn:ietf:params:netconf:capability:name:1.x

A basic NETCONF session is depicted in Figure 2.26. First, the capabilities exchange between server and client takes place. Each peer, both client and server, must send a <hello> message with the set of its capabilities, see Listings 2.4 and 2.5. For example, two base capabilities: the writable-running capability means the support for <edit-config> and <copy-config> operations; the validation capability means that the device supports the <validate> protocol operation and checks at least for syntax errors. Customized capabilities are defined by an URI.

Listing 2.4: NETCONF hello message from client [190]

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.1">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
```

After the capabilities are exchanged, the client can perform RPC operations on the NETCONF server. An RPC call involves all NETCONF protocol layers as shown in the example in Listing 2.6 [190]: the configuration data layer in the <config> tags, operations layer in the <edit-config>, the message layer in the <rpc> tags, the transport layer e.g. over SSH. The NETCONF server responds with an <rpc-reply>

on the operation with <ok> or <rpc-error> element. Finally, the client closes the session graceful with <close-session> or force termination with <kill-session>.

Listing 2.5: NETCONF hello message from server [190]

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.1">
  <capabilities>
        <capability>urn:ietf:params:netconf:base:1.1</capability>

        <capability>urn:ietf:params:netconf:capability:writable-
            running:1.0</capability
        <capability>urn:ietf:params:netconf:capability:validate:1
            .1</capability>
        <capability>http://example.com/syslog</capability>
                ...
  </capabilities>
<session-id>3</session-id>
</hello>
```

An XML Schema for NETCONF messages layer is used and thus the messages can be validated. Additionally, operations like <get> and <get-config> support subtree filters to respond only parts of the configuration and operation data. The XPath expressions can be used in the filter element.

Data modeling and content layer are out of scope of the NETCONF protocol. It is assumed that the device's data model is known to the application and it is responsible for the management of these data. The configuration data is carried inside of the <config> element and is specific to the device's data model. The content of that element is agnostic to the protocol. The device uses capabilities to disclose the set of data models that the device implements. The capability definition contains the operation and constraints imposed by data model. Devices and managers can support multiple data models, including both standard and proprietary data models.

Listing 2.6: NETCONF RPC construction

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
        <config>
                ... Content ...
        </config>
  </edit-config>
</rpc>
```

Data models written in YANG usually do not need to define protocol capabilities because the usage of YANG automatically leads to a capability announcing the data model. Thus, the data model-driven management is enforced by using YANG in NETCONF. An architecture for network management in [352] describes how NETCONF and YANG help to build network management applications that meet the needs of network operators.

2.4.3 YANG

The Yet Another Next Generation (YANG) data model language is developed by IETF working group Network Modeling (netmod). YANG is compatible with Structure of Management Information SMIv2 language. The SMIv2-based MIB modules used by the SNMP protocol can be automatically translated into YANG modules for read-only access [332]. However, YANG does not consider reverse translation from YANG to SMIv2.

The RFC 3444 [280] explains the difference between Information Models and Data Models. Information model is an conceptual/abstract model for designers and operators, e.g. UML. Data model is a concrete/detailed model for implementors, e.g. YANG data models and Management Information Base (MIB) modules.

YANG is specified in 2010 by RFC 6020 [38]. YANG was proposed to be used for other protocols and purposes (e.g., RESTCONF [32], a draft for the CoAP Management Interface (CORECONF) [403], Manufacturer Usage Description (MUD) [209]). In the CORECONF, CoAP is used to access datastore and data node resources specified in YANG, or SMIv2 converted to YANG. CORECONF uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction.

MUD should provide a means for end devices to signalize to the network what sort of access and network functionality they need for proper functioning. The initial focus of MUD is access control. The MUD description will be provided by an URL sent by a device.

Further encodings besides XML have been proposed (e.g., JSON [211]). The YANG 1.1 version is specified by RFC 7950 [37] in 2016 and tries to stay compatible with the previous version. With the RFC 8199 [39] the YANG Module Classification was introduced for consistent terminology to help with the categorization of YANG modules.

Listing 2.7: YANG Example for RPC [38]

```
rpc activate-software-image {
 input {
  leaf image-name { type string; }
 }
 output {
  leaf status { type string; }
 }
}
```

YANG is a model language used for configuration and state data manipulated by the NETCONF protocol, NETCONF RPCs, and NETCONF notifications. This allows a complete description of all data sent between a NETCONF client and a server. The data is organized in a hierarchical way as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides descriptions of the nodes and the interaction between those nodes. YANG structures data models into modules and submodules. Thus, modules can import modules or data nodes from another modules. Further definitions in YANG are: a set of built-in types (similar to data types in programming languages), constraints on the data, groups of nodes,

lists, conditional statements. YANG is an extensible language and allows to define augment statements by standardization organizations, vendors, and individuals.

Listing 2.8: NETCONF Example for RPC [38]

```
<rpc message-id="101" xmlns="
    urn:ietf:params:xml:ns:netconf:base:1.0">
        <activate-software-image xmlns="http://acme.example.com/
            system">
                <image-name>acmefw-2.3</image-name>
        </activate-software-image>
</rpc>

<rpc-reply message-id="101" xmlns="
    urn:ietf:params:xml:ns:netconf:base:1.0">
        <status xmlns="http://acme.example.com/system">
                The image acmefw-2.3 is being installed.
        </status>
</rpc-reply>
```

YANG modules can be translated into so called YANG Independent Notation (YIN) which is an equivalent XML syntax. YIN allows applications to use XML parsers and Extensible Stylesheet Language Transformations (XSLT) scripts to operate on the models. The conversion from YANG to YIN is lossless, thus, content in YIN can be converted back into YANG.

YANG tries to balance between high-level data modeling and low-level encoding. The reader of a YANG module can see the high-level view of the data model while understanding how the data will be encoded in NETCONF operations. The following example illustrates this principle. The YANG snippet in Listing 2.7 describes the RPC call. When this RPC call is performed by a NETCONF client, the XML-based RPC message will be sent to the NETCONF server and replied as shown in Listing 2.8.

Listing 2.9: YANG Example for notification [38]

```
notification link-failure {
        description "A link failure has been detected";
        leaf if-name {
                type leafref { path "/interface/name"; }
        }
        leaf if-admin-status { type admin-status; }
        leaf if-oper-status { type oper-status; }
}
```

YANG allows to define notifications for the NETCONF protocol. The notification statement can be used to define the contents of event notifications. Examples in Listings 2.9 and 2.10 show a notification in YANG and NETCONF XML. The RFC 8632 [400] specifies a YANG Data Model for Alarm Management which is even more crucial than the notification.

IETF encourages working groups to use YANG data models for configuration and the number of the YANG modules grows. For standardization reasons several further

Listing 2.10: NETCONF Example for notification [38]

```
<notification xmlns="
    urn:ietf:params:netconf:capability:notification:1.0">
        <eventTime>2007-09-01T10:00:00Z</eventTime>
        <link-failure xmlns="http://acme.example.com/system">
                <if-name>so-1/2/3.0</if-name>
                <if-admin-status>up</if-admin-status>
                <if-oper-status>down</if-oper-status>
        </link-failure>
</notification>
```

RFCs were defined: common YANG data types [330], guidelines for authors of YANG models [29], YANG module classification [39].

2.5 Semantic Web and Ontology

Thinking about semantics leads to the W3C Semantic Web standards (initiated by Tim Berners-Lee) which is a branch of artificial intelligence (AI) focusing on knowledge representation and reasoning, creating new knowledge from known facts. Semantic Web tackles machine-readable and human-understandable knowledge representation and logic.

Ontology plays a central role in the Semantic Web. The shortest definition of ontology can be found as "ontology is a formalization of a conceptualization". A definition from W3C documents says:

Definition 6 (Ontology).
An ontology formally defines a common set of terms that are used to describe and represent a domain . . . An ontology defines the terms used to describe and represent an area of knowledge. [142]

Several terms must be explained from this definition. Ontology is domain-specific and is used to describe and represent an area of knowledge. A domain is a specific subject area, for example building automation, agriculture, metering, etc. Ontology contains terms and the relationships among these terms. Terms are called *concepts* or *classes*. The relationships between these classes can be expressed by using a hierarchical structure: super-classes represent higher-level concepts, and subclasses represent finer concepts. Another level of relationship can be expressed by using a special group of terms: *properties*. These property terms describe various features and attributes of the concepts, and they can also be used to associate different classes together. Therefore, the relationships among classes are not only super-class or subclass relationships, but also relationships expressed in the term of properties.

By having the terms and the relationships among these terms clearly defined, ontology encodes the knowledge of the domain in such a way that the knowledge can be understood by a computer. This is the basic idea of ontology [421].

An article from 1999 [16] describes what ontologies are (vocabularies) and why we need them: "Ontological analysis clarifies the structure of knowledge". An ontology

is a representation vocabulary which describes a certain domain of interest (e.g. metering, medicine, sensors). The advantages of ontology can be summarized as follows [421]:

- It provides a common and shared understanding/definition about certain key concepts in the domain.
- It offers the terms one can use when creating an ontology in the domain.
- It provides a way to reuse domain knowledge.
- It makes the domain assumptions explicit.
- Together with ontology description languages (such as RDFS and OWL), it provides a way to encode knowledge and semantics such that machines can understand.
- It makes automatic large-scale machine processing possible.

Therefore, an ontology describes semantics of data and helps to understand the domain of interest, can be processed in a structured way, and new facts can be inferred.

2.5.1 RDF and RDFS

One form of knowledge representation is a graph which defines a model based on triples (subject-predicate-object). These triples describe a meaning of data, also called *semantics*. A predicate logic can be applied to an RDF graph. The ontology defines concepts and their relationships. *Concept* is a shared understanding of a term. Two concepts are connected by a directed relationship. Thus, they build so-called *triples*:
subject concept $\rightarrow$ ***relationship*** $\rightarrow$ ***object concept***
Concepts identify classes of individuals. An example for *classes* within the IoT is:
Device $\rightarrow$ ***hasFunctionality*** $\rightarrow$ ***MeasuringFunctionality***

These triples are formalized by RDF. The extension of the RDF is the RDF Schema (RDFS). The Web Ontology Language (OWL) (based on RDFS) was defined as "an ontology language for the Semantic Web with formally defined meaning", [269]. Such knowledge base can be queried by SPARQL Protocol And RDF Query Language (SPARQL) standard. Data linked in the Web with help on Semantic Web Standards is called Linked Data (LD)[16]. Semantic Web standards found broad use by software developers [421, 85].

Resource Description Framework (RDF) [235, 288] was originally applied to web resources, however, was revised in 2004 and since then applied as generic representation of semantic information [147]. In RDF, each resource (i. e. individual, class, property) is identified by a unique Uniform Resource Identifier (URI) or Internationalized Resource Identifier (IRI). In our example, the annotation is `http://www.onem2m.org/ontology/Base_Ontology/base_ontology#Device`. Namespaces can be shortened with a prefix as applied in XML. RDF described a directed graph: a set of nodes which are connected by directed arcs.

There are several syntax representations (serializations) defined for RDF, which should improve, simplify or extend the predecessor, among them:

[16] `http://www.w3.org/DesignIssues/LinkedData.html`

- RDF/XML (default), 2004 [235], 2014(1.1) [288]
- Turtle, 2004, 2014(1.1) [54]
- JSON-LD (JSON syntax for RDF graphs), 2014(1.0) [206]

RDF triples have a formal meaning which determines, with mathematical precision, the conclusions (or *entailments*) that machines can draw from a given RDF graph. Positive and negative entailments as well as a datatype-aware entailment are possible.

RDF Schema (RDFS) v1.1 [126] was developed for lightweight ontologies. Formal semantics were introduced in the latest version (distinction between assertional knowledge (RDF) and terminological knowledge (RDFS)). A vocabulary created by RDFS can be distributed and the terms from this vocabulary can be reused. RDFS introduced classes such as rdfs:Resource, rdfs:Class, rdfs:Literal, rdfs:Datatype, and properties such as rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:label. The RDF graph can be extended with constructs like these: ***MeasuringFunctionality*** $\rightarrow$ ***subClassOf*** $\rightarrow$ ***Functionality***
hasFunctionality $\rightarrow$ ***domain*** $\rightarrow$ ***Device***
hasFunctionality $\rightarrow$ ***range*** $\rightarrow$ ***Functionality***

The knowledge in a RDFS vocabulary is encoded in the meaning of a term which is expressed and understood by defining the following: "all the properties that can be used on it, and the types of those objects that can be used as the values of these properties" [421]. In this way, machine inferencing based on RDFS can be proceeded.

2.5.2 OWL and Logic

However, a more advanced language is required to proceed ontology based reasoning. This language should be able to do the following [421]:

- express relationships among classes defined in different documents across the Web;
- construct new classes by unions, intersections and complements of other existing classes;
- add constraints on the number and type for properties of classes;
- determine if all members of a class will have a particular property, or if only some of them might;

This new language is the Web Ontology Language (OWL). OWL provides us with the capability to express much more complex and richer relationships. OWL is built upon RDF Schema and contains new constructs for better expressiveness. All the terms contained in RDFS vocabulary can still be used when creating OWL documents. OWL ontologies support *reasoning/inference*.

The W3C defined OWL [269] as a standard for ontology description. The OWL standard provides not only a standard for structuring vocabulary, but also has some main advantages:

- information is represented in a formal, machine-readable way;

- a reasoner can check consistency and infer new information;
- two concepts in different ontologies and similar meaning can be mapped (i.e. using the property `owl:sameAs`)

OWL 2 is an extension and revision of the OWL 1. OWL introduced further constructs: owl:Class, owl:Thing (generic class), owl:Nothing(empty class), owl:ObjectProperty, owl:DatatypeProperty, owl:sameAs, owl:DisjointWith, owl:AllDifferent, owl:equivalentClass, owl:Restriction, owl:maxCardinality, owl:inverseOf etc.

For example, *Datatype Properties* are used, when the object concept is data (typed or untyped), and *Object Properties*, when individuals are connected
OperationState → ***hasValue*** → ***rdfs:Literal***
hasValue → ***rdf:type*** → ***owl:DatatypeProperty***
hasFunctionality → ***rdf:type*** → ***owl:ObjectProperty***
myDevice → ***hasFunctionality*** → ***getTemperature***

OWL2 can be represented in different syntax variants, among them:

- RDF/XML-Syntax: extension of existing OWL/RDF
- OWL/XML-Syntax: independent XML serialization
- Turtle: concise and easy readable
- JSON-LD [206] format

Different syntax representation can influence the file size of an ontology.

Logic

The strength of the Semantic Web standard is the logic formalism. A knowledge base is consisting of data sets. These data sets are assumed as true and build a *model* otherwise the knowledge base is inconsistent or contradictory. If one data set can be logically concluded from other data sets, then we speak of *entailment (or logical consequence)*. The process of finding entailment in the knowledge base is called *inference*. This process should be complete (decidable and of polynomial complexity) otherwise the inference will never come to a result. Simple entailment in RDF is NP-complete [380]. Formal semantics in RDFS make use of entailment with deductive rules. However, RDFS has limits modeling restrictions (e.g. negations are not possible).

There are two semantics in OWL2 [269] the *Direct Semantics* and the *RDF-Based Semantics*. They provide two alternative ways of assigning meaning to OWL 2 ontologies and a correspondence theorem providing a mapping between them. These two semantics are used by reasoners and other tools. OWL 2 FULL adopts the *RDF-Based Semantics* whereas OWL 2 Descriptive Logic (DL) takes the *Direct Semantics*. OWL 2 DL provides three profiles (syntactic subsets): OWL 2 EL, OWL 2 QL, and OWL 2 RL (see Figure 2.27). The OWL DL is decidable [361]. OWL DL is also well supported by common developer tools [147].

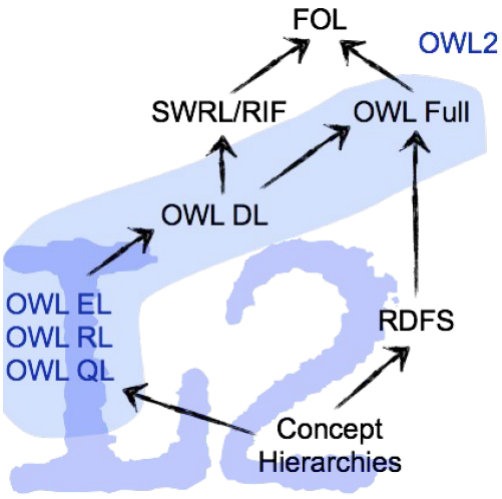


Fig. 2.27: OWL2 Concept hierarchies [310]

The FOL has *declarative and compositional semantics*, and is *context-independent and unambiguous*. It is based on a formal model which defines ontological bindings. These bindings are facts, objects and relationships. Some of these relationships are functions. Further, FOL can be extended by time defining a temporal logic. The *domain* of a model is the set of objects or *domain elements*. There is a vast amount of literature about FOL and it's use in Semantic Web (e.g. [305], [399], [215], [148], [79]), therefore no further details are provided here.

DLs are Descriptive Logics which are classified as fragments of FOL which allows highly expressive constructors if they are limited on certain complexity classes. DLs were developed for semantic networks. DLs are related to modal logics. DLs build a compromise between expressivity and scalability. Usually, DLs are decidable and there are efficient algorithms to make inferences. But there are also DLs which have strong expressivity with polynomial complexity. The Attributive Language with Complement (ALC) is the basis of descriptive logic. ALC includes classes, properties(also called roles) and individuals which can build relationships. OWL DL corresponds to descriptive logic SHOIN(D) which stands for:

- S stands for ALC plus transitive roles/properties.
- H stand for role hierarchies (subproperties).
- O stand for nominals (complete classes with owl:oneOf or owl:unionOf).
- I stand for inverse roles.
- N stand for (unqualified) number restrictions.
- D stand for data types.

The change from SHOIN(D) to SHOIQ(D) is negligible. OWL 2 corresponds to descriptive logic SROIQ(D) [155] where R stands for role constructors. Description logics [304] differs between data complexity and overall complexity. Data complexity means the complexity measured with respect to the total size of the assertions in the ontology. For ALC and SHIQ(D) is data complexity for checking of concept satisfiability NP-complete, for checking of assertions co-NP-complete [147]. The overall complexity is the complexity measured with respect to the total size of the

axioms and the assertions in the ontology. ALC and SHIQ are ExpTime-complete, SHOIN(D) – also OWL DL – is NExpTime-complete [147]. These results are according to OWL 1. Further later references show other results considering complexity, see Computational Properties in OWL 2 [270] and the website [423].

2.5.3 RDF HDT Compression

RDF Header-Dictionary-Triples is a W3C Member Submission [101] from 2011. HDT was developed to share big semantic datasets in RDF format on the web. The data are compressed to save space, but enable search and browse operations without decompression. HDT addresses high levels of verbosity and redundancy in RDF files and machine-understandability.

Technically, as its name implies, HDT splits RDF data in three logical components: Header, Dictionary and Triples. The Header holds metadata about the file using a plain RDF structure. The Dictionary is a catalog of triples for the used terms such as IRIs in an ontology. The dictionary encoding is based on the Front-Coding [416] which is again based on words with similar prefix. Subjects, Predicates and Objects are subsumed in sections to avoid repeating. Each section is sorted lexicographically and then IDs are assigned to each term. RDF Triples are now tuples of three IDs. HDT proposes an encoding of Triples called Bitmap Triples (BT). SPO trees are sorted in a specific order like many trees in a forest and a binary encoding is applied layer by layer, as shown in Figure 2.28.

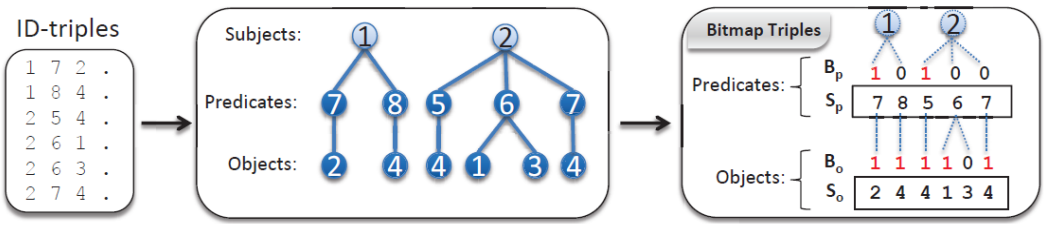


Fig. 2.28: Encoding of Bitmap Triples [224] in HDT

2.5.4 SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) is a query language for RDF in the semantic web. The SPARQL queries have some similarities with SQL queries with its *SELECT ... WHERE...* notation. Queries include RDF graph-patterns as RDF triples which can be combined and filtered. Unknown variables are marked with question mark like *?var*. Besides the keywords SELECT and WHERE, there are further keywords defined: OPTIONAL, FILTER, UNION, REGEX, ORDER BY. An Example in Figure 2.11 shows a SPARQL SELECT query for device and its controlling functionalities in an ontology.

The results of a query are displayed as a table with columns of variables in the SELECTS command. This is useful for users but not very useful for developers. Therefore the output format can be defined, i.e. as JSON-LD, or new subgraphs can be created with the keywords CONSTRUCT and DESCRIBE. The keyword ASK is only used to check if something is defined in the graph. SPARQL can be easily applied to RDF but not that easy to OWL because OWL has more expressivity [147, p. 233ff].

Therefore SPARQL was extended with SPARQL 1.1 Entailment Regimes [116] for OWL 2 DL ontologies.

Listing 2.11: A SPARQL Query Example

```
PREFIX onem2m: <http://www.onem2m.org/ontology/Base_Ontology/base_ontology#>

SELECT ?device ?functionality
WHERE {
  ?device rdf:type onem2m:Device.
  ?device onem2m:hasFunctionality ?functionality.
  ?functionality rdf:type onem2m:ControllingFunctionality.
}
```

2.6 Conclusion

For CoAP protocol, devices maintain a server and wait for a call from an application and responds e.g. with a sensor measurement result. In terms on energy, this is not efficient. CoAP recognizes this problem and tries to solve it with observation pattern [378, 140] and a draft for a Publish-Subscribe Broker [194]. For the stable connection, CoAP over TCP was also specified [43].

MQTT was the protocol of choice because the publish/subscribe paradigm is energy-saving and a proven approach for Wireless Sensor Networks (WSN). Moreover, it is suitable for constrained as well as non-constrained devices. Devices on the LISTENING state need almost the same energy as on the SENDING state [186]. MQTT is a TCP based protocol which implies more stable network connections but also requires more overhead than UDP. MQTT 5.0 was specified in 2018 which was optimized for constrained IoT devices and scalability. The MQTT protocol has become the de-facto IoT standard [366]. For this reason, we have chosen the MQTT protocol for our framework.

The NETCONF protocol is the successor of the SNMP protocol and was chosen for network configuration. Finally, the Semantic Web Standards are currently the state-of-the-art for ontology definition and processing.

3 Related Work and State-of-the-Art: Interoperability

> *Every old idea will be proposed again with a different name and a different presentation, regardless of whether it works.*
>
> **— RFC 1925, Nr. 11**
> (The Twelve Networking Truths)

Interoperability in network management of heterogeneous constrained devices in the IoT is still an unresolved question. The challenge is to bring everything (i.e. heterogeneous devices, networking tasks) together on a framework or system architecture. As already mentioned, interoperability goes with standardization. Also, many researchers have investigated this question. Meanwhile, some vendors and alliances developed their solutions.

Surveys [5, 67] consider the layers of interoperability: technical, syntactical, semantic and organizational interoperability. While technical interoperability refers gateways, API and IoT platforms, the semantic interoperability focuses on the use of semantic technologies in the IoT and states that "it is still open issue to design a semantic IoT framework and open data standards to support full interoperability" [67].

Many approaches were proposed to solve the interoperability issue in the IoT. Most of them use a semantic approach. The semantic interoperability will be discussed in Section 5.1 in context of semantic device descriptions.

Holistic approaches for interoperability which are related to this thesis are introduced in the next sections: first, the efforts of open standardization organizations and commercial alliances are presented; then, some open source and vendor solutions are outlined.

3.1 Organizations and Standards

Many Standards Development Organization (SDO)s and alliances try to standardize or specify an interoperable approach in the IoT with the goal to manage devices. There are essential differences between such organizations. SDOs such as Internet Engineering Task Force (IETF) or World Wide Web Consortium (W3C) are open international communities and the standards are free available to everybody. Implementations based on these standards are mostly free available as libraries for developers. This is the main reason for the broad acceptance and dissemination of such standards (e.g. the IP-based Internet and World Wide Web (WWW)).

On the other side, commercial industrial alliances such as Open Mobile Alliance (OMA) or Open Platform Communications (OPC) Foundation (where members pay fees for the membership) represent the interests of vendors. Their specifications

or implementations are often not available to the public. Additionally, there is a lot of movements in these alliances and consortia, so that new ones appear and other disappear or are overtaken by others (e.g. IPSO merges into OMA, Alljoyn and Open Internet Consortium (OIC) build now OCF). Some alliances build partnerships and specify interworking capabilities to support both sides technologies (e.g. oneM2M defines interworking specifications to OMA LwM2M and OCF) and offer interoperability at least between these specifications.

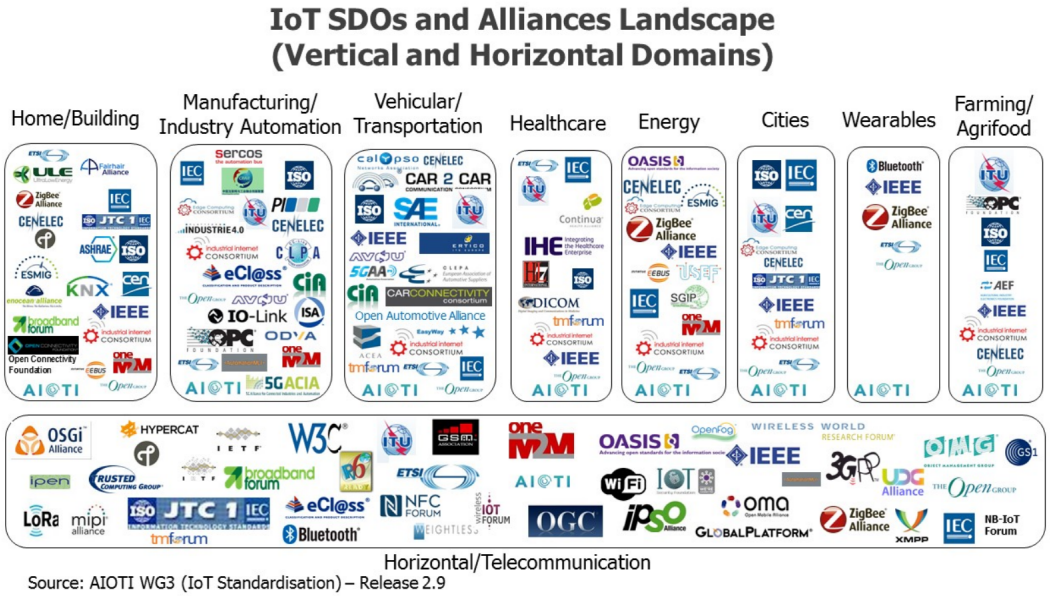


Fig. 3.1: IoT Landscape with vertical and horizontal domains [165]

The Alliance for Internet of Things Innovation (AIOTI) recently made a comprehensive analysis on the ongoing IoT standardization. A vast number of SDOs and alliances are shown in see Figure 3.1. The landscape is sorted into vertical and horizontal domains. Vertical domains serve solutions for a certain application domain like home and building or farming. Horizontal domain presents standards for the common IoT infrastructure independent from any application use case. We give a rough overview of the most active organizations and alliances and their standardization efforts in the IoT without claim of completeness (see Table 3.1 and 3.2). We focus on the horizontal solutions and do not consider vertical domains (e.g. Industrial IoT (IIoT), also called Industry 4.0).

This vast amount of standards (vendor solutions are not considered so far) makes it very difficult to choose one solution for developers. So far only a few standards are obviously leading in the practice (i.e. MQTT, WLAN, Bluetooth). Many specifications are just a draft, on research state or still not established. A detailed view and assessment on some related standards and specifications of this thesis is provided in the next chapters. They are surveyed from the perspective of network management of devices, sometimes also called "device management". The criteria are the used protocols, semantics, discovery methods and security.

3.1.1 IoT-related work in the IETF

Several surveys [359, 167, 189] show the work on IETF standards for the IoT. Working groups for CoAP protocol, 6LoWPAN and CBOR encoding were introduced

Tab. 3.1: Overview of Standardization Organizations in the IoT

Name	Standards
IETF	RFC 7252 (CoAP), RFC 4944 (6LoWPAN), etc.
IEEE	IEEE 802.15.4, IEEE 802.11 (WLAN)
W3C	Web of Things, SSN Ontology, etc.
ETSI	SAREF Ontology etc.
OASIS	MQTT Protocol 3.1.1 and 5.0
OGC	Sensor Web Enablement (SWE), SensorML
ITU-T	ITU-T Y.4111/Y.2076 (Semantics based requirements and framework of the Internet of Things), etc.
ISO/IEC	ISO/IEC 20924:2018 IoT Vocabulary, ISO/IEC 30141:2018 IoT Reference architecture

Tab. 3.2: Overview of Alliances and Consortia in the IoT

Name	Specifications
OMA SpecWorks	LwM2M (CoAP based), IPSO Smart Object Registry
oneM2M	Functional Architecture and amendments
OCF (former OIC)	OCF Core Framework, oneIoTa Model Tool, Alljoyn Framework, UPnP, Fairhair
AIOTI	IoT LSP Standard Framework Concepts, IoT High Level Architecture (HLA), IoT Semantic interoperability recommendations
OPC Foundation	OPC UA, Milo as Eclipse implementation
OMG	DDS Protocol, Cyclone DDS as Eclipse implementation

in Section 2.2.2 and 2.3.1. The Constrained RESTful Environments (core) working group introduced also the Sensor Measurement Lists (SenML) in RFC 8428 [177] as a format for representing simple sensor measurements. The SenML is described in Section 5.3.6 in context of semantic device descriptions. The NETCONF protocol and the YANG language were already introduced in Sections 2.4.2 and 2.4.3.

IETF has further groups dedicated to the IoT application field. The Thing-to-Thing Research Group (t2trg) investigates open research issues in the IoT and specified the RFC 8576 [112] where they analyzed the state of the art and challenges in the IoT security.

The working group Light-Weight Implementation Guidance (lwig) supports developers of the smallest devices. The goal is to build minimal yet interoperable IP-capable devices for the most constrained environments. This group worked on the terminology, RFC 7228 [40] and specified RFC 8352 [119] about energy-efficient features of IoT protocols, and RFC 8387 [346] with practical considerations in securing IoT networks.

The working group Software Updates for Internet of Things (suit) focuses on defining a firmware update solution that will be usable on Class 1 (as defined in RFC 7228) devices, i.e. devices with about 10 KB RAM and 100 KB flash, but also may apply to more capable devices as well. This group will not define any new transport or discovery mechanisms, but may describe how to use existing mechanisms within the architecture. The group defined various requirements and challenges on software

updates that are specific to IoT devices in RFC 8240 [393]. One promising draft is a firmware update architecture [239]. This document lists requirements and describes an architecture for a firmware update mechanism suitable for IoT devices. Another draft [238] describes the information that must be present in the manifest for firmware updates.

The working group Authentication and Authorization for Constrained Environments (ace) aims to produce a standardized solution for authentication and authorization to enable authorized access to resources identified by a URI and hosted on a resource server in constrained environments. As a starting point, the working group assumes that access to resources at a resource server by a client device takes place using CoAP and is protected by DTLS. Both resource server and client may be constrained. This access will be mediated by an authorization server, which is not constrained. The group finalized two RFCs: RFC 8392 [180] with CBOR Web Token (CWT), and RFC 7744 [338] with use cases for authentication and authorization in constrained environments.

IETF work is in the process of figuring out how to help a device without user interface onboard to find the correct network in a secure way. There are some drafts from the Remote ATtestation ProcedureS (RATS) working group which proposes an Entity Attestation Token (EAT) and architecture [36]. However, they are not explicitly specified for the IoT.

There is a draft for MQTT-TLS profile [343] from the Authentication and Authorization for Constrained Environments (ace) working group. This document specifies a profile for the ACE framework to enable authorization in an MQTT-based publish-subscribe messaging system. Proof-of-possession keys, bound to OAuth2.0 access tokens, should be used to authenticate and authorize MQTT Clients. The protocol relies on TLS for confidentiality and server authentication.

The working group Home Networking (homenet) focuses on the evolving networking technology within small "residential home" networks. They specified five RFCs, among them the IPv6 home networking architecture principles in the RFC 7368 [63].

The Operations and Management Area (opsawg) group is working on management of network with constrained devices, e.g. RFCs 7547 and 7548 [94, 95] identify problems, specify use cases and requirements for such management.

Discussion
IETF working groups recognize requirements and challenges in the IoT field. Many of them are formulated as RFCs. A lot of drafts are arising but only partly achieved a RFC state. However, IETF groups focus only on the particular aspects ("islands") of the IoT system architecture (i.e. network and transport protocols, application protocol CoAP and enhancements, security). The big picture is not completed and has some gaps (e.g. considering management of devices and semantics of data). However, these gaps were recognized as some recent activities show, for example, the Manufacturer Usage Descriptions (MUD) [209] and the Semantic Definition Format (SDF) for data and interactions of things [193]. IETF tries to build an ecosystem around CoAP with CoRE Resource Directory (RD), CBOR, YANG and CORECONF [403] for network configuration management. However, the MQTT protocol is often not considered in the RFCs.

3.1.2 W3C Web of Things (WoT)

The term *Web of Things (WoT)* [131] was introduced by Guinard and Trifa [129, 130, 128, 390, 131]. While IoT is located on the layers 1-6 of OSI-Model, the WoT is located on the layer 7 (see Figure 3.2) and concerned about web application protocols (e.g. HTTP) and tools. When data and services in WoT are semantically described then they build a **Semantic Web of Things(SWoT)** [131, p. 239ff].

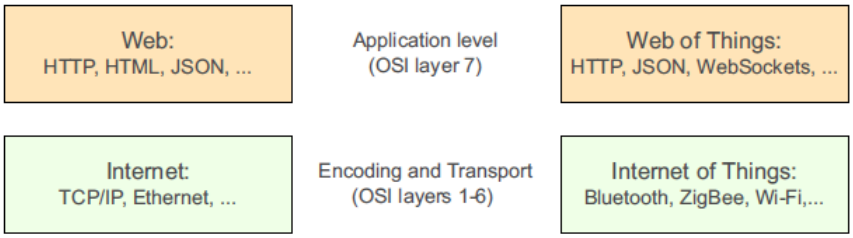


Fig. 3.2: Internet of Things (IoT) vs. Web of Things(WoT) [131, p. 9]

The idea of the WoT is to reuse existing web standards for integration of things/devices into a web platform or application. Guinard proposed to split the WoT application layer into four layers of the WoT architecture (see Figure 3.3): (1) access, (2) find, (3) share and (4) compose. Below them is a network layer. The access layer connects the things to the web using e.g. a RESTful API or HTTP, built on top of the TCP/IP, and using the JSON data format. The find layer enables search for things and can be automatically used by applications. The approach here is to reuse Semantic Web Standards which describe things and their services. This process is also called *discovery*. The share layer specifies how data generated by things can be shared in an efficient and secure manner over the web. The goal of the compose layer is to integrate data and services from heterogeneous things into an ecosystem of web tools such as analytics software and mashup platforms.

Subsequently, Guinard and Trifa proposed the W3C Member Submission for the Web Thing Model [391]. The Web Thing Model is a contract between clients and so called extended web things (see Figure 3.4) which allow clients to automatically discover and use its properties. They use JSON format for this model to describe such a thing. Further, extended web things must be accessible via an HTTP URL and supports HTTP GET requests on that URL (e.g. for exposing the Web Thing Model).

Discussion
Guinard and Trifa submitted their proposal for the Web Thing Model in 2017, it was however not standardized. The reason could be that the JSON format for description of things defines only the syntax. But the meaning of the single fields is described in the external documentation. They assume that things expose an HTTP access or at least "on an intermediate host in the network such as a Gateway or a Cloud service (for Things that aren't accessible through the Internet)". Even they remark that the Web Thing Model can also be applied to CoAP or MQTT, they don't specify further details. However, the idea of Guinard and Trifa was a starter for further development and refinements of the Web of Things.

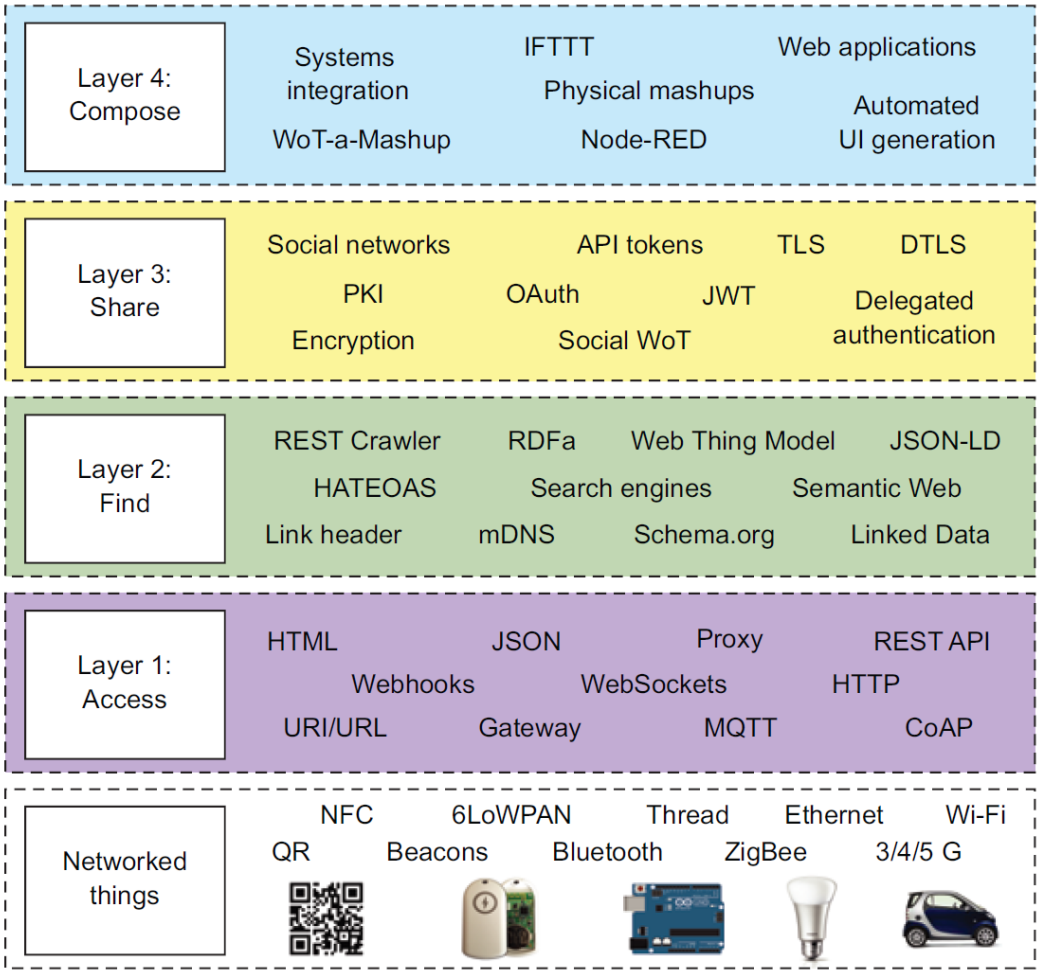


Fig. 3.3: The Web of Things architecture stack with its various layers [131]

W3C WoT Architecture

The W3C consortium has two Working Groups: Web of Things Interest Group (IG) und Web of Things Working Group (WG). The Web of Things Interest Group (IG) explores and test-drives concepts and technologies to determine WoT building blocks which are ready for progression with the W3C Recommendation track. Furthermore, the IG re-evaluates the current working assumptions of the Web of Things Working Group (WG) based on "running code". The W3C WoT PlugFests are the central mechanism where both, member and non-member implementors of WoT building blocks, come together to test their solutions and showcase their proposals.

Kovatsch et al. proposed WoT architecture [108, 226]. The first public working draft for the WoT Architecture was released in September 2017. The WoT Architecture and WoT Thing Description achieve the state of W3C Proposed Recommendation on 30 January 2020.

The WoT Architecture, shown in Figure 3.5, outlines various architecture patterns where several functional entities are involved such as devices, controllers, gateways and cloud servers. The WoT Architecture has four building blocks: the WoT Thing Description (TD) [184], the WoT Binding Templates [192], the WoT Scripting

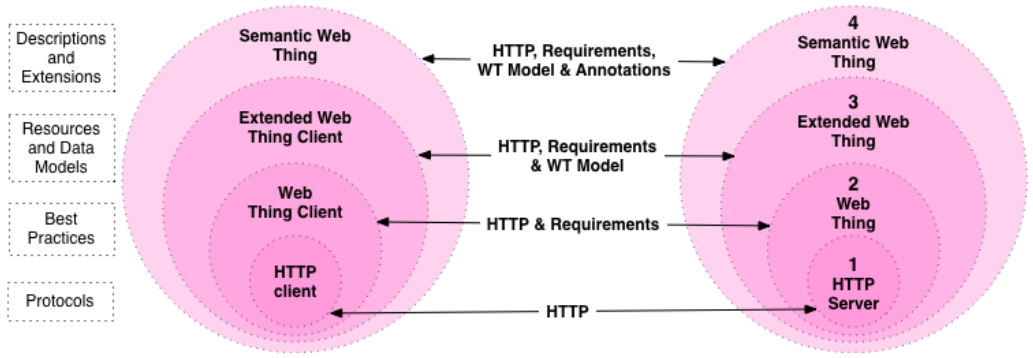


Fig. 3.4: Web Thing Model: the various levels for describing web Things [131]

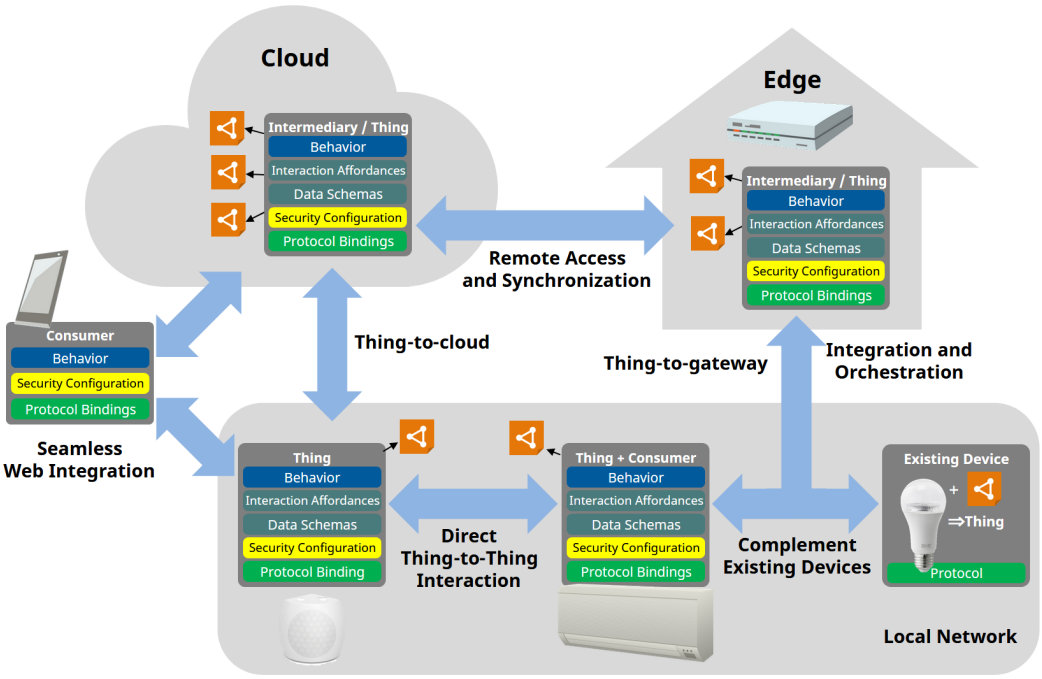


Fig. 3.5: Abstract Architecture of W3C Web of Things (WoT) [226]

API [274], the WoT Security and Privacy Guidelines [297]. A software that implements a WoT building block is called a *Servient*. A *Servient* can host and expose *Things* and/or host *Consumers* that consume *Things*. Servients can support multiple Protocol Bindings. Consumers parse TD descriptions and interact with Things.

The WoT Thing Description (developed among others by Charpenay and Kaebisch) plays a central role in the WoT architecture, which is discussed in Section 5.3.1 in context of semantic device descriptions. The WoT Binding Templates enable to adapt the Thing Description to different protocols and define vocabulary for HTTP, MQTT and CoAP. The WoT Scripting API describes a programming interface that "allows scripts to discover, operate Things and to expose locally defined Things". The WoT Security and Privacy Guidelines provide a guidance for security and privacy issues in the WoT architecture (i.e. requirements, privacy considerations, best practices and security validation).

At the time of writing, the new version 1.1 of the WoT architecture was proposed [389] without Kovatsch. The thing lifecycle (discovery and bootstrap) is still in discussion. The Thing Description is also work in progress. It seems to be a notable re-work of the WoT architecture.

Discussion
The WoT Thing Description provides a semantic description of devices in a simple form. But the WoT architecture is still lacking a proper discovery and bootstrap process. Even vocabulary terms for CoAP and MQTT are defined in the WoT Binding Templates, it is not specified in detail how to embed these protocols. These vocabularies are also very generic which is not sufficient for concrete implementation. From our experience, such protocols should follow some at least (naming) conventions to satisfy implementation requirements. However, WoT architecture does not consider the special needs of constrained devices. There implementation of the W3C WoT architecture is still proprietary[17].

Novo et al. [258] remark that the WoT "architecture does not cover all possible use cases and still has important limitations". Therefore, they extend this architecture to achieve a higher level of semantic interoperability for the IoT. They propose an architecture that enhances the semantic compatibility between components in the W3C and the IETF organizations.

W3C WebSub

Besides HTTP, W3C is working on publish/subscribe mechanisms for the Web. WebSub [113] achieved the W3C Recommendation state in 2018. The communication between publishers and subscribers, shown in Figure 3.6, relies on *hubs* which validate and verify requests. Then, hubs distribute new and updated content to subscribers when it becomes available. Hub are HTTP-based and Topics are URL-based.

Discussion
The purpose of the WebSub is not clear yet. There is similarity to MQTT but WebSub is HTTP-based which is not appropriate for constrained devices.

3.1.3 oneM2M

oneM2M is the worldwide standards initiative which main goal is to specify requirements, architecture, security and interoperability solutions for Machine-to-Machine (M2M) and IoT technologies. oneM2M was formed in 2012 and consists of 8 standards development organizations: ETSI (Europe) (see Section 3.1.4), ARIB (Japan), ATIS (USA), CCSA (China), TIA (USA), TSDSI (India), TTA (Korea), TTC (Japan); together with two industry consortia (GlobalPlatform, OMA SpecWorks) and over 200-member organizations (companies worldwide). The technical specifications and technical reports provided by oneM2M are free available. By the time of writing, the release 3[18] deliverables were ratified by the oneM2M Technical Plenary in December 2018. Further two releases have a draft state and are not completed yet.

[17] `https://projects.eclipse.org/projects/iot.thingweb`
[18] `http://www.onem2m.org/technical/published-drafts/release-3`

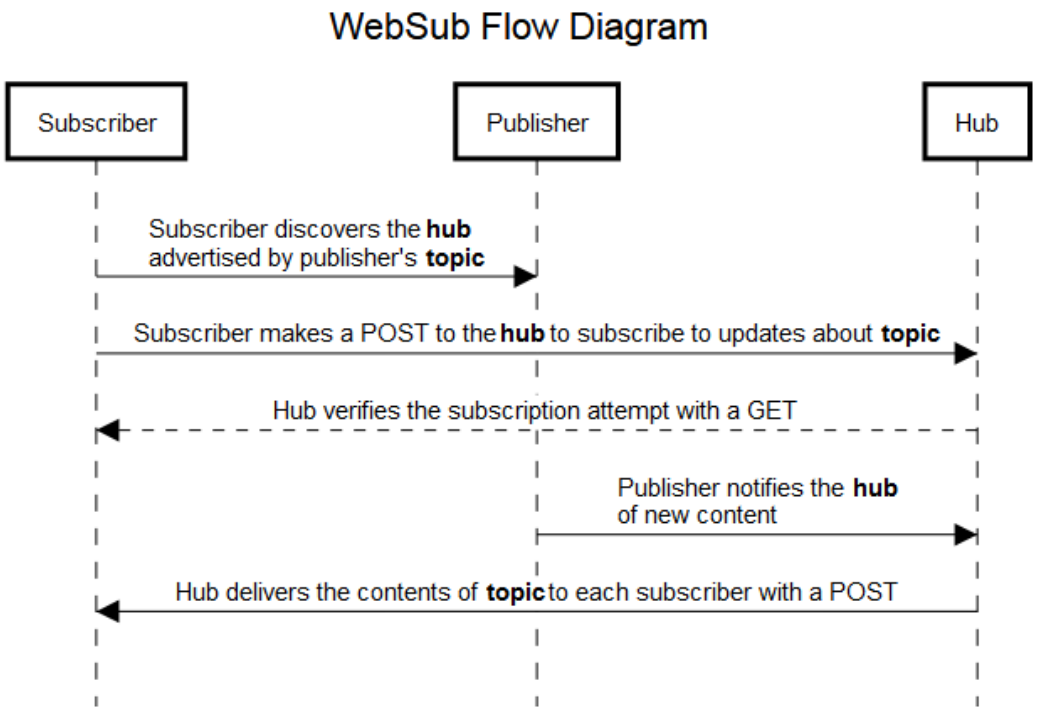


Fig. 3.6: WebSub Flow Diagram [113]

oneM2M Functional Architecture

The main technical specification is the end-to-end oneM2M functional architecture [110]. This specification focuses on the M2M Service Layer which connects devices with M2M application servers. The common services layer comprises oneM2M service functions that enable oneM2M applications (e.g. management, discovery and policy enforcement). The specification takes the underlying network-independent view of the end-to-end services. Underlying networks provide data transport services between entities in the oneM2M System.

The oneM2M functional architecture consists of three entity layers, shown in Figure 3.7: Application Entity (AE), Common Services Entity (CSE), Network Service Entity (NSE). The CSE is a subset of Common Services Functions (CSFs). The abbreviations Mca, Mcn and Mcc stand for "M2M communication" with application, network, common service entities. The CSFs are identified tasks which are necessary in an IoT field. We confined the related CSF functions: Device Management (DMG), Discovery (DIS), Group Management (GMG), Network Service Exposure, Service Execution and Triggering (NSSE), Semantics (SEM).

A *resource* is defined as a uniquely addressable entity in oneM2M architecture (e.g. AEs, CSEs, data, commands, etc.). A *resource* is transferred and manipulated using CRUD operations and can contain child resource(s) and attribute(s), which are also uniquely addressable. There are three categories of resources: normal, virtual and announced. Normal resources "include the complete set of representations of data which constitutes the base of the information to be managed". A virtual resource is used "to trigger processing and/or retrieve results, but they do not have a permanent representation in a CSE". An announced resource contains a set of attributes of the

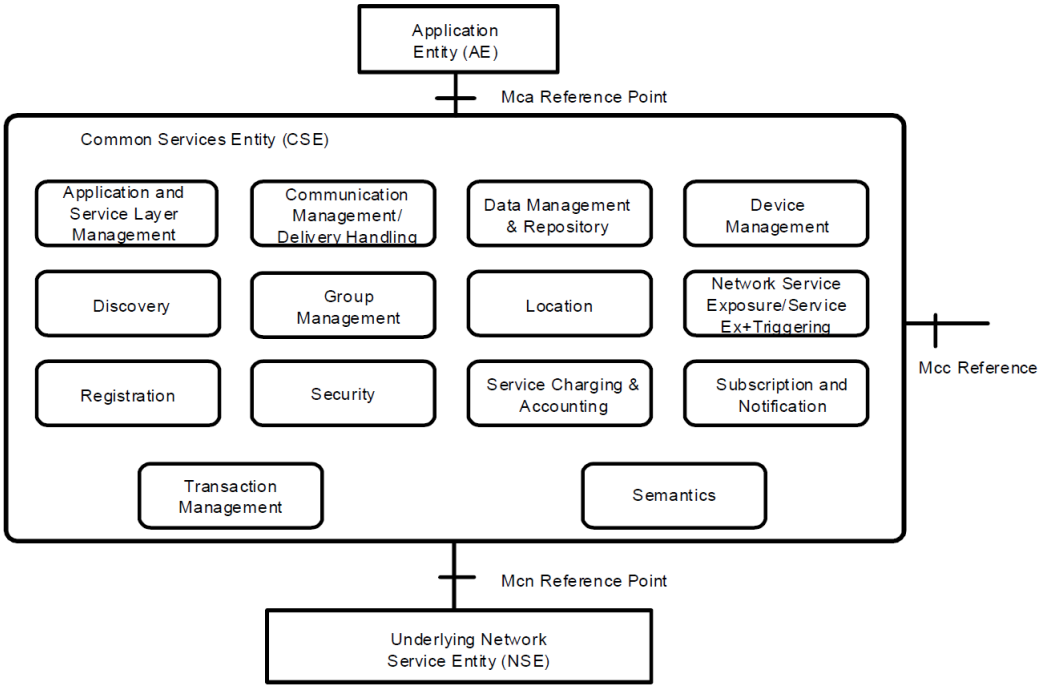


Fig. 3.7: oneM2M Common Services Functions [110]

original resource. A range of resource types and attributes are specified in oneM2M (e.g. <node>, <mgmtObj>, <mgmtCmd>, <deviceInfo>).

The DMG provides management of device capabilities on Middle Nodes (MNs, e.g. M2M Gateways), Application Service Nodes (ASNs, e.g. M2M devices) and Application Dedicated Nodes (ADNs, e.g. constrained M2M Devices). In order to manage the device capabilities, the DMG can utilize existing technology specific protocols (e.g. BBF TR-069 [48], OMA-DM [265], LwM2M [266]). For other protocols a Management Adapter has to be used. The DMG provides following functions: Configuration Function (DCF), Device Diagnostic and Monitoring Function (DDMF), Device Firmware Management Function (DFMF), Device Topology Management Function (DTMF). The tasks of these functions are quoted from the specification.

The DCF includes "discovery of a device's management objects and attributes, ability to enable or disable a device capability, provisioning configuration parameters of a device".

The DDMF includes following tasks: configuration of diagnostics and monitoring parameters on the device; retrieval of device information that identifies a device and its model and manufacturer as well as the software and firmware installed on the device; retrieval of information related to a battery within the device and the memory in use by a device; device reboot diagnostic operation. The DFMF manages the firmware lifecycle including actions to download, update or remove a firmware image.

The DTMF is responsible for the M2M Area Network, e.g. configuration of the topology, retrieval of information related to the devices, the transport protocol, the characteristics associated with online/offline status of devices.

The Discovery (DIS) function searches information about applications and services as contained in attributes and resources. The Group Management (GMG) function is responsible for handling group related requests (e.g. operations on multiple devices via multicast or broadcast). The Network Service Exposure, Service Execution and Triggering (NSSE) function manages communications with the underlying networks for accessing network service functions over the Mcn reference point. The network service functions include e.g. device triggering, small data transmission, location notification, policy rules setting, location queries, IMS services, device management [110].

The Semantics (SEM) function enables applications to manage semantic information and provides functionalities based on this information. The SEM function brings features related to the meaning of data and resources. The SEM function uses semantic descriptions provided by resources and based on ontologies. The SEM function supports features like annotation, resource filtering and discovery, querying, validation, mash-up, reasoning, analytics, etc. The SEM function handles the processes of discovery of resources and querying of semantic information, based on syntactic, semantic and structural information. Such descriptions are contained in semantic content data (such as RDF triples). The SEM function enables also the creation, execution and result retrieval of functions based on semantic mash-up, the validation of semantic content, and the use and management of ontologies.

The communication between the entities, i.e. Originator (CSE or AE) and Receiver(Hosting CSE), is based on request/response paradigm and consists of CRUD methods as well as notification. All AEs and CSEs are registered in a registry. Access policies determine the access of these entities. CSEs subscribe on each other's services and will be notified accordingly.

The DMG is based on the node management procedures over Mca and Mcc reference points, using the <node> resource which represents information about M2M Nodes. A <node> resource is hosted by the represented CSE. The Hosting CSE first collects the original technology specific data model object on the managed entity via technology specific protocol (e.g. OMA DM, BBF TR-069 or LwM2M), then transforms the object into the <mgmtObj> resource representation and create the <mgmtObj> resource locally in the CSE. The DMG describes how RESTful management operations may be performed using <mgmtCmd> and <execInstance> resources over the Mca and Mcc reference points (e.g. using technology BBF TR-069).

In order to provide a unique identifier for M2M devices, oneM2M proposes to use the international Object Identifier (OID) (developed by ITU-T 8638 and ISO/IEC in the Recommendation ITU-T X.660 and ISO/IEC 8648 9834-1 [294]). The OID has a hierarchical tree structure and is represented as a sequence of integer values. The M2M device ID is built for example of: M2M Device Indication ID, Manufacturer ID, Model ID, Serial No ID, Expanded ID.

oneM2M provides also specifications for protocol bindings like MQTT, CoAP, HTTP and WebSocket. Basically it includes the CRUD operations between the entities.

Discussion
oneM2M introduces its own vocabulary for architecture components (e.g. ADN stands for Application Dedicated Node and maps physically to a constrained M2M Device) which is generic and makes it difficult to understand the specifications. The oneM2M functional architecture is a high level architecture focusing on the

middleware services and does not consider the underlying network and devices. The discovery relies on a registry e.g. where consumers can lookup for sensor resources. However, the specification provides a basic concept for integration of non-oneM2M solutions via Interworking Proxy Application Entities (IPE)s introduced in Section 5.3.3.

3.1.4 ETSI

European Telecommunications Standards Institute (ETSI) is a member of the oneM2M and is actively working on the specifications there (e.g. oneM2M Functional Architecture published also as ETSI TS 118 101 V2.10.0 (2016-10))[19]. ETSI published nearly 100 documents around the IoT field (39 of them from SmartM2M committee). Some of them address protocols (e.g. "IPv6-based Internet of Things Deployment of IPv6-based Internet of Things"), analysis and domain-specific studies (e.g. industrial, home, energy), requirements, guidelines.

The ETSI Technical Committee (TC) SmartM2M enables connected devices to exchange information through the Smart Applications REFerence (SAREF) ontology that runs with oneM2M-compliant communication platforms. With SAREF, SmartM2M is promoting oneM2M Base Ontology with extensions in six IoT domains by now: industry and manufacturing, smart cities, building, environment, energy, smart agriculture and food chain. The SAREF ontology is discussed in Section 5.3.4 in context of semantic device descriptions.

The SAREF ontology claims to provide semantic interoperability and can be mapped to the oneM2M Base ontology by ETSI TS 103 264 V2.1.1 (2017-03) [367, 368]. This package together with four testing specifications build the ETSI M2M Communication Framework. However, "technically the oneM2M system is an evolution and extension of the ETSI M2M system & specifications" and allows interworking from ETSI M2M to oneM2M M2M systems.

The Eclipse OM2M project [6] is an open source implementation of oneM2M and SmartM2M standard, initiated by the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS) in France. It provides a horizontal Common Service Entity (CSE), specified by oneM2M, which can be deployed on servers, gateways, and devices. It provides a RESTful API interface and is running on top of an OSGi container, which is extensible via plug-ins.

Discussion
The most significant ETSI contributions are the SAREF ontology and oneM2M specifications. ETSI targets semantic interoperability issue in the IoT and defines domain-specific extensions for ontologies which are not defined in the oneM2M Base ontology.

3.1.5 OMA - Open Mobile Alliance and LwM2M

The Open Mobile Alliance (OMA) SpecWorks is a standardization organization, originally for mobile network, nowadays also for the IoT. The OMA members are well-known companies which are interested in the interoperability of their services and products. In 2018, OMA has joint the IPSO Alliance to enforce the standardization on device management for the IoT using the smart objects defined by IPSO (Internet

[19] `https://www.etsi.org/committee/onem2m`

Protocols for Smart Objects)[20]. All specifications by OMA are free available. Among them, the LwM2M protocol for the IoT.

OMA specified the Lightweight Machine to Machine (LwM2M) [266] as a device management protocol for M2M networks. Supported functions are Bootstrapping, Device Configuration, Firmware Update, Fault Management, Configuration & Control, Reporting. The LwM2M protocol stack is based on CoAP for CRUD and notification communication, DTLS and OSCORE [340] (end-to-end) for security, IPSO smart objects for data modeling. As shown in Figure 3.8, the LwM2M architecture consists of LwM2M server and bootstrap server as well as a LwM2M client and four interfaces: (1) Bootstrap, (2) Client Registration, (3) Device Management and Service Enablement, (4) Information Reporting.

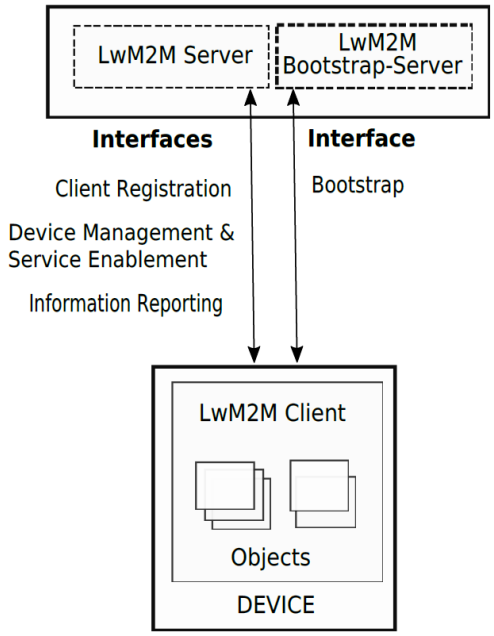


Fig. 3.8: The overall architecture of the LwM2M Enabler [266]

An LwM2M client provides a Resource Model (see Figure 3.9). A Resource is a "piece of information" made available by a client. These resources are logically organized into Objects. Every resource has a unique identifier and a data type. For example, the Firmware Update Object contains all the Resources used for firmware update purposes. Each Object, defined for the LwM2M Enabler, is assigned an unique OMA LwM2M Object identifier allocated and maintained by the OMA LwM2M Object and Resource Registry[21], former IPSO Object Model. OMA also hosts IPSO Vorto models (see Section 5.3.8). Further Objects may be added by OMA or other organizations to enable additional M2M Services. Following objects are registered among others: Security, Server, Access Control, Device, Device Capability Management, Connectivity Monitoring, Firmware Update, Location, Connectivity Statistics.

For example, the Device object provides information about manufacturer, firmware version, power sources, memory, device type, etc. The following data formats are

[20] https://www.omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/
[21] http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html

defined for transferring of Resource information: plain text, opaque, Type-Length-Value (TLV), JSON, CoRE Link, CBOR, SenML JSON, and SenML CBOR.

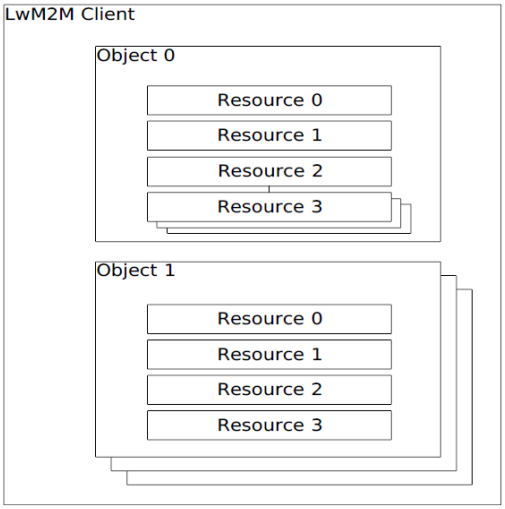


Fig. 3.9: Relationship between LwM2M Client, Object, and Resources [266]

The bootstrap process is initiated by a client (see Figure 3.10). Server initiated bootstrap is also available. Additionally, the required information to function with the LwM2M server can be stored during manufacturing on the flash or a smartcard. In order, to start the bootstrap process, the client must have a *Bootstrap-Server* account pre-provisioned. The bootstrap-server provides the LwM2M server account to the client. The *Bootstrap-Discover* operation is used to discover which LwM2M Objects and Object Instances are supported by the client. The retrieved payload is a list of application/link-format CoRE Links.

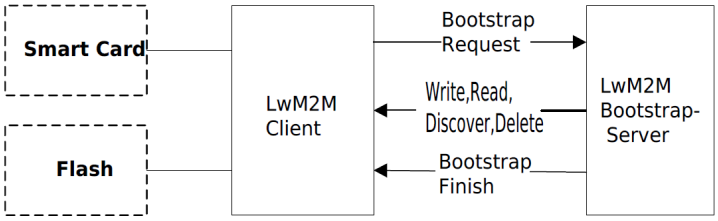


Fig. 3.10: LwM2M Bootstrap Interface [266]

For the device management and service enablement(see Figure 3.11) only downlink operations are defined. The *Discover* operation is used to discover attributes (i.e. metadata) and to discover which resources are implemented in a certain object (i.e. functionality).

The *Reporting* (see Figure 3.12) is initiated by sending *Observe* or *Observe-Composite* operation from the server. The *Notify* is an uplink operation, which is used to send a new value of a *Resource* from the LwM2M Client to the LwM2M Server. The *Send* operation is another uplink operation which is used by the LwM2M Client to send data to the LwM2M Server.

LwM2M claims to support a range of security technologies: certificates, pre-shared key, public raw key, PKI deployment; then DTLs and SMS, and Object Security for

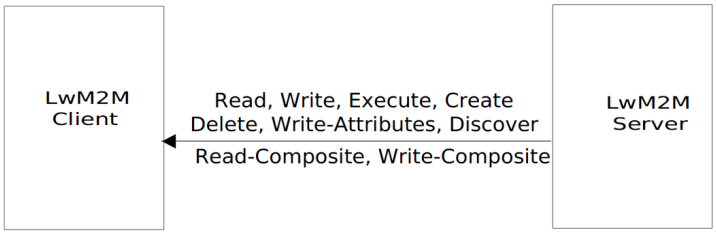


Fig. 3.11: LwM2M Device Management and Service Enablement [266]

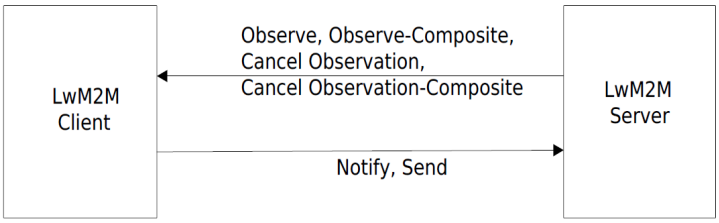


Fig. 3.12: LwM2M Information Reporting [266]

Constrained RESTful Environments (OSCORE) (RFC 8613). It should be able to replace initial keys during bootstrap process.

An open implementation for LwM2M server (Leshan[22] in Java) and LwM2M client (Wakaama[23] in C) exists as Eclipse projects. Datta et al. [309] used LwM2M to provide M2M service enablement for end users and integrated it into oneM2M architecture. Robles et al. [301] proposed a performance measurements platform designed with LwM2M.

Discussion
The LwM2M protocol offers a comprehensive number of features for device management. However, it has some limitations: support only of one protocol namely CoAP; proprietary data modeling. A proposal for LwM2M over MQTT was made on Eclipse IoT Days in 2017 [24]. LwM2M claims to support battery-constrained devices (low client footprint) but the evaluation results show that LWM2M is feasible on Class 2 device [329].

3.1.6 OGC and SWE

Open Geospatial Consortium (OGC) was founded in 1994, originally named GIS Consortium. The OGC is an international consortium with about 500 organizations "driven to make geospatial (location) information and services, like FAIR - Findable, Accessible, Interoperable, and Reusable". OGC defines Abstract Specifications and Implementation Standards for Geo Information Systems (GIS). The Abstract Specification is a reference architecture of the OGC vision of geospatial technology and data interoperability. The Abstract Specification provides the conceptual foundation for the Implementation Standards. The Implementation Standards are written for technical audience like software engineers.

[22] https://projects.eclipse.org/projects/iot.leshan
[23] https://projects.eclipse.org/projects/iot.wakaama
[24] https://wiki.eclipse.org/images/e/e1/LWM2M_MQTT_EclipseIoTDaysGrenoble.pdf

A Geo Sensor Network (GSN) [28, 34] is one of the OGC's application fields. A GSN is a special kind of a WSN which considers the spatial information of nodes or the whole network. A GSN is used to monitor environment (i.e. forest, landscape area). Usually, sensor data in such network is collected in a sink. In 2006, OGC specified the Sensor Web Enablement (SWE) framework with the idea to connect such GSN networks through the Web in a global Geo Data Infrastructure (GDI). Thus, the sensor data becomes accessible and findable on the Internet through SWE services-based interfaces. SWE specified languages for describing sensors, their capabilities and measurements.

The SWE specifications are listed in Table 3.3, based on overviews [232, 263]. Over time, some modifications were made. The specification for the Transducer Markup Language (TML) was retired. Further, JavaScript Object Notation (JSON) encodings were introduced to the O & M and the SensorML as well as SWE Common Data Model, and specified in version 2.0.

Tab. 3.3: OGC SWE specifications

Specification	**Description**
Observations and Measurements Schema (O&M)	XML (OMXML), JSON schema and models for standardized description of sensor data
Sensor Model Language (SensorML)	schema and models for standardized description of sensors and sensor processes
SWE Common Data Model Encoding Standard	low level data models for exchanging sensor related data between nodes
Sensor Observation Service (SOS)	standardized service interface for querying observations
Sensor Planning Service (SPS)	standardized service interface for queries about the capabilities of a sensor and how to task the sensor
SWE Service Model Implementation Standard	8 packages with data types for common use across SWE services
PUCK Protocol Standard	protocol for RS232 and Ethernet connected instruments
Sensor Alert Service (SAS)	service for advertising, subscribing, and publishing alerts (best practice, not a standard)
Web Notification Service (WNS)	service for asynchronous message interchanges (best practice, not a standard)

SensorML and O&M provide complementary viewpoints. SensorML [45] is provider-centric and encodes details of the sensor with raw observation data. Their properties like self-containing and flexibility make life easy for data producers but are demanding on consumers. In contrast, O&M [73] was designed to be more user-centric with the target of the observation and the observed property (see Figure 3.1). O&M works at a higher semantic level than SensorML. However, it provides only abstract classes for sensors, features of interest and observable properties and expects the details to be provided by specific applications and domains. O&M also provides a model for sampling and feature of interest.

Since 2015, OGC is working on the SensorThings API which should provide an open, geospatial-enabled and unified way to interconnect the IoT devices, data and

```
{"id":"measure-instance-test",
"type": "Measurement",
"phenomenonTime": { "instant":"2011-05-11T00:00:00+10:00" },
"observedProperty": {"href":"http://environment.data.gov.au/def/property/
    air_temperature"},
"procedure": {"href":"http://www.opengis.net/def/waterml/2.0/processType/Sensor"},
"featureOfInterest": {"href":"http://waterml2.csiro.au/rgs-api/v1/monitoring-point
    /419009/"},
"resultTime": "2011-05-12T09:00:00+10:00",
"result": {"value": 3.2,
"uom": "http://qudt.org/vocab/unit#DegreeCelsius"}}
```

Listing 3.1: JSON encoding of the O & M Observation

applications over the Web. The API builds on open standards like Web protocols and SWE (incl. O&M data model) and consists of two parts: (1) Sensing and (2) Tasking Core. The Sensing part provides "a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems" (i.e. functions similar to the OGC SOS service). The Tasking part provides a "standard way for parameterizing - also called tasking - of task-able IoT devices, such as sensors or actuators" (i.e. functions similar to the OGC SPS service). The main difference between the SensorThings API and the OGC services is that the API is designed specifically for the resource constrained IoT devices and the Web developer community. Therefore, the API "follows the REST principles, the use of an efficient JSON encoding, the use of MQTT protocol, the use of the flexible OASIS OData protocol and URL conventions".

Discussion
The SWE framework enables GSNs to provide a standardized interface to the Internet. The SWE does not specify internals of a single GSN. These internals are left to the GSN providers. The SWE standards, namely SensorML and O&M, provide syntactic interoperability but do not address semantic interoperability and do not provide a basis for reasoning offered by semantic technologies. These shortcomings and the development towards IoT were already addressed by research [50, 49, 198]. First evaluations [401] of the OGC SensorThings API were promising. Further, OGC and W3C are working together on the SSN ontology for semantic interoperability (see Section 5.3.2).

3.1.7 Conclusion

After the detailed review of related specifications and standards about network management in the IoT, we found many efforts towards interoperability. However, the organizations tend to use their previous standards and specifications. For example, IETF creates an CoAP ecosystem and W3C the Web of Things. This could also lead to a kind of vendor lock-in. Nevertheless, oneM2M specifies a more generic architecture and also proposes interworking solutions with other systems. Only few standards consider MQTT protocol in their IoT solutions, although it has become the de-facto IoT standard [366].

We do not review further standardizations and alliances because their contribution and further development to the IoT implementations is not clear. The recommendations of ITU-T and ISO/IEC regarding the IoT are mostly high-level and summarize the state of the art. The Alliance for Internet of Things Innovation (AIOTI) represents the member interests and tries to bring innovation development in the IoT to support

digitization in Europe. OCF(former OIC) has undergone many changes in the past. OCF provides a certification program to enable manufacturers to bring OCF certified products to the market and has commercial interests. The OPC and OMG propose architecture and protocols for industrial IoT which are not in focus of this thesis.

3.2 Vendors and Open-Source Implementations

Many vendor implementations for the IoT exist. The solutions differ greatly, as the survey [236] shows: cloud-based, centralized, decentralized, open-source, closed-source, etc. Some of them support heterogeneous devices and service discovery, and other do not. In this section, the general approaches of some well-known vendors and open-source solutions are introduced.

3.2.1 Vendor Lock-in

Some vendors offer a platform for development of customized IoT solutions (i.e. Amazon AWS IoT and Microsoft Azure). Other vendors like IBM offer completed IoT services. Third vendors change their IoT strategy over the years. Google has started with Google Brillo and Weave in 2015, moved to Android Things Console and Nest Weave, and offers now Cloud IoT Core and Nest Device Access (focusing on smart home).

Amazon
Amazon IoT[25] uses the existing cloud services, provides SDK and Tools, supports MQTT, HTTP, WebSockets protocols. Amazon IoT uses semantics (JSON-formatted) in so-called Thing Registry and in Rules which are formulated as SQL-similar language. Amazon AWS IoT Device Management[26] provides a cloud-based management solution. Amazon defines device management in the IoT as following: it is a device management service that manages (remotely) IoT devices throughout their lifecycle (i.e. onboard device information and configuration, organize their device inventory, monitor their fleet of devices, and remotely manage devices deployed across many locations including over-the-air (OTA) updates to device software). This definition is close to network configuration.

Amazon Greengras is Amazon's answer on edge computing. A part of the cloud-based services it can be exported and deployed on the edge to support basis functionality in case the internet connection gets lost.

Microsoft Azure
Microsoft Azure IoT[27] offers a collection of cloud-based services which connect to IoT devices, manage and control them. Azure IoT Hub provides a catalogue of certified devices which can be connected to Azure IoT. It supports protocols like HTTP, MQTT, AMQP. Azure IoT Edge application analyzes data on the edge network. Azure Digital Twins is a platform where a digital representation of an environment can be modeled. Azure RTOS is a customized operation system for constrained devices.

[25] https://docs.aws.amazon.com/de_de/iot/latest/developerguide/what-is-aws-iot.html
[26] https://docs.aws.amazon.com/iot-device-management/index.html
[27] https://azure.microsoft.com/de-de/overview/iot/

IBM
IBM developed the MQTT protocol originally for the monitoring of pipelines with sensors. MQTT protocol is the core component of the IBM Bluemix, the Cloud-Platform as a Service (PaaS). The IBM Watson IoT uses artificial intelligence base on technologies like semantics (Natural Language Processing) and Machine Learning for video and image analysis. IBM developed the Node-RED[28] and offered it open-source. Node-RED is a browser-based application, developed with Node.js. It offers a visual and extendable building block concept for connecting IoT devices, APIs and services.

3.2.2 Eclipse Projects

The well-known Eclipse Foundation hosts a range of IoT open source projects united under the top-level project *Eclipse IoT*[29]. Related Eclipse projects to this thesis are summarized in Table 3.4. Only three projects have a stable state: Californium CoAP Framework, Kura, Paho. All other projects are incubating. Remarkable is that the Bosch company is an active contributor and supports many IoT projects. Finally, they offer an IoT Suite[30] and offer some parts as open source trying to establish their frameworks. We will review Ditto and Vorto (in Section 5.3.8). The other active company contributor is Eurotech with projects Kura and Kapua. Kura is MQTT-based and provides guidelines for the MQTT topic namespaces. They segregate between persistent and transient data which results in distinction between Data Topics for sensors and Control Topics for actuators. Further, they use keywords and response codes similar to HTTP or REST (e.g. keywords like GET, POST, PUT, DEL, REPLY and code like 200 (RESPONSE_CODE_OK), 404 (RESPONSE_CODE_NOTFOUND) etc).

Discussion
Vendors provide vertical solutions for the IoT. However, the disadvantage is the dependency on the vendor for enhancements or upgrades, so-called *vendor lock-in*. Connecting two vendor solutions or migrating to another platform is nearly impossible.

There are over 40 Eclipse IoT projects but most of them have an incubating state. Some implementations, particularly of standards (MQTT, CoAP), are established in the developer world. Open-source implementations of standards have clear advantages. Many vendors are following the strategy to offer their solutions as Open Source Solutions (OSS). It is questionable whether these software will be further developed when the vendors do not support the project anymore.

3.3 Conclusion

Even though there are many IoT solutions, there is no clear winner yet. SDOs offer only partial solutions depending on their roots and technical focus. Vendors produce vendor lock-in because a service consumer is dependent on the platform and has to pay for services and upgrades. Only some SDOs and open-source solutions address constrained devices and use specific IoT protocols like MQTT.

[28] https://nodered.org/
[29] https://projects.eclipse.org/projects/iot
[30] https://www.bosch-iot-suite.com/

Tab. 3.4: Related Eclipse IoT Projects

Project	Description	last commit
Bosch Company as Supporter and Contributor		
Californium (Cf) CoAP Framework	CoAP protocol implementation in Java	2020
Ditto	framework for providing the "Digital Twin" pattern for IoT applications	2020
hawkBit & Hara	domain independent back end solution for rolling out software updates to constrained edge devices and reference agent software implementation	2020
Hono	IoT Connector cloud component	2020
IoT Packages	getting started packages for Eclipse IoT projects	2020
Kiso	RTOS integration component	2020
Vorto	comprises of the meta information model, the tool set to create information models, the code generators and the repository to manage existing information models	2020
Eurotech Company as Supporter and Contributor		
Kura	Java/OSGi-based container for M2M applications running in service gateways	2019
Kapua	cloud-based IoT architecture for device connectivity and management	2020
Implementations of IoT Standards		
Mosquitto	MQTT Server/Broker implementation	2020
Paho	MQTT Client implementation	2020
tinydtls	library for DTLS in CoAP	2017
Leshan & Wakaama	LwM2M Server and Client implementation	2020/2018
OM2M	implementation of the ETSI M2M standards	2018
Thingweb	W3C WoT Servient implementation	2019
Cyclone DDS	OMG Data Distribution Service (DDS) implementation	2019
Milo	implementation of OPC Unified Architecture	2019
Implementations for IoT Hardware		
MRAA	high-level, easy-to-use set of APIs for I/O access on Linux boards and systems, similar to Arduino offerings for MCU boards, support by Intel	2020
UPM	builds on the solutions of MRAA	2019

4 Network Configuration Management with MYNO Framework

> *Every networking problem always takes longer to solve than it seems like it should.*
>
> **— RFC 1925, Nr. 9a**
> (The Twelve Networking Truths)

The concept of the MYNO framework is outlined in this chapter as an answer on the requirements from the introduction (see Section 1). Since none of the solutions, analyzed in Section 3, is convincing.

This chapter is organized as follows. First, related work is provided on network and device management in the IoT. Then, the MYNO architecture is introduced. Afterwards, the NETCONF-MQTT bridge as one of the central components is explained. Further, the concepts for the core functionalities are introduced: discovery and bootstrapping; actuator, sensor and event triggering scenario. Further concepts about MYNO framework are introduced in separate chapters because they are topics in themselves.

4.1 Related Work: Device and Network Management

Network configuration management can help to automate the more error prone tasks and therefore to save time and reduce the risk of errors. Network configuration management is designed to allow to take control of network changes.

Silva et al. [363] defines two categories for IoT management: *IoT Network Management* and *IoT Device Management*.

> *IoT Network Management* is required to collection and analysis large volumes of data from IoT platforms and, consequently, provides efficient decisions and/or actions. *IoT Device Management* is required to provide the device location and status information, e.g., update embedded software, disconnect some stolen or unrecognized device, modify security and hardware configurations, locate a lost device, and even enable interaction between devices. [363]

In the IoT Network Management, Silva et al. differ between protocols (e.g. SNMP, NETCONF, LNMP protocols) and platforms (e.g. OpenNMS also supports SNMP, NETCONF). However, some of the surveyed protocols are not mature. For example, LNMP [139, 254] is a protocol that adapts the SNMP messages for 6LoWPAN networks because running SNMP on the device is impracticable due to the limited

device's resources. Nevertheless, the LNMP protocol was not developed further. Some platforms are not developed specially for the IoT needs: e.g. OpenNMS[31] is an open-source platform with SNMP and NETCONF support but do not consider IoT and constrained devices exactly.

Further, Silva et al. differ between IoT Device Management Protocols and Platforms. IETF Draft identified some candidates for the so called COnstrained Networks and Devices MANagement (COMAN) [122] like OMA-LwM2M, OMA Device Management (OMA-DM), CoAP, SNMP, NETCONF, etc. The authors criticize no support for the heterogeneity for OMA-LwM2M, OMA-DM and CoAP. The surveyed IoT Device Management Platforms (among them oneM2M) are mostly a middleware or cloud-based software. The authors of surveys [363, 67] conclude, heterogeneity among devices and interoperability are still open issues even more in IoT Network Management than in IoT Device Management.

The management of devices is a network task and occurs at the edge of the network. Therefore, standard protocols such as Simple Networking Management Protocol (SNMP) and its successor Network Configuration Protocol (NETCONF) can be used for network management. SNMP [55] protocol manages and controls network elements such as hosts, gateways, terminal servers, and the like. However, SNMP has some shortcomings identified in [331, 363]. For example, no distinction between configuration and operational state; no configuration change notifications; no support of standard tools, etc. SNMP is focused on monitoring and not on network configuration. NETCONF [93] is the newer protocol and provides mechanisms to install, manipulate, and delete the configuration of network devices. Furthermore, NETCONF uses standards like XML-based configuration database and Remote Procedure Call (RPC).

One of the first IoT devices [303] was a toaster in 1990, connected to the Internet by TCP/IP and controlled by a SNMP with the help of Management Information Base (MIB). It had one control operation to turn the power on and off. The authors wanted to demonstrate the possibilities of SNMP. However, SNMP and NETCONF protocols require powerful devices and are not appropriated for running on constrained devices [336, 337, 334]. Schönwälder, Sehgal et al. [337, 334] investigated how existing IP based network management protocols can be implemented on resource constrained devices and measured the high resource requirements for SNMP and NETCONF. This work resulted in the proposal for NETCONF Light [336] which describes an approach using a subset of the NETCONF protocol operations on constrained devices. However, NETCONF light was not standardized.

Years later, the management of networks with constrained devices was analyzed by the Internet Engineering Task Force (IETF) and problem statement and requirements as well as use cases were formulated in Request for Comments (RFC)s [94, 95]. They state SNMP and NETCONF as possible candidates but remark that problems with constrained devices and networks may occur in terms of memory and CPU as well as unreliable network connection.

Some approaches [420, 77] considered to use NETCONF over RESTful web services for IoT. In [420], a device must implement a HTTP RESTful web service in order to become a managed object. HTTP services are not an option for constrained devices because they always have to be on and cannot save energy. YANG [38] is a data

[31] `https://www.opennms.com/`

modeling language for the NETCONF. In [77], a YANG model is defined for a room which has sensors and a controlling device. This device with a NETCONF server installed was a Raspberry Pi which is not a very constrained device.

Recent research works [404, 339] show management and monitoring with NETCONF and YANG of home appliances but devices like a washing machine. This related work shows that the NETCONF protocol is a considerable approach for network management of IoT devices. However, the challenge is still the network management of constrained devices.

Vijay et al.[404] used NETCONF and YANG for management of IoT Devices in home network via intelligent home gateway. They installed NETCONF server on a washing machine. Wallin et al. [409] introduced an approach for automating network and service configuration using NETCONF and YANG. They run performance tests in a cloud managing 2000 devices. Ontology-based approaches are also known in the network management [214] and can improve several tasks in the network management value chain. They describe several research projects where service ontologies are applied to network management. Ontology mapping for the interoperability problem in network management was studied in [417]. They describe an approach with heterogeneous router configuration management.

Discovery of devices is a part of network management: semantic discovery [230] using a resource directory and a multi-proxy module, MQTT-based discovery [285] with BLE and fog environment; and evaluation of service discovery protocols (CoAP, DNS, DDS) [52].

While the IoT platforms started with cloud-based solutions, they move towards edge computing. For example, the big two: AWS IoT Greengrass extends AWS services to edge devices and Microsoft is pushing Azure IoT Edge as a managed service to deploy cloud workloads on edge devices. Because edge computing has a big advantage comparing to the cloud: if the internet connection gets lost the edge computing systems do not fail. Even reasoning on the edge can be processed [351]. However, Edge computing in the IoT is still work in progress, as IETF draft shows Challenges and Functions for the IoT Edge computing [153].

4.2 MYNO Architecture

Scheffler and Bonneß [327] propose another approach using NETCONF for constrained devices. Instead of installing NETCONF on constrained devices, they propose a NETCONF-MQTT bridge which translates between the IoT specific MQTT protocol and NETCONF. Furthermore, they use a dynamically generated YANG model instead of static pre-defined configuration and state data model. The YANG model is generated when a new device is joining a network.

The bridge approach [327] shows that the translation between the MQTT protocol and NETCONF is feasible. This bridge is particularly useful when many IoT devices must be managed in the network. The IoT devices can describe their capabilities and disclose them. The NETCONF server operates as a network manager at the edge of the network.

This thesis picks up the idea of the NETCONF-MQTT bridge. The MQTT [243] protocol uses the publish/subscribe principles and is well established in the IoT

domain, [366]. Furthermore, the publish/subscribe paradigm is popular and long standing approach for Wireless Sensor Network (WSN) [186]. WSN is the predecessor and enabler of the IoT because constrained devices and constrained networks were the research subject there.

We have extended their approach and built the MQTT-YANG-NETCONF-Ontology (MYNO) framework. In contrast to their experiment in [327], we have extended the bridge as following:

- we adjusted the setup for constrained microcontroller boards connected either by 6LoWPAN or by WLAN, while they used a LIFX LED bulb connected by WLAN.
- we used boards with sensors not only with actuators like the bulb.
- we proposed to use semantic device description based on the extended oneM2M Base ontology [24] for the description of device capabilities instead of a proprietary description format JSON.
- we introduced a bootstrap mechanism to make the framework robust.
- we designed a web-based user-interface for the NETCONF Client to visualize the network configuration management.
- we proposed a concept of virtual device for aggregation of capabilities at the edge.
- we developed the MYNO update protocol (MUP) for distribution of firmware updates for IoT devices.

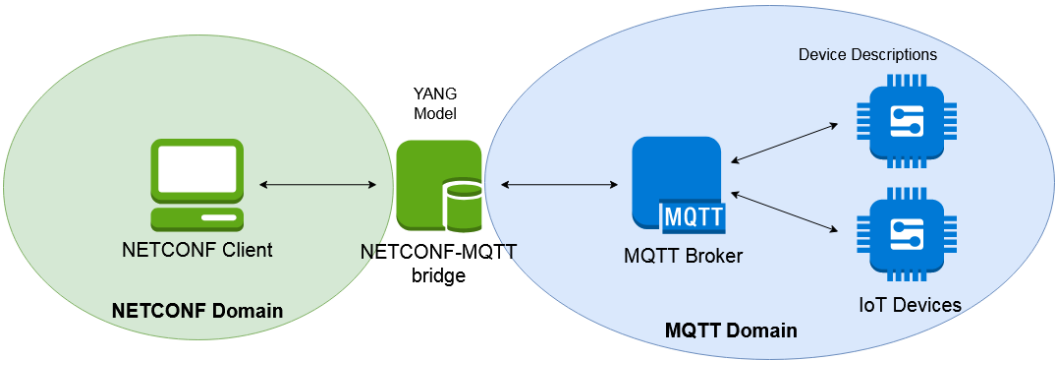


Fig. 4.1: System-Architecture with MQTT and NETCONF protocol domains

The system architecture of the MYNO framework is shown in Figure 4.1. The main components are the web-based NETCONF client, the NETCONF-MQTT bridge, MQTT broker and IoT devices (microcontroller boards). The NETCONF-MQTT bridge manages the YANG model generated from the device descriptions published by the IoT devices. The concepts behind the single components are introduced subsequently.

4.3 NETCONF-MQTT Bridge

The NETCONF-MQTT bridge (subsequently referenced as the bridge) mediates between two separate network management domains: the constrained node network (e.g. 6LoWPAN) via MQTT and the enterprise network with NETCONF configuration for the administration through a user. The NETCONF-MQTT bridge has three components:

1. A discovery and bootstrapping mechanism for discovery of new devices and their capabilities.
2. A translation component which generates a YANG model from the device descriptions.
3. A mapping component that transforms a NETCONF RPC into the device specific MQTT command.

The discovery mechanism is based on the MQTT Publish/Subscribe principle and device descriptions. The discovery is a part of the bootstrap process. The devices use device descriptions for sharing of their capabilities. The MQTT broker coordinates *Topics* and messages between the constrained devices and the NETCONF domain.

The NETCONF-MQTT bridge translates the device descriptions into a YANG model as soon as a device joins the network and publishes its description. Further, the bridge acts as a NETCONF server and translates the RPC calls from a NETCONF client into MQTT Publish messages. The information for this mapping, namely MQTT Topics and functions, must be provided by the device description.

4.3.1 MQTT Topics in MYNO

MQTT Broker is agnostic to the content of the message payload and also to the topic names. However, it is reasonable to determine a topic schema or naming conventions for the framework because it simplifies the topic management for the developing process and manufacturers. Following types of the MQTT topics are defined:

- sensor topics to send sensor data: *sensor/type/name/UUID*.
- actuator topics for controlling actuators: *actuator/type/name/UUID* .
- response topics as response channels for the actuator topics: *response/UUID*.
- error topics for separate error reporting: *error/UUID*.
- default topics for bootstrapping process, see Section 4.4.
- topics for events and their configuration: *event/type/name/UUID* and *config/-type/identifier/UUID*.
- topics for automation function: *automation/sensor/type/name/UUID*

Only the topics for bootstrapping are default and must be pre-configured on all IoT devices, see Section 4.4. The bootstrapping process is used by devices for publishing their capabilities for the network configuration. These topics are also used by default from the NETCONF-MQTT bridge.

Device-specific topics are deduced from device descriptions. Considering the topics for sensors and actuators, we determined that there is useful information in the

topic which at least must be contained: the distinction between sensor or actuator, type and name of sensor or actuator, and a unique identifier of the device. The naming schema for topics is defined as following: *actuator/type/name/UUID* and *sensor/type/name/UUID*. The topic schema consists of the first keyword "sensor" or "actuator", following by the type of sensor or actuator. The type is also a keyword from a domain vocabulary, i.e. temperature, humidity, LED, etc. The name is the device-specific name, for example temp1, temp2, led3, led4. Finally, the Universally Unique Identifier (UUID) of the device is the last part which makes the topic unique. While building the MQTT topics this way, the power of wildcards for subscriptions can be used. For example, the subscription to all temperature sensors or LEDs will result in *sensor/temperature/#* and *actuator/led/#* using the # wildcard which includes all subtrees. The subscription to certain UUID for all sensors and actuators will result in topics like *sensor/+/+/UUID* and *actuator/+/+/UUID* because the plus sign + can be used at more than one level in the so called Topic Filter.

Such topics can be quite long, therefore Vogli et al. [406] proposed an efficient resource naming for enabling constrained devices in ETSI SmartM2M architecture. They evaluated this approach with short URLs on devices with CoAP and 6LoWPAN.

As UUID can be used either the manufacturer UUID or MAC-address or other generated UUIDs. The UUIDs as URNs are defined in [208] which is compatible to the ISO/IEC 9834-8 Standard and ITU-T Recommendation. Basically, any unique IDs can be used as UUID. The only requirement is that they are compatible with MQTT Topic name requirements (e.g. must not contain wildcard characters).

In order to provide a unique identifier for M2M devices, oneM2M propose to use the international Object Identifier (OID) (developed by ITU-T 8638 and ISO/IEC in the Recommendation ITU-T X.660 and ISO/IEC 8648 9834-1 [294]). The BLE use Bluetooth Base UUID (128 bit = 16 bytes) which can be combined with a short 16-bit Bluetooth UUIDs. However, BLE use UUIDs for everything as keys in the key/values pairs.

Often a location or a location hierarchy is caught in the topic name but this is not useful for several reasons: (i)the device itself originally doesn't know where it will be placed; (ii) the device location can change over time. However, for better separation of concerns, the location should be configured as a context from outside in the framework.

The topic schema for responses published by a device is built as *response/UUID*. Since MQTT 5.0, the corresponding response topic is part of the standard and can be provided in the published "request" message.

The topic schema for errors published by a device is *error/UUID*. Such errors can be collected by a monitoring software and alert network administrators.

An MQTT topic can be quite long which is impedimental for constrained networks (e.g. 6LoWPAN). For this reason, MQTT 5.0 introduced the *Topic Alias* which replaces a topic by an alias for subsequent messages. We use them for the MUP protocol optimization.

4.4 Discovery and Bootstrapping

The discovery mechanism assumes working connectivity on the lower protocol layers as 6LoWPAN and WLAN. The device discovery works with 6LoWPAN quite simple: the default IPv6 network address *fd00::1* from the border router is also used for the MQTT broker address. Depending on the use case and devices, security mechanisms for 6LoWPAN can be applied on Layer 2 and 3 as mentioned in 2.3.1.

Sensor boards with WLAN access must be pre-configured, e.g. the WPA2. However, the MQTT broker address can vary. To achieve a dynamic configuration anyway, the DHCP Options [86] and DHCPv6 [245] can be used. Some sensor boards (e.g. ESP32, Arduino) provide a configuration interface through Bluetooth or WLAN hotspot. An initial configuration for WLAN access and MQTT broker can be made in this way.

The devices must use the bootstrapping topics (see Section 4.4.1) and provide their device descriptions (see Section 5.4).

After the network connection is established between a device and MQTT broker, the question of trust must be resolved. The authentication and authorization mechanisms depend on the security strategy and discussed in Section 7.

The discovery mechanism is shown in Figure 4.2. These steps are executed only **once** unless device capabilities or network has changed, or network connection was lost:

1. device → MQTT-Broker: The device publishes the device description as the payload to the MQTT broker's *Topic yang/config/create*;
2. MQTT-Broker → bridge: the NETCONF-MQTT bridge as the MQTT client receives this publication message;
3. The bridge parses the device description using ontology knowledge, adds RPCs to the NETCONF server and generates a YANG model;
4. A user obtains the YANG schema including all possible RPC calls through a web-interface or a (command-based) NETCONF client, see RPC example in Listing 4.1.
5. The generated YANG does not contain the MQTT topics. These are cached by the bridge for the mapping on a executed RPC call.

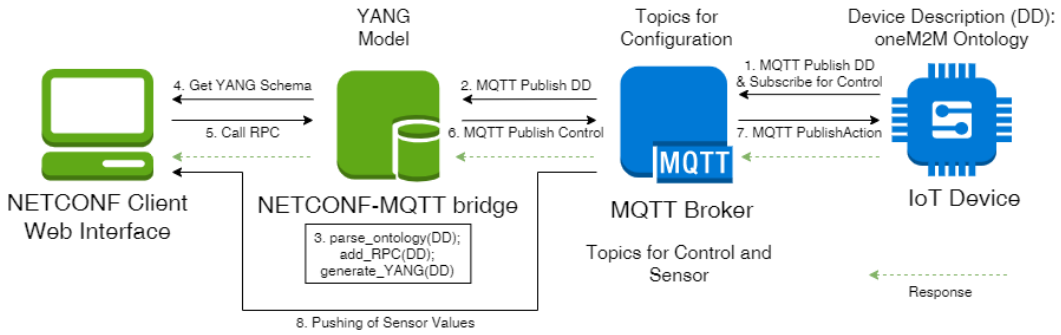


Fig. 4.2: Bootstrapping in the MYNO framework

At the end of this process, the bridge has a model of the operational state because managing constrained devices via RPC calls reflects the state of the IoT domain.

The input and the output parameters can be defined for a RPC call in the YANG model. A typical parameter for the input is the UUID. Even the UUID is already a part of the topics, the UUID in the message ensures unique addressing.

The output parameter can be a state achieved by the device through an actuator. Such states can be defined for all devices in a generic way. For example, Eclipse Kura define them like HTTP states 200, 404, etc.; and the oneM2M Service Layer Core Protocol [345] specified a range of Response Status Codes from 1000 till 6xxx. For simplicity, we defined three basic states: OK if RPC call was successful and the actuator state has changed; NOOP if the actuator state did not changed; ERROR if something went wrong.

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="urn:uuid:7d6da1c2-6c90-4243-bd9c-5523c8cc6890">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
  <![CDATA[module mqtt-LED-LAMP {
  namespace "https://www.cs.uni-potsdam.de/bs/research/myno/LED-LAMP";
  prefix LED-LAMP;

  rpc funcSwitchOn {
    description
      "Switches the led on";
    input {
      leaf uuidInput {
        description
          "Target UUID for request";
        type string;      }
    }
    output {
      leaf retval {
        type enumeration {
          enum ERROR {
            description
              "error";   }
          enum NOOP {
            description
              "nothing to do";   }
          enum OK {
            description
              "successful";   }
        }  }  }
  }
]]></data>
</nc:rpc-reply>
```

Listing 4.1: Example of generated YANG model by the NETCONF-MQTT bridge

A survey of secure IoT bootstrapping was made in [347]. It shows that bootstrapping is more than discovery. The discovery mechanism must be repeatable in case something went wrong during discovery. Further, it should be possible to update or delete a device from the network configuration. This results in a bootstrapping process and its CRUD operations: create, retrieve, update, delete. We use the term *retrieve* instead of *read* because it better expresses the process of retrieving the wanted device UUID from the network configuration.

Here are some examples of situations where the CRUD operations are necessary. CREATE operation is used to add a device to the network configuration (see discovery).

In case the first configuration is failed, the bridge does not get the device description (no matter for what reason). Therefore the device has to be notified whether this CREATE operation worked and, if necessary, repeat it. RETRIEVE operation is used when a device has to check whether it is already known to the network (e.g. after sleep, and also before create, update, delete). UPDATE operation is necessary when a device is already configured in the network but its device description has changed (e.g. due to firmware update or hardware change) and the device will send the new device description. DELETE operation is used when a device has to remove itself (actively) from the network.

However, bootstrapping is a whole process, not only CRUD operations. Here are requirements for bootstrap, including consequences for the MQTT-NETCONF bridge:

- Devices must be notified from the bridge of successful create/retrieve/update/delete operation: Response time out (upon MQTT publish) must be defined for the whole system.
- Devices must be able to repeat an operation after time out: Devices must ensure that they are known by the bridge (retrieve) before executing these operations: create, update and delete. Bridge must remember the device UUID and be able to send a response.
- A Device must be able to add/create its device description: Bridge must be able to add a new device description to the network configuration.
- A Device must be able to update its device description: Bridge must be able to replace the existing device description in the configuration.
- A Device must be able to delete its device description: Bridge must be able to remove the existing device description from the configuration.
- Bridge can send regular requests to devices to detect if the devices are still present (ping): Devices must respond to the ping, otherwise the devices will be removed after time out.

The time out, mentioned above, should be configured uniformly for the whole system to avoid inconsistency and side effects.

4.4.1 MQTT Topics in Bootstrapping

Following default MQTT topics are defined for controlling the bootstrapping process. There are two topics per operation: one for the CRUD operation and one for the respective response. The devices publish to the CRUD topic and subscribe to the response topic.

Tab. 4.1: Concept for MQTT Topics in Bootstrap process

Operation	Topic	Message
CREATE	yang/config/create	<UUID>;<MSG>
RETRIEVE	yang/config/retrieve	<UUID>
UPDATE	yang/config/update	<UUID>;<MSG>
DELETE	yang/config/delete	<UUID>

Tab. 4.2: MQTT Response Topics in Bootstrap process

Operation	Response Topic	Response Message
CREATE	yang/config/create/response/UUID	<RESPONSE_CODE>
RETRIEVE	yang/config/retrieve/response/UUID	<RESPONSE_CODE>
UPDATE	yang/config/update/response/UUID	<RESPONSE_CODE>
DELETE	yang/config/delete/response/UUID	<RESPONSE_CODE>

Alternatively, for legacy devices of our version 1.0, the topic *yang/config* was used, therefore the message should be built like *<UUID>;CREATE;<MSG>*. The device subscribes to the response topic for */UUID* and the message payload contains *<RESPONSE_CODE>*.

On the other side, the bridge should know whether the devices are still present in the network. For this purpose the PING operation should be implemented which works in the other direction, from the bridge to a device. The bridge publishes to the topic */config/ping/UUID* a message containing *<UUID>*. The bridge subscribes to the topic *yang/config/ping/response/UUID* and receive a response message with *<RESPONSE_CODE>* from the device.

The <RESPONSE_CODE> can be defined in the HTTP style (e.g. 200 (RESPONSE_CODE_OK), 404 (RESPONSE_CODE_NOTFOUND)) or in the style of oneM2M (e.g. 2000 OK, 4004 NOT_FOUND) [345].

In MQTT 5.0, the property *Response Topic* in the Publish message can be used. A MQTT Client (the Requester) publishes a Request Message to a topic. A Request Message is an Application Message with a Response Topic.

4.5 Web-based NETCONF Client

Generally, the NETCONF-MQTT bridge can be accessed by any NETCONF client, for example, a CLI netconf client. However, a web-based client provides better usability.

The requirements for the web-based interface were determined as following: Python 3 as programming language; light-weight library; graphical user interface (GUI) optimized for mobile smartphones (so-called Responsive Web Design); secure access to the NETCONF-MQTT bridge through SSH; dynamically created GUI based on the YANG model; user friendly interface with actuators and sensor data shown.

The client should provide buttons for RPC calls and show the responses if there are any. The client should subscribe to the MQTT topics of sensors and display dynamically new sensor values. Dropdown menus should be displayed for configuration of parameters for RPC calls. Incoming events should be dynamically displayed as they occur.

4.6 Device Functionalities

IoT devices can have different capabilities but there are some common functionalities which are provided by the microcontroller boards: actuators, sensors, event

triggering, configuration. The next sections describe how these functionalities are supported by MYNO framework.

4.6.1 Actuators

IoT devices can have actuators on board. A user can control them, for example switch on and off a led on a device. We describe the functionality of the NETCONF-MQTT bridge according to Figure 4.2 and the actuator scenario, which contains following steps:

5) User → bridge: A user triggers the RPC call for switch on functionality;

6) Bridge → MQTT-Broker: The bridge maps the RPC call to the MQTT message and publishes it to the topic e.g. *actuator/led/led1/UUID*;

7) MQTT-Broker → device: The device is receiving the message because it is subscribed to the associated topic before and finally switches on the led.

Afterwards, the device publishes the response to the topic *response/UUID*. The response will be propagated to the web client (dashed arrows).

4.6.2 Sensors

Besides controlling actuators, current sensor values must be published by a microcontroller board to a sink or a client application. Measured sensor values are published periodically to the MQTT Topic e.g. *sensor/brightness/br1/UUID*. There are several reasons, why sensor data should not be transported through the bridge: (i) pushing sensor data periodically is not a network configuration task; (ii) big amount and high rate of sensor data would cause overload for the bridge. Although, NETCONF protocol provides a notification mechanism but this is determined for state changes of the network configuration.

Therefore, the sensor topics are provided by the generated YANG model to a client. The client can subscribe directly to the MQTT-broker in order to pick up the sensor values, see Step 8 in Figure 4.2.

4.6.3 Events and Configurations

We assume that an IoT device itself is not context-aware. It has some sensing and actuating capabilities but it is not aware where it is located or whether, for example, the measured temperature is lower or higher than it should be. Such context-awareness can be achieved through configuration of device. The context-aware configurations on a device are thresholds for event triggering and automation rules (like "if this than that"). This decentralized approach differs from the common centralized approach where the sensor values will be analyzed in the cloud. However, such configurations have two advantages:

- to configure a threshold on a device for event triggering helps to save energy because a notification will be only sent if the threshold is exceeded;
- to save automation rules on a device makes a device more independent comparing to a centralized approach where the rules are saved in a rules engine in the cloud;

There are several challenges to overcome for configuration and event functionality on an IoT device especially if they are constrained: the configuration must be small (because of low-rates networks) and simple (because of limited computational power) but interpretable by devices (well structured condition).

We made the following assumptions:

- a device can have several configurations which can trigger different events;
- CRUD operations can be performed in these configurations;
- alarms are special kinds of events (whether an event is an alarm is dependent on an application use case);
- the relation of a configuration to an event occurs through a sensor because such sensor measures the threshold as a condition for the event (e.g. trigger an event if the temperature is over 40 degrees);
- a configuration must include min and max sensor values as well as further parameter to enable to build a condition: possible comparing operators, interval and duration.
- an interval determines how often an event must be sent after the occurrence, e.g. every 10 seconds.
- a duration determines how long an event must be sent after the occurrence, e.g. for 100 seconds.
- interval and duration parameter help to estimate whether the situation gets worse or better.

The assumption is that every sensor on a device can have just one generic event functionality with one MQTT topic where all events from this sensor will be published. The distinction between the events is made through the event name in the payload.

The automation functionality is a concatenation of configuration functionality and controlling functionality. For example, if the temperature is over 40 degrees, turn the LED light to red. Therefore, these two RPC calls just concatenated by the AND keyword.

The configuration and automation capabilities must be defined in the device description. For the bridge, they look like a special type of RPC calls with several parameters. The opposite is the event triggering, this functionality looks like the sensor functionality for the bridge.

4.7 Conclusion

The MYNO framework has to fulfill several tasks and every task claims a specific attention. The NETCONF-MQTT bridge manages the network configuration. The discovery mechanism allows devices to proceed auto-configuration in a network. Bootstrapping makes this process robust. Different scenarios (sensor, actuator, event, automation) are supported by the framework. An MQTT broker and its topics make these mechanisms working. The processing of semantic device descriptions enables the execution of these tasks.

Semantic Device Description

5

> *One size never fits all.*
>
> **— RFC 1925, Nr. 10**
> (The Twelve Networking Truths)

Ontology-driven interoperability should ensure semantic interoperability [222]. The question pursued here is: Which ontology can be used for a device description in the MYNO framework? The chapter is organized as follows. First of all, related work on semantic interoperability is presented. Then, the common IoT ontologies are evaluated in terms of suitability for device descriptions in the MYNO framework. After that, an ontology will be chosen and extended for device descriptions. Finally, an ontology for sensor data will be discussed.

5.1 Related Work: Semantic Interoperability

IETF uses their own terms for (semantic) interoperability definition, see [178] and [280]. They differ between three models: Information Model (IM), Data Model (DM) and Interaction Model (IN). IM describes the environment on a higher abstraction level, e.g. in UML. DM is a concrete data model, e.g. Management Information Base (MIB) in SNMP protocol, W3C Thing Description (TD) Things, YANG models, LwM2M Schemas, OCF Schemas etc. IN defines how data will be exchanged, e.g. on behalf of REST, Publish/Subscribe or RPC.

The intersection with our definition of interoperability layers in Section 1 is obvious. To achieve a certain level of interoperability in an IoT system, connectivity and communication protocols are not sufficient. The meaning and structure of exchanged data and the context of this data exchange are necessary to be defined.

Barnaghi et al. [23] surveyed semantics at different levels of the IoT: services and applications; data processing; devices, resources and networks; real world objects (things); and domain knowledge. Their conclusion was that IoT and using semantics in the IoT is still in its early days. They identified research challenges like dynamicity and complexity, scalability, semantic service computing for IoT, distributed data storage/query. One of the first semantic applications was the representation of observation and measurement data from sensor networks defined by Open Geospatial Consortium (OGC) and the early W3C's SSN-XG [342] ontology.

On the question, why sensor-based devices should describe their data semantically, the researchers from Ireland have a clear opinion [284]. They proved that the sensor data from weather stations is in the best case, immediately delivered as Linked Data (RDF-based data available on the Web and interlinked). This allows to answer semantic data queries easily, for instance: How is the average temperature today in New York?. The weather stations deliver their semantic data with SSN-XG [342] ontology. Additionally to sensor data, data for spatial and time is added. The

researchers implemented an intermediate layer which categorized delivered data and stored it as Linked Data. The above mentioned questions can be answered.

The authors in [22] describe a semantic model for representation of heterogeneous sensor data. They use frameworks from the Semantic Web Community. This work describes a sensor data ontology which is based on the Sensor Web Enablement (SWE) and SensorML [45].

Mikhaylov et al. [233] ask the question, how data can be exchanged in WSNs where resources are limited. With the result that SensorML and SWE can be applied. Sensor Model Language (SensorML) is a XML-based language from OGC, which defines syntactical and (later semantical) processes and components for measurement and thereafter transformation of data.

Sheth et al. [360] propose a Semantic Sensor Web (SSW) where sensor data with semantic meta data increase interoperability and offer context-information situation-depending. Their meta data especially refers to spatial, time and topic (theme) semantic data.

Semantic interoperability for the Web of Things was surveyed in [248] with the conclusion, that deployment of semantic technology is still at the beginning and to achieve semantic interoperability at scale, requires collaboration across standards organizations, consortia, alliances, and open source projects. Therefore many organizations work on semantics for IoT. For example: AIOTI, ISO/IEC JTC1, ETSI, oneM2M and W3C collaborate on two joint white papers on Semantic Interoperability targeting developers and standardization engineers [221, 222]. Also, European Research Cluster on the Internet of Things (IERC) published a report about semantic interoperability with best practices and recommendations [223].

Shi et al. [361] provide an overview about data semantization in the IoT. They conclude that ontology modeling has become the most pervasive technique so far: "Every entity, context, user and activity can be modeled through ontologies, with strong expressivity, expansibility and reasoning ability".

5.1.1 Semantic Web of Things (SWoT)

The term Semantic Web of Things (SWoT) is not formed by W3C but by researchers like Amelie Gyrard [132] who developed the Machine-to-Machine Measurement (M3) framework[32]. SWoT describes basically an approach to develop WoT applications by using semantics. M3 framework consists of 7 components: Generating semantic-based IoT applications; Reusing domain knowledge, Semantic Web Best Practices, Interpreting IoT data, Visualizing IoT ontologies, Interoperable IoT Data & Domain Knowledge, Securing IoT applications. However, the M3 framework is not widely disseminated in the industry.

Seydoux[349, 350, 348] researches on semantic data lowering approach to make constrained devices semantically interoperable. This approach reduces the semantics on a constrained device and enriches it on a non-constrained device.

[32] https://github.com/gyrard/M3Framework

Wu et al. [419] proposed a semantic Web of Things (SWoT) framework for CPS (SWoT4CPS). SWoT4CPS provides a hybrid solution with both ontological engineering methods by extending SSN and machine learning methods based on an entity linking (EL) model.

The research group in [176] analyzed the use of semantics in the IoT. They see Semantic Web of Things (SWoT) as the next challenge after the IoT and Web of Things. The goal of the SWoT is „to integrate information which is semantically rich and easily accessible into the physical world, thus connecting smart objects and digital entities“. They analyzed several IoT standards and came to conclusion that the IoT and M2M markets are strongly fragmented. Additionally, they see Cloud and Big Data as a key for semantic data processing in the IoT. They recognize the necessity of semantic interoperability but do not suggest a solution..

Ruta et al. researched the use of semantics for CoAP Protocol in [307] and [306]. They describe in [306] a SWoT Framework which is based on the backward compatible extension of CoAP Protocol and supports non-standardized inference-services for semantic matching. Therefore, they extend the CoAP Protocol with further attributes for semantic data and additionally introduced an algorithm which finds appropriated sensors.

The SPITFIRE project[277] integrated sensors as a part of the Linked Open Data (LOD) Cloud. They showed their vision and architecture of the Semantic Web of Things (SWoT). However, the project is not maintained anymore.

Rojas et al. [302] propose a related approach to use semantics with constrained devices. Their constrained device provides a CoAP server which serves a Hydra API [205] documentation, as well as responds to client requests concerning its capabilities. This server only exposes the parts of its documentation and refers to a common ontology. They validated their implementation on Arduino UNO, Class 0 device. The sensor values are buffered and the output JSON-LD payload is generated using a Service Template. They use the CoAP Block-wise Transfer option. Even though they achieved to parse small pieces of JSON-LD data in the Arduino, there are some shortcomings on this approach. Their ontology does not reuse an existing ontology. The authors assume that the ontology is shared with all other distributed servers. The ontology defines the capabilities of devices but does not describe the technical details, e.i. the CoAP links for access.

5.1.2 Device Descriptions

The term "Ontology-based Device Descriptions(ODDs)" was introduced by Dibowsky [84, 83]. He introduced a hierarchical ontology layer architecture for device descriptions in home automation field. He focused on the semantics itself, namely, the Electronic Device Description Language (EDDL). EDDL is an international standard IEC 61804-3 and is used in process- and factory automation for protocols like PROFIBUS, HART and FOUNDATION field bus. However, his research field is the industrial IoT and not the WSN-based IoT.

An e-Health system based on SSN using IETF YANG was designed in [179] to address the device interoperability issue. They used IETF YANG for modeling the semantic e-Health data to represent the information of e-Health sensors. They developed an ontology for e-Health data that supports different styles of data formats. The

ontology is defined in YANG for provisioning semantic interpretation of sensing data in the system by constructing meta-models of e-Health sensors.

Recently, IETF is working on a draft for Semantic Definition Format (SDF) [193] for data and interactions of things. Taking a closer look, it uses similar terminology and structure as W3C WoT Thing Description [184]. Obviously, there is a demand for such semantic device descriptions.

5.2 Ontology Choice for Device Description

We have chosen to use an ontology-based device description, instead of plain JSON. The main reasons are: (i) model-driven approach to describe common device capabilities; (ii) the meaning of capabilities, so called semantics are included; (iii) Semantic Web Standards are used to formalize, format and query the descriptions. Additional reasons for using an ontology are: sharing a common understanding of the structure of information among system components; enabling reuse of domain knowledge; making domain assumptions explicit; separating domain knowledge from the operational knowledge; analyzing domain knowledge.

The next step is to develop an ontology for device description. Noy and McGuinness proposed Ontology 101[262], a simple but practical guide for building an ontology. This methodology consists of seven steps:

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumerate important terms in the ontology
4. Define the classes and the class hierarchy
5. Define the properties of classes
6. Define the facets of the properties
7. Create instances

For the domain and scope determination of the ontology in the first step, we have to answer the following questions:

- *Which domain should be covered by the ontology?* The horizontal IoT device domain which is generic for all devices but extendable up to vertical domains.
- *What should the ontology be used for?* For network configuration management of IoT devices.
- *What types of questions should be answered by the knowledge represented in the ontology?* We have following types of questions: Which devices are connected? How are they identified? Which sensor and actuator capabilities do they have?
- *Who will use and maintain the ontology?* The ontology should by produced and maintained by the devices manufacturer. The developers of the MYNO framework integrate these device descriptions.

Moreover, *competency questions* [124] have to be formulated. These are some of the *competency questions* which arise for our device descriptions:

- *Which properties of the device should be considered for modeling?* Identification of a device via UUID, short description of the device and its capabilities, technical details how to access these capabilities i.e. how to send requests to the device and receive responses from the device.
- *How is the parsing to a YANG model enabled?* Since the NETCONF-MQTT bridge will be used to parse device descriptions, we can consider the JSON profiles originally used in [327].
- *How is the bridging to other protocols supported?* An ontology must contain data about application protocol i.e. MQTT Topics.
- *Which common classes and properties do IoT devices have to enable inference?* This question will be answered by the concept of a Virtual Device.
- *How can the chosen ontology stay small, in terms of the amount of classes and properties?* This criteria must be considered by the choice of ontology. Otherwise, a compression mechanisms should be taken into consideration.

The *competency questions* will be answered during the modeling in the next steps. The second step of Ontology 101 considers reusing existing ontologies. We surveyed the existing IoT ontologies by the beginning of the year 2017, trying to answer these competency questions. At that time, some of the related IoT ontologies, mentioned in the Section 5.3, didn't even exist. Hence, the choices were limited. The results are summarized in closing of this overview.

5.3 IoT Ontologies

Many ontologies are developed for the IoT domain and especially for interoperability purpose. For example, the Linked Open Vocabularies (LOV)[33] repository counts by May 2020 about 30 vocabularies tagged with "IoT". Some of them are outdated like the SPITFIRE ontology [277], some of them were developed by research projects like FIESTA-IoT Ontology [2], and some were developed by standardization organizations like SSN ontology [138].

Additionally, different IoT ontologies serve various purposes. In this thesis, ontologies are used to serve two interoperability purposes: (i) to describe IoT device capabilities enabling network configuration management and (ii) to describe sensor data.

Related work was done by VTT Finland [191] and the University of Toulouse [218, 349] serving as an example for the primary purpose. Konstantinos and Katasonov [191] developed an ontology for the automated deployment of applications in heterogeneous IoT environments. They layered several ontologies: IoT entities layer, IoT entities' alignment level, reusing the DUL and SSN ontology. However, the proof of concept is limited to the semantic evaluation of the designed ontology.

Alaya et al. [218] defined the IoT-O ontology as an extension of the oneM2M standard to support semantic data interoperability. Several ontologies were merged (i.e. DUL, SSN, MSM, HREST, ACT, QUDV, TIME ontologies) and the missing concepts relevant for IoT like a thing, actuator, actuation, or manager were defined by the IoT-O ontology. The proof of concept was made by appliance of the ontology to the

[33] `https://lov.linkeddata.es/dataset/lov/vocabs?&tag=IoT`

oneM2M system. The IoT-O is a comprehensive ontology towards the right direction but limited to the horizontal domain.

In [349], the authors refined the initial IoT-O ontology [218], proposing stronger modularization: Acting module based on the Semantic Actuator Network (SAN) ontology); Sensing module based on the SSN; Lifecycle module based on Lifecycle ontology; Service module based on the HREST, MSM and WSMO-Lite ontologies; Energy modules based on the PowerOnt. However, the implementation of the IoT-O ontology was applied to a robot in a home automation use case and not to a constrained device in WSN.

SensorML 2.01 from OGC [45] is a model language describing the semantic meaning of sensors, actuators and processors. Janes proposed an IOTDB vocabulary based on JSON which describes the Things in the IoT (i.e. facets, purposes and units) [173]. Both vocabularies do not use Semantic Web Standards and therefore do not meet our requirements.

FIESTA-IoT Ontology[34] and as a part of this, M3-lite Taxonomy, are developed within the EU H2020 FIESTA-IoT project. FIESTA-IoT Ontology aims to achieve semantic interoperability among heterogeneous sensor testbeds in the data format of observations. M3-lite Taxonomy is a taxonomy for a variety of quantities (i.e. physical and environmental phenomenons), unit of measurements, different types of sensor and different types of domain of interests. FIESTA-IoT reuses and interconnects existing ontologies. Therefore, these ontologies are used for description of produced observations in a testbed and not appropriate for our use case of device descriptions.

There are also many ontologies developed for vertical domains. For example, the Brick Ontology[35] for Smart Building. Another approach is the M3 ontology [133] which is designed as a cross-domain ontology.

Recently, some of the ontologies describing a device were specified by standardization organizations like W3C Thing Description, oneM2M, SAREF Ontologies, and Schema.org are introduced in detail in the following subsections.

For the second purpose, describing sensor data by an ontology, several ontologies were defined such as SSN ontology and SenML. They are also introduced in the following subsections. Also, a transformation from SenML to RDF [376] was applied to connect IoT Sensors to knowledge-based systems.

5.3.1 W3C WoT Thing Description

The Thing Description (TD) [184] is the machine-understandable metadata and "is self-descriptive, so that Consumers are able to identify what capabilities a Thing provides and how to use the provided capabilities." The First Public Working Draft was published on 14 September 2017. The Thing Description is based on an ontology [36] and use the JSON-LD syntax format.

Each thing must have such Thing Description. This Thing Description includes statements about interaction *affordances* which "refers to the perceived and actual

[34] http://ontology.fiesta-iot.eu/ontologyDocs/fiesta-iot/doc
[35] https://brickschema.org/ontology/
[36] https://www.w3.org/2019/wot/td/v1

properties of the thing... that determine just how the thing could possibly be used" (e.g. switch on and off the light), data schemas (referred ontology), security configuration (control access to the interaction affordances, secure management of data), protocols bindings (mapping from interaction affordances to concrete messages of a specific protocol). The TD core vocabulary is shown in Figure 5.1 including three kinds of affordance for property (state of the Thing), action and event.

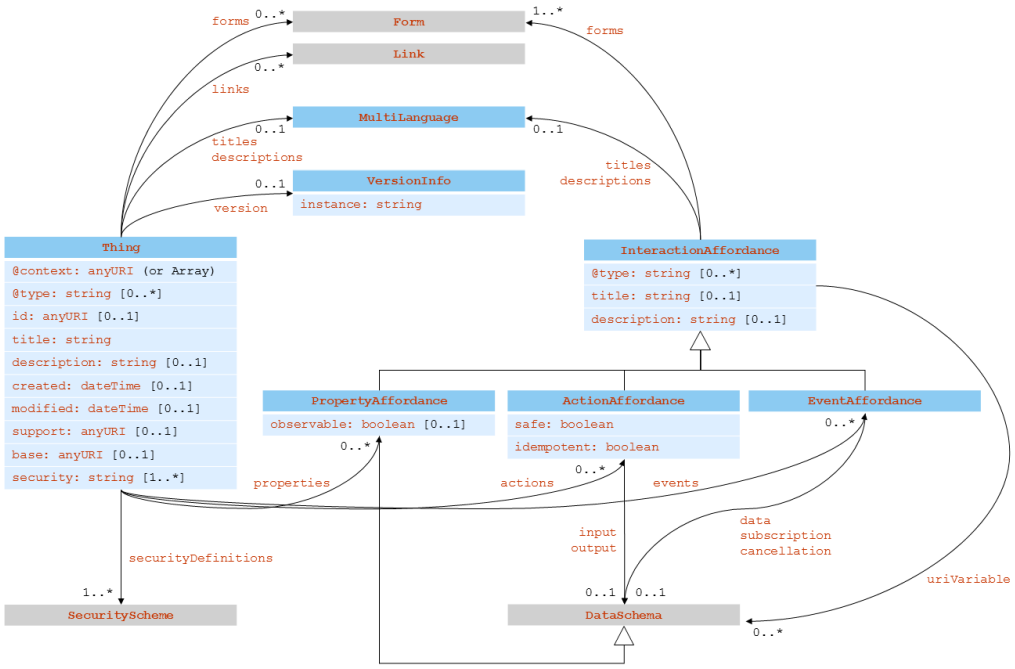


Fig. 5.1: WoT Thing Description (TD) core vocabulary [184]

Listing 5.1 shows an example of the TD for a lamp instance with protocol binding for HTTP. The MyLampThing is the title and has an URN but no type defined. The input and output parameters for HTTP operations are not defined further.

Discussion
Thus, in the opposite of the Web Thing Model above, the WoT Thing Description is based on a model defined with semantic web standards. Such ontology-based description can be queried by SPARQL and offers also other semantic web features like reasoning. The Thing Model is a base model. For semantic interoperability is must be extended with domain-specific vocabulary like Schema Extensions for IoT or the W3C SSN ontology (see Section 5.3). However, the WoT TD is missing many definitions (e.g. type definition like a type of saref:LightSwitch ontology, input and output parameters for operations, etc.) about a Thing, that are necessary for mature development.

The IoT-Lite Ontology [164, 26] which represents IoT resources, entities and services stays as a W3C member submission. Two years later, W3C published Web of Things (WoT) Thing Description(TD) [184] as a First Public Working Draft. The WoT TD is encoded in JSON-LD [206] format which is simple like JSON but has more semantic

```
{    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"} },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{"op": "readproperty",
                       "href": "https://mylamp.example.com/status",
                       "htv:methodName": "GET"}] } },
    "actions": {
        "toggle" : {
            "forms": [{"op": "invokeaction",
                       "href": "https://mylamp.example.com/toggle",
                        "htv:methodName": "POST"}] } },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{ "op": "subscribeevent",
                       "href": "https://mylamp.example.com/oh",
                   "subprotocol": "longpoll"
            }] } } }
```

Listing 5.1: Thing Description Example for a Lamp [184]

expressiveness. It is an interaction model with WoT's Properties, Actions, and Events, a semantic scheme to make data models machine-understandable, and features for Web Linking to express relations among things. At the time of our implementation, the WoT TD was a very first draft and immature. Although, by now the WoT TD ontology was developed further [184] and became W3C Recommendation by April 2020.

5.3.2 SSN Ontology

The vision of Semantic Sensor Web (SSW) [360] is to diminish the shortcomings of SWE from OGC. The OGC standards (e.g. SensorML [45] and O & M [73]) provide syntactic interoperability but do not address semantic interoperability. In SSW, sensor data is annotated with semantic metadata like spatial, temporal, and context.

Initially, the W3C Semantic Sensor Network Incubator group (the SSN-XG) worked on a Semantic Sensor Network Ontology (SSNO) [341, 342, 69], also supported by the OGC community. This ontology is built around an ontology design pattern called the Stimulus Sensor Observation (SSO) pattern [174]. The SSNO was designed as a base for heavy-weight ontologies in the Semantic Sensor Web as well as addressing light-weight semantics requested by the Linked Data community. The SSO was also aligned to the Dolce-Ultralite upper ontology (DUL). The usage experience and the development of Web of Things required a re-design of SSNO ontology.

The new SSN [138, 137, 175] was a joint W3C-OGC project. The SSN is based on a revised and expanded version of the SSO pattern, namely the Sensor, Observation, Sample, and Actuator (SOSA) ontology. SOSA is the central building block and puts more emphasis on light-weight use. SOSA can be used standalone because of the modular approach of SSN (see figure 5.2). The scope of the ontology was extended by classes and properties for actuators and sampling. An optional alignment via the SSN-DUL is provided.

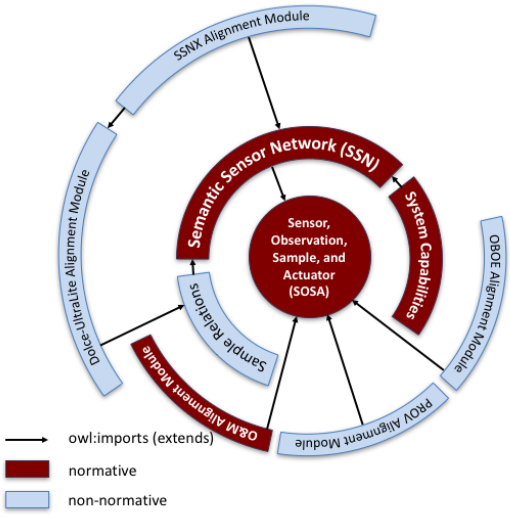


Fig. 5.2: The SOSA and SSN ontologies and their vertical and horizontal modules [138]

The SSN ontology includes the modules System, System Property, Feature, Condition (blue color in Figures 5.3, 5.3). The SOSA ontology considers modules Observation/Actuation/Sampling and Result (green color in Figures 5.3, 5.3). The SSN ontology also extends modules Deployment and Procedure as well as Observation/Actuation/Sampling. The SSN System Capabilities Module and Sample Relations Module are called *horizontal segmentation*.

The SSN ontology can be extended by further ontologies to describe location (e.g. by GeoSPARQL [276]), quantity values and unit of measures (e.g. by Quantities, Units, Dimensions and Data Types Ontologies (QUDT) [369] and the Ontology of Units of Measure (OM 2). The SSN ontology can be aligned (mapped) to the O & M model by OGC, W3C provenance (e.g. PROV-O [319]) as well as the upper ontology DUL. This alignment is also called *vertical segmentation*.

We modeled an evaluation example with SOSA, shown in Listing 5.2. This excerpt illustrates the observation with descriptions on sensor and observed property, feature of interest and measured result.

During the modeling we experienced some difficulties. First, SSN intentionally does not specify quantities and measurement units. Other special ontologies for such domain-specific terms are necessary. The Ontology of units of Measure (OM) 2.0 [298] seemed to be promising for our use case.

The second question was, how to model an *ObservableProperty*? An *ObservableProperty* is the temperature in our example but it could be also humidity or brightness as well as height and depth. In case of many devices and observations, the *Generic Instances* (e.g. Temperature) of the *ObservableProperty* are better to handle than *Specific Instances* (e.g. room1temperature) for several reasons: (i) avoiding name conflicts or many creative names with the same meaning; (ii) improving aggregation/reasoning using *Generic Instances*; (iii) re-using ontologies with *Generic Instances* like OM 2.

The third question was, how to model a measured result? The result of a sensor measurement is not only the value but also the kind of unit and data type. SOSA

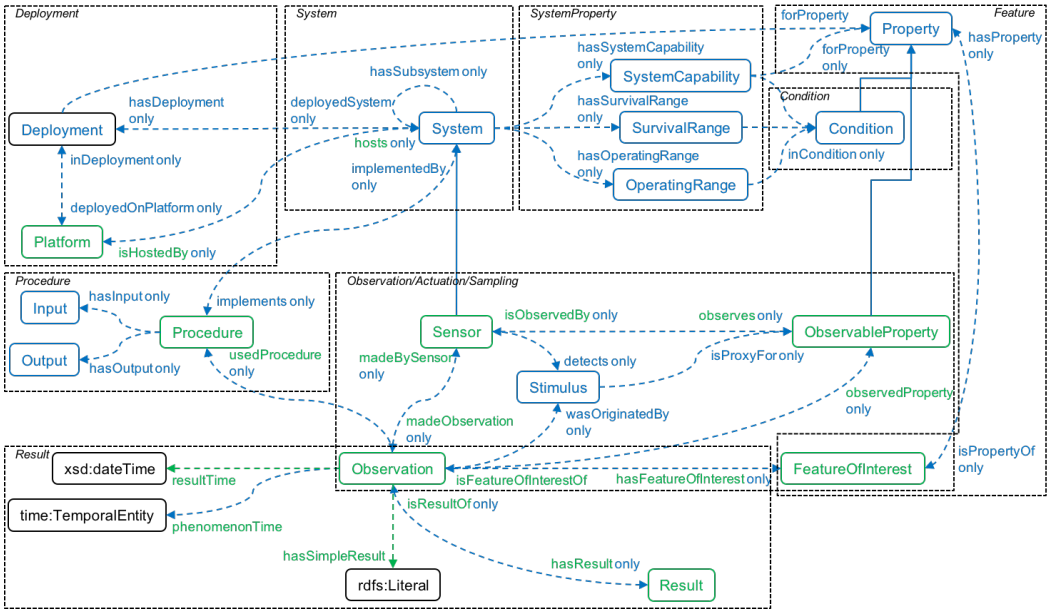


Fig. 5.3: Classes and relationships involved in Observation (SOSA/SSN) [138]

stops at the *Result* class and does not define the details. For this purpose we also used the OM 2 ontology and had to inherit the result from the *Measure* class of OM 2.

The next question was about the *Procedure* class. The used procedure (i.e. how to make an observation) can be described as well as its output. The fact is, once a sensor-equipped device is installed, the procedure never changes. Why should the every observation carry this information? It is sufficient to send this information once with the device description. This will also save space in data packets which is important for constrained devices.

The last question was about platform and system capabilities module defined by SSN. These describe the deployment and system properties which usually do not change during the lifecycle of the device. This information can be also sent only once upon a deployment. This information can also only be sent once upon a deployment.

Discussion
SOSA and SSN ontologies build a base for semantic description of sensor data. Nevertheless, further ontologies are required to define domain specific knowledge but some ontologies can be broadly reused like ontologies for location, approver, quantities and units. SOSA and SSN define even more details than our dimensions require, see Chapter 5.5. This is questionable transmitting these static data with every observation. For aggregation/reasoning of sensor data there are some best practices to follow such as using *Generic Instances*. The new SSN ontology is improved in terms of modularization and alignment as well as adjusted to roles in the IoT (actuators, sampling). The ontology is detailed enough to be used standalone or can be extended upon requirements. However, to describe a single observation requires a lot of information which can be redundant (see Section 9.2).

The new version of the Semantic Sensor Network Ontology (SSN) [138] is defined by W3C and OGC together as a vocabulary for sensors data description and observations

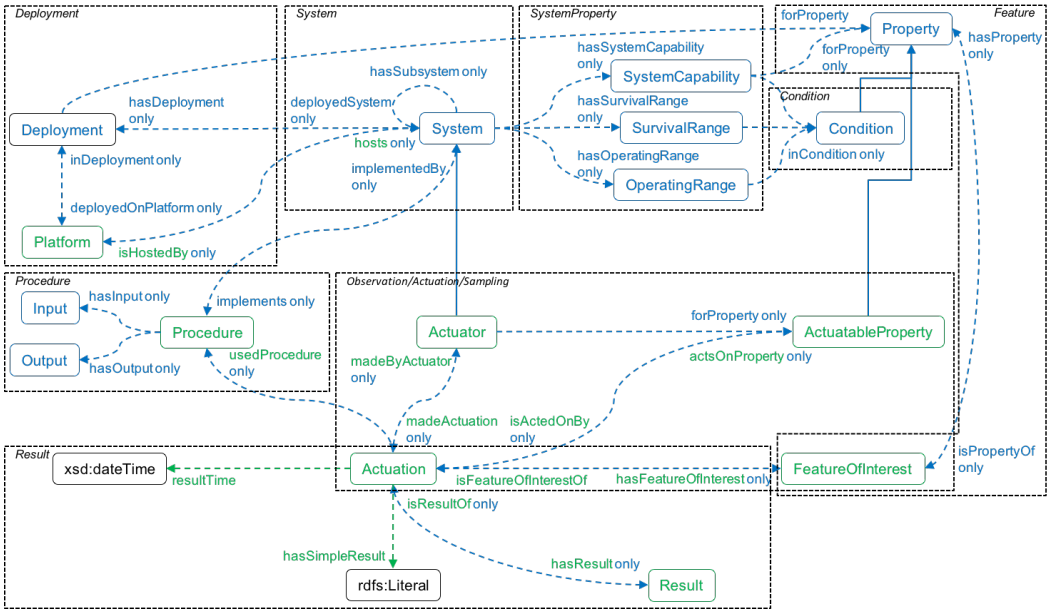


Fig. 5.4: Classes and relationships involved in Actuation (SOSA/SSN) [138]

processed by sensor networks. Therefore, SSN describes the collected sensor data but not the provided device capabilities how to access devices and collect the data.

5.3.3 oneM2M Base Ontology

The oneM2M Base ontology [24] (further referred as oneM2M ontology) is intended for interworking with non-oneM2M systems and shall provide syntactic and semantic interoperability. The basic concept of the ontology-based interworking is introduced in Section 3.1.3. The oneM2MM ontology is shown in Figure 5.5. The central class is the *Device* which is subclassed from *Thing* and can have *ThingProperties*. The ontology provides two views: human-understandable and machine-interpretable. The human-understandable meaning is provided by the class *Function*: what the device does. The Function is subclassed by *ControllingFunction* and *MeasuringFunction*. The class *Command* represents an action that can be performed to support the Function. The Function is exposed by the class *Service* which is a machine-interpretable representation of a Function to a communication network that "makes the Function discoverable, registerable, remotely controllable in the network". The Service is dependent on the technology of the network. The Service has an *Operation* which is the machine-interpretable exposure of a human-understandable Command to a network. Further, a Service has an *OperationInput* and *OperationOutput* parameters.

Discussion
The oneM2M ontology is a base model and not sufficient for direct use and must be extended (see Section 5.4). The oneM2M ontology is represented by OWL-DL which ensures that queries are decidable. However, the ontology uses existential, universal and cardinality restrictions which make such queries more complex (see Evaluation 9.2). On the other side, no restrictions at all are also drawbacks (see Evaluation 9.2).

```
{"@context": {
  "owl": "http://www.w3.org/2002/07/owl#",
  "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
  "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
  "xsd": "http://www.w3.org/2001/XMLSchema#",
  "om-2": "http://www.ontology-of-units-of-measure.org/resource/om-2/",
  "sosa": "http://www.w3.org/ns/sosa/",
  "myno": "https://www.cs.uni-potsdam.de/bs/research/myno#"
  },
...
{
      "@id": "myno:myobservation30102018",
      "@type": [
        "owl:NamedIndividual",
        "sosa:Observation"      ],
      "sosa:hasFeatureOfInterest": {
        "@id": "myno:room21"    },
      "sosa:hasResult": {
        "@id": "myno:measuredResult" },
      "sosa:madeBySensor": {
        "@id": "myno:mytempsensor" },
      "sosa:observedProperty": {
        "@id": "om-2:Temperature" },
      "sosa:resultTime": {
        "@type": "xsd:dateTime",
        "@value": "2018-06-20T21:49:18+00:00" },
...
}
```

Listing 5.2: Observation example based on SOSA in JSON-LD format

oneM2M Ontology-based Interworking

oneM2M system can interact with non-oneM2M system via Interworking Proxy Application Entity (IPE). There are several specifications for specific interworking with LwM2M, OCF, 3GPP, etc. For all other technologies, the ontology-based interworking [267] can be used. For this interworking the oneM2M's Base Ontology [24] in OWL language is defined. The Device Information Model of the interworked devices must be provided as an ontology derived from the Base Ontology. The IPE creates proxied devices as oneM2M Resources (e.g. <AE>, Application Entities) in the oneM2M Solution that can be accessed by communicating entities (e.g. oneM2M Applications) in the usual way. Figure 5.6 shows an example how a light switch implemented as a ZigBee device is abstracted as oneM2M device according to the Home Appliance Information Model (HAIM). Both types of Services expose a Function "On Off Function" and a Command "On Command" which is e.g. described in the SAREF ontology.

Such ontology-based Device Information Models are mapped to the oneM2M Resources by IPE. IPE can create XSD files for mapping of device types, service types and operation types, according to the class definitions (sub-classes of class:Device, class:Service, class:Operation) in the ontology. Rules for creation of XSDs from ontologies are provided by the oneM2M specification. IPE in an ontology-based interworking needs to communicate with other systems either in a RESTful communication style or in a procedure call (RPC) style. The interaction between IPE and interworked device can be better modeled using Operations and their Inputs/-Outputs.

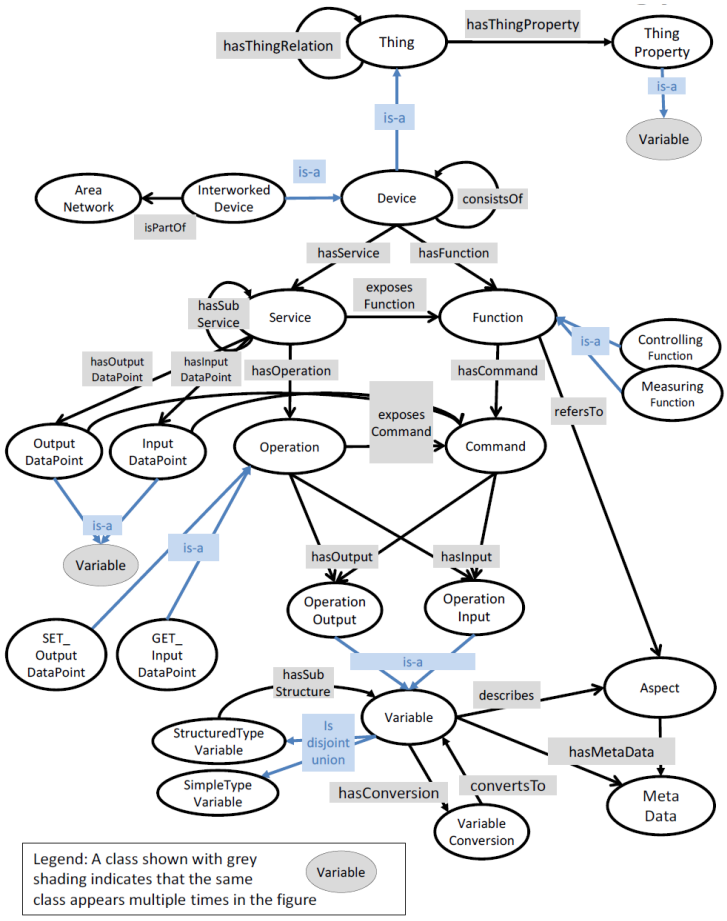


Fig. 5.5: The oneM2M Base Ontology [24]

Discussion
The ontology-based interworking by the oneM2M is provided to connect non-oneM2M systems which do not have specific interworking. The oneM2M's Base Ontology is used as an interface between those systems. However, so called Device Information Models which are based on this ontology will be mapped by IPE to the oneM2M resources and are not described directly by the oneM2M resources. Another point is, that in the whole architecture only CRUD operations are defined. This is a very generic approach and might not be sufficient in practice.

5.3.4 SAREF Ontology

The Smart Applications REFerence (SAREF) ontology is free available [78]. SAREF is intended to enable interoperability between solutions from different providers and among various domains. SAREF shall use the ETSI M2M Communication framework (see Section 3.1.4).

SAREF ontology is based on the following principles: reuse and alignment, modularity, extensibility, maintainability. Figure 5.7 shows the central OWL classes of the SAREF ontology. A washing machine is used as an example for a *device*. The washing machine has start and stop *functions*. The device acts upon a ON/OFF/STANDBY *state*. A ON/OFF *command* is a directive that the washing machine shall support to perform its function. A `SwitchOnService` *service* is a representation of a function to a

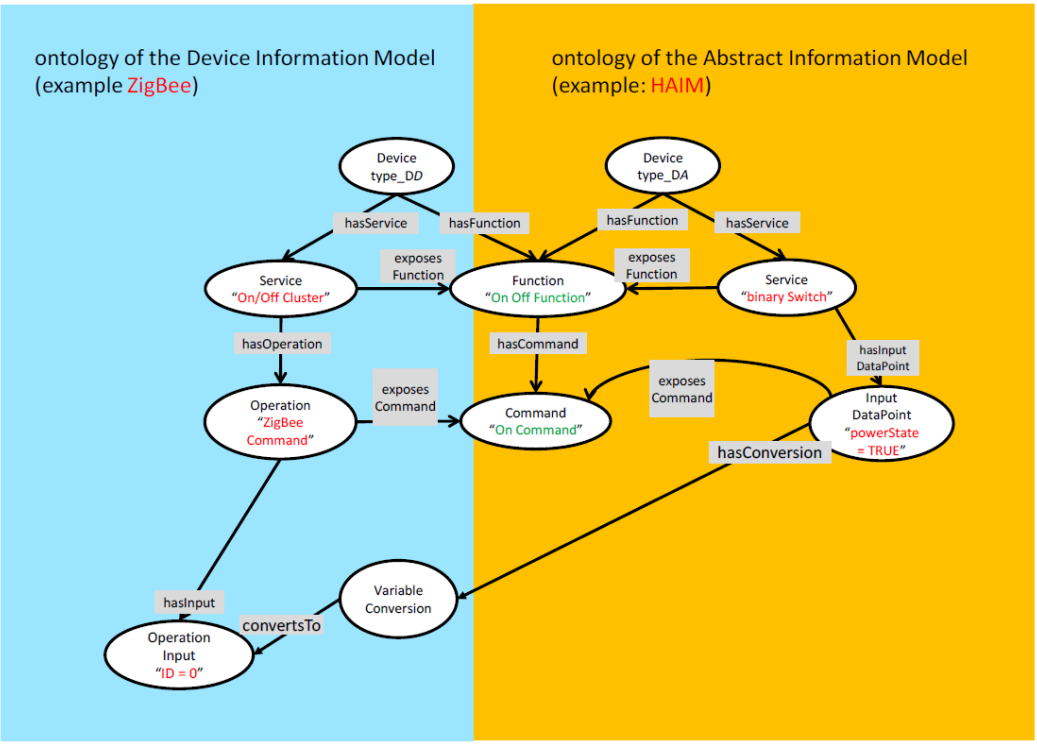


Fig. 5.6: oneM2M Ontology based Interworking with Device Abstraction [267]

network that makes the function discoverable, registerable, remotely controllable by other devices in the network. The washing machine is designed for the washing *task*, seen from a user perspective. A *property* is anything that can be sensed, measured or controlled in households. A *feature of interest* represents any real world entity from which a property is measured. A *measurement* and *unit of measure* represent a result of a measurement function. A *commodity* is a marketable item like electricity, gas, water, oil. A *profile* is "a specification associated to a device to collect information about a certain property or commodity (e.g. energy or water) for optimizing its usage in the home/building in which the device is located" [78]. An example of a profile is the Power Profile defined in the SAREF4ENER extension.

The SAREF ontology provides more details than the oneM2 Base Ontology. For example, several basic types of devices are defined as subclasses of device, as shown in Figure 5.8. Also, functions (Figure 5.9), commands and states have finder granularity.

The SAREF Ontology and oneM2M Mapping [368] is provided using owl:equivalentClass (see Figure 5.10) and owl:equivalentProperty relationships for: classes as Device, Service, Function, SensingFunction, ActuatingFunction, Command; object properties as offers, hasFunction, represents, hasCommand, consistsOf.

Discussion

The SAREF ontology is taking the right direction considering finer granularity on classes. As we will see later (Section6), this is a key for semantic inference and aggregation. Several domain-specific extensions are also defined for SAREF which can be used in the vertical IoT domains. As an improvement, some new concepts (profile, commodity, time, measurement) and some shortcuts are introduced by SAREF in the opposite to the oneM2M ontology.

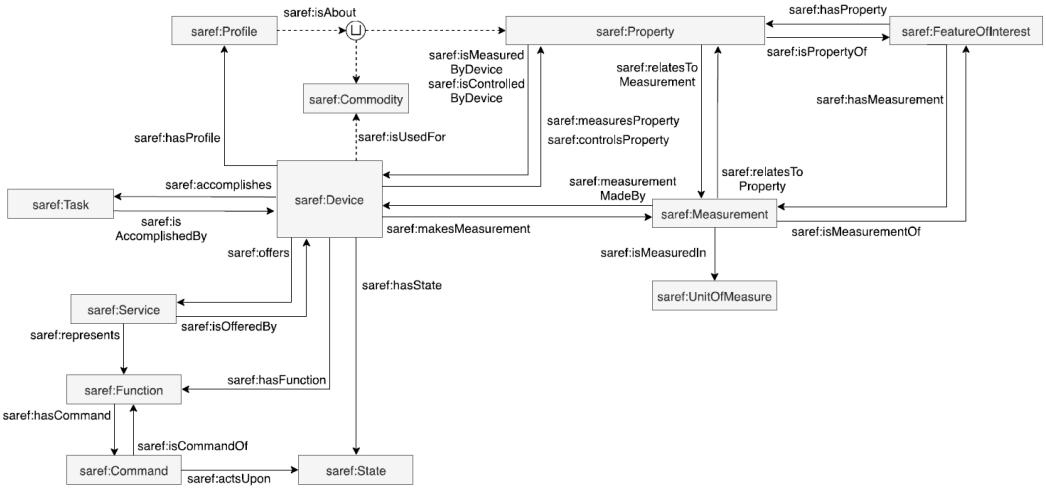


Fig. 5.7: Overview of the SAREF ontology classes [368]

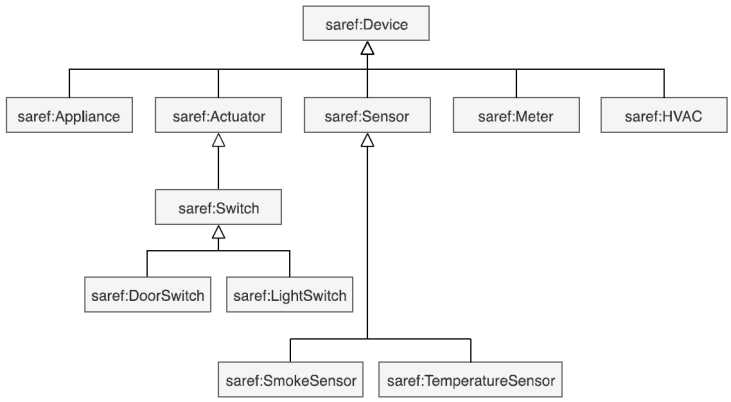


Fig. 5.8: SAREF Types of Devices [368]

5.3.5 Bluetooth Sensing Profile

Bluetooth version 5.0, also called Bluetooth Low-Energy (BLE), was optimized for IoT. Especially technical details like mesh-networking, more bandwidth or range, bigger data packets, more channels for broadcast, were improved for the usage on small devices. Bluetooth Profiles define *services* and *characteristics*. Services are human-readable specifications of a set of characteristics and their associated server behavior. Characteristics are attributes with Handle, Type, Value, Permissions; e.g. a Device Name characteristic. Characteristics is data with known schema and labeled with UUID. Characteristics have a computer-readable format and are reusable.

Most Bluetooth profiles are based on the Generic Attribute Profile (GATT). There are 61 GATT specifications: 25 GATT profiles and 36 services[37]. Additionally, 27 traditional GATT profiles[38]. Among them is the Environmental Sensing Profile (ESP) for sensor data. The ESP profile consists of three services: Device Information Service (DIS), Battery Service (BAS), and Environmental Sensing Service (ESS). According to the specification, this service exposes measurement data from an environmental

[37] `https://www.bluetooth.com/specifications/gatt`
[38] `https://www.bluetooth.com/specifications/profiles-overview`

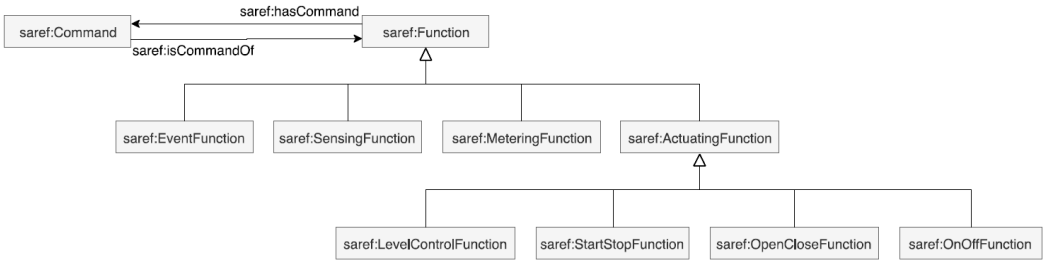


Fig. 5.9: SAREF Function types [368]

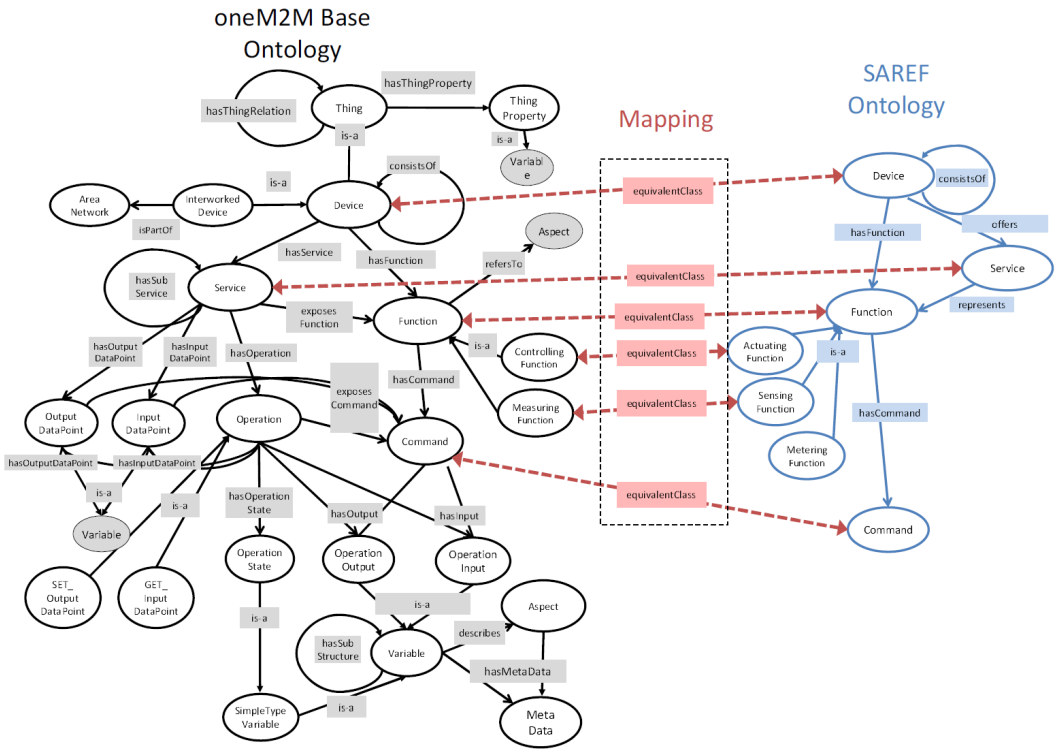


Fig. 5.10: Mapping between SAREF and the oneM2M Base Ontology [368]

sensor intended for sports and fitness applications. ESS Characteristics are among others Pressure, Temperature, Humidity, etc. Each characteristic has an UUID and is described by XML Schema. For example, temperature has the UUID 2A6E and defined as the unit `org.bluetooth.unit.thermodynamic_temperature.degree_celsius`. It has the format sint16, short integer, and the Decimal Exponent of -2. Such Characteristics Example for ESS Temperature is shown in Listing 5.3.

Additional information about the measurement is described by the ESS Descriptor Value Changed. This includes ES Measurement (associated ESS Characteristic by providing additional information pertaining to the value, e.g. Measurement Period, Update Interval, etc.), ES Trigger Setting (two parts: a Condition field and an Operand field), ES Configuration (shall be present if more than one ES Trigger Setting descriptor is present), Characteristic User Description (human-readable label to be associated with the measurement), Valid Range (the upper and lower bounds (inclusive)).

```
<Characteristic xsi:noNamespaceSchemaLocation="http://schemas.bluetooth.org/
    Documents/characteristic.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" name="Temperature" type="org.bluetooth.characteristic.temperature"
    uuid="2A6E" last-modified="2014-11-20" approved="Yes">
    <Value>
        <Field name="Temperature">
            <InformativeText>Unit is in degrees Celsius with a resolution of 0.01
                degrees Celsius</InformativeText>
            <Requirement>Mandatory</Requirement>
            <Format>sint16</Format>
            <Unit>org.bluetooth.unit.thermodynamic_temperature.degree_celsius</
                Unit>
            <DecimalExponent>-2</DecimalExponent>
        </Field>
    </Value>
</Characteristic>
```

Listing 5.3: Bluetooth Characteristics Example for ESS Temperature

Discussion
Interoperability between vendors is guaranteed when two devices (client and server) use the same Bluetooth profile. However the number of specifications is limited and defining a new one is a long process. An alternative is the definition of own customized profiles which leads to the vendor lock-in. While characteristics define the meaning of data, they do not use Semantic Web technologies. Moreover, the number of Bluetooth profile specifications is limited. Aggregation of data from characteristics requires further transformation steps.

5.3.6 SenML

Core Working group specified Sensor Measurement Lists (SenML) in the RFC 8428 [177] for compact transport of sensor data. It defines the syntax and very simple semantics for a data from the same sensor type. The goal of this format is a small memory footprint. Before in 2002, the Entity Sensor Management Information Base was specified in RFC 3433 for sensors.

SenML were finalized after eight years as RFC 8428 [177] by IETF in August 2018. This RFC defines an encoding format for sensor measurements into the media type and is intended for constrained devices in IoT applications. There are four representations to choose from: JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), Extensible Markup Language (XML), and Efficient XML Interchange (EXI). The main fields in a data set are: name, time, unit, value and sum. The name is the name of sensor or parameter. It is recommended to use concatenated names be represented as URIs [27] or URNs [320]. Other names like Universally Unique Identifiers (UUIDs) [208] can be used but have to consider the restricted character set specified in the RFC 8428. The time represents an absolute time relative to the Unix epoch (1970-01-01T00:00Z in UTC time). The units are defined by IANA in the "SenML Units" registry. The value is a single measurement per Record. The sum represents the sum of the values over the time. The example in Listing 5.4 shows a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with value of 23.5 degrees Celsius with its time. SenML messages are intended to carry the minimum information about measurements. It is assumed that the static metadata about the device is carried out of band e.g. using the CoRE Link Format [354]. New fields can be only added to the registration by Expert Review as defined in [72].

```
[
  {"n":"urn:dev:ow:10e2073a01080063","u":"Cel","t":1.276020076e+09,
   "v":23.5}
]
```

Listing 5.4: SenML example in JSON format [177]

Discussion
The SenML has a simple and short structure for arrays of sensor data but it is missing some semantic descriptions for the topic, type, location and approver. It is assumed that these semantic descriptions can be obtained from somewhere else e.g. the used protocol. However, extension of the SenML structure requires the registration process.

5.3.7 Schema.org Extensions for IoT

Once, schema.org was established by the biggest search engines (i.e. Google, Bing, Yahoo, Yandex) as a common vocabulary for metadata on the Web to provide schemas for structured data on web pages and documents. This vocabulary simplifies the assessment of the content for the search engines. Schema.org uses Semantic Web standards for encoding: RDFa, Microdata and JSON-LD.

Following the success of the vocabulary schema.org, a W3C community group is developing an extension for IoT, called iotschema.org [166]. The goal is the same[39] the definition of a common vocabulary for the IoT to support semantic syntactic and semantic interoperability.

For example, iotschema.org defines a class *Device*, subclassed by *Sensor* and *Actuator*. The Sensor has properties observes, madeObservation, and isObservedBy, see Figure 5.11. Such terms are a basis in the IoT domain. On the other side, domain-specific vocabulary is defined, for example the *Capability* class describes general measurement and actuation and has subclasses like AirConditioner, Humidity Sensing, IlluminanceSensing, Thermostat, etc. The definitions are in RDF, currently using the formats JSON-LD and Turtle.

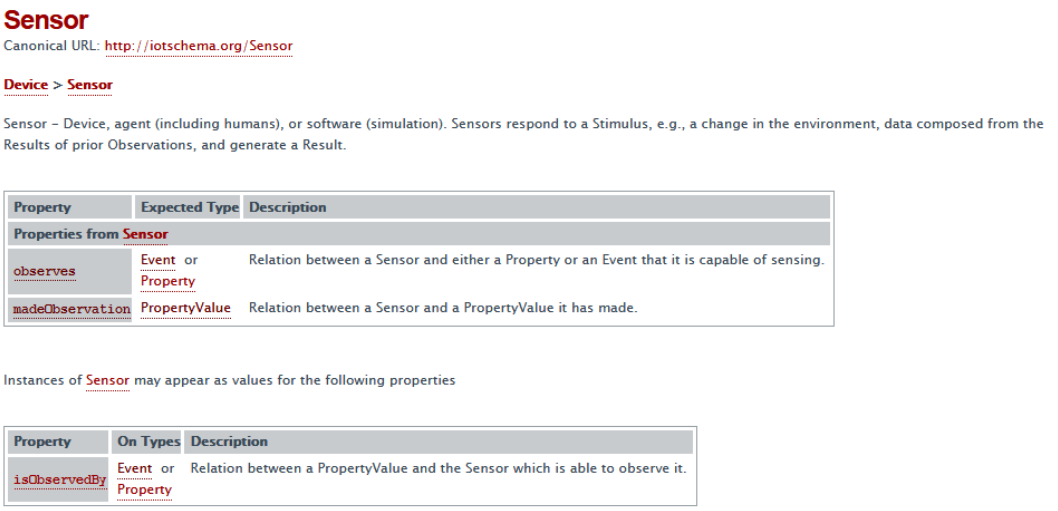
The iotschema.org is supposed to enable the annotation of W3C Thing Descriptions. Thing Description enables applications to interact with connected things using diverse protocols and data formats. The iot.schema.org enables web pages to be annotated using schema.org style annotation in RDFa or microformats.

Discussion
The iotschema.org has good intentions. However, the vocabulary is in an early stage and differs from the W3C Thing Description approach. Defined in the right way, it could be an extension or enhancement of the W3C Thing Description.

5.3.8 Vorto Information Model

Vorto Information Model describes the capabilities and functionality of a device. Vorto uses a Domain Specific Language (DSL) language, also called "Vorto Language for Digital Twins" [92]. Such Information Models consists of reusable and abstract *Function Blocks*. The Function Blocks can use three classes to describe capabilities

[39] https://github.com/iot-schema-collab/intro-materials/blob/master/iotschema-intro.md



Sensor

Canonical URL: http://iotschema.org/Sensor

Device > Sensor

Sensor – Device, agent (including humans), or software (simulation). Sensors respond to a Stimulus, e.g., a change in the environment, data composed from the Results of prior Observations, and generate a Result.

Property	Expected Type	Description
Properties from Sensor		
observes	Event or Property	Relation between a Sensor and either a Property or an Event that it is capable of sensing.
madeObservation	PropertyValue	Relation between a Sensor and a PropertyValue it has made.

Instances of Sensor may appear as values for the following properties

Property	On Types	Description
isObservedBy	Event or Property	Relation between a PropertyValue and the Sensor which is able to observe it.

Fig. 5.11: Sensor description in the iotschema.org

of device: *Properties*, *Events*, and *Operations*. A simple example of Vorto DSL is provided in Listing 5.5.

Besides the Vorto language, the Eclipse Vorto project produces "the meta information model, the tool set to create information models, the code generators and the repository to manage existing information models" [92]. The metamodel is used to define the relationship between the different entities like Information Models, Function Blocks, and Datatypes. The tool set is realized as an Eclipse plug-in which is based on the Eclipse Modeling Framework (Eclipse EMF) framework. Thus, the model creator can use the visualized tool (for non-programmers) or edit DSL directly (for programmers). Next, EMF enables code generator to generate logic, written in any language, for example XML, JSON, Java or C/C++. This generated code can be integrated in any IoT solution. Defined models are stored in an Information Model Repository [40].

The main Vorto Eclipse contributor, Bosch company, intends to keep the Vorto models compatible to other models defined e.g. by EEBus, ETSI and oneM2M as well as Smart Home Device Template (SDT) [41] which has been specified by HGi (Home Gateway Initiave) in 2015.

Here an example that outlines the working process of Vorto: a manufacturer creates a new WiFi smoke detector which can measure the temperature, return the battery status and fire an alarm event in case of fire. Using the Vorto tool set, the manufacturer creates an information model with these three functionalities. Then, the manufacturer publishes the model to the Repository. Now, a user who bought such a smoke detector wants to include it into his openHAB [268] environment. Using the Vorto tool set, the user can browse the Repository and download the information model created by the manufacturer. Then the user could create the openHAB representation of the device using a specific code generator. Finally, the user would complete this representation by adding required WiFi configurations. Since the

[40] `https://vorto.eclipse.org/`

[41] `https://github.com/Homegateway/SmartDeviceTemplate/blob/master/SDT/schema3.0/docs/Introduction.md`

function blocks of the information model are reusable, another manufacturer could use it for his smoke detector.

```
vortolang 1.0
namespace com.bosch.iot.suite.examples.digitaltwin
version 1.0.0
displayname "DigitaltwinExample"
description "Information Model for DigitaltwinExample"
using com.bosch.iot.suite.examples.digitaltwin.D100;1.0.0
using com.bosch.iot.suite.examples.digitaltwin.Accessories;2.0.0
using com.bosch.iot.suite.standard.Descriptive;1.0.0

infomodel DigitaltwinExample {
        functionblocks {
                Device as D100
                Description as Descriptive
                Accessories as Accessories
        }
}
```

Listing 5.5: Eclipse Vorto DSL short example

The Vorto information models can be exported to the *Ditto* format (see Figure 5.6), another Eclipse project supported by Bosch. The Ditto framework mirrors physical devices as digital representations in the cloud. Developers can interact with such "digital twins" as with other services ("device as a service" paradigm). Ditto exposes a unified resource-based HTTP JSON API representing devices, defines "Digital Twin State Management Protocol" using JSON for command- and events-based interaction with devices, provides connectors to AMQP, MQTT and Kafka endpoints, manages states for digital twins, integrates with Eclipse Hono cloud component.

```
{
  "definition": "com.bosch.iot.suite.examples.digitaltwin:DigitaltwinExample:1.0.0
      ",
  "attributes": {
        "modelDisplayName": "DigitaltwinExample" },
        "features": {
        "Device" : {
        "definition": [
                "com.bosch.iot.suite.examples.digitaltwin:D100:1.0.0"   ],
        "properties": {
                "status": {
                        "temperature" : 0.0             },
        "configuration": {
                "threshold" : 0.0}  } },
        "Description" : {
        "definition": [
                "com.bosch.iot.suite.standard:Descriptive:1.0.0"        ],
        "properties": {
        "configuration": {
                "displayName" : "" } }  },
        "Accessories" : {
        "definition": [
                "com.bosch.iot.suite.examples.digitaltwin:Accessories:2.0.0"    ],
        "properties": {         }       } }
}
```

Listing 5.6: Ditto JSON short example

Discussion

The Eclipse Vorto project provides a convenient plug-in with an information model and tools for integration into an IoT solution. Vorto is also compatible with other Eclipse projects supported by Bosch. However, to participate in the model repository,

a Bosch account is required. Thus, the repository is owned by the vendor. Further, Vorto uses the model-driven approach with DSL which has some disadvantages e.g. long ways from model to code through a code generator, problems with details and exceptions, no semantically interoperability is ensured.

5.3.9 Conclusion

The most suitable ontology for device descriptions is the oneM2M Base ontology. As an extension for vertical domains, the SAREF ontology can replace the oneM2M ontology. The SSN ontology is appropriated for the description of sensor data.

Finally, we have chosen the oneM2M Base Ontology [24] (hereinafter referred to as oneM2M ontology) for two reasons: (i) it is a small ontology for service and the functionality description of devices which answers our competency questions; and (ii) it is represented by the OWL standard. Additionally, the SAREF ontology from ETSI is related to oneM2M ontology and developed vertical domain ontologies. This might be a sign for a potential establishment of these ontologies.

The oneM2M ontology contains specific types of communication parameters (names of operations, input/output parameter names, their types and structures, etc.). It is used to allocate resources in the local area network and execute reads/writes from/into these resources. The provided vocabulary is a expedient starting point for description of device capabilities and functions. In addition, the ontology is small enough to fit into constrained devices.

5.4 Ontology Extension for Device Description

The oneM2M Base ontology will be reused for device description but must be extended in a minimal way. This extension is described in this Section and include the Steps 3-7 of the Ontology 101 method [262]. Our namespace is `https://www.cs.uni-potsdam.de/bs/research/myno#`.

The device description is a self-description of the capabilities provided by the device, based on the oneM2M ontology. The oneM2M ontology is a high-level ontology where the classes are defined in a generic way. For example, the *Operation* class is defined together with *OperationState*, *Input*- and *OutputDataPoint*, *OperationInput* and *OperationOutput*. We have to extend the oneM2M ontology to represent properties required for the implementation of the NETCONF-MQTT bridge:

- *YangDescription* as a subclass of *ThingProperty*;
  The *YangDescription* is used to generate a description of the RPC call in the YANG model.
- *mqttTopic* and *mqttMethod* as Data Properties for MQTT operation description;
- return values (e.g. OK, NOOP, ERROR) for *OperationState*;
- parameter values (e.g. red, yellow, green for LED colors) as *OperationInput* for actuators.

These ontology extensions are sufficient to map the NETCONF RPC calls to the MQTT Publish/Subscribe operations.

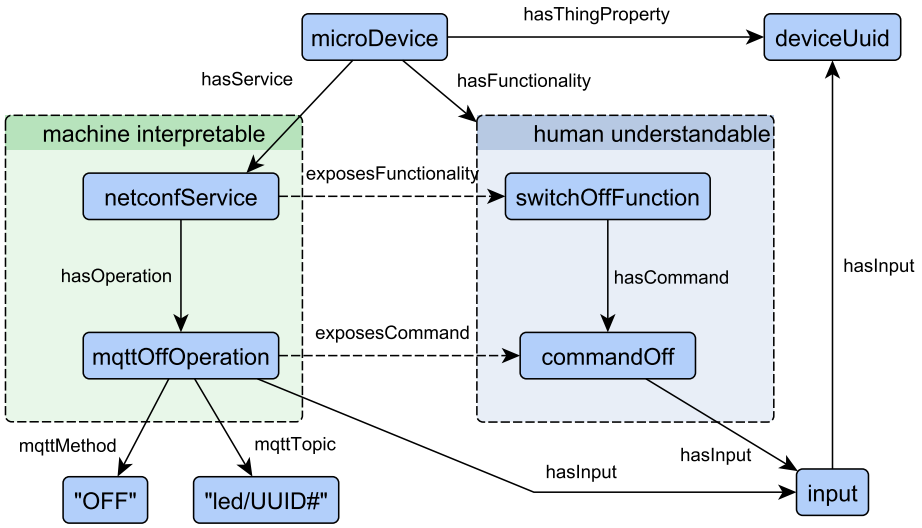


Fig. 5.12: Device Description based on the oneM2M Base Ontology [315]

A device is a whole electronic device identified by UUID. A sensor or actuator built within it enables a capability of the device. A device is controlled by a single piece of software instead of a set of software pieces.

In the following, we show a couple examples how the capabilities of a device can be modeled using oneM2M. First, we present the ontology modeling for the actuator scenario and next the sensor scenario.

ControllingFunctionality
We instantiate our ontology from the oneM2M Ontology to describe how an LED on a microcontroller board as actuator can be controlled, as introduced in [315]. There are three control methods: to switch on the light, to set a pre-defined LED color and to switch the LED off. For simplicity, Figure 5.12 shows the part of our ontology which describes the switch-off control of the LED. Our individual device with the name *microDevice* is an instance of the oneM2M ontology class *Device*. In OWL, each resource (i. e. individual, class, property) is identified by an unique URI or IRI. The annotations in OWL are `https://www.cs.uni-potsdam.de/bs/research/myno#microDevice` for the instance and `http://www.onem2m.org/ontology/Base_Ontology/base_ontology#Device` for the class. The *microDevice* has a property *deviceUuid* which is an instance of the oneM2M class *ThingProperty*.

Further, the oneM2M Ontology distinguishes between machine-interpretable exposures and human-understandable meanings. A *Device* has a *Functionality* (*Controlling* or *Measuring*) which provides human-understandable meaning what the device "does". A *Functionality* has a *Command* which is the human- understandable name of an action that is invoked in the device. We name our functionality *switchOffFunction* and the command *commandOff*.

On the other side, a *Device* provides a *Service* which is the machine- interpretable exposure of *Functionality*. Therefore, the *Service* is discoverable, registerable and remotely controllable in the network. A *Service* has an *Operation* which is the machine-interpretable exposure of a *Command* to the network. Our service individual is called *netconfService* and the operation *mqttOffOperation*. *OperationInput* and

OperationOutput of the *Operation* can parameterize the *Command*. This is the *input* individual which has the *deviceUuid* as an input data.

Consequently, a finite state machine for actuators has to be implemented internally on the microcontroller. It allows to keep an internal state like LIGHT ON or LIGHT OFF and commands turn on, turn off and toggle, similar to Figures [145, p. 184] which makes the device interoperable and efficient in communication.

MeasuringFunctionality
The sensor functionality is modeled different from the actuator in the ontology. The operation where and how sensor data can be subscribed is described by device capabilities. We use the *OutputDataPoint* of *Operation* to set an *mqttTopic* for MQTT protocol. When a client subscribes to this topic, it will receive sensor values periodically. The units of measurements are defined by reusing an OM-2 ontology [298] and the appropriated unit `http://www.ontology-of-units-of-measure.org/resource/om-2/degreeCelsius`. Further properties of measurements could be defined if required (i.e. what has been measured: air temperature).

5.4.1 Event Notification

The concept of event functionality will be found in several IoT ontologies, for example in WoT Thing Description and SAREF. In the WoT Thing Description [184], an *EventAffordance* was defined as an Interaction Affordance that describes an event source, which asynchronously pushes event data to Consumers (e.g., overheating alerts). A *subscription* defines data that needs to be passed upon subscription, e.g., filters or message format for setting up Webhooks. *Data* defines the data schema of the Event instance messages pushed by the Thing. A *cancellation* defines any data that needs to be passed to cancel a subscription e.g., a specific message to remove a Webhook. A Webhook is a customer-oriented event mechanism in this case.

The SAREF ontology defines the *saref:EventFunction* which allow to "notify another device that a certain threshold value has been exceeded". This function allows the command *saref:NotifyCommand* and *saref:hasThresholdMeasurement* with *saref:Measurement* classes.

Concepts for event notification and alarm management are defined for NETCONF and YANG. The RFC 5277 [61] from 2008 specifies NETCONF Event Notifications. An event is "something that happens which may be of interest - a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system". There is an asynchronous message notification delivery service for NETCONF which is not further specified how it is realized technically. The operation <create-subscription> initiates an event notification subscription that will send asynchronous event notifications to the initiator of the command until the subscription terminates. Parameter like stream, filter, start and stop time can be defined. An event notification <notification> is sent to the client who initiated a <create-subscription> command asynchronously when an event of interest (i.e., meeting the specified filtering criteria) has occurred [61]. An event notification is a complete and well-formed XML document. The <notification> is not a Remote Procedure Call (RPC) method but rather the top-level element identifying the one way message as a notification.

The RFC 6470 [30] specified the YANG Module for NETCONF Base Notifications in 2012. The client will be notified about that the NETCONF server state has changed: <netconf-config-change>, <netconf-capability-change>.

The RFC 8632 [400] specified a YANG Data Model for Alarm Management in 2019. This RFC includes functions for alarm-list management, alarm shelving, and notifications to inform management systems (alarm-management application that consumes the alarms). The purpose is to define a standardized alarm interface for network devices that can be easily integrated into management applications. The alarm definition is provided as following: An alarm signifies an undesirable state in a resource that requires corrective action. Alarms are viewed as states on resources and not as discrete notifications. The RFC provides Alarm-Usability Requirements which is a summary adopted to networking based on the ISA and Engineering Equipment Materials Users Association (EEMUA) standards. The mapping to "Information technology - Open Systems Interconnection – Systems Management: Alarm reporting function", the ITU-T Recommendation X.733, is achievable. An example for smoke detector is provided.

However, NETCONF specified the subscription operation for notification as an RPC but did not specify how the notification should be realized technically (some commercial solutions use Websocket based notifications). This is an open issue. YANG modules for events and alarms [400] are specified but alarm modules can be highly detailed.

EventFunctionality
The ontology will be extended by two new subclasses of the *Functionality* class, see Figure 5.13. These classes should enable aggregation of such functionalities.

- *ConfigurationFunctionality* for configuration of thresholds. Also, the *Command* and *OperationInput* classes (including *mqttTopic* and *mqttMethod* properties) are involved, just like the controlling functionality for actuators.

- *EventFunctionality* for event triggering from device through a MQTT Topic defined in the *OutputDataPoint*, just like published sensor data.

Our early experiments in [398] showed that parsing JSON format is challenging on constrained devices i.e. the CC2538 board. Based on analysis in Section 5.4.1, we decided to form tuples for configuration of event triggering on the devices. The configuration parameters must be defined in the device description.

ConfigurationFunctionality
Following steps are required to model a new configuration functionality in the ontology-based device description:

- a new instance of the class *ConfigurationFunctionality*: e.g. funcConfTemperature

- a MQTT Topic Schema: `config/$sensor/$sensortype/$sensorname/$UUID` (for example `config/sensor/temperature/temp_1/UUID`)

- an instance of *Command* class with value CONFIGEVENT. This value will be sent in the RPC.

- a name for the event that will be triggered as an instance of the *OperationInput* class. This name will be sent in the event payload (string datatype, no white spaces and no special characters).
- possible min and max sensor values for the thresholds as an instance of the *OperationInput* class with properties *hasDataRestriction*. This will be mapped to the YANG module as: "typedef range min...max" in <container> element.
- Interval as an instance of the class *Interval* from the W3C Time ontology [74]
- Duration as an instance of the class *Duration* from the W3C Time ontology [74]
- Operator as an instance of the *OperationInput* class with properties *hasDataRestriction_pattern* with values like <,=, >, >=, <=
- CRUD operations as an instance of the *OperationInput* class with properties *hasDataRestriction_pattern* with values CREATE, READ, UPDATE, DELETE

AutomationFunctionality
The automation functionality is a sublcass of *ConfigurationFunctionality* and concatenates existing controlling functionality. However, it is modeled like a *Controlling-Functionality* with a *Command* and a *Description*.

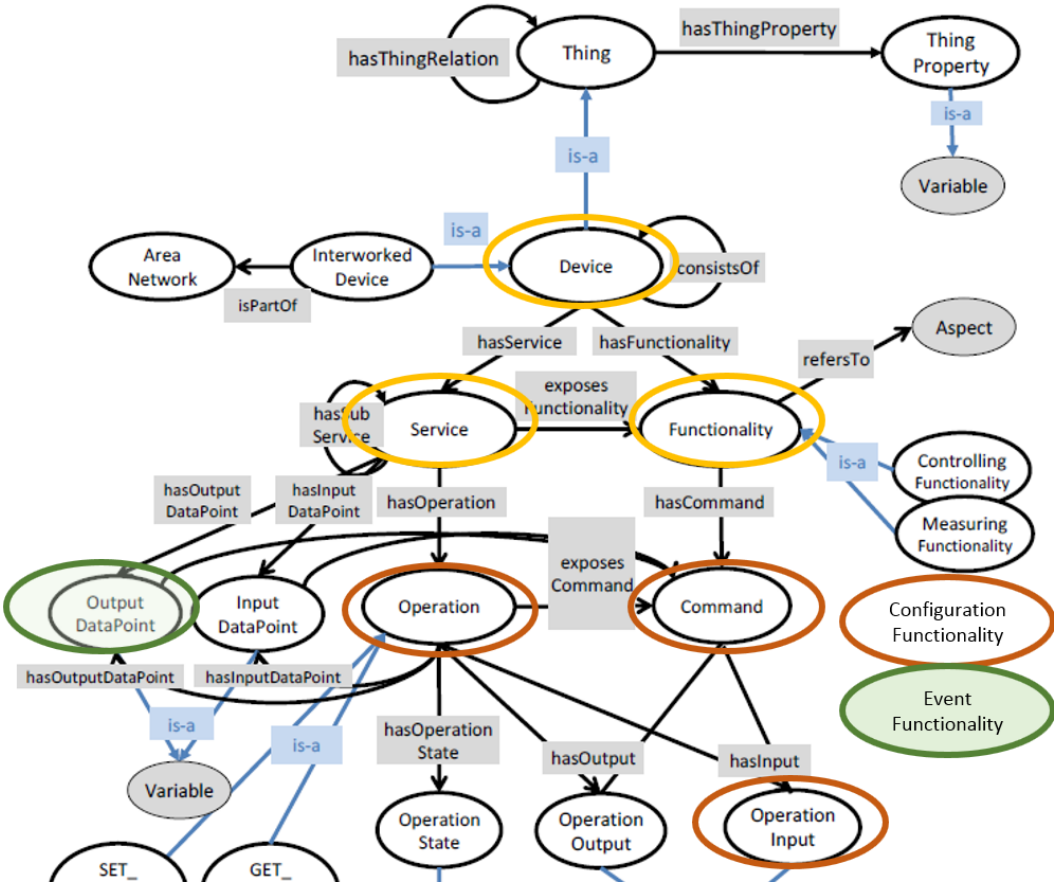


Fig. 5.13: Configuration- and EventFunctionality in Device Description

The *EventFunctionality* instance is designed like *MeasuredFunctionality* for sensors. An instance of the class*OutputDataPoint* with a MQTT Topic must be defined. The

Event MQTT Topic schema is: `event/$sensor/$sensortype/$sensorname/$UUID`, e.g. `event/sensor/temperature/temp_1/UUID`.

5.5 Sensor Data

The sensor data is collected in a sink. Sensor data stored as Semantic Web data has advantages because SPARQL can be used as query language [284]. There are some typical queries for the processing of such data, for example:

- calculate an average value of all light sensors in a room XY on 2020.04.20 between 2 and 3 pm.
- find min. and max. values of these light sensors (same period of time)
- find notifications from the motion detector (same period of time)
- retrieve all kinds of sensors which are installed in the building A
- retrieve the measure units of temperature sensors
- prove sensing conditions in the real live time and raise events or alarms

Such data processing is only possible if all kinds of information are provided. We identified and classified this information into dimensions [318] which we refined for the MYNO framework. For example, a temperature sensor can provide these dimensions:

- The *sensor value* is the physical value itself (e.g. 25) and answers the question (what?).
- The *topic* dimension describes the kind of a sensor value (e.g. temperature) and answers the question (what is it about?).
- The *type* dimension describes the type of value (e.g. measurement in Celsius) and answers the question (what kind of information?).
- The *time* defines the time of a measurement (e.g. time stamp in xsd:dateTime schema) and answers the question (when?).
- The *Location* is the place of a measurement (e.g. the room 1 or geo coordinates) and answers the question (where?).
- The *sender* describes a source or a sensor node (e.g. a device UUID) and answers the question (what is sending?).
- The *approver* is any entity which can confirm the provenance of the information and answers the question (what does confirm the provenance?).

Such context information can be described by an ontology. The SSN ontology is exactly defined for this purpose. SOSA ontology is the core part of the SSN ontology. Using this ontology, the collected sensor data can be queried by SPARQL. Figure 5.14 show the common patterns used by the three activities with classes stacked where they play a similar role in the SSN ontology. In MYNO framework the observation and actuation pattern is provided:

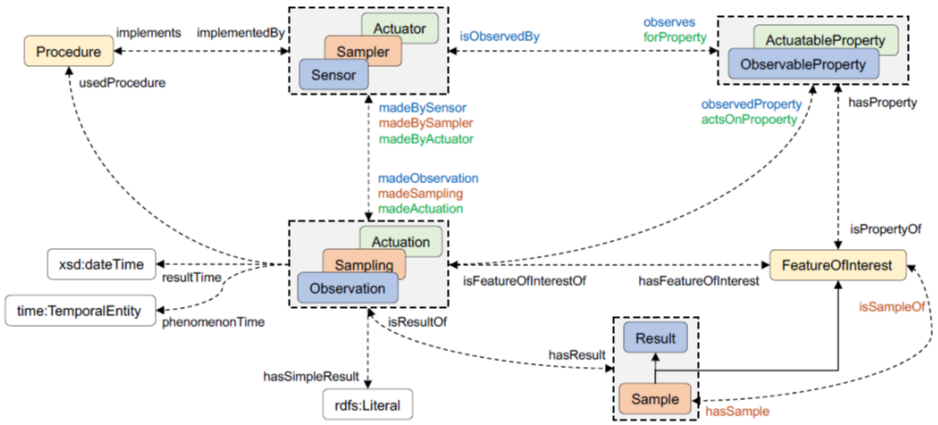


Fig. 5.14: Activities in the SSN ontology: Sensor, Actuator, Sampler [137]

- every Sensor sends an observation, either periodically (e.g. light sensor) or if an observation occurred (e.g. motion sensor) and sends a value to an MQTT Topic e.g. `sensor/temperature/temp1/ssn/UUID`

- every actuator sends an actuation if an action was triggered (e.g. switch on or off the light) through an MQTT Topic e.g. `actuator/led/rgb-led1/ssn/UUID`

```
{       "@id": "om-2:Temperature",
       "@type": "sosa:ObservableProperty",
       "sosa:isObservedBy": {"@id":"myno:mytempsensor" } }

{       "@id": "myno:espBoard",
       "@type": ["owl:NamedIndividual", "sosa:Platform" ],
       "sosa:hosts": {"@id": "myno:mytempsensor" } }

{       "@id": "myno:myprocedure",
       "@type": ["owl:NamedIndividual", "sosa:Procedure" ],
       "sosa:madeBySensor": {"@id": "myno:mytempsensor" },
       "http://www.w3.org/ns/ssn/hasOutput": {"@id": "myno:procedureOutput"} }
{       "@id": "myno:mytempsensor",
       "@type": ["owl:NamedIndividual", "sosa:Sensor"],
       "sosa:isHostedBy": {"@id": "myno:espBoard" },
       "sosa:madeObservation": {"@id": "myno:myobservation/07082019-1"},
       "sosa:observes": {"@id": "om-2:Temperature"} }

    {"@id": "myno:procedureOutput",
       "@type": ["owl:NamedIndividual", "http://www.w3.org/ns/ssn/Output" ],
       "rdfs:comment": "current measured value"}

    {    "@id": "myno:room21",
       "@type": ["owl:NamedIndividual", "sosa:FeatureOfInterest"],
       "sosa:isFeatureOfInterestOf": {"@id": "myno:myobservation/07082019-1"}}
```

Listing 5.7: Example of sensor data in a device description

Table 5.1 provides an overview how the dimensions can be modeled using SSN ontology. The results from the table are applied to the device description in Listing 5.7. The sensor data is modeled in SSN which will be sent from a device is shown in Listing 5.8.

Tab. 5.1: Semantic Sensor Data Modeling with dimensions and SSN

Dimension	SOSA/SSN	When and Where this information comes from?
sensor value (what?)	sosa:Observation -> sosa:hasSimpleResult, sosa:hasResult (With QUDT 1.1, a sosa:Result would be a qudt:QuantityValue. With OM 2, a sosa:Result would be a om:Measure or om:Point.)	every time from sensor board
topic (what is it about?)	subclass or instance of *sosa:ObservableProperty* (external Ontologies e.g. OM-2)	only once from the Device Description
type (what kind of information?)	external Ontologies for units (e.g. OM-2 or QUDT 1.1)	only once from the Device Description
time (when?)	sosa:resultTime (xsd:dateTime)	every time from MQTT Broker
location (where?)	sosa:FeatureOfInterest, subclass of geo:Feature (GeoSPARQL ontology)	once configured for the static devices, every time from mobile devices
sender (who?)	sosa:Sensor, sosa:Platform, subclass or instance of sosa:System, UUID is used in the URI in SOSA/SSN e.g. `apartment/134`	every time through the UUID in MQTT Topic
approver (provenance?)	W3C PROV-O ontology	every time from MQTT Broker

5.5.1 Integration of SSN Ontology into Device Description

The oneM2M ontology based device description should include information about the sensor output format which is SSN ontology based. As secondary goal, the output must have a relation to the device description because sensor data is stored separately from device descriptions.

So far following ontology classes are used for sensors: *Service* with *OutputDataPoint*, *MeasuringFunctionality*. There is no *Operation* and no *Command* classes involved. An instance of the *OutputDataPoint* class has a property *mqttTopic* e.g. with value `sensor/brightness/led01/UUID`.

A concept is to extend the device description as following: an instance of the *OutputDataPoint* has a property *hasOutput* to an instance of a new class *ServiceOutput* similar to *OperationOutput* for controlling functions. This *ServiceOutput* class is *owl:equivalentClass* to the *sosa:Observation* class. For the actuator, the *OperationOutput* can be used which is *owl:equivalentClass* to the *sosa:Actuation* class.

Ontology-Alignment can be used to match these two ontologies: oneM2M and SSN:

- *oneM2M:Device* and *ssn:Platform* are identical in their meaning;

```
{    "@id": "myno:measuredResult/123",
     "@type": ["owl:NamedIndividual", "om-2:Measure", "sosa:Result" ],
     "om-2:hasNumericalValue": {"@type": "xsd:double", "@value": "23.5" },
     "om-2:hasUnit": {"@id": "om-2:CelsiusScale"},
     "sosa:isResultOf": {"@id": "myno:myobservation/07082019-1"} }

   { "@id": "myno:myobservation/07082019-1",
     "@type": ["owl:NamedIndividual", "sosa:Observation"],
     "sosa:hasFeatureOfInterest": {"@id": "myno:room21" },
     "sosa:hasResult": {"@id": "myno:measuredResult/123" },
     "sosa:hasSimpleResult": {"@type": "xsd:double", "@value": "23.5"},
     "sosa:madeBySensor": {"@id": "myno:mytempsensor"},
     "sosa:observedProperty": {"@id": "om-2:Temperature"},
     "sosa:resultTime": {"@type": "xsd:dateTime", "@value": "2018-06-20T21:49:18
         +00:00" },
     "sosa:usedProcedure": {"@id": "myno:myprocedure" } }
```

Listing 5.8: Example of sensor data sent from a device

- relationship on the class level can be expressed by *owl:equivalentTo*;
- relationship on the instance level can be expressed by *owl:sameAs*;
- UUID as the device property is not exactly defined neither in SSN nor in oneM2M. In SSN it is indirectly referenced in the URI name or instance name. Thus an UUID must follow naming convention for URIs. It would be possible to define an UUID as a part or URI in the oneM2M ontology.

A summary for this concept is so far:

- both ontologies, oneM2M and SSN, can be linked together.
- Output format or SSN template in the device description will be read only once, e.g. from the Virtual Device in Scenario 3 (Aggregation of Sensor Data).
- Virtual Device is subscribed to the Topic with semantic sensor data and enriches the sensor data with time stamp, etc.
- Virtual Device can aggregate sensor data on the edge. Aggregation rules are pre-configured (with SPARQL-Queries or RDF Streams). Possible implementation with RDFlib-Web (SPARQL endpoint based on Flask framework).

However, there are some improvements for this concept because to transmit data, energy is required. It is better to collect data and send it less frequently. The amount of data should be also minimized, e.g. pure sensor data. The semantic enrichment can be completed on the edge, or sensor data must be compressed, or another format (e.g. SenML) should be used.

5.6 Conclusion

Choosing an ontology for device descriptions and sensor data is a demanding task. Especially in case of reusing of an ontology, many IoT ontologies had to be surveyed. Still, there is no perfect ontology but the most appropriate one was chosen: the oneM2M Base ontology. This ontology was extended to meet the requirements for network configuration management with the MYNO framework. Additional classes (*ConfigurationFunctionality*, *EventFunctionality*, *AutomationFunctionality*) and properties (e.g. *mqttTopic*, *mqttMethod*) were defined. The SARFE ontology

was developed by ETSI as an extension of the oneM2M Base Ontology and can be considered for vertical domains.

Virtual Device

6

> *It is always possible to add another level of indirection.*
>
> **— RFC 1925, Nr. 6a**
> (The Twelve Networking Truths)

The term *virtuality* appears in IoT literature but it is used in various ways as shown in related work. Then, we propose the concept of a *Virtual Device* for the MYNO framework.

6.1 Related Work: Virtual Objects and Digital Twins

Nitti et al. [257] provided a survey of the *Virtual Object(VO)* in the IoT. They analyzed the definitions from the historical perspective of VOs. They found manifold definitions of virtualness in the past decades. Starting with RFID tags in the year 2000, every real object has a *digital representation* of itself and also *identification and addressing scheme*. The latter enabled *enhanced service discovery*. The project SENSEI [395] introduced *semantically enriched* models and descriptions of sensors, actuators and processing elements and sub-sequentially *context awareness and service orchestration*. The project COMPOSE [68] evolved an assisted service composition engine where virtual objects acquire a *cognitive ability* i.e. the engine automatically generate possible compositions based on the semantics of the data. iCore Cognitive Framework [115] used the composite virtual objects concept with the similar aim. Finally, the authors concluded with the following definition:

Definition 7 (Virtual Object (VO)).
A Virtual Object is a digital representation, semantically enriched, of a real world object (human or lifeless, static or mobile, solid or intangible), which is able to acquire, analyze and interpret information about its context, to augment the potentialities of the associated services for the benefits of the quality of life of humans as final consumer of the real world data. [257]

Furthermore, they noticed that the association between real and virtual objects can be *one-to-many* [68], *many-to-one* [395] or *many-to-many* [115]. Moreover, *accounting and authentication* functionality should enhance virtual objects to provide authorized access.

Dibowski [84] used the term "Virtual Properties" for ontology properties which do not exist in his ontology-based device descriptions but will be computed on demand. Physical Entities are represented by Virtual Entities in the IoT Architectural Reference Model (ARM) [25] and in the ISO/IEC CD 30141, Internet of Things

Reference Architecture (IoT RA) [161]. The Ontology-Defined Middleware [51] uses Virtual Objects (VOs) to embody a template that describes the behavior for physical devices.

A WSN Management Framework was proposed in [228]. The core component is Virtual Sensor Network where *Virtual Entities* (VE) are managed. The authors move beyond a simple virtual representation of the real world object and include cognitive capabilities to effectively compose an intelligent agent to mediate and manage the physical WSN within the context of service demands and system capabilities.

Another term often used in the IoT is the "Digital Twin". Digital Twins are the focus of the Eclipse project "Ditto" [92] supported by Bosch company. The definition according to W3C WoT Architecture:

> A digital twin is a virtual representation of a device or a group of devices that resides on a cloud or edge node. It can be used to represent real-world devices which may not be continuously online, or to run simulations of new applications and services, before they get deployed to the real devices. [226]

Many approaches were proposed for virtual objects. Mrissa et at. [244] proposed an avatar for the Web of Things. Bader et al. [19] introduced virtual representations based on RESTful Web APIs for Industrial IoT. Negash et al. [252] proposed a web of virtual things at the fog layer for integration of different platforms. Based on semantics, data aggregation in Home Automation was proposed by Ramparany et al. [290].

Nitti et al.[257] conclude that interoperability is one of the main key challenges still need to be addressed because proposed virtual objects belong to different architectures and speak different languages. Another challenge is the scalability. Lifecycle of virtual objects must be managed and also deleted when they are not needed anymore.

6.2 Virtual Device Concept

We introduce an extension of the NETCONF-MQTT bridge with a **Virtual Device (VD)** [317]. In opposite to the Virtual Objects, defined in 6.1, such Virtual Devices are more than just a digital representation of a real device. In the MYNO framework, a *Virtual Device* is a digital representation of the (aggregated) services of real world objects. The granularity is determined by the association between real and virtual objects (one-to-one, one-to-many, many-to-one or many-to-many). The VD provides a semantic device description of itself. The VD provides search and discovery of devices and services as well as service orchestration. Based on the aggregated semantic device descriptions, it provides context awareness of the environment information and has cognitive ability. The VD is a device, because it acts as a device to the NETCONF similar to real devices.

Figure 6.1 depicts the flow of device descriptions and shows the hierarchical tree topology of devices and virtual devices. The devices (actuators and sensors) on the left/right side send their device descriptions to the next MQTT broker. This step takes place only once upon deployment of the system. These descriptions are passed to a virtual device which is subscribed to the same MQTT Topics as the bridge.

Device descriptions will be semantically analyzed by a virtual device. The virtual device collects device descriptions and aggregate the functionalities for sensors and actuators at the edge (e.g. a room or a floor). The scalability is possible through hierarchical tree topology (i.e. functionalities of virtual devices of all floors will be aggregated by a functionality of virtual device of a building). The aggregated device description represents the virtual device at the edge and is integrated into the bootstrap process. We assembled the main four scenarios for virtual devices and split them into tasks.

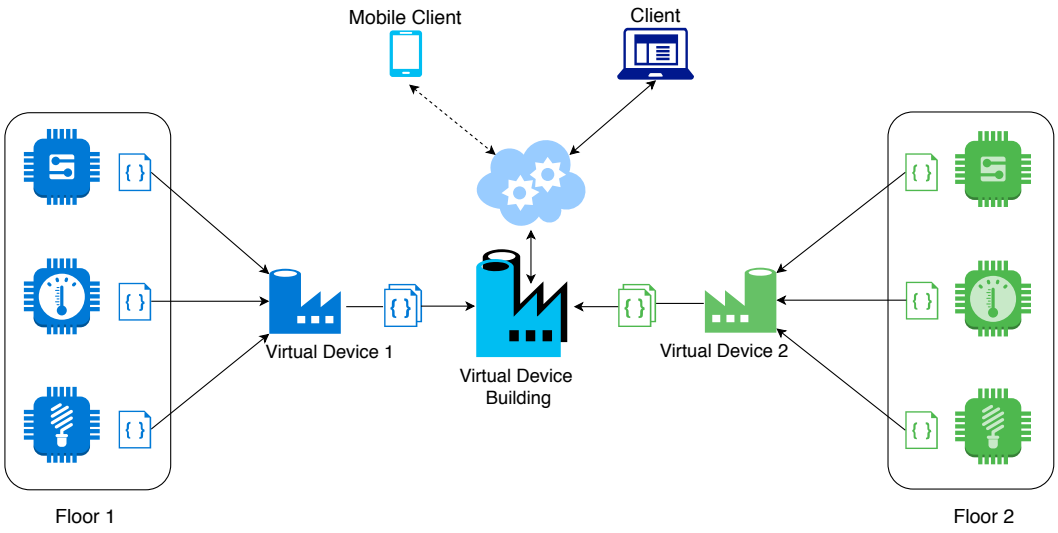


Fig. 6.1: Virtual Devices Topology

Scenario 1 Aggregation of Device Description
Semantic Discovery contains tasks for performing semantic processing of device descriptions: collect and aggregate device capabilities from device descriptions; identify similar capabilities between the devices; infer new system capabilities from device descriptions; generate a virtual device description and publish it to the MQTT broker.

Scenario 2 Aggregation of Controlling Services
Controlling Services comprises tasks for processing requests for actuators: delegate controlling functionalities, e.g. switch off all lights; map aggregated functionalities to the single device functionality. The aggregation of services also applies for firmware update functionality.

Scenario 3 Aggregation of Sensor Data
Aggregation of Sensor Data contains tasks for aggregating sensor data i.e. calculating a current average sensor value for temperature or state: collect and pre-process sensor data; aggregate sensor data; calculate average value or state.

Scenario 4 Aggregation of Events
Event Triggering comprises tasks for checking conditions and creating push events: monitor sensor data; prove conditions; trigger events or alarms if necessary. Such events can be collected at the virtual device.

A discussion point is where the alarm or event should be triggered: on the sensor board or on the edge. In the BLE, this is defined in the client profile and not on

the device which is the server [145, p. 215]. However, we decided to save the configuration for event triggering on a device, see Section 4.6.3.

6.2.1 Extension of the Device Description

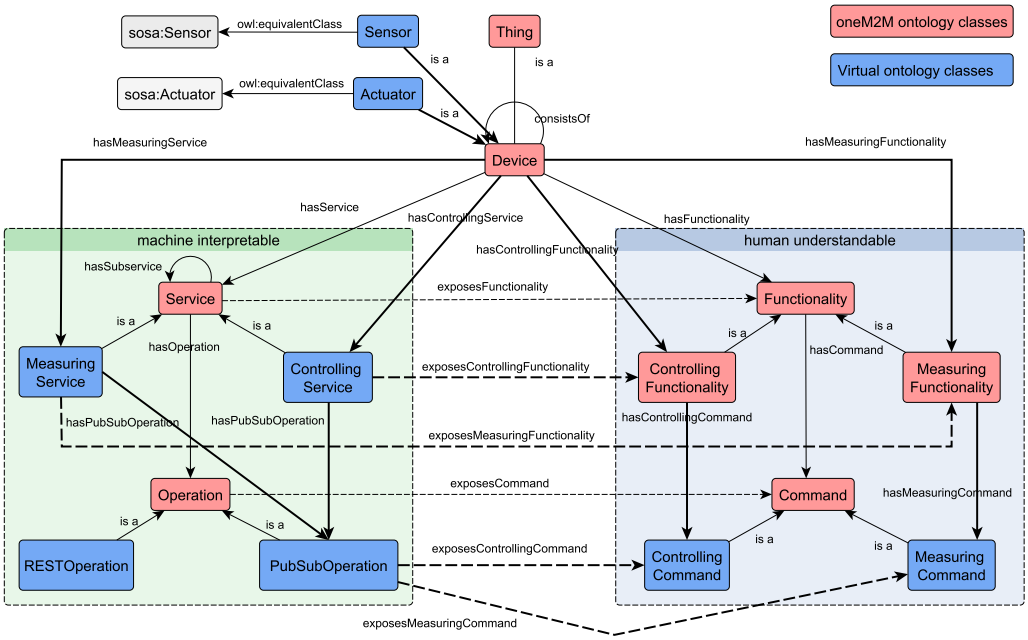


Fig. 6.2: Extended oneM2M ontology for Virtual Devices

Our device descriptions derive directly from the oneM2M Base ontology. The oneM2M ontology is organized in two parts, Figure 6.2: human-understandable functionality descriptions (on the right) and machine-interpretable technical service descriptions (on the left) of *Device* class. Both parts are corresponding with each other. oneM2M defines a high-level base ontology. The only specific classes are *Controlling* and *Measuring Functionality* subclasses of the *Functionality* class. IoT devices are usually equipped with sensors and/or actuators. Actuators have controlling functionality e.g. switch on and off light. Sensors have measuring functionality e.g. brightness, temperature, humidity.

The device descriptions based on ontology are eligible for inference. Matching mechanisms can be defined with the help of Semantic Web technologies. Looking closer on the oneM2M ontology, we noticed that there is a gap between the high-level oneM2M ontology and the specific device description ontology. This gap makes inference difficult e.g. to infer similar functionalities of different or same kind of devices. The oneM2M ontology needs to be extended in order to enable inference. Thus, these extensions can describe *abstract ontology classes* and are used for ontology matching and aggregation as described bellow.

Our extension approach, depicted in Figure 6.2, has two levels. First, we introduce measuring and controlling subclasses also for *Service* and *Command* classes. Thus, all exposed classes distinguish in measuring and controlling subclasses, except the *Operation*. On this level, we can at least infer all controlling and measuring services and commands in device descriptions.

Additionally, we define two subclasses of *Operation* class: for REST-based services (or Pull Services) and Publish/Subscribe Services (or Push Services). The distinction of these two types of operations is based on the fact that there are basically two types of M2M protocols in the IoT: either REST-based e.g. CoAP protocol or Publish/Subscribe based e.g. MQTT protocol. Each of these operations are exposed to *Controlling* and *Measuring Command* classes. Our implementation with MQTT is focused on the *PubSubOperation* class.

The second part of our extension is the introduction of so-called *Abstract Classes* (similar to object-oriented programming) and a set of description logic and rules which are application dependent. For example, we introduce a *AbstractSwitchOff-Functionality* class as a subclass of *Controlling Functionality* which means in common sense the functionality for switching off something. This functionality is exposed by a service. Thus, we can make inferences on device capabilities in a fine-grained way and find all devices with such functionality. This extended ontology is used as a base for virtual devices, further referred to as **Virtual Ontology**.

6.2.2 Inference

The device descriptions can either use directly the virtual ontology which simplifies the matching. Or they inherit from the first level of the extension if the exact ontology of the virtual device is not known. Then the description logics and rules can be applied for inference. Overall, the ontology matching can be done on two levels [99, p. 65], [79, p. 143]: (i) structurally, and (ii) lexically. The structural level is based on the hierarchical comparison of the semantic network of ontology. The first extension of measuring and controlling subclasses will pay off here. The lexical level is the string-based comparison which is more fine-grained. The instance names will be analyzed and inferred from the *Abstract Classes* of the virtual device. Thus, a combination of both levels seems to be appropriated in this case.

One of the questions is how Semantic Web Standards based on logic formalism, namely OWL2, can be used for inference. We applied several mechanisms on Ontology Matching to investigate this question, see also [317]. In Section 6.2.1, ontology matching on combination of two levels, structurally and lexically, was introduced. Our use case is an aggregation of device capabilities: switch off all devices. The goal is to infer all switch off functionalities from different devices. We will evaluate how this can be achieved using different semantic web technologies.

The oneM2M ontology is defined in OWL 2 DL flavor. Therefore, the definition of properties and rules can be done with semantic tools on board:

- OWL constructs: <owl:Restriction>,<owl:someValuesFrom>, <owl:sameAs>, >owl:equivalentClass>, etc.;
- SWRL rules;
- SPARQL requests.

The example in the Listing 6.1 shows a Restriction in the OWL. Unfortunately, the possible names of the controlling functionality in the <owl:someValuesFrom> expression should be written as a whole word. Regular expressions would shorten this collection but are not provided by OWL.

The example in the Listing 6.2 shows a SWRL rule and a common String comparison method *containsIgnoreCase*. There is also another SWRL built-in <swrlb:matches>

Listing 6.1: OWL Abstract Class for switchOff Functionality

```
<owl:Class rdf:about="service#AbstractSwitchOffFunctionality">
<rdfs:subClassOf rdf:resource="onem2m#ControllingFunctionality"/>
<rdfs:subClassOf>
 <owl:Restriction>
  <owl:onProperty rdf:resource="ext-onem2m#hasControllingFunctionality"/>
   <owl:someValuesFrom>
    <owl:Class>
     <owl:oneOf rdf:parseType="Collection">
      <onem2m#ControllingFunctionality rdf:about="dev#switchOff"/>
      <onem2m#ControllingFunctionality rdf:about="dev#turnOff"/>
     </owl:oneOf>
    </owl:Class>
   </owl:someValuesFrom>
 </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

which satisfied if and only if (iff) the first argument matches the regular expression the second argument. The SWRL expression is an improvement compared to the OWL restriction.

Listing 6.2: SWRL Rule for switchOff Functionality

```
onem2m:Device(?d)
  ^ service:hasControllingFunctionality(?d, ?f)
  ^ swrlb:containsIgnoreCase(?f, "switchOff")
  -> AbstractSwitchOffFunctionality(?f)

onem2m:Device(?d)
  ^ service:hasControllingFunctionality(?d, ?f)
  ^ swrlb:containsIgnoreCase(?f, "turnOff")
  -> AbstractSwitchOffFunctionality(?f)
```

The example in the Listing 6.3 shows a SPARQL request. This allows filters which are defined by regular expressions. Besides, SPARQL has more expressiveness than the other two above and has more common implementations. On the other side, the result of a SPARQL request is a new RDF sub-graph. The device description of a virtual device is a kind of a new RDF sub-graph.

Listing 6.3: SPARQL Request for switchOff Functionality

```
PREFIX onem2m: <http://www.onem2m.org/ontology/Base_Ontology/base_ontology#>

SELECT ?functionality ?type
WHERE {
  ?functionality rdf:type owl:NamedIndividual.
  ?functionality rdf:type onem2m:ControllingFunctionality.
  FILTER REGEX (?functionality, "switchOff", "i").
}
```

The virtual device is running on an edge node with constrained resources. Therefore, the ontology matching must be as simple as possible as well as the underlying ontology itself. Additional OWL constructs and SWRL rules require more powerful processing nodes and software. Therefore, the SPARQL is sufficient for our intention. The virtual ontology and set of SPARQL requests for inference are pre-configured.

6.3 Conclusion

Virtual Device concept proposed in this chapter is more than just a digital representation of a device. It performs the aggregation of capabilities of real devices deployed at the edge. Some ontology extensions are required to perform the aggregation. Especially when these devices are heterogeneous in their capabilities, the stronger classification by class and property definition is required.

MYNO Security

7

> *Some things in life can never be fully appreciated nor understood unless experienced firsthand.*
>
> **— RFC 1925, Nr. 4**
> (The Twelve Networking Truths)

Security by Design should be always considered by developing a framework for the Internet of Things. First, an overview over security issues in the IoT is provided by related work. Then, related work about firmware updates over the air is introduced. This knowledge is applied to the MYNO framework. First, the MYNO framework is analyzed for threats and security requirements. Then, the design of the MYNO Update Protocol (MUP) is described because firmware update was identified as a security key feature.

7.1 Related Work: IoT Security

Security issues in the IoT are versatile. As a part of an IT infrastructure, the common security practice is also appropriated to the IoT. However, some IoT characteristics have particular security requirements. These requirements result from the WSN security research.

Leloglu [210] proposes a layered architecture to guide theoretical research in the IoT security, see Figure 7.1. He divides an IoT architecture into 4 layers: perception, network, support and application. The perception layer includes for example sensors and tagging technology to collect information from the physical world. The network layer contains for example WSN, mobile and IP-based networks to transfer the collected sensor data to the processing systems. The support layer involves transformation and storage of these data. Finally, the application layer includes domain applications based on user or industry requirements such as precise agriculture, smart home, etc.

The threats and related risks according to these layers were also classified by Leloglu [210]. Threats of perception layer and related security requirements are: spoofing (authenticity, integrity and confidentiality); signal/radio jamming (availability and integrity); device-tampering/node-capturing (availability, integrity, authenticity and confidentiality); path-based DoS attack (PDoS) (availability and authenticity); node outage (availability and authenticity); eavesdropping (confidentiality).

Threats of network layer are: selective forwarding, sybil attack, sinkhole attack (blackhole), wormhole, Man-in-the-Middle attack, Hello-flood attack, acknowledgement flooding. Threats of support layer are: tampering with data, DoS attack, unauthorized access. Threats of application layer: sniffer/loggers, injection, session hijacking, Distributed Denial of Service (DDoS), social engineering.

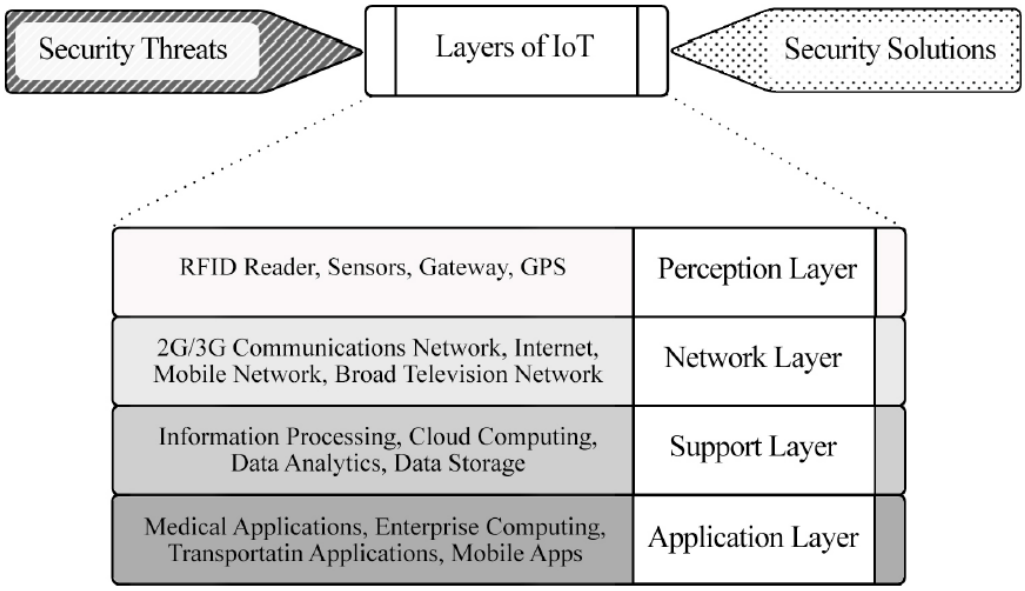


Fig. 7.1: IoT Architecture layers [210]

Leloglu discusses possible solutions on layers of IoT as shown in Figure 7.2. Physical identity and access management as well as cryptographic processing are possible solutions on the perception layer. Protocols for authentication and key management such as SSL/TLS or PPSK (Private Pre-Shared Key) can be applied to the network layer. Data and application access solutions have to be applied to the support and application layers. Leloglu reviewed the security research in the IoT and concluded that there are still many key security concerns which need more research effort to be resolved. To the same conclusion came also surveys from Alharbi et al. [8] and Haus et al.[141].

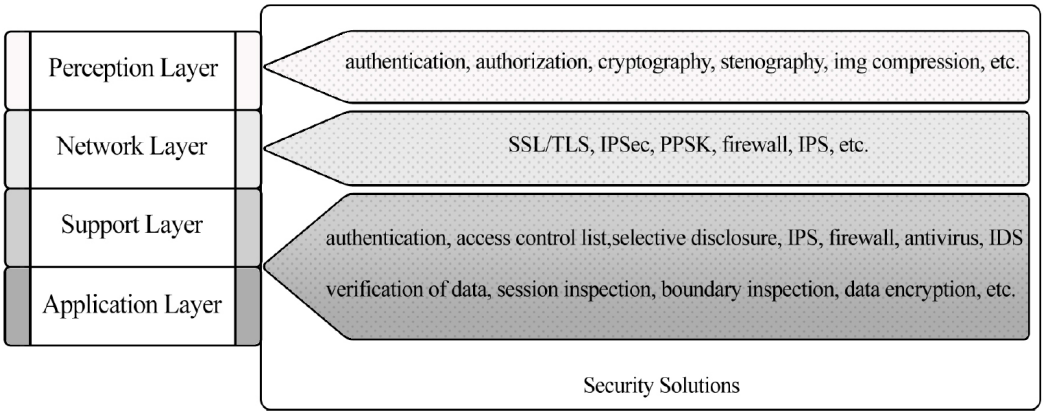


Fig. 7.2: Security solutions on layers of IoT [210]

Samaila et al. [322] provides a comprehensive survey on challenges of securing IoT devices. They analyzed IoT architectures and domains, defined security requirements, presented a system and threat models, as well as outlined typical protocols and communication technologies for nine application domains (e.g. smart home

and smart agriculture). The authors review five different security aspects, namely: cryptographic primitives, authentication protocols, hardware, specific application domains, and current security mechanisms. Finally, they presented a list with unresolved research challenges. There is need for lightweight solutions which address constrained devices considering cryptography, authentication, key management schemes.

The security of commercial IoT applications was surveyed by Ammar et al. [9]. Among these applications were Amazon AWS IoT, Microsoft Azure IoT Suite, Eclipse Kura. Although, most of them follow a similar philosophy in terms of identifying cloud-based applications by centralizing distributed data sources. The vendors follow various approaches in order to implement this philosophy.

Tschofenig and Baccelli [394] provide more concrete survey on cyberphysical security. They define three categories of attacks in the IoT considering hardware, software and communication. These categories are also reflected in the ENISA guidelines [98]. These categories are: authentication and communication security; object security; authorization and access control; key management; state-of-the-art crypto; restrictive communication; firmware and software updates. Tschofenig and Baccelli mapped these guidelines to areas of work in the IETF and analyzed them, see also RFC 8576 [112]. Many known standards are discussed: TCP and UDP; TLS and DTLS; CoAP, CBOR and COSE; drafts of Software Updates for Internet of Things (SUIT) [239] and Remote ATtestation ProcedureS (RATS) [36] working groups. Although there is various ongoing standardization work, the practice has still gaps and challenges. Sometimes, standardization are ahead of implementations. The security libraries are often difficult to integrate and have limited features and bugs.

Granjal et al. [121] surveyed the CoAP protocol for security issues with the stack of IPv6, RPL, 6LoWPAN, and IEEE 802.15.4. They discussed research challenges and proposals at different layers. The results are many mechanisms and proposals on every layer which can make IoT security quite complex.

Regarding MQTT, attack scenarios and security analysis of MQTT protocol were provided by Andy et al. [308]. In the local network, attackers can sniff and modify packet data from the network to attack data privacy, data integrity, and MQTT authentication mechanism. Therefore, using TLS is recommended. However, it is not always possible on constrained devices. Proposals for secure MQTT in the IoT were made in [365] and in [375]. The authors in [365] propose a version of MQTT and MQTT-SN protocols (SMQTT and SMQTT-SN) in which a security feature is augmented to the existing MQTT protocol based on Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) using lightweight Elliptic Curve Cryptography. Su et al. [375] proposed MQTT Thing-to-Thing Security (MQTT-TTS) which provides thing-to-thing security which prevents data leak because MQTT TLS provides security only between MQTT broker and a thing.

7.1.1 Over-The-Air (OTA) Updates

In case a security exploit is identified, the firmware should be fixed as soon as possible. Otherwise, the risk is that these IoT devices are hijacked and misused, e.g., as a botnet. Hence, a firmware update solution for such devices is essential to deal with vulnerabilities [394].

It must be ensured that patches and updates are only obtained from trustworthy sources. In the update process, there are two main security properties to prove:

1. *Authenticated firmware:* The device has to be able to verify that the received firmware is sent by a trustworthy source.
2. *Freshness of the firmware:* The freshness property claims that the device has to be able to verify that the new firmware has a higher version number than the installed firmware.

The freshness property is even more important in the IoT: to save energy, only *fresh* updates should be transferred to the IoT device. Hence, the freshness of the firmware should be proven before the firmware transmission is started. The freshness property also prevents replay attacks.

Samuel et al. [323] present The Update Framework (TUF) which builds the basis of the update system Thandy [225]. Thandy was originally developed for secure updates for the Tor project [388]. The design of TUF focuses on the security principle *survivability*, defined as the ability of the system to function correctly while under attack or partial compromise. For better resilience against key compromise, they propose the separation of duties and multi-signature trust. Multi-signature trust may be achieved by signatures of multiple roles or by threshold signatures, where at least t signers are required out of a set of n potential signers. Further, TUF uses a two-step approach where signed metadata describing the new update is downloaded and checked first before the update file is downloaded and installed. This two-step approach is also very suitable for the update of constrained devices and used in the presented MUP protocol (see Section 7.3).

Uptane [187], a software update system for automobiles, adapts TUF in order to address the specific automotive requirements. For example, it adds a director role at the repository site to blacklist faulty software and for customizing software when the vehicle owners may have paid for extra features. Further, they combine TUF with an edge computing architecture by adding the concept of primary and secondary components. The primary is connected to the software repository, downloads the new update and distributes it to the secondary components.

Then there are several research groups investigating update protocols for IoT devices. Some of them propose MQTT-based solutions ([382, 212, 109]) and others propose customized/proprietary solutions [207, 204]. Further, we discuss work in progress at the IETF [239, 238, 422].

The Open Mobile Alliance has specified a TLS-based update process within the Lightweight Machine-to-Machine (LwM2M) protocol [266]. LwM2M is based on the Constrained Application Protocol (CoAP) [356] and uses a pull approach where the device/client periodically polls a server for new updates. Then the client connects to the URI provided by the LwM2M server and downloads the firmware. In this architecture the devices need Internet access, while in the proposed MYNO architecture the constrained devices are separated from the Internet by the edge node.

Thantharate et al. [382] compare CoAP and MQTT for delivering software and security updates. They argue that in the IoT the constrained devices are meant to last for a number of years with limited power, and therefore most of the time the devices will be inactive (sleeping). For transmitting a new firmware, a robust transport of the data is necessary. Therefore, Thantharate et al. evaluate the performance of

MQTT and CoAP for reliable data transport. They compare MQTT with Quality of Service (QoS) level 1 (deliver the message at least once, with confirmation required) and QoS level 2 (exactly-once) against CoAP in CON mode (Confirmable messages). They use a simulator in their performance study. The results show that MQTT performs faster and has fewer spikes in transmission durations due to retransmissions. This is not surprising, since MQTT uses TCP, while CoAP uses UDP. Hence, they recommend MQTT for IoT updates. While Thantharate et al. draw their conclusion from simulation results, this work presents results with a real implementation of an update framework.

Langiu et al. [204] have recently proposed a new update protocol called UpKit dedicated for IoT devices. UpKit installs authenticated firmware and guarantees the freshness of the update. The benefit of that framework is that freshness is guaranteed without the use of an Network Time Protocol (NTP) server and authenticated clocks. The evaluation shows that UpKit has a small memory footprint. Further optimizations are the support for A/B updates and the support of differential updates. A/B updates require two bootable slots which keep two images, and the bootloader jumps to the newest slot. Differential updates reduce the amount of data which have to be transferred over the network which saves energy and the actual update time. The UpKit implementation supports CoAP or Bluetooth Low Energy [145] for communication. Our proposed MUP adapts this approach for an MQTT-based IoT environment.

Frisch et al. [109] also consider an over-the-air (OTA) update process via MQTT, but present no performance numbers. They experiment with the ESP8266 microcontroller board which has $96\,\mathrm{kB}$ main memory and integrated WiFi. Instead, we investigate the OTA update procedure for much more constrained devices and networks. Further, their approach does not guarantee the freshness of the firmware (version numbers are sent in clear text and not signed). Another weakness is that the firmware verification is done after download, but that is too late when dealing with constrained devices. If the verification fails, the device has spent much energy for the transmission of the malicious firmware. This makes Denial of Service (DoS) attacks possible. MUP avoids this weakness by a two-phase approach similar to UpKit [204] and TUF [323].

An approach for the secure distribution of firmware using MQTT is proposed by Lo and Hsu [212]. However, the MQTT protocol is only used between the firmware patch server, the firmware broker server and the gateway. The gateway is connected to the Internet and communicates with the IoT devices via wireless connections such as Wi-Fi or Bluetooth. The protocol between gateway and device is not further specified. In opposite, the proposed MUP protocol relies on MQTT for the communication with the devices. Further, while MUP only needs to pre-install the public vendor key, in the approach of Lo and Hsu one secret value and one secret key have to be pre-installed on the devices.

Laukkarinen et al. [207] present the design and implementation of a firmware update protocol for resource constrained Wireless Sensor Nodes (WSN). They propose the use of a Message Authentication Code (MAC) for integrity checking. This needs shared secret keys between update server and device. Again, this approach is not scalable for vendors. Instead, MYNO uses signatures, and therefore only public keys have to be distributed.

There is work in progress at the IETF [239, 238]. The draft for software updates for IoT (SUIT) [239] assumes asymmetric cryptography and a public key infrastructure. A data structure called manifest [238] specifies 24 elements with detailed information about the firmware. The manifest has an optional Expiration Time, but this needs a secure source of time which is not available on most IoT devices. Instead, our proposed MYNO Update Protocol uses a Nonce to avoid replay attacks. Zandberg et al. [422] implemented and evaluated a prototype to compare the surveyed firmware update methods, among them the SUIT-OTA update. They use CoAP blockwise transfer to pull the firmware image onto the device.

7.2 MYNO Security Analysis

Every component and communication channel can have vulnerabilities and therefore can be a target for attacks. The distribution of the MYNO framework components in a 6LoWPAN network is visualized in Figure 7.3.

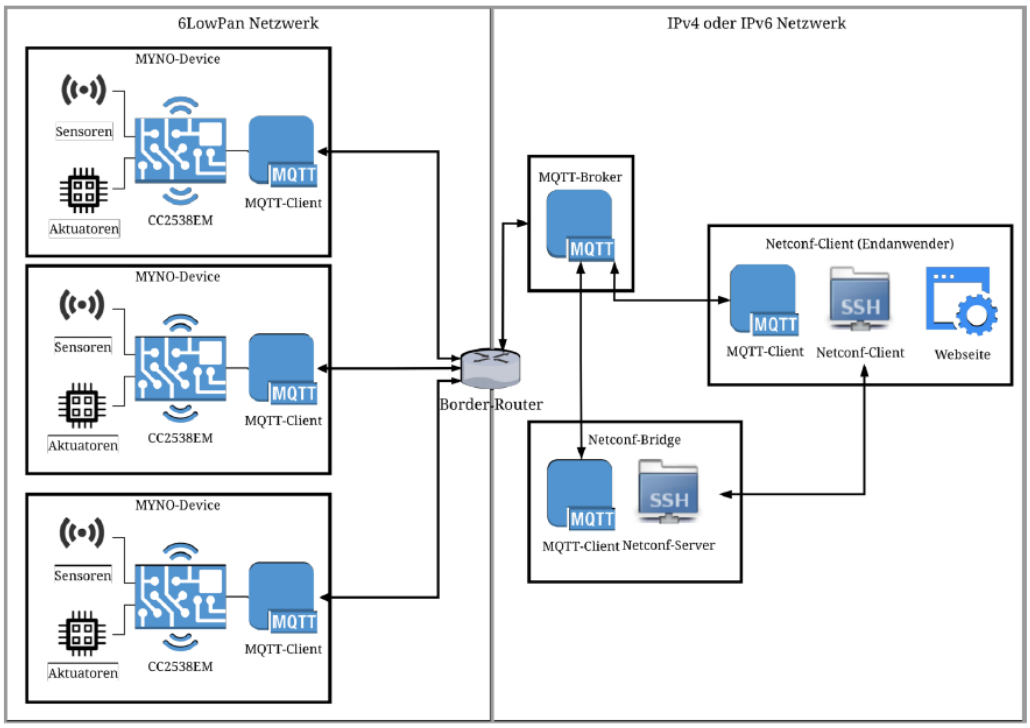


Fig. 7.3: MYNO network components [261]

The main security goals (also called protection goals) for the MYNO framework are identified as following:

- **Availability**: The operation of the infrastructure is not interrupted and the system can be used continuously.
- **Confidentiality**: Information that is exchanged or stored in the framework is not accessible and not visible to third parties.
- **Integrity**: Information remains correct anytime during communication or storing. Third parties are not able to manipulate information unauthorized and unobtrusively.

Further security goals are authenticity (i.e. components are authenticated) and non-repudiation (i.e. an attack is monitored) are indirectly included in the above mentioned goals.

The security analysis of the MYNO framework is surveyed according to the security guide of the Federal Office for Information Security (BSI) [170]. First, a threat analysis was accomplished. Then, security requirements for MYNO framework were identified and prioritized. Finally, the most important security goal was chosen for design and implementation.

7.2.1 Threat Analysis

Every MYNO component is analyzed for possible threats according to the security goals [261]: availability, confidentiality and integrity. Such components are IoT devices, border- or WLAN-router, MQTT broker, NETCONF-MQTT bridge, NETCONF client, as well as communication channels between the components.

Availability of the distributed components is threatened if they are out of order. This is particular the case for the MQTT broker and the NETCONF-MQTT-bridge. They could be attacked e.g. by a DoS attack. If the Border- or WLAN-router is not available, the devices cannot connect to the network. The NETCONF-client is a web-server which can be set out of order by the common attacks for web-based components. If some devices are not available, there might be lack of sensor data.

Confidentiality of data is threatened during transport in 6LoWPAN and WLAN network between the devices and MQTT-broker. This communication channel is potentially insecure and therefore deserves protection. The communication between MQTT-Broker and NETCONF-MQTT-Bridge can be secured by using the security mechanisms provided by MQTT standard. The communication between NETCONF-MQTT-Bridge and NETCONF-Client uses SSH and is therefore potentially secure.

Integrity of data can be ensured when communication parties can be authenticated and authorized. The NETCONF-Client uses SSH to exchange data with NETCONF-MQTT Bridge and therefore fulfills integrity. NETCONF-MQTT bridge and devices can use user/password to authenticate to the MQTT broker. The MQTT broker can authorize them based on this information. But there is no mechanism for the MQTT clients to authenticate the broker. TLS Certificates can be used for connection with MQTT-broker.

Many threats and countermeasures were identified. The next step is to identify the security requirements and to prioritize them.

7.2.2 Security Requirements for MYNO

The security requirements for MYNO are modeled according to BSI guide [170], see Table 7.1. The modeling results in three security requirements [261, p.20]: data traffic, management interface and firmware update process. Data traffic between the bridge, the MQTT Broker, the 6LoWPAN border router and the CC2538dk boards should be secured. Security mechanisms can be applied on the 6LoWPAN layer (e.g. using 6TiSCH [88]) or on the MQTT layer (e.g. using TLS) or using cryptographic libraries on the boards. Such mechanisms require a lot of effort and are only partly useful for all devices. A management interface (e.g. as proposed in [121]) could have tasks such as state monitoring of the devices, key provisioning, analyzing

network state and providing a early warning system for attacks. However, such network monitoring tools are customized to the user requirements and could be quite complex.

Tab. 7.1: Modeling according to BSI guide [261, p.12]

MYNO Components	BSI System Components
MYNO-Device	SYS.4.4 Allgemeines IoT-Gerät
Border-Router	NET.3.1 Router und Switches
MQTT-Broker	SYS.1.1 Allgemeiner Server
NETCONF-Bridge	SYS.1.1 Allgemeiner Server
NETCONF-Client	SYS.1.1 Allgemeiner Server, APP.3.1 Webanwendungen, APP.3.2 Webserver

Finally, the firmware update process was identified as the most prioritized security requirement for MYNO framework. Recent incidents in the IoT where no firmware update process is supported by the vendors[42] show how important such a feature is. When a security bug is discovered and fixed, all devices must get a firmware update as soon as possible. The concept for the MYNO update process is introduced in the next section.

7.3 Update Over-The-Air (OTA) with MYNO

We propose an update protocol for IoT devices managed by the MYNO framework, called MUP. In the MYNO architecture, the IoT devices are separated from the Internet via a gateway running on the edge node. Since the edge node is a more powerful device compared to the IoT devices, it is suited to overtake more complex and energy-consuming tasks, for example to distribute firmware updates. This section shows how a secure update process can be integrated into the MYNO architecture, shown in Figure 7.4.

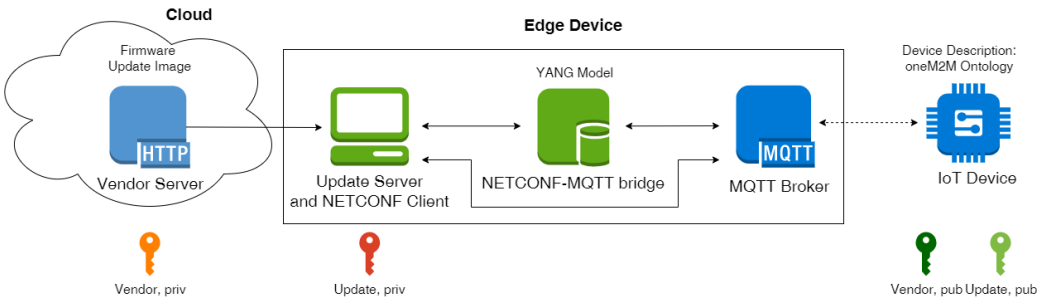


Fig. 7.4: System architecture of the MYNO Update Protocol (MUP)

7.3.1 Prerequisites

The NETCONF protocol provides only operations for configuration update. However, RPC calls can be used to define further operations like a firmware update on the device. Such device capabilities are described in the device description.

[42] https://www.heise.de/security/meldung/Ripple20-erschuettert-das-Internet-der-Dinge-4786249.html

During the *bootstrap process*, the device description is published by an IoT device to the MQTT broker. The NETCONF-MQTT bridge parses this ontology and generates the YANG data model with corresponding RPC operations. The device description has to be extended for the update capabilities. After the bootstrap process, the update functionality is activated in the NETCONF client.

Private and public keys must be distributed before the update process starts. The vendor possesses a Private/Public key pair $(K_{pub}^{Vendor}, K_{priv}^{Vendor})$, and pre-installs the vendor public key K_{pub}^{Vendor} on the device. The Update Server possesses also a Private/Public key pair $(K_{pub}^{Update}, K_{priv}^{Update})$ and propagates its public key K_{pub}^{Update} to the device during the bootstrap process.

7.3.2 MYNO Update Protocol (MUP)

We adapt the UpKit approach [204] where the Update Server at the edge verifies the freshness of the firmware before it is transmitted to the device.

We designed the update process as an push approach. The network administrator initiates the download of a new firmware image from the vendor server (see Figure 7.4). The vendor provides the firmware, and a so-called `vendor manifest` (see Table 7.2) which describes the firmware image characteristics such as size and version number. The `manifest` includes also a vendor signature called *inner signature*.

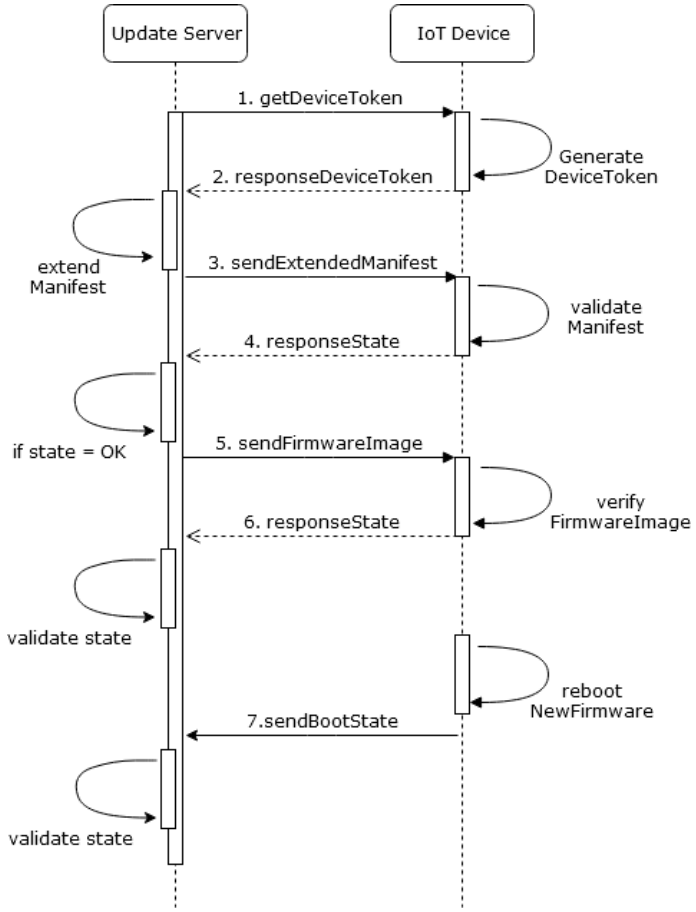


Fig. 7.5: MYNO update protocol as a sequence diagram

The MYNO update protocol is shown in Figure 7.5. For simplicity, we omit the components NETCONF–MQTT bridge and the MQTT broker because they are agnostic to the messages and act only as intermediaries during the update process. Messages starting with `response` are sent as a MQTT response which were introduced in MQTT v5 [243]. The update protocol works as follows:

(1) The Update Server requests a `device token` from the IoT device.

(2) The `device token` contains the device Universally Unique Identifier (UUID), the current version of the firmware, and a nonce (see Table 7.3). The generated token is sent within a response.

(3) The `device token` information is used to generate the `extended manifest` which grants the freshness of the update: The Update Server appends the device UUID, version and the nonce from the `device token` to the vendor manifest and signs this `extended manifest` with his private key (see Table 7.2). Now the `extended manifest` carries a double signature. The `extended manifest` is then sent to the device for validation.

(4) The device validates the `extended manifest` using the public keys of the vendor and Update Server. The following fields are checked for the freshness of the firmware: nonce and device UUID must be the same as sent before with the device token. Further, the new version must be higher. The old version is required for differential updates only. If the `extended manifest` was successfully validated, the device responds with the state ok.

(5) If the `responseState` was ok, the Update Server starts the transmission of the firmware image.

(6) When the firmware is fully transmitted, the device performs an integrity check: It calculates a digest of the firmware and compares it with the digest included in the manifest. Since the digest in the `vendor manifest` was correctly signed by the vendor, this proves the authenticity of the firmware. If this verification is successful, the device responds with the state ok.

(7) The device reboots with the new firmware and notifies the Update Server about success.

If the validation of the manifest or the firmware is not successful, the update process will be cancelled by the device. If the reboot fails because of other reasons, the Update Server will get a timeout and reports the error.

7.3.3 MUP Security Discussion

In this section, we show that MUP achieves the security properties defined in Section 7.1.1. Further, we discuss its robustness against resource exhaustion. However, the security aspect is linked with configuration and management effort. Therefore, we also discuss MUP's key distribution process.

Tab. 7.2: MUP Manifest

Vendor Manifest	
Field	**Description**
App ID	unique id for application
Link offset	memory address
Digest	hash value of the firmware
Size	size of the firmware in bytes
New Version	new firmware version
Old version	old firmware version
Inner signature	vendor signature
Manifest Extension	
Device UUID	unique device ID
Nonce	nonce generated by device
Outer signature	Update Server signature

Tab. 7.3: MUP Device token

Field	Description
Device UUID	unique device ID
Nonce	nonce generated by device
Version	current firmware version

Guaranteed Security Properties

The proposed MYNO update protocol achieves both security properties: an authenticated firmware and freshness of the firmware.

Two steps are necessary for the authentication of the new firmware. First, the device validates the vendor manifest using the public key of the vendor. If the validation is correct, the device has trust into the digest of the manifest (which is the hash value of the firmware update). In a second step after the download of the firmware, the device checks whether the received firmware corresponds to the manifest. Therefore, it calculates the firmware digest and compares it with the digest in the manifest. If they are the same, the device has also trust in the firmware.

The new version number is signed by the vendor in the vendor manifest and checked by the device whether it is higher than the current version number. Further, the freshness of the firmware is guaranteed by the double signature process where a nonce is generated by the device and signed by the Update Server in the extended manifest. Since also the device UUID is included in the signature, this challenge is unique for each device.

Replay Attacks

The MUP protocol does not rely on TLS. All messages are sent in clear text. Hence, an adversary may resend these messages to initiate more firmware updates. Even installing the same firmware again and again would be a DoS attack ending when

the device battery is empty. Lo and Hsu [212] rely only on signed version numbers to guarantee the freshness of the firmware. MYNO follows the UpKit approach and use nonces to verify the freshness of the firmware. This hardens the protocol against First-Pre-Image-attacks since the time for an attacker to prepare such an attack is shortened. Further, the window for the DoS attacks is minimized in MUP because a device subscribes to the topic for the firmware image just before receiving it (before step 5) and unsubscribes as soon as the firmware image is received (after step 6).

Confidentiality

There may be several reasons why a firmware vendor may prefer to send the firmware update encrypted. First, this may be important due to licensing. Since the transmitted firmware is opaque to MUP, the vendor may send the firmware update encrypted and the Update Server forwards it to the device. However, this assumes appropriate keying material on the device.

Alternatively, the connection channel can be encrypted. The channel between vendor and update server may be secured by TLS [295]. The MUP messages may also be sent encrypted by MQTT over TLS, but this has to be supported by the device. For example, the Arduino Nano 33 IoT supports MQTT over TLS, while Contiki-NG does not support it [241].

Since support for efficient encryption is considered as an important feature, crypto chips are getting more wide-spread in IoT devices. For example, the Arduino Nano 33 IoT [13] is equipped with the crypto chip ATECC608A. The crypto chip has a data zone where up to 16 keys or compressed certificates may be stored [14]. We evaluated TLS on the Arduino Nano 33 IoT in Section 9.4.

Edge Architecture and Man-in-the-Middle Attacks

In cloud based solutions like the Arduino IoT Cloud [12], AWS IoT [15] or IBM IoT Cloud [159], the devices are directly connected to the Internet. In contrast, MYNO relies on an edge computing architecture where the IoT devices are separated from the Internet via a gateway. Hence, all Internet connections are terminated at the edge node.

If an attacker breaks into the edge node system, he has a powerful man-in-the-middle position and may for example inhibit software updates. Hence, the edge node has to be managed with the same care as every other machine which has Internet connection.

Key Distribution and Update

The MUP protocol requires two public keys on the devices: the pre-installed public vendor key and the public key of the Update Server. While mechanisms for key distribution and key update are important building blocks belonging to a security architecture, we do not focus on this topic here. A survey of key bootstrapping protocols in the Internet of Things based on public-key cryptography can be found in [216].

Lo and Hsu [212] propose Diffie–Hellman for key exchange, since they have to create different keys for every device. Instead of managing secrets for every device, MUP uses public/private key pairs. Only the public key of the vendor has to be pre-installed, and the public key of the Update Server is propagated during the bootstrap phase. This makes MUP scalable.

Currently, MYNO uses the bootstrap process also for the distribution of the key of the Update Server. The bootstrap process consists of two steps (see Figure 7.6): (1) when a new device enters a network, it publishes its device description; the NETCONF–MQTT bridge parses this description and adds the device to the YANG model; (2) if successful, the bridge publishes this state and the public key of the Update Server to the response topic of the device. Obviously, this approach is vulnerable against eavesdropping. In case an attacker is able to reply faster than the update agent to the first message, the device will take over the wrong key and get compromised, too.

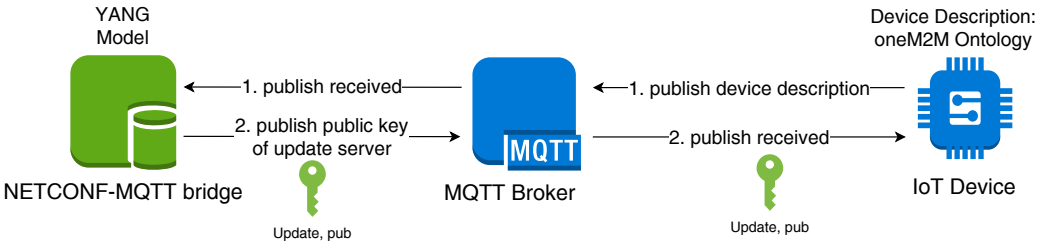


Fig. 7.6: Bootstrap process in MUP

In the Arduino IoT Cloud solution [12] the Arduino certificate is stored as trust anchor on the device. During bootstrap the Arduino client sends a *Certificate Signing Request* to the Arduino Cloud to generate a client certificate. In a similar way, the bootstrap phase of MYNO can be improved by using self-signed certificates where the vendor certificate is used as a trust anchor. The Update Server has to be equipped with a certificate signed by the vendor. This certificate can be verified by the device using the pre-installed vendor certificate.

An update of the vendor key may be supported by the Update Server running on the edge. For the verification of the new vendor key, the Update Server may employ DNSSec/DANE [82, 151].

Robust against Resource Exhaustion

Considering the constraints of IoT devices in terms of network bandwidth, memory, storage and energy, the update process must ensure that no unnecessary data transmissions and reboot occurs. The proposed MUP protocol ensures this by two steps: In the first phase, only the extended manifest is transferred to the device and checked. The extended manifest is much smaller than the new firmware. If the validation of the manifest guarantees freshness, the firmware will be downloaded in the second phase. This separation avoids unnecessary transfers and reboots.

Since an attacker could periodically send manifests promising a new update without a valid signature, the signature verification process may drain the battery. Since MUP is built upon the UpKit approach and uses an extended manifest, the device

checks first the manifest extension carrying the Nonce. A correct signature of the manifest extension authenticates the Update Server as the origin.

Usability

The additional effort on the vendor side is minimal: The devices have to be pre-installed with the following components: the device description, the public vendor key, the application and bootloader. Further, the bootstrap protocol has to support the exchange of the public key of the Update Server.

The presented vendor manifest and the manifest extension carry no information about the used crypto algorithms. This approach is not feasible in a productive environment with devices from different vendors. Hence, the vendor manifest has to be extended with this information. At the IETF, there are efforts underway for standardizing a suited manifest that describes the firmware image and processing steps [238]. This approach may also be combined with MUP.

MUP has been easily integrated into the existing MYNO architecture. Only a few adjustments were necessary on the bridge (e.g., adding a binary data type for signatures for new parameters). The update server is an extension of the NETCONF client for the web-interface.

The automated distribution of update images to IoT devices is relevant to prevent security gaps. MUP can be easily integrated into the DevOps processes and Continuous Delivery (CD) pipeline using scripts.

7.4 Conclusion

We identified firmware update as a main security issue for IoT devices to prevent the dissemination of security risks. We propose an MQTT-based architecture for IoT management and show how the update process is integrated. We called the proposed protocol as MYNO Update Protocol (MUP). We assess the security properties of MUP and show that MUP fulfills the security properties *authenticated* and *fresh firmware* and is also guarded against replay attacks.

Prototype Implementation

8

> *It Has To Work.*
>
> **— RFC 1925, Nr. 1**
> (The Twelve Networking Truths)

In the previous chapters, the concepts of the MYNO framework, the ontology-based device descriptions, a virtual device and the MYNO update protocol were introduced. The feasibility of these concepts will be shown through a prototype implementation. First, the software libraries for the MYNO components were chosen as the base for implementation. Then, the hardware for IoT devices was selected, implemented and corresponding device descriptions were created. Afterwards, the processing of these ontology-based device descriptions is outlined. Finally, the special features of the implementation for the MYNO update protocol are described. For all implementations, experienced challenges and problems are also documented in this chapter.

8.1 MYNO Framework Components

The MYNO framework has several components which must be implemented and configured. As the starting option, we rely on the open-source implementation provided by Scheffler [327]. Python 3 was chosen as the programming language for all components running on the edge. Python language became popular in recent years because of simplicity for developers and the combination of functional and object-oriented programming. This homogeneity also simplifies the development process and reduces effort.

First, an MQTT broker software is necessary. The broker software is interchangeable as long as it implements the MQTT specification. The most known commercial broker is HiveMQ [149] but it is not considered here due to the high cost. There are several open-source broker: for example, HBMQTT[43] in Python, Apache ActiveMQ Artemis[44] in Java, Aedes (former Mosca)[45] in Node.js; three brokers in Erlang as EMQ[46], VerneMQ [47], RabbitMQ Server[48]. However, they support only MQTT protocol versions 3.1 and 3.1.1 and it is not clear whether they will be developed further.

We use the open-source software Eclipse Mosquitto v1.6.10 [90]. Mosquitto is developed in C and C++ programming languages. Mosquitto is suitable for all devices from low power single board computers to full servers. Mosquitto was originally developed by IBM, and is still actively supported. This broker claims to

[43] https://hbmqtt.readthedocs.io
[44] https://activemq.apache.org/components/artemis/
[45] https://github.com/moscajs/aedes
[46] https://emqtt.io
[47] https://vernemq.com/
[48] https://github.com/rabbitmq/rabbitmq-server

implement the MQTT protocol versions 3.1, 3.1.1 and 5.0. We use the version 3.1.1 because the v5.0 was recently specified and is not fully supported. We evaluated the support for v5.0 together with the MUP protocol in Section 9.3.

The implementation of the second and third components, namely the NETCONF-MQTT Bridge and the web-based NETCONF client, is introduced in the next two subsections. The MQTT broker, the bridge and the NETCONF client are running on a Raspberry Pi 3B at the edge of network.

8.1.1 NETCONF-MQTT Bridge Implementation

The NETCONF-MQTT bridge mediates between two separate network management domains: the constrained network via MQTT and the public network with NETCONF configuration management. The NETCONF-MQTT bridge translates dynamically the semantic device descriptions into a YANG model. Further, it acts as a NETCONF server interface and translates the RPC calls into MQTT messages.

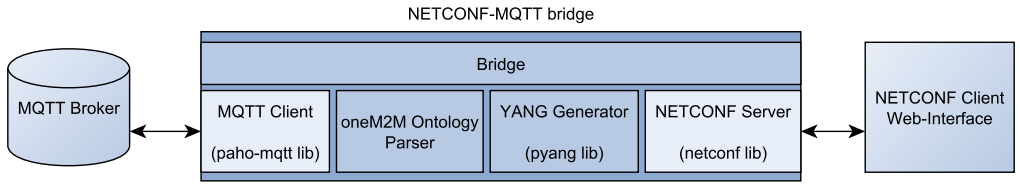


Fig. 8.1: NETCONF-MQTT bridge software components

Software libraries for the bridge were chosen by the following criteria: Python 3 as the programming language; open source software; active support and development of libraries; support of NETCONF specification; documentation.

The software architecture of the NETCONF-MQTT bridge is shown in Figure 8.1. We decided to use following libraries for the implementation of the NETCONF-MQTT bridge:

- NETCONF Server [253] (v0.5.3, since 2020 v2.1.0);
- Pyang [282](v1.7.3, since 2020 v2.4.0);
- Eclipse Paho MQTT Client [271] (v1.3.0, since 2020 v1.5.0);
- Paramiko SSH library [273] (v1.17.6, since 2020 2.7.2);
- RDFLib [293] (versions 4.2.2, since 2020 v5.0.0);

The NETCONF Server library enables to implement the NETCONF server interface. The NETCONF server offers an access over SSH [411]. This access was implemented by the Paramiko SSH library.

The Pyang library is used for generation and validation of a YANG model. Pyang is compatible with the following IETF RFCs regarding YANG, among them are RFC 6020 [38], RFC 7950 [37], RFC 8040 [32].

The Eclipse Paho MQTT Client allows to implement the access to a MQTT broker. This library offers client implementations in many programming languages.

The RDFLib library was chosen for parsing of the ontology-based device descriptions. This library is discussed separately in Section 8.3.1.

The versions of the libraries were updated over time. The first version of MYNO was implemented in 2017 [315]. Last changes to the MYNO framework were done in 2020 [313].

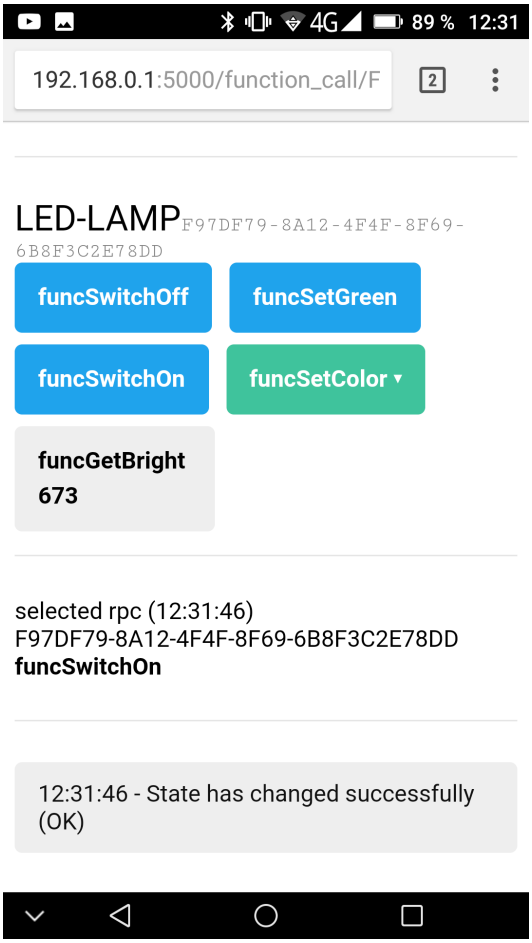


Fig. 8.2: Web-based NETCONF Client in Browser

8.1.2 Web-based NETCONF Client Implementation

At the beginning of our implementation project, we used the Python library for NETCONF client [251] by command line. Then, we implemented a web-based client [259].

Following libraries were chosen for the web application:

- Flask [107](v0.12.2);
- Flask MQTT [106] (v0.0.9);
- ncclient [251] (v0.5.4);
- Pyang [282](v1.7.3, since 2020 v2.4.0);

The Flask was chosen as the web framework in Python. Even official Raspberry Pi projects recommend to use Flask for Web applications because of its lightweight implementation. Alternatively, Django[49] was considered but was too heavyweight for our purpose.

The Flask MQTT library was used as MQTT client for flask to receive sensor values in the web application. The ncclient library was used for accessing the NETCONF-MQTT bridge.

The Pyang library was needed for converting the YANG model to YIN format. YIN [37] format is an XML syntax of a YANG model. Using XML format simplifies the processing of the YANG model because the native Python interface supports XML processing and no further libraries are required.

For dynamic updating of the web interface (e.g. new devices or new sensor values), the AJAX technology was used. AJAX is an acronym for Asynchronous JavaScript And XML and is based on JavaScript. The web-based client can be accessed via URL in a browser or smartphone, see the screenshot in Figure 8.2.

8.2 IoT Devices

One of the goals of the MYNO framework is to manage heterogeneous IoT devices. Thus, different microcontroller boards must be implemented for the prototype. We defined following requirements for selection of such boards:

- support for 6LoWPAN or WLAN connectivity
- common programming software interface
- software support for MQTT protocol and TCP
- constrained resources: RAM and ROM storage, energy
- sensors and actuators equipment and extension

According to these requirements, following hardware was chosen, see Table 8.1. The Arduino Nano 33 IoT and Raspberry Pi Zero were used only for evaluation of selected aspects, see Section 9.

[49] https://www.djangoproject.com/

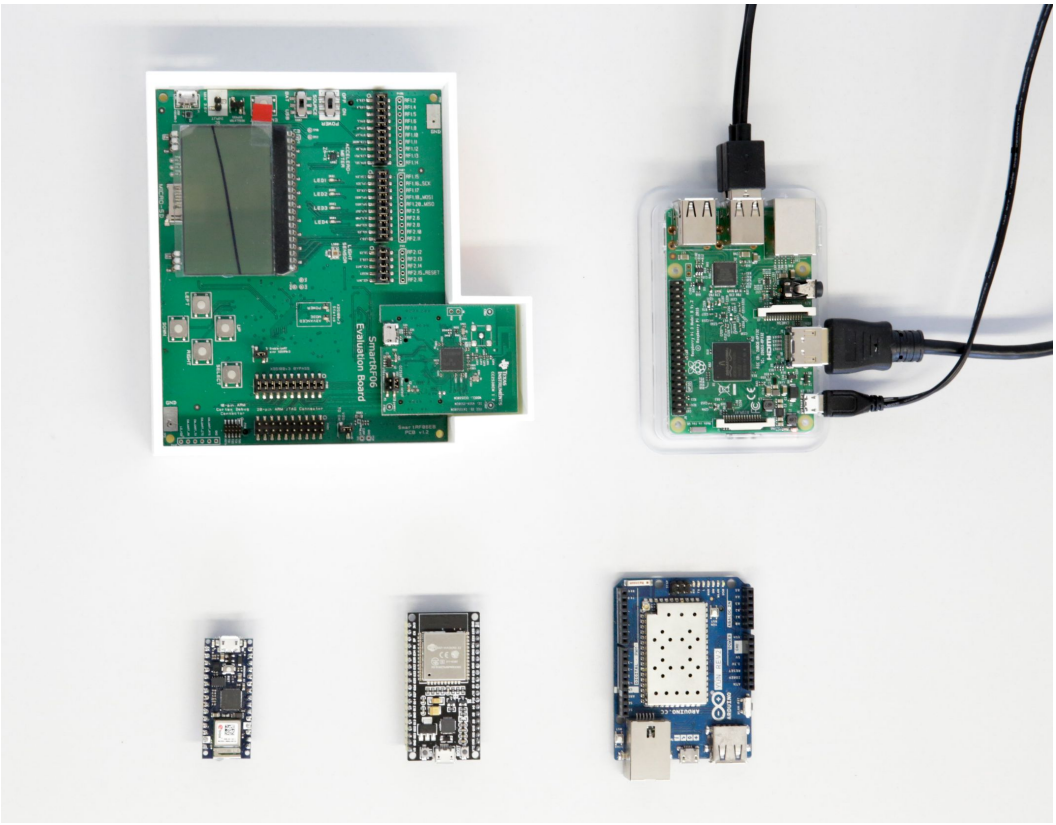


Fig. 8.3: Used Devices from left to right: CC2538dk board and Raspberry Pi 3B at the top; Arduino Nano 33 IoT, ESP-32 NodeMCU and Arduino Yún at the bottom

Tab. 8.1: Device Zoo for the MYNO framework

Requirement	CC2538	Arduino Yún rev. 2	ESP32 NodeMCU	Raspberry Pi 3B	Arduino Nano 33 IoT	Raspberry Pi Zero w
Price per board	499.00 USD	49.00 EUR	8.99 EUR	45.90 EUR	16 EUR	20 EUR
Processor	32-bit Arm Cortex-M3	ATmega32U4, Atheros AR9331	160MHz Tensilica L108 32 bit Dual-Core CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU	SAMD21 Cortex-M0 + 32bit low power ARM MCU	1GHz single-core CPU
Network	6LoWPAN	WLAN	WLAN	WLAN	WLAN	WLAN
Software	Contiki OS, Contiki-NG	Arduino IDE, OpenWRT	Arduino IDE	Raspbian OS	Arduino IDE	Raspberry Pi OS Lite
Programming language	C	C/C++, Python	C/C++	Python	C/C++	Python
MQTT library	Texas Instruments mqtt-client for Contiki	paho.mqtt.client	Arduino Client for MQTT	Mosquitto broker	ArduinoMqttClient	–
RAM	32 KB	2.5 KB, 16 MB DDR2	512 KB SRAM	1 GB RAM	32 KB	512 MB RAM
ROM/Flash	512 KB	32 KB, 64 MB	16 MB	16 GB (SD Card)	256 KB	16 GB (SD Card)
Energy	battery, Micro USB	Micro USB	Micro USB	Micro USB	Micro USB	Micro USB
Sensors	light, on-chip temperature, voltage, buttons (onboard)	extended over breadboard	extended over breadboard	none	extensible	none
Actuators	LEDs (onboard)	extended over breadboard	extended over breadboard	none	extensible	none

8.2.1 CC2538 Development Kit

The constrained device of Class 1 in our experimental setup is a CC2538EM microcontroller board of the Development Kit [59] from Texas Instruments (TI). The TI-board is an ARM Cortex M3-based and supports the IEEE 802.15.4 and 6LowPAN standards. The TI-Board is a very constrained device with 32 kiB RAM and 512 kiB Flash memory. The TI-Board can be powered by two batteries of AAA type or by electricity over micro USB. The CC2538 Development Kit consists of two microcontroller boards with a chip and antenna plugged on the two development boards with micro USB, sensors and actuators, micro SD card slot and I/O pins, LCD display, and a CC2531 USB Dongle for packet sniffing. The development kits are intended for industrial employment.

Texas Instruments provides a foundation firmware which is implemented in C and supports low level programming. This makes it difficult to extend it for support of higher protocols. The protocol stack for the communication with the constrained devices consists of the MQTT, TCP, IPv6, 6LoWPAN, IEEE 802.15.4 and must be supported by these microcontroller boards. The CC2538 board provides just enough memory and storage to run an operation system.

We run Contiki OS [71] and its implementation of MQTT client on the TI-Board. Contiki-OS is an open-source library with a broad support for different microcontroller boards and supports the desired protocol stack. One of the reasons for the hardware choice and Contiki OS was the fact that we already possessed several CC2538 boards for evaluation and had experience with Contiki OS [326]. The other reason is that Texas Instruments actively develops Contiki and also provide hardware support for new boards like CC26xx and CC13xx families[50].

Other known OS candidates for IoT devices are TinyOS and RIOT. TinyOS[51] is one of the oldest operation systems for WSN (first release in 2000) but the most recent release, TinyOS 2.1.2, was in 2012 and thus, TinyOS achieved obviously the end of its cycle. The code is now provided on Github[52] as open source and is not actively developed. The RIOT OS [18, 17, 299] is a new OS for IoT, first introduced in 2013. RIOT is significantly supported by Freie Universität Berlin, by INRIA (France), and by Hamburg University of Applied Sciences. RIOT claims to support CC2538dk boards, 6LoWPAN and MQTT-SN protocol. While RIOT supports more features [18] in comparison with other operations systems, the RIOT still doesn't support the MQTT protocol.

The CC2538dk implementation for the MYNO project included following sensors and actuators: light sensor, on-chip temperature sensor, voltage sensor, button sensor (selection button), LEDs lights.

These controlling functions were modeled in the device description: `setColor` with parameters green, red and yellow; `setGreen` without parameters; `switchOn` a LED; `switchOff` a LED.

These measuring functions were modeled for sensors in the device description: `funcGetBright`, `funcGetSelButton`, `funcGetTemp`, `funcGetVoltage`.

[50] `https://github.com/contiki-ng/contiki-ng/wiki/Platform-simplelink`
[51] `http://www.tinyos.net/`
[52] `https://github.com/tinyos/tinyos-main`

The implemented functions are shown in the web-based NETCONF client in Figure 8.4. The colorful buttons represent the actuators and the grey sensor fields show the values.

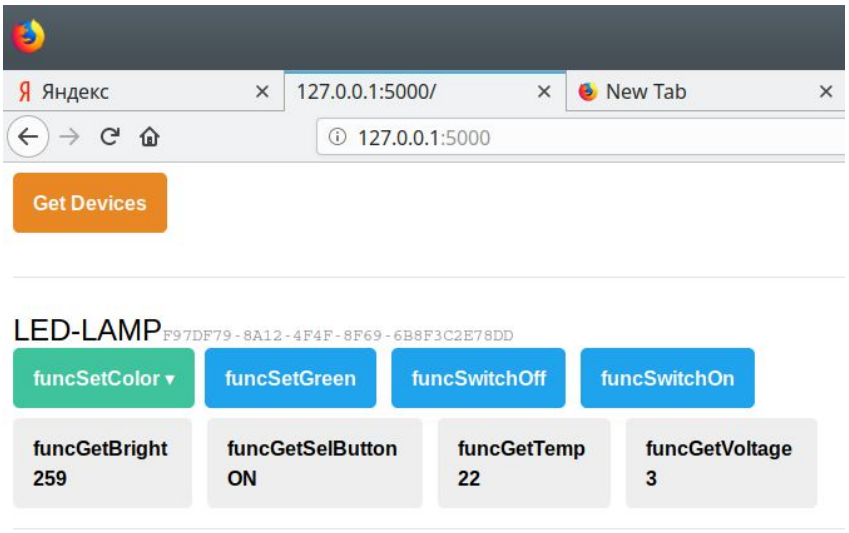


Fig. 8.4: CC2538dk Functions in the web-based Client

Contiki OS and Contiki-NG

Contiki OS [71] was founded by Adam Dunkels as a part of his thesis in Sweden [383]. After refining the OS with the help of community and some big players (Texas Instruments, Atmel, Cisco, Redwire, SAP, Sensinode, Zolertia, RWTH Aachen University, Oxford University, ETH Zurich, SICS (Swedish Institute of Computer Science) and others) the Contiki OS started in 2003 as an open source (BSD license) IoT operating system. The main goal of the Contiki development was the support of IP-based internet protocols, especially IPv6.

In year 2008, Cisco, Atmel, and the Swedish Institute of Computer Science released uIPv6 [418], as open source for the Contiki OS. They claim that it is the world's smallest IPv6 stack. The intent was to bring IP addresses to the masses by giving devices such as thermometers or lightbulbs an IPv6 stack. They claim also: "with a code size of 11 kilobytes and a dynamic memory usage of less than 2 kilobytes, it certainly fits the bill of the ultra-low-power microcontrollers typically used in such devices".

Finally, these features were developed in Contiki OS:

- support constrained devices (8 to 32 bit MCU, kBs RAM)
- operate in low-power systems (even run on batteries)
- IPv6-ready stack: 6LoWPAN, RPL, MQTT, CoAP, 6TiSCH, etc.
- Cooja simulator

Since the last release of Contiki OS v3.0 in 2015, it is maintained by Thingsquare. The company was founded by Dunkels and built the cloud backend for Contiki

devices. But there are deprecated features, outdated development practice and no new roadmap or release cycles [364] are planned.

Therefore, the fork project Contiki-NG (NG stands for Next Generation of IoT Devices) started in 2017 by co-founders Simon Duquennoy and others, and distributed under the terms of the 3-clause BSD license. The project is supported by RISE SICS (RISE Research Institutes of Sweden), Bristol University and Toshiba. The goals of Contiki-NG development are:

- re-focus on modern protocols (additionally to IPv6-stack, RPL-Lite, LWM2M) and platforms (16-32 bits with low-power radio, homogenized interfaces for hardware features)
- focus on dependable communication: security (link-layer(IEEE 802.15.4) and application layer(DTLS) and reliability (RPL/TiSCH mesh, RPL Lite)
- improved development cycle: public roadmap, periodic release (2/year), current v4.5 in 2020
- improved and modern documentation, tutorials and communication channels for community
- usability: rehauled configuration system, new data logging and shell system
- systematic continuous integration(CI)

The Contiki operating system (OS) uses a Protothread, which combines multithreading and event-driven programming. On the hardware side, the Contiki project provides hardware schemes so that developers can build their own Contiki boards. The Contiki programming language uses C syntax for writing programs. Contiki provides hardware abstractions that encapsulate hardware complexity. This approach makes Contiki work with various hardware, including MCUs and radio modules. The general architecture of the Contiki-NG [201] is illustrated in Figure 8.5. Kernel,

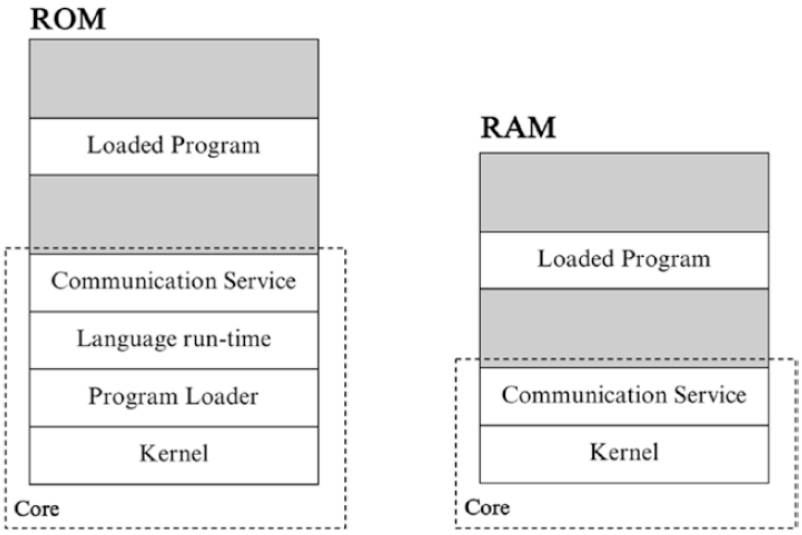


Fig. 8.5: Contiki NG general architecture [201]

the program loader, the language run-time, and the communication service are static modules within the ROM of Contiki. All user programs will be loaded into

Loaded Program. Only the kernel and the communication service will be used by the Contiki RAM. Contiki uses a GCC compiler to compile C source code files. Contiki applications are written in *.c files. After they are compiled, a binary file will be produced and can be deployed on a hardware. A toolchain must be installed on a PC for this process, see Figure 8.6. Linux OS has more native support for such toolchain as Windows OS.

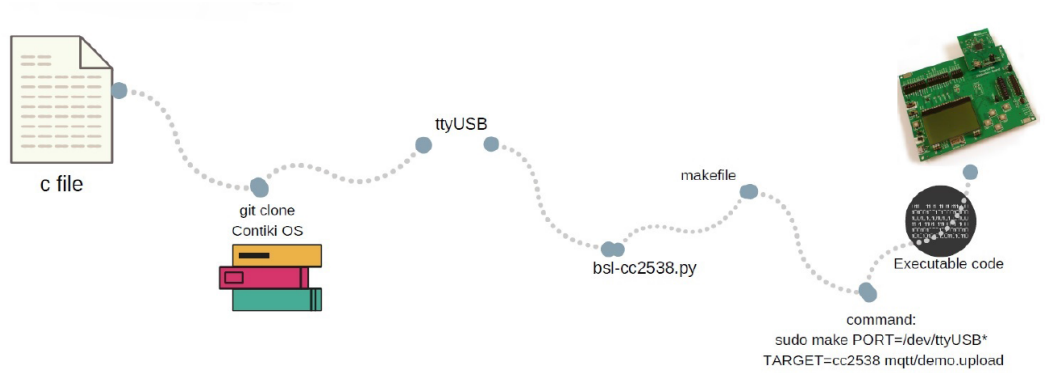


Fig. 8.6: Toolchain for Contiki development on Linux OS [185]

6LoWPAN with Contiki

Contiki runs a 6LoWPAN/IP stack on TI hardware such as the CC2538 and C26xx (new generation of TI devices) families. A border router (also called edge router) is required to connect the 6LoWPAN network with constrained devices with the existing IPv4 or IPv6 network. A CC2538 board can act as a border router, the corresponding implementation is provided by Contiki. Another possibility is to install the border router software (so-called slip-radio) on a Linux computer. Naidu et al. [249] successfully implemented a 6LoWPAN Border Router on the Raspberry Pi.

Contiki-NG NETSTACK implements four layers, as follows [201]:

- Network layer (NETSTACK_NETWORK)
- MAC layer (NETSTACK_MAC)
- RDC (Radio Duty Cycling) layer (NETSTACK_RDC)
- Radio layer (NETSTACK_RADIO)

The Network layer (NETSTACK_NETWORK) in an OSI layer can be represented as Application, Transport, Network, Routing, and Adaptation.

The RPL routing implements the DODAG routing graph form. On the MAC layer, Contiki-NG uses CSMA/CA on the IEEE 802.15.4 protocol. The Radio Duty Cycling (RDC) layer saves energy by allowing a node to keep its radio transceiver off most of the time. Contiki-NG supports the ContikiMAC protocol based on the principles behind low-power listening. ContikiMAC uses Time Slotted Channel Hopping (TSCH) that is a part of the MAC layer of the IEEE 802.15.4e-2012 amendment. The radio layer is handled by radio module from the Contiki-NG mote. Most radio layers work on the IEEE 802.15.4 protocol mechanism.

8.2.2 Arduino

Arduino is an open-source electronics platform consisting of hardware and software. Arduino boards sense the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators. Arduino software consists of Arduino programming language (based on Wiring[53]), and the Arduino IDE, based on Processing[54]. The Arduino programming language can be expanded through C++ libraries. Wiring is an open-source programming framework for microcontrollers and allows writing cross-platform software to control devices attached to a wide range of microcontroller boards. Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts.

The advantages of Arduino are inexpensive microcontroller boards (less than 50 USD), the cross-platform IDE, simple and clear programming environment, open source and extensible software, open source and extensible hardware. For this reason, we have chosen two Arduino boards: Arduino Yún rev. 2 and Arduino Nano 33 IoT which are introduced in the next sections.

Arduino Yún rev. 2

Arduino Yún rev. 2[55] was developed specially for IoT projects. The peculiarity of this board is that it consists of two hardware components. The AVR Arduino microcontroller with only 2.5 KB RAM and 32 KB (minus 4 KB for bootloader) ROM is programmed with Arduino IDE and provide the access to the I/O pins. The Linux Microprocessor has 16 MB RAM and 64 MB ROM and runs an OpenWRT[56] wireless stack and enables the board to connect to WLAN and Ethernet networks. Python is used as programming language. A Bridge Library, provided by Arduino, simplifies communication between these two components.

The big advantage of Arduino is the easy extensibility of hardware by using a breadboard, some wires and a variety of sensors and actuators which are inexpensive and available through online shops. For heterogeneity reasons, we built two Arduino Yún prototype devices [260] with different sensors and actuators. The board in Figure 8.8 was extended by a relais actuator and a combined DHT11 humidity and temperature sensor. The device in Figure 8.9 was extended by a light sensor, smoke sensor and four LEDs in different colors.

Arduino Nano 33 IoT

Arduino Nano 33 IoT [13] is comparable with CC2538 board due to the constrained memory and storage resources, i.e. the 32KB RAM and 256KB ROM. The board provides WLAN and BLE connectivity and has a low power architecture. Arduino Nano 33 IoT supports full TLS secure transport: the ATECC608A cryptochip stores certificates and pre shared cryptographic keys in hardware. The board is compatible with the Arduino IoT Cloud and other Cloud services, i.e. IFTTT or Amazon AWS

[53] http://wiring.org.co/
[54] https://processing.org/
[55] https://store.arduino.cc/arduino-yun-rev-2
[56] https://openwrt.org/about

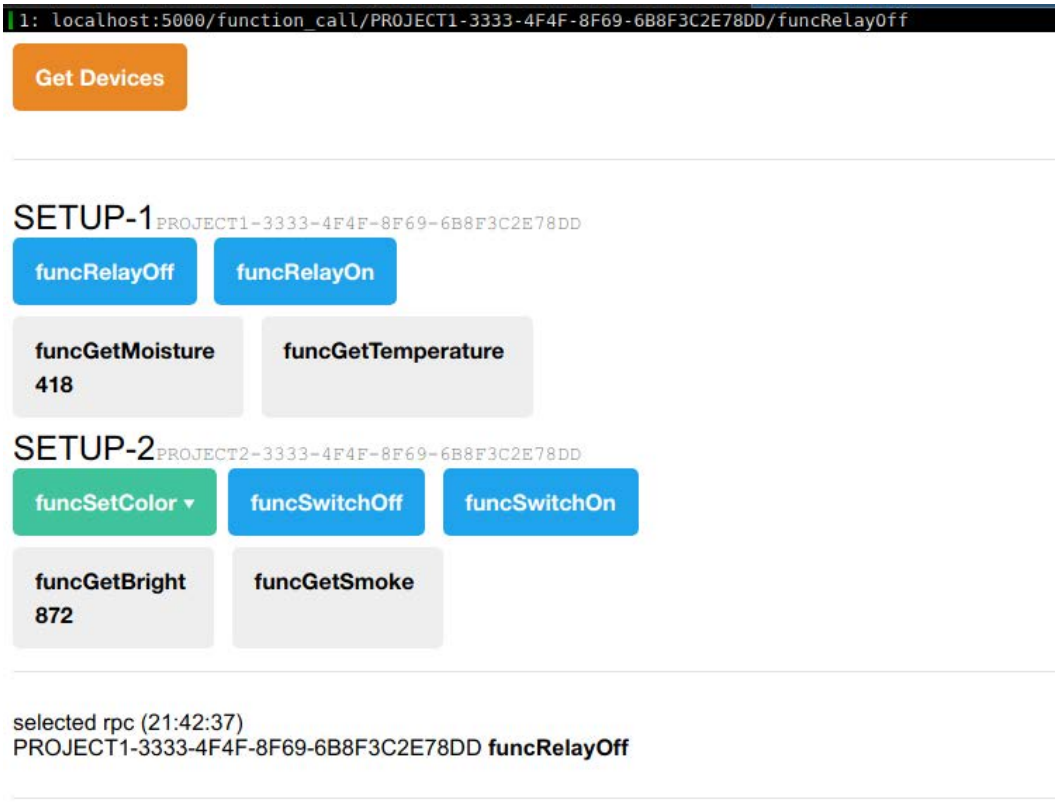


Fig. 8.7: Arduino Yún functions in the Web Client

IoT Core. The board Arduino Nano 33 IoT was chosen for evaluation of security hardware chip and TLS benchmark, see Section 9.4.

8.2.3 ESP-32 NodeMCU

ESP-32 [96] is a series of low cost, low power system-on-chip (SoC) microcontrollers with integrated WLAN and dual-mode Bluetooth. They are able to achieve ultra-low power consumption. The chips come from the Chinese company Espressif Systems. The ESP-32 NodeMCU from AZ-Delivery has a built-in ESP-WROOM-32 chip. It has 520 KB SRAM and 16 MB Flash ROM. They are programmable with NodeMCU, Arduino IDE, MicroPython, Lua RTOS. We decided to use Arduino IDE. The boards are extendable via breadboard with Arduino compatible sensors and actuators. This software development variety, low cost and low energy consumption make ESP-32 chips attractive for IoT devices in industry. Its popular predecessor was an SP8266 chip.

For our feasibility study, we extended an ESP-32 board with sensors and actuators. The first one is shown in Figure 8.10 [234] with PIR motion sensor, DHT22 humidity and temperature sensor, RGB-LED, Button, LED, BH1750 light sensor. The second one is shown in Figure 8.11 [234] with PIR motion sensor, red and green LEDs, light sensor.

The bootstrapping process was implemented on ESP-32 NodeMCU boards. The CRUD operations were implemented on the board and in the NETCONF-MQTT bridge. As soon as a device joined the network, the RETRIEVE operation was

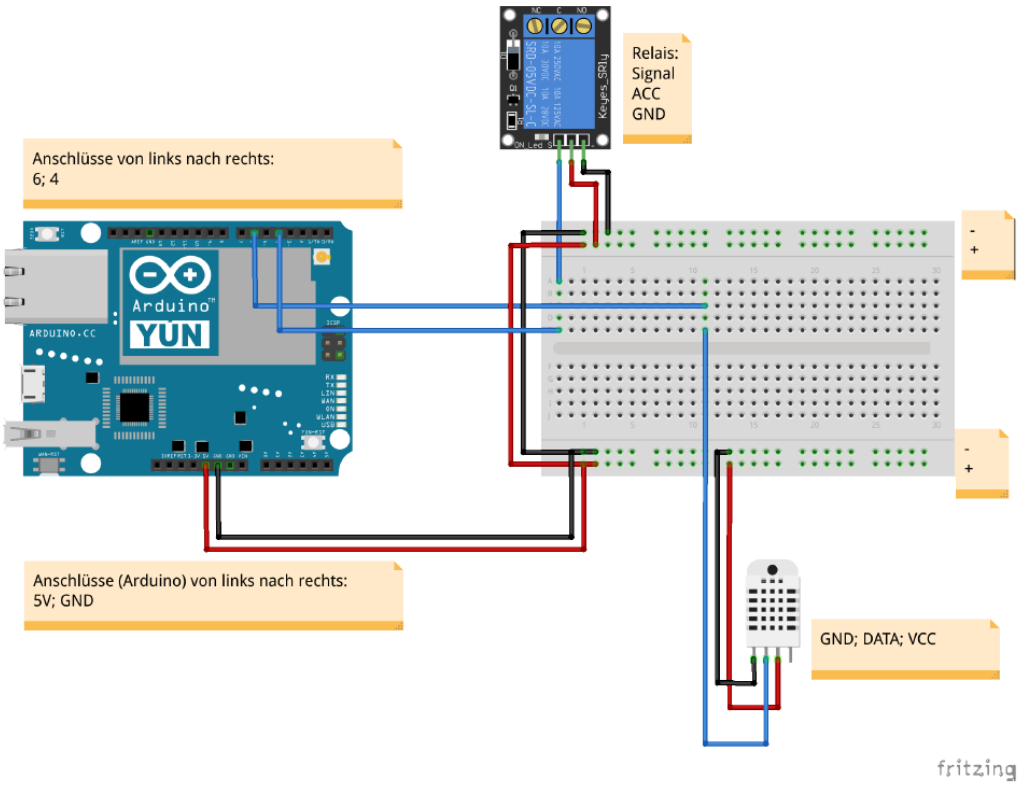


Fig. 8.8: Prototype Arduino Yún with Relais Actuator and Humidity and Temperature Sensor

performed to check if the device is already known in the network configuration. This check is performed also before UPDATE and DELETE operations. If the device was known, then UPDATE or DELETE operations could proceed. Otherwise, the CREATE operation proceeded. All this operations get acknowledged by the bridge. In a case of time-out, a board repeats the operation. This mechanism allows automate device discovery process. ESP-32 NodeMCU boards were also used for evaluation within the precision agriculture project, see Section 9.5.

8.2.4 Raspberry-Pi for Edge Computing

We use a Raspberry-Pi 3B for edge computing. Raspberry Pi is a single-board computer with WLAN and Bluetooth connectivity. It has a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1 GB RAM and a slot with 16 GB Micro SD card.

We use Raspbian OS as operation system and Python as the programming language. The Raspberry Pi is running:

- 6LoWPAN border routing with help of a connected CC2538 board;
- Mosquitto MQTT broker;
- NETCONF-MQTT bridge;
- Virtual Device;
- web-based NETCONF-Client;

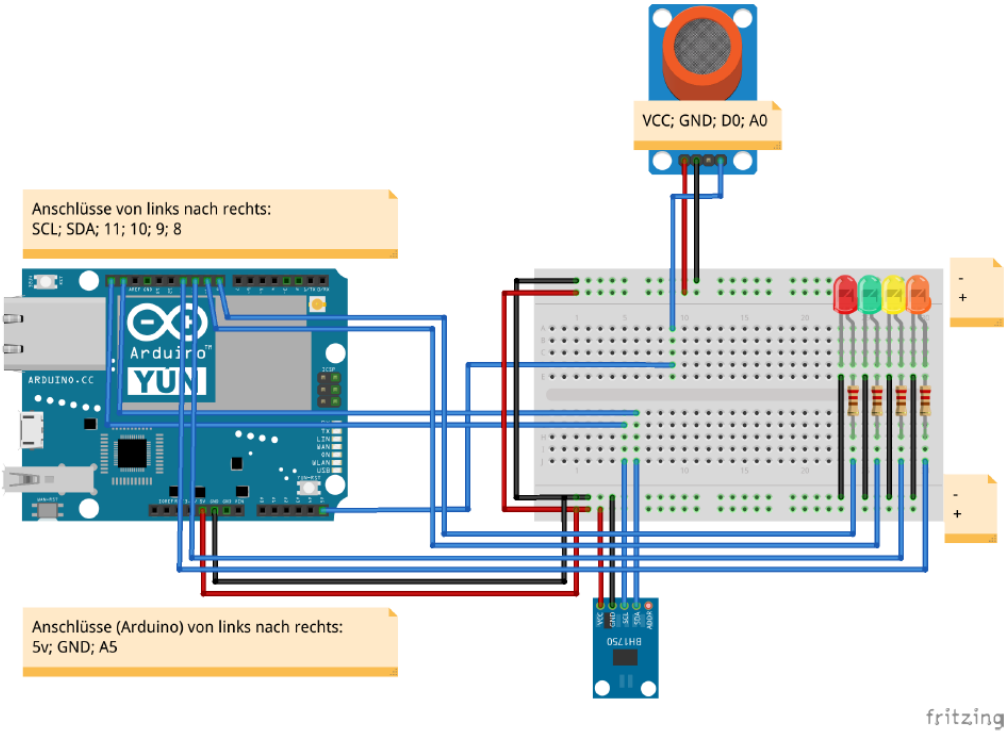


Fig. 8.9: Prototype Arduino Yún with a Light and a Smoke Sensor, and 4 LEDs as Actuators

The challenging part was to set up the 6LoWPAN border router. Special instructions has to be followed[57]. First step is to identify boards as ttyUSB devices. Then the Contiki software for border router must be installed on a CC2538dk board. Finally, the tool called *tunslip6* must be running. This tool is provided by Contiki and bridges the Contiki border router to the host (here, Raspberry Pi) via a tun interface.

Another model, the Raspberry Pi Zero was evaluated with RDF library for semantic processing in Section 9.2.5. This board, with 1 GB RAM and single-core 1GHz CPU, is less powerful than Raspberry Pi 3B. However, the Raspberry Pi OS Lite can be used as operation system and Python as programming language.

8.3 Processing of Semantic Device Descriptions

There are two implementations which deal with semantic device descriptions: the NETCONF-MQTT bridge and the Virtual Device. Both implementations are using the RDFLib library. There are not many Python libraries available for processing ontologies in RDF. This library was chosen because it has a broad and active community which developed many plugins around it, and it is actively maintained and further developed. The compared alternative was Owlready2[58] but its development is has not been active since 2017.

[57] https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-RPL-border-router
[58] https://pythonhosted.org/Owlready2/

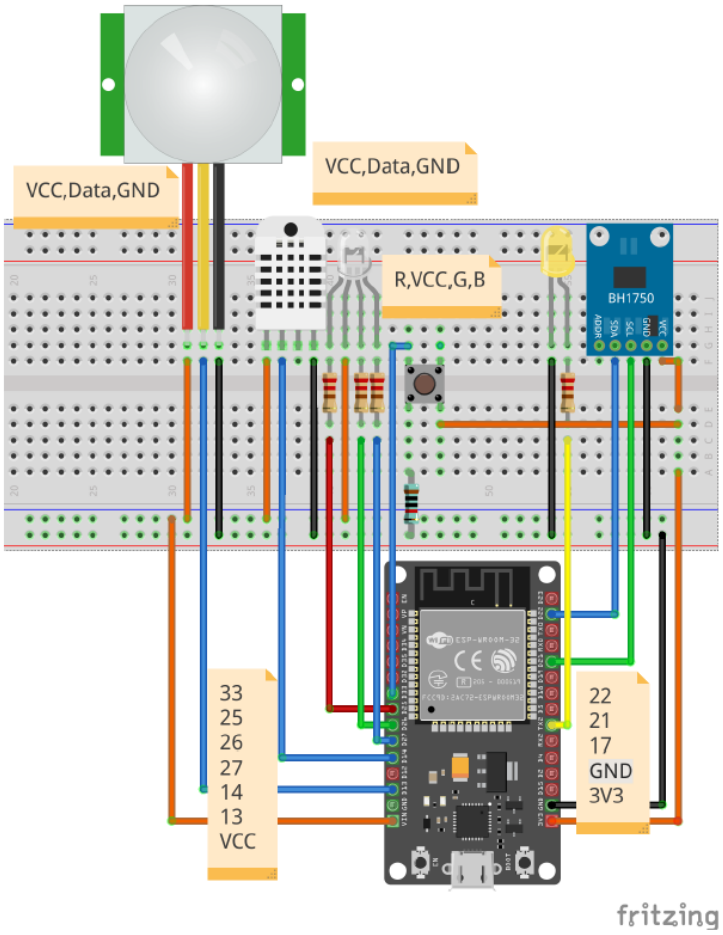


Fig. 8.10: Prototype ESP-32 NodeMCU with motion sensor, humidity and temperature sensor, RGB-LED, Button, LED, light sensor

8.3.1 RDFLib

RDFLib [293] is a Python package working with RDF. The last stable release was 4.2.2 in January 29, 2017. The Version 5.0 was released in 2020. RDFLib is open source under BSD license and is maintained in a GitHub repository. RDFLib is a library that provides support for parsing, storing, querying, and serializing RDF. RDFLib provides a core API and is extensible by plugins. RDFLib supports SPARQL 1.1 and different syntax for RDF triples, among them RDF/XML, Turtle, and JSON-LD (via a plugin).

An example of RDFLib code in Listing 8.1 shows a SPARQL query over the graph which is an ontology-based device description. First, an RDF graph is loaded, then a query is prepared, and finally executed. All instances of class types Services, Subservices, Functions and Operations can be queried at once. The *WHERE* clause defines the RDF triples which must match. The variables with a question mark are unknown, or in other words, these are variables, we are looking for. Namespaces must be given as input parameters for the graph.

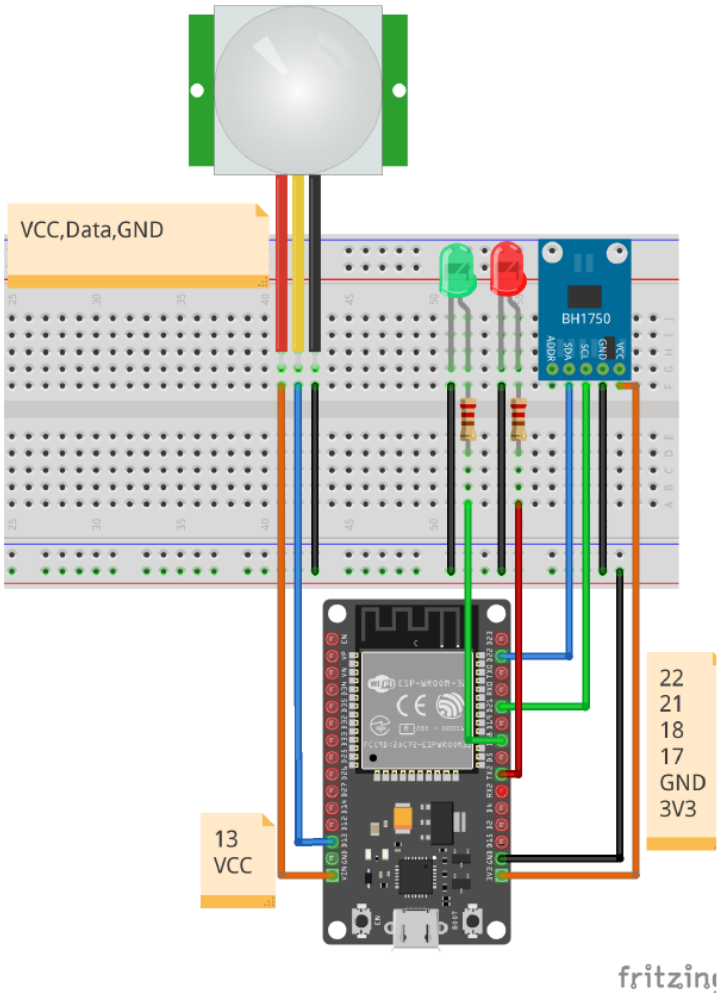


Fig. 8.11: Prototype ESP-32 NodeMCU with motion sensor, red and green LEDs, light sensor

8.3.2 RDFLib in Bridge and Virtual Device

The original implementation of the bridge used a plain JSON parser [327] because the device descriptions came along as proprietary device profiles with JSON format. After parsing, the obtained data was used for the generation of a YANG model. This parser was replaced by the implementation with RDFLib which use SPARQL query language to obtain data.

The virtual device uses also RDFLib library for processing of ontology-based device descriptions. Three RDFLib modules are used: the RDFLib core, rdflib-jsonld (a parser plugin for JSON-LD format) and rdflib-sparql (a plugin for SPARQL queries).

The virtual device implementation consists of several modules for scenarios mentioned in Section 6. The virtual device processes new device descriptions and creates its own device description. This VD device description will be published to the MQTT broker. The bridge adds this virtual device into the configuration namely the YANG model. During the processing of a device description, the virtual device collects data about the device functions, especially their MQTT topics, used for aggregation mapping. For example, when a user of the NETCONF client calls the RPC *switchOn*

```
from rdflib import Graph
from rdflib.plugins.sparql import prepareQuery

def parseServices(g, serviceList):

    g = Graph()
    g.parse(data=json_str, format="json-ld", base=" https://www.cs.uni-potsdam.de/
        bs/research/myno#")

    q = prepareQuery(
        'SELECT DISTINCT ?services ?functions ?subservices ?operations '
        'WHERE { ?device onem2m:hasService ?services .
        ?services onem2m:exposesFunctionality ?functions .
        ?services onem2m:hasOperation ?operations .'
        '?services onem2m:hasSubService ?subservices .
        ?subservices onem2m:hasOutputDataPoint ?outDps . }
        ORDER BY DESC(?functions) ',
        initNs={"base": " https://www.cs.uni-potsdam.de/bs/research/myno#",
        "onem2m": "http://www.onem2m.org/ontology/Base_Ontology/base_ontology#"},
        base=" https://www.cs.uni-potsdam.de/bs/research/myno#")
    result = g.query(q)
```

Listing 8.1: RDFlib Code example

on the virtual device, the virtual device receiving this call, will propagate it to all devices with such functions.

8.3.3 Ontology Compression

The device description is based on the oneM2M ontology and uses RDF-XML syntax by default. However, different syntax makes the device description longer or shorter. The size of device descriptions matters because they should be used on constrained devices. An example is given in Listings 8.2 and 8.3 which show non-optimized Turtle [54] and optimized JSON-LD [213] syntax.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix oneM2M: <http://www.onem2m.org/ontology/Base_Ontology/base_ontology#> .
<https://www.cs.uni-potsdam.de/bs/research/myno#myDevice>
  a owl:NamedIndividual, oneM2M:Device ;
  oneM2M:hasFunctionality <https://www.cs.uni-potsdam.de/bs/research/myno#
      funcSwitchOff> .
```

Listing 8.2: RDF Example in Turtle, non-optimized

```
{ "@context": {
   "oneM2M": "http://www.onem2m.org/ontology/Base_Ontology/base_ontology#",
   "myno": "https://www.cs.uni-potsdam.de/bs/research/myno#" },
  "@graph": [
{       "@id": "myno:myDevice",  ...  ],
      "oneM2M:hasFunctionality": [
        { "@id": "myno:funcSwitchOff" },   ...
```

Listing 8.3: RDF Example in optimized JSON-LD syntax

The device description in our example comprises descriptions for three functionalities [314]: (i) switch on the LED with a pre-defined color; (ii) switch off the LED; (iii) request the value of the brightness sensor. These capabilities described in the oneM2M ontology result in a file size of 23,966 Bytes using the RDF-XML syntax.

The implementation was done on a CC2538EM microcontroller board. The TI-Board is also connected by constrained network namely 6LoWPAN [237]. The file size has to be manageable by these resources.

The file size is reduced from 23,966 Bytes (RDF-XML) to 19,771 Bytes by converting it to the JSON-LD syntax. However, the device description will grow again when further services are added. Therefore, we investigate compression possibilities to minimize the file size. Since the compression is performed on a capable compute device before the deployment on the microcontroller board, the compute complexity is not an important criteria and we concentrate only on the file size reduction in our evaluation. We use the same input file in all of our experiments.

We can summarize our requirements as follows:

1. small file size regarding 32 KB RAM;
2. ontology data structures particularly strings and structural elements like curly and square braces should be compressed efficiently;
3. small count of fragments concerning small MTUs in 6LoWPAN;
4. enabling to edit the file e.g. to change the UUID.

The last requirement makes it possible to paste the individual data of a board like the UUID into a generic device description which may be written once for this kind of device. The device itself is not able to this, but a client application receiving this data can process it.

There is a variety of compression algorithms. Nevertheless, most of them are not designed for resource-constrained devices. There are few efforts applied to sensor networks e.g. S-LZW [311], SBZIP [396]. We have chosen to compare CBOR [41], a relatively new standard for the representation of data on constrained devices, and RDF HDT [101] which supports the compression of RDF data and therefore may be well-suited for the binary representation of device descriptions. The evaluation of CBOR and RDF HDT is provided in Section 9.2.4.

CBOR and RDF HDT are relatively new representations, but already investigated by some research groups ([300, 135, 136, 102, 103, 224]).

For the transmission of data within a SIEM system, Rix et al. [300] developed an approach of mapping XML to CBOR using JSON as an intermediate step. Additionally, they applied GZIP compression. By this combination, the example files were compressed from 358 Bytes (XML) to 251 Bytes (XML/GZIP) and to 63 Bytes using CBOR and GZIP resp. 506 (XML), 331 (XML/GZIP) and 141 Bytes (CBOR/GZIP). This shows the benefit of CBOR/GZIP in their use-case scenario.

Pöhls et al. [135] considered CBOR and COSE [325] for the signature of sensor messages in the IoT. Ilgner et al. [136] evaluated CBOR for Bluetooth and 3G Communication to monitor remote pipelines. They used a 8-bit microcontroller (AT Mega 128) with 4 Kbytes of internal static RAM. He experienced some challenges because CBOR consumes a lot of static memory (2.5 Kbytes) for CBOR object variables. CBOR libraries do not support 64-bit integers and longer objects needed to be broken.

RDF HDT was originally developed for the exchange of RDF Datasets in the Web of Data. A comprehensive evaluation on HDT was done by Fernandez et al. [102], [103]

and Martinez et al. [224]. They show examples where the HDT files took only between 6% and 11% space compared to the original N-Triples[59]. These compression results are impressive and make HDT an interesting candidate.

8.3.4 Sensor Data with SSN

The concept of semantic sensor data with SSN from Section 5.5 was implemented for experiments on CC2538dk [185] and ESP32 NodeMCU [234] boards. The sensor data with semantic annotation was published additionally to the plain values. Such SSN-based descriptions were also published if an RPC call for actuator was triggered.

The main challenges during implementation were the file size of such data description and placeholders. Even the file size is smaller than the device description, it is still a lot for Class 1 devices such as CC2538dk (e.g. non-optimized JSON-LD syntax: 24,1 KB of device description versus 4,86 KB for temperature). The second challenge was the replacement of placeholders through real values. It works well on the ESP-32 NodeMCU in Arduino Sketch. It was tricky but not impossible on the CC2538dk. During the loading of the SSN template into the buffer, the placeholders can be replaced. These implementations were analyzed and suggestions for improvement are outlined in Section 9.2.6.

8.4 MYNO Update Protocol (MUP)

In normal operation mode, an IoT device is sending data to the edge. In case of a firmware update, several kilobytes of data have to be transferred to the device. This is not a common task in the IoT and brings challenges for the communication layer.

This section describes the details of the MUP implementation in the MQTT-based MYNO architecture. Starting with the testbed and the MYNO device description, we describe the efficient transmission of an update image to constrained devices over a 6LoWPAN network.

8.4.1 Testbed

We implemented the proposed MYNO update approach in a testbed for the microcontroller board CC2538dk [59] from Texas Instruments with Contiki-NG v4.5 [70] to show the feasibility of our approach. The CC2538dk is a constrained device with an ARM Cortex-M3 processor, $32\,\mathrm{kB}$ RAM, $512\,\mathrm{kB}$ flash memory and an IEEE 802.15.4 compliant system-on-chip. Software support for 6LoWPAN is provided by Contiki-NG. One CC2538dk board is used as 6LoWPAN router and the other boards are used as IoT devices with sensors and actuators. Our testbed is shown in Figure 8.12. The edge node, the Raspberry Pi 3B with Raspberry Pi OS, is running an MQTT Broker (Mosquitto v1.6.10), the NETCONF-MQTT bridge and the Update Server as well as the *tunslip6* tool for tunneling the IP-Traffic over the serial port for the 6LoWPAN router.

We used the open-source free library *crypto-algorithms* [76] for computation of SHA-256 hash values. For signatures, the Elliptic Curve Digital Signature Algorithm

[59] `http://www.rdfhdt.org/hdt-internals/`

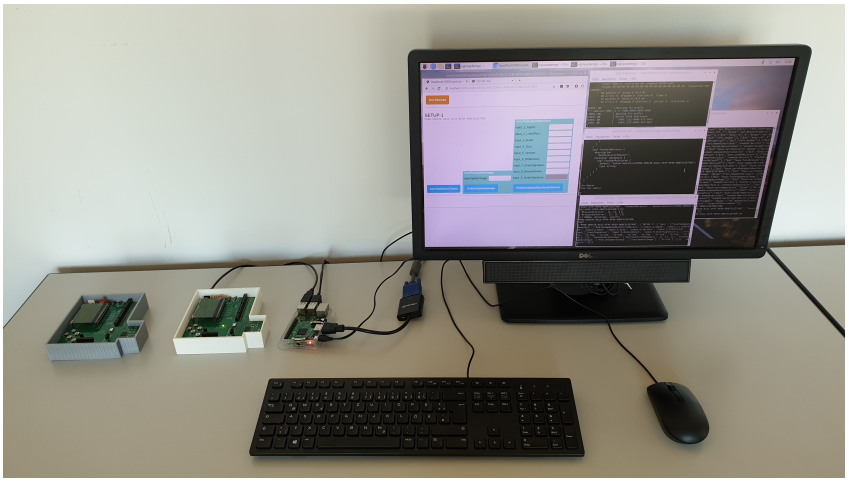


Fig. 8.12: Testbed with a Raspberry Pi 3B and a CC2538dk Development Kit consisting of two CC2538EM microcontrollers plugged into the SmartRF06 Evaluation Boards and used as a 6LowPAN Border Router and an IoT device (wireless and battery-powered) on the left.

(ECDSA) on the curve *secp256r1* is used. On the CC2538dk board, we used the library *micro-ecc* [231] for the validation of inner and outer signature. This is a small and fast ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit processors.

8.4.2 Device Description

We extended the MYNO semantic device descriptions based on the oneM2M Ontology. There are three RPC calls required which must be translated to MQTT publish messages: `getDeviceToken`, `sendExtendedManifest`, and `sendFirmwareImage`.

The extension of the device description for the first call, the `getDeviceToken` is shown as a snippet in JSON-LD format in Listing 8.5 in the Appendix. This example shows that a *Device* named `myDevice` has a *Service* `servGetDeviceToken` with an *Operation* `opGetDeviceToken` with MQTT properties `mqttMethod` and `mqttTopic`. Additionally, `myDevice` has a *Controlling Functionality* `funcGetDeviceToken` with a *Command* `cmdGetDeviceToken`. This functionality description is intended for human readability, and also the NETCONF client uses it for RPC calls generated by YANG.

The MQTT topics for the update process were defined in the device description (see Table 8.2). The devices expect this topic pattern and their device UUID at the end of the topics. In this way every device can be uniquely addressed via MQTT topic. The topics are used by the NETCONF-MQTT bridge for translation between RPC calls and MQTT publish and subscribe messages.

Tab. 8.2: MQTT topics for publish/subscribe in MUP protocol

MQTT Topic	Message
mup/token/UUID	publish request for device token
mup/manifest/UUID	publish manifest
mup/firmware/UUID	publish firmware
mup/response/UUID	publish all responses from device

We evaluated the overhead introduced by the proposed semantic device descriptions. The size of the device description increased from $10.88\,\mathrm{kB}$ to $27.49\,\mathrm{kB}$, since we added three controlling functions for MUP and the descriptions of the manifest parameters and error definitions as well as new MQTT topics. This device description was already reduced in size by using the compacted JSON-LD format [213]. The expanded document format has a size of $41.27\,\mathrm{kB}$. The size of the device descriptions could be further reduced by RDF/HDT compression for semantic datasets as shown in [314]. However, the device description is transmitted only once, during the bootstrap process. Constrained devices send the device description in pieces to the bridge using QoS 1. The size of these pieces depends on the MQTT buffer implementation on a device. The last piece of the device description has a tag *END* so that the bridge can process the device description. Opposite to the device description which is published by the device, the firmware image is sent from the Update Server to the device.

8.4.3 Transmitting the Firmware Image

In our test setup, the firmware image had a size of $87.8\,\mathrm{kB}$. Hence, the transfer of the image to the constrained device was a challenging task. Table 8.3 shows the communication stack of our testbed. Due to the limited resources, the communication stack and the implemented protocols try to be lightweight. Contiki-NG relies on uIP [87], the IPv6 compliant TCP/IP stack, designed to be used with tiny 8 and 16 bit microcontrollers [397].

Tab. 8.3: MUP Protocol stack

MQTT
TCP
IPv6
6LoWPAN
IEEE 802.15.4

8.4.4 Transfer via NETCONF–MQTT Bridge versus MQTT Publish

If the update image is transmitted via the NETCONF-MQTT bridge, XML RPCs will be used which results in ASCII-encoded hex strings. For example, a hexadecimal „A“ ($= 1010_2$) is transmitted as its ASCII code 65 ($= 1000001_2$). This is highly inefficient since it doubles the image size.

Alternatively, the update image could be sent directly to the MQTT broker instead of passing it through the NETCONF-MQTT bridge. In that case, it would not have to be encoded in ASCII format and would remain at its original size. Only one RPC is needed at the beginning that instructs the device to start the verification process, as well as one RPC reply where the device indicates the result of the verification (see Listing 8.4).

Listing 8.4: Example NETCONF RPC and reply.

```
<?xml version="1.0" encoding="UTF-8"?>
<funcPubUpdateImage>
    <uuidInput xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
        F97DF79-8A12-4F4F-8F69-6B8F3C2E78DD</uuidInput>
    <inputUpdateImage xmlns:nc="urn:ietf:params:xml:ns:netconf:base
        :1.0">
        START</inputUpdateImage>
</funcPubUpdateImage>


<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="urn:uuid:2e8eef63-fe1f-4b4d-a585-c3e86ee23374">
    <data>
        <retval>FW-SUCCESS</retval>
    </data>
</nc:rpc-reply>
```

8.4.5 MQTT Slicing

Since constrained devices can receive only a limited MQTT packet size, the Maximum Packet Size property was introduced in version 5 of the MQTT protocol [243]. Thereby, a client can inform the MQTT broker about the packet size it is willing to accept. When a packet is too large, the broker must discard it without sending. However, only the MQTT broker is informed about the `Maximum Packet Size` value and not the application willing to publish a message intended for a such constrained device. In other words, the Update Server and the device must agree on the same packet size implementing MUP. The CC2538dk is a constrained device and the firmware image is too big for one message. Therefore, we introduced `slicing` on the application level for the firmware image transmission. We sliced the image like a salami into smaller packets. The slices were numbered and sent in order. Flow control was necessary to ensure the message order, as described in the next section.

8.4.6 Flow Control

An important requirement on the receiver side is that the MUP implementation on the device expects all slices to be delivered in the correct order. This approach avoids additional buffer space which would be necessary for slices that arrive out of order. Further, it simplifies the calculation of the hash values, since each received slice can be immediately piped into the hash function and processed further.

While the reliable delivery of data is typically handled by TCP, the situation is not so easy on constrained devices. For example, the uIP TCP/IP stack in Contiki-NG only allows each TCP connection to have a single TCP segment in flight at any given time.

To solve this problem, we implemented a simple *Stop-and-Wait* protocol on the application layer shown in Figure 8.13 left. Each slice is published to the MQTT broker, arrives at the device and is acknowledged by the MQTT client. These acknowledgments are published as MQTT messages on a response topic. When all

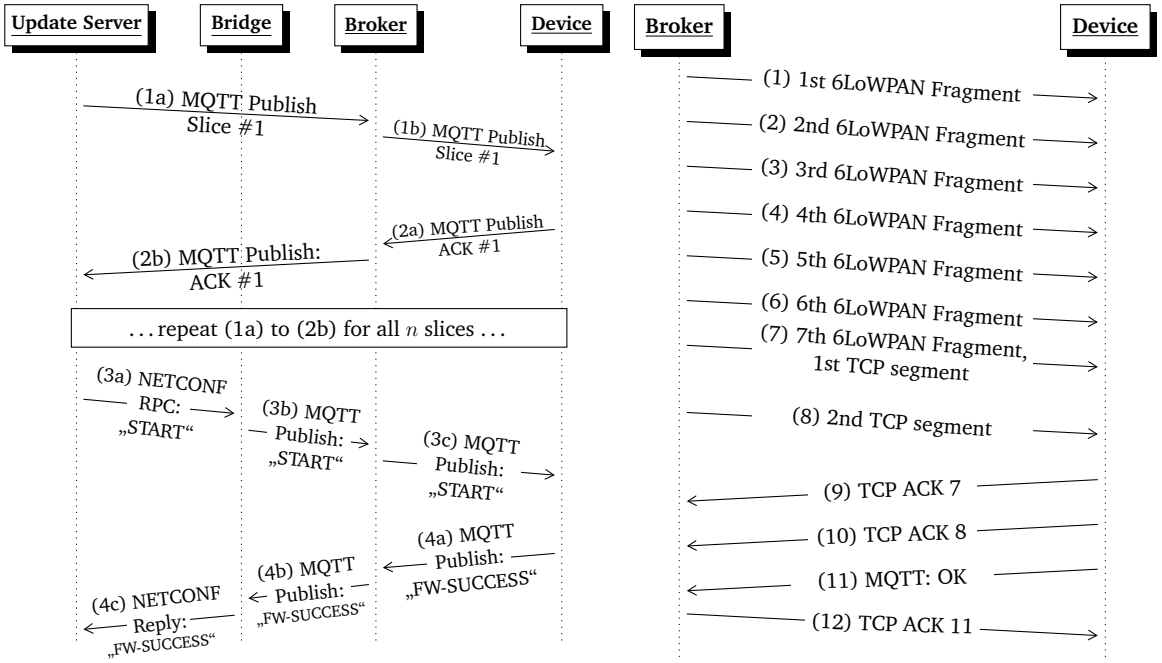


Fig. 8.13: Sequence of packets sent during the transmission of a complete update file (slices 1 to n) (left). Each slice is fragmented by the 6LoWPAN Router (right). Example for slice size of 600 bytes.

slices have been published, a NETCONF RPC call starts the verification of the update image on the device.

The *Stop-and-Wait* protocol makes the transfer of the firmware image robust, but causes an overhead which will be analyzed in Section 9.

8.4.7 Slice Size and Fragmentation

Choosing the appropriate slice size is a challenging task. Increasing the slice size reduces the number of MQTT messages and therefore the MQTT protocol header overhead. However, it must be kept in mind that each update image slice is further fragmented by the 6LoWPAN router, since IEEE 802.15.4's physical layer payload size is limited to $127\,\mathrm{B}$. Hence, the larger the slice size, the more fragments have to be created. This also negatively impacts the performance, e.g., due to the complexity of the reassembly process and the large reassembly buffers that are required [357, p. 59],[188].

Figure 8.13 right shows this procedure for a slice size of $600\,\mathrm{B}$. The slice was transmitted along with a slice number ($2\,\mathrm{B}$), the MQTT header with the topic name (in the example $57\,\mathrm{B}$), and protocol headers (TCP/IP, 6LoWPAN). The slice was sent in two TCP segments where the first TCP segment consisted of $610\,\mathrm{B}$ and was divided into seven fragments, and the second segment with $54\,\mathrm{B}$ was small enough to fit into one IEEE 802.15.4 packet. Once the IoT device received and reassembled all fragments of a slice, it first acknowledged the receipt using a TCP ACK (message 9 and 10). It then published the MQTT response message (message 11). The broker acknowledged the receipt of the response with a TCP ACK (message 12).

The overhead due to the long MQTT topic names is a well-known problem. Hence the `Topic Alias` feature was introduced in MQTT v5 [243]. A `Topic Alias` could be set by including a two-byte integer alias with the full topic name in the first published message on any topic. All following published messages could include the alias and a zero-length topic. However, in Contiki-NG the `Topic Alias` feature is supported only for sending messages, but not for receiving messages.

The trade-off between slice size and fragmentation along with other performance issues is evaluated and discussed in the next section.

8.5 Conclusion

The implementation of all devices and system components was challenging and required different skills (e.g. ontology design and processing, different programming languages and libraries) even we tried to reduce the number of languages and libraries. However, is was possible to implement the MYNO concept on the chosen devices and showed that with every further device, it became easier to follow the concept.

The proposed MYNO Update Protocol (MUP) is suited for constrained devices which is demonstrated with our prototype on an IoT device with only $32\,\mathrm{kB}$ RAM and the firmware transmission over 6LoWPAN. The implementation challenges are discussed, especially, the need of slicing is motivated. We discuss the implementation issues in Section 9.3.6 and identify optimization potential in MQTT implementations and the MQTT standard to further improve the support of constrained devices.

Listing 8.5: Device description: `getDeviceToken` controlling functionality in the ontology.

```
{
  "@context": {
    "owl": "http://www.w3.org/2002/07/owl#",
    "myno": "https:// www.cs.uni-potsdam.de/bs/research/myno#",
    "onem2m": "http://www.onem2m.org/ontology/Base_Ontology/
       base_ontology#"},
  "@graph": [
...
{      "@id": "myno:myDevice",
       "@type": ["owl:NamedIndividual", "onem2m:Device"],
       "onem2m:hasFunctionality": [ {"@id": "myno:
          funcGetDeviceToken"},
...
       "onem2m:hasService": {"@id": "myno:servGetDeviceToken"},
       "onem2m:hasThingProperty": [ {"@id": "myno:deviceUuid"}
...
{      "@id": "myno:servGetDeviceToken",
       "@type": ["owl:NamedIndividual", "onem2m:Service"],
       "onem2m:exposesFunctionality": {"@id": "myno:
          funcGetDeviceToken"},
       "onem2m:hasOperation": {"@id": "myno:opGetDeviceToken" } },
...
{      "@id": "myno:opGetDeviceToken",
       "@type": ["owl:NamedIndividual", "onem2m:Operation"],
       "onem2m:exposesCommand": {"@id": "myno:cmdGetDeviceToken"},
       "onem2m:hasInput": {"@id": "myno:uuidInput"},
       "onem2m:hasOperationState": {"@id": "myno:opState"},
       "myno:mqttMethod": "GET-DEVICE-TOKEN",
       "myno:mqttTopic": "mup/update/token"  }
...
{      "@id": "myno:funcGetDeviceToken",
       "@type": ["owl:NamedIndividual", "onem2m:
          ControllingFunctionality"],
       "onem2m:hasCommand": {"@id": "myno:cmdGetDeviceToken"},
       "onem2m:hasThingProperty": {"@id": "myno:
          funcDescGetDeviceToken"} },
...
{      "@id": "myno:cmdGetDeviceToken",
       "@type": ["owl:NamedIndividual","onem2m:Command"],
       "onem2m:hasInput": {"@id": "myno:uuidInput" } },
...
{      "@id": "base:opState",
       "@type": [ "owl:NamedIndividual", "onem2m:OperationState"
          ],
       "onem2m:hasDataRestriction_pattern": [
         "10_OK",
         "12_ERROR",
       ],
...
```

9 Evaluation

> *For all resources, whatever it is, you need more.*
>
> **— RFC 1925, Nr. 9**
> (The Twelve Networking Truths)

The prototype implementation allows to proof the concept and feasibility of the MYNO framework. This is the first step in this evaluation chapter. Then, the use of the ontology-based device descriptions will be assessed in many facets. Afterwards, the MYNO Update Protocol is evaluated with CC2538dk boards and a TLS benchmark is performed. Finally, performance evaluation with 10 ESP32 NodeMCU boards is provided for scalability.

9.1 Proof of Concept and Feasibility

The concept of the MYNO framework was implemented and several IoT devices were implemented and tested with the framework. The criteria for the proof are:

- overall MYNO concept in Chapter 4;
- prototype implementation in Chapter 8;
- requirements for management of networks with constrained devices from the RFC 7547 [94];
- interoperability model in Figure 1.3, introduced in Chapter 1;
- heterogeneity of IoT devices from Section 8.2;

The concept of the MYNO framework was implemented in Python and is running on a Raspberry Pi 3B as well as on the more powerful instance (for example, a notebook). All functionalities from the concept were realized: discovery and bootstrapping operations; processing of semantic device descriptions and generation of a YANG model; NETCONF server interface.

RFC 6241
Regarding the NETCONF server, we compare the functionality with the specification RFC 6241 [93] introduced in Section 2.4.2. The NETCONF protocol layers are shown in Table 9.1.

Tab. 9.1: Comparison between the MYNO Framework and NETCONF protocol

Layer	MYNO Framework	NETCONF Protocol
Content	YANG	YANG, Notification data
Operations	get, get-config	get, get-config, edit-config, copy-config, delete-config etc.
Messages	RPC, RPC-Reply	RPC, RPC-Reply, Notification
Secure Transport	SSH support	SSH, TLS, etc.

Summarizing the comparison results, the MYNO framework has some limitations. The NETCONF-MQTT bridge can retrieve the edge device configuration as a YANG model but this configuration cannot be edited, replaced or deleted. Because the device descriptions cannot be edited on the device due to limited resources [398]. As configuration, we understand the controlling and measuring functionalities on a device. These functionalities can only be changed by a firmware update. Then, this new firmware would also provide a new device description. This description will be published to the MYNO framework as the UPDATE operation during the bootstrap. Thus, the YANG model as the edge configuration will be also updated. We implemented and tested the bootstrap process on ESP32 NodeMCU boards and showed the robustness of this mechanism when devices join and leave a network, for example because of the sleep state during the duty cycle.

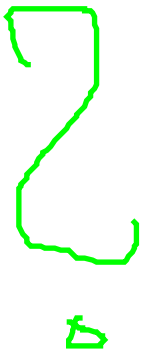
The NETCONF Notifications are not provided by the bridge. Such notifications are asynchronous messages sent by a NETCONF server. The first reason is that NETCONF base notifications should notify about configuration changes [30] but the device descriptions on the devices are static. The event notifications, specified in RFC 5277 [61] are optional. The second reason is that the NETCONF server library, we used for implementation, does not support the notification mechanism. Finally, we do not miss them in our network configuration scenarios. The "old" notifications work only during the NETCONF session and are provided by so-called "Pull" approach. The new YANG "Push" notifications are specified in 2019 by RFC 8639 [408], RFC 8640 [407], RFC 8641 [65]. These are only supported by commercial solutions [114].

Finally, the SSH support is implemented in the bridge and is used by the web-based NETCONF client, introduced in Section 4.5.

RFC 7547

von Juergen

The next proof criteria is a number of requirements for management of networks with constrained devices addressed by the RFC 7547 [94]. The limited resources of constrained devices and networks have a strong impact on these requirements. The RFC 7547 avoids a selection of the requirements as mandatory to implement. However, it defines priorities for the requirements showing their importance for a particular type of device. We have considered a subset of these requirements with high priority for devices of C1 and C2 classes and prove our concept against them:



- Support multiple device classes within a single network (Req-ID: 1.001): support of heterogeneous devices i.e. CC2538dk and ESP32 NodeMCU.
- Management scalability (Req-ID: 1.002): performance evaluation with 10 devices in Section 9.5.
- Automatic resynchronization with eventual consistency (Req-ID: 1.005): bootstrap process support for duty cycle.
- Support for lossy links and unreachable devices (Req-ID: 1.006): support of 6LoWPAN devices such as CC2538dk boards.
- Compact encoding of management data (Req-ID: 2.002): compression of ontology-based device descriptions in Section 9.2.4.

- Consistency of data models with the underlying information model (Req-ID: 2.005), to support data interoperability, consistency, and model reuse: constrained and non-constrained devices are supported by the MYNO framework and use device descriptions based on the same ontology.
- Lossless mapping of management data models (Req-ID: 2.006),(medium priority): device descriptions, mentioned above, are parsed lossless and automated to a YANG model.
- Self-configuration capability (Req-ID: 3.001): using IP-based networks, such as 6LoWPAN and WLAN, in combination with semantic device descriptions and MQTT allow automated self-configuration without human intervention.
- Capability discovery (Req-ID: 3.002), (medium priority): enabled through ontology-based device descriptions.
- Authentication of management system and devices (Req-ID: 6.001): authentication is only considered by the MYNO Update Protocol.
- Select cryptographic algorithms that are efficient in both code space and execution time (Req-ID: 6.004): use of cryptographic libraries during the firmware update process, see Section 7.3.
- Secure message transport (Req-ID: 10.004): authentication and data integrity are provided by the MUP; TLS evaluation is shown in Section 9.4.
- Avoid complex application-layer transactions requiring large application-layer messages (Req-ID: 11.001): slicing of a firmware update was evaluated by MUP in Section 9.3.

  auch beim Schicken der device description
- Avoid reassembly of messages at multiple layers in the protocol stack (Req-ID: 11.002): optimization of MUP protocol on constrained devices was evaluated in Section 9.3.

Summarizing, we considered the most requirements with high priority for devices of C1 and C2 classes. However, we do not consider a few requirements regarding security: access control on management system and devices (Req-ID: 6.003); support suitable security bootstrapping mechanisms (Req-ID: 6.002). The implementation of such security mechanisms require a security infrastructure provisioning and were out of scope in this thesis.

Heterogeneous Devices
Feasibility of our approach with proposed scenarios and heterogeneous devices in terms of capabilities and constraints was shown by implementation. The ontology-based device description was extended at minimum required. Every following device was easy to integrate because many features were already supported by the bridge.

CC2538
The CC2538 boards were challenging in every point of view. A special toolchain must be installed to compile the Contiki image, to communicate with the boards, and to upload the image over Micro USB. Then, the constrained memory, storage and 6LoWPAN capabilities were challenging when it comes to publish the device description for bootstrapping. This was solved by publishing it in pieces. The implementation for accelerometer sensor, LCD display and micro SD card was not

possible because Contiki does not provide the drivers for these hardware components. The next challenge was the transport of firmware update, see details in 9.3.

Arduino Yún Rev 3
Arduino Yún consists of two implementations: Arduino microcontroller and Linux microprocessor. They are connected by a bridge library. Microconroller provides access to the input and output pins, and the Python implementation on Linux provide MQTT client and WLAN network connectivity. The challenge was to synchronize the measuring and controlling activities with networking and publish/subscribe mechanisms through this bridge.

ESP-32 NodeMCU
The ESP-32 was extended by new kinds of sensors and actuators: PIR motion sensor and RDG LED. The PIR motion sensor has different output than the other sensors before: instead of a value, there is a boolean value (motion or no motion); the RGB LED requires three parameter values (red, green, blue) to control the color. These particular requirements must be designed in the device description and YANG. We also used these boards for performance evaluation in Section 9.5.

Interoperability
The interoperability model in Figure 1.3 contains four levels. The MYNO framework achieves the interoperability in all four levels as following:

1. physical interoperability: IEEE 802.15.4 by CC2538dk, WLAN by ESP32 NodeMCU and Arduino Yún Rev 2.
2. network and transport interoperability: 6LoWPAN and TCP/IPv6 by CC2538dk, TCP/IPv4 by ESP32 NodeMCU, Arduino Yún Rev 2.
3. integration interoperability: MQTT protocol for devices and NETCONF protocol for network configuration management are used as application protocol standards .
4. data interoperability: ontology-based device descriptions formalized with RDF and OWL standards and JSON-LD syntax format; YANG model for NETCONF configuration.

The semantic interoperability and the detailed evaluation of device descriptions is discussed in the next Section.

9.2 Evaluation of Semantic Device Descriptions

An ontology is defined as "a formal specification of a conceptualization where the conceptualization refers to the abstraction of a domain of interest" [150]. In our case, this is the device description based on the oneM2M Base ontology which we would like to evaluate. We analyzed some surveys made on the ontology evaluation methods and approaches [47, 286, 150, 80, 134].

Ontology evaluation is a process of deciding the quality on the ontology of an ontology in respect to a particular criterion with the view of determining which in a collection of ontologies would best suit in a specific purpose, see [40].

Another definition is given by [120, 80]: Ontology evaluation is defined in the contexts of two concepts: verification and validation. Ontology verification is concerned

with building an ontology correctly: checks the encoding of the specification; detects errors, as e.g. circular, class hierarchies, redundant axioms, inconsistent naming schemes etc.; confirms that the ontology has been built according to certain specified ontology quality criteria. Ontology validation on the other hand is concerned with building the correct ontology: checks whether the meaning of the definitions matches with the conceptualization the ontology is meant to specify; the goal is to show that the world model is compliant with the formal model.

Both definitions can be summarized as ontology evaluation which is concerning two perspectives: quality and correctness. The measuring *criteria* for ontology evaluation [81, 150] are categorized into these two perspectives:

- Correctness: Completeness, Conciseness, Consistency, also known as 3Cs [255].
- Quality: Computational Efficiency, Adaptability, Clarity.

Correctness of Ontology
The 3Cs problems in the correctness are briefly explained as follows [150]. *Completeness* measures if the domain of interest is appropriately covered and if all questions the ontology should be able to answer can be solved. *Conciseness* is "intended to reflect if the ontology defines irrelevant elements with regards to the domain to be covered or redundant representations of the semantics". *Consistency* describes that the ontology does not include or allow for any contradictions.

We use several tools to ensure the correctness of the ontology used for our device descriptions. First, we use Protégé [35] as an open-source ontology editor which automatically supports users during the editing process: provides graphical user interface, provides autocompletion and RDF graph visualization, generates different syntax formats, checks RDF and OWL consistency, executes SPARQL Queries, supports reasoning, etc. Protégé has an active community and offers an amount of various plug-ins.

We also used the W3C RDF Validator [292] which validates the RDF triples and generates a graphical visualization of the data model. Unfortunately, it was not further developed since 2004 and does not support newer specifications of Semantic Web Standards and has problems with graphics generation. For example, RDF Validator supports only the RDF/XML syntax and could not generate a model of our device description. However, it recognizes the RDF triples properly.

For conversion between different syntax formats, we used EasyRdf Converter [157], RDF Translator [373] and finally for JSON-LD optimization, the JSON-LD Playground [181] which also supports the new JSON-LD 1.1 version. These three tools are still actively developed and supported. These tools aim to be conform with Semantic Web W3C Recommendations. The EasyRdf Converter is developed in PHP 7 and the last version v1.1.1 was released in December 2020. RDF Translator was developed as part of a project supported by the BMBF authority at the Universität der Bundeswehr München. It is implemented in Python and is built on top of RDFLib v5.0 library. The JSON-LD Playground is developed in Node.js and is maintained by the W3C JSON-LD Working Group[60]. By using these tools, we could ensure that our device descriptions are consistent and syntactically well-formatted.

[60] `https://github.com/digitalbazaar/jsonld.js`

Additionally, we evaluated a device description with the OOPS!(OntOlogy Pitfall Scanner!) [279] tool. The OOPS is an on-line tool for ontology evaluation. The strength of this tool is that the authors reviewed literature about ontology evaluation and best practices including the design methodology 101 [262], evaluation criteria [120], 3Cs problems. Then, they presented a catalogue of common pitfalls in ontology development. These pitfalls were classified according to the three main types of measures for evaluation defined by Gangemi et al. [111]: structural measures, that are typical of ontologies represented as graphs; functional measures, that are related to the intended use of an ontology and of its components; and usability-profiling measures, that depend on the level of annotation of the considered ontology. Additionally, they prioritized and weighted the pitfalls. The OOPS tool was built upon this knowledge and some pitfall detection methods.

We have run the device description from the agriculture project (see Section 9.5) through the OOPS tool [61]. The evaluation result is shown in Figure 9.1.

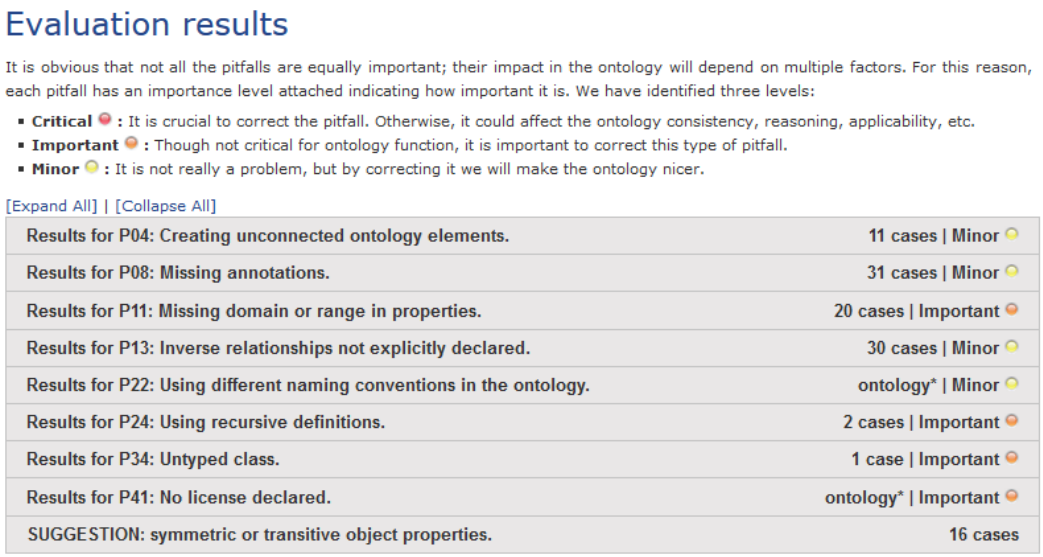


Fig. 9.1: Evaluation Results through OntOlogy Pitfall Scanner (OOPS)

The OOPS evaluation shows that no critical pitfalls were found. However, the device description has 4 important (24 cases) and 4 minor pitfalls (73 cases). Looking at the details, we can state that the most pitfall cases are coming from the oneM2M and the W3C Time ontologies. Only a few cases came from the MYNO device description.

The minor pitfall P08 means missing annotations which should be provided for human- readability (e.g. *rdfs:label* or *rdfs:comment*). These entities are: *EventFunctionality*, *ConfigurationFunctionality*, *AutomationFunctionality*, *YangDescription*, *mqttTopic*, *mqttMethod*.

The important pitfall P11 means missing domain or range in properties. Domain or range restrict the use of properties to certain objects or datatypes. These entities are; *mqttMethod* and *mqttTopic*.

Overall, we could show the correctness of device descriptions based on oneM2M ontology.

[61] `http://oops.linkeddata.es/`

Quality of Ontology
The second perspective, quality of an ontology will be evaluated with following criteria [150, 80]: adaptability, clarity, computational efficiency. Adaptability shows the ease of use of an ontology in different contexts by allowing to be extended and specialized monotonically, i.e. without the need to remove axioms. Clarity defines how effectively the ontology communicates the intended meaning of the defined terms. Adaptability and clarity of device descriptions based on the oneM2M Base ontology are discussed in Section 9.2.1. Computational efficiency relates to the speed at which tools can work with the ontology (e.g. querying, reasoners) and is measured in an experiment in Section 9.5.4.

9.2.1 Conclusion about the oneM2M Base Ontology

One of the research questions was whether the oneM2M Base ontology was the right choice. The oneM2M Base ontology is divided into machine-interpretable and human-understandable parts. The human-understandable part describes device functions in human language for device users. The machine-interpretable part describes technical interfaces of services how to use these functions. On one side, the parts seems to be doubled. On the other side, from the view of semantics, this distinction is more precise. Both is eligible. There are following questions for adaptability and clarity:

1. Is it useful to model the human-understandable part or is it simpler to add a description to the machine-interpretable part?
2. Realizing that oneM2M Base ontology is very generic, is it not better to use categories of devices/sensors for description?
3. Which external ontologies were used for extensions?

These questions are answered in the following subsections.

1. Human-understandable part
Generally from the point of semantic technologies, it is possible to model descriptions of semantic subjects and objects. There are two possibilities: *rdfs:label* and annotation properties.

rdfs:label is used to label subjects and objects. However this element is from the RDF Schema namespace and not from OWL DL (in which oneM2M ontology is defined). For this reason *rdfs:label* is not used.

The definition of *owl:AnnotationProperty* is used and described in the documentation by oneM2M as a "property that can be used to add information (metadata/data about data) to classes, individuals and Object/Data Properties."

In OWL, an annotation property is "used to provide additional information about ontology elements like classes and instances, which typically are external to the ontology and would not be used for reasoning" [24]. Example usages for such additional information are for providing a creator, a version or a comment. They will be ignored during document parsing by OWL parsers.

Summarizing, advantages about human-understandable part: differentiated descriptions and reasoning. Disadvantages about alternatives: *rdfs:label* cannot be used, *owl:AnnotationProperty* must be defined additionally and cannot be used for reasoning.

2. Usage of categories of devices/sensors for description
The oneM2M Base ontology is a very generic ontology. There are different points of view, how IoT ontologies can be defined more specifically.

Kreuzer discusses two perspectives for device descriptions in [195]: a device (or Thing) and a functional perspective of users. First, from the device perspective, such description includes technical details specific to device like its configuration, discovery, access and authentication i.e. the capabilities of the device. This information is useful for the system integration including network management. Device description can also include physical properties (e.g. temperature sensor with measurement in degree Celsius). Second, from the application view, the functionalities need to be described. The whole system is a result of aggregation, grouping and processing functionalities of devices (e.g. switch on the light in all rooms, retrieve an average humidity in a house).

Another example, Eclipse SmartHome has a strict separation between the physical world (the “Things”) and the application:

> Things are the entities that can physically be added to a system and which can potentially provide many functionalities in one. [91]

Application is "built around the notion of *Items* (also called the virtual layer)” [91]. *Items* represent functionality that is used by the application (mainly user interfaces or automation logic). *Items* have a state and are used through events. *Items* are Color, Image, DateTime, Switch, etc. Further, Eclipse SmartHome defines categories which provide meta information about Things. Categories are Camera, Door, Lock, etc. Eclipse SmartHome was archived in May 2020. The openHAB [268] is based on Eclipse SmartHome and continues to maintain its core bundles.

Michael Koster started a discussion in WoT and T2TRG about iot.schema.org and what it should do: describe the services which are available on a Thing or choose the Thing from a List? He concluded, that the description of services is more useful and flexible instead of long lists of device types.

Summarizing, depending on context, a mixed description of physical properties and offered services need to be described in the IoT.

3. Ontology extension
The device description was extended for description of heterogeneous devices. We tried to reuse existing definitions of the oneM2M ontology and from external ontologies when necessary. At the same time, as few extensions as possible were made.

The SSN ontology [138] was used to describe sensors capabilities. However, this ontology is more useful for detailed description of sensor observation results, see Section 9.2.6.

The time ontology [74] was used for configuration of events. Time ontology classes such as Interval and Duration were reused.

The Ontology of units of Measure (OM) 2.0 [298] was reused for description of units for sensor measurements. The difference whether a temperature value is given in Fahrenheit or Degree Celsius is very important.

The only new data properties defined in the device description are *mqttMethod* and *mqttTopic* for technical description of the MQTT operations. The new functionality classes were defined: Automation-, Configuration- and Event-Functionality. A *YangDescription* class was defined as a subclass of *ThingProperty* for better distinction from other properties. All other extensions are instances of classes.

For more specific domain definitions, the SAREF ontology [78] can be considered for extension or replacement of the oneM2M ontology because they have some identical core definitions.

9.2.2 Editor for Semantic device descriptions

Developers who should create and edit the ontology-based device descriptions need a knowledge of Semantic Web Standards and of appropriated tools like Protégé. However, the experience with heterogeneous devices shows that the amount of sensor and actuator types is limited. For example, a temperature sensor has always the same semantics, and a bulb has always the same functionalities: switch on, switch off and maybe dim or change color. For this reason, a simple editor would help manufacturers to create and edit device descriptions. Such editor can be a guided online form where a backend component creates the ontology-based device description, see Figures 9.2, 9.3, 9.4.

sprechende Felderbzw. Beispiel mit C0

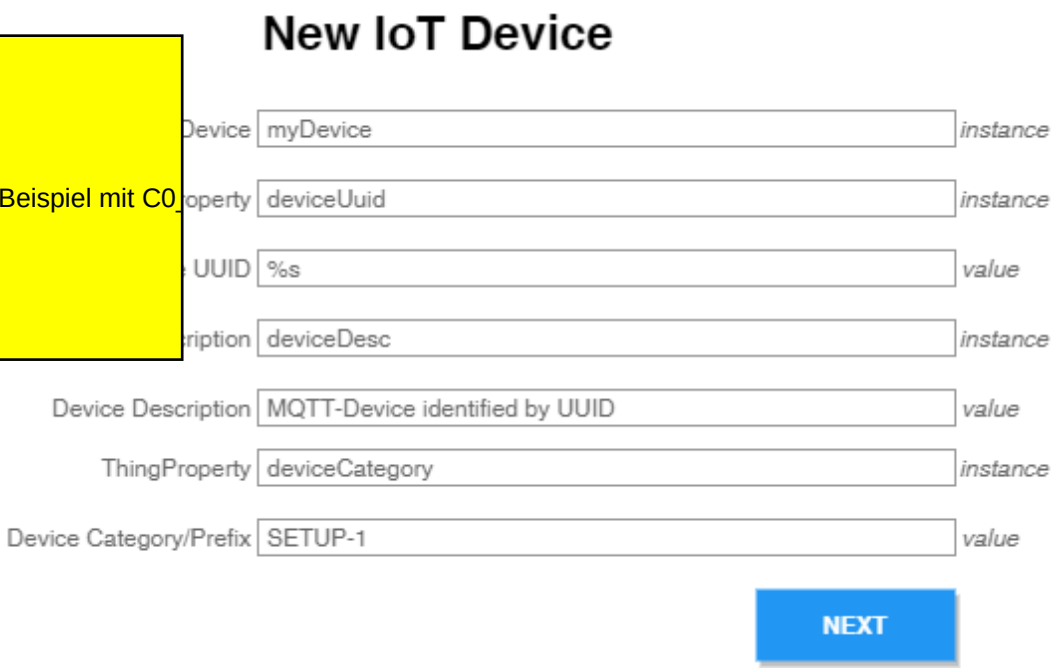


Fig. 9.2: Mockup Design for a new IoT Device

9.2.3 YANG Evaluation

One of the evaluation questions was "Why not to use YANG for device descriptions?". A YANG module defines a hierarchy of data for NETCONF-based operations: configurations, state data, Remote Procedure Calls (RPCs), notifications. The YANG modeling language tries to balance between high-level data modeling and low-level encoding. YANG can be encoded in two syntax representations: the Extensible

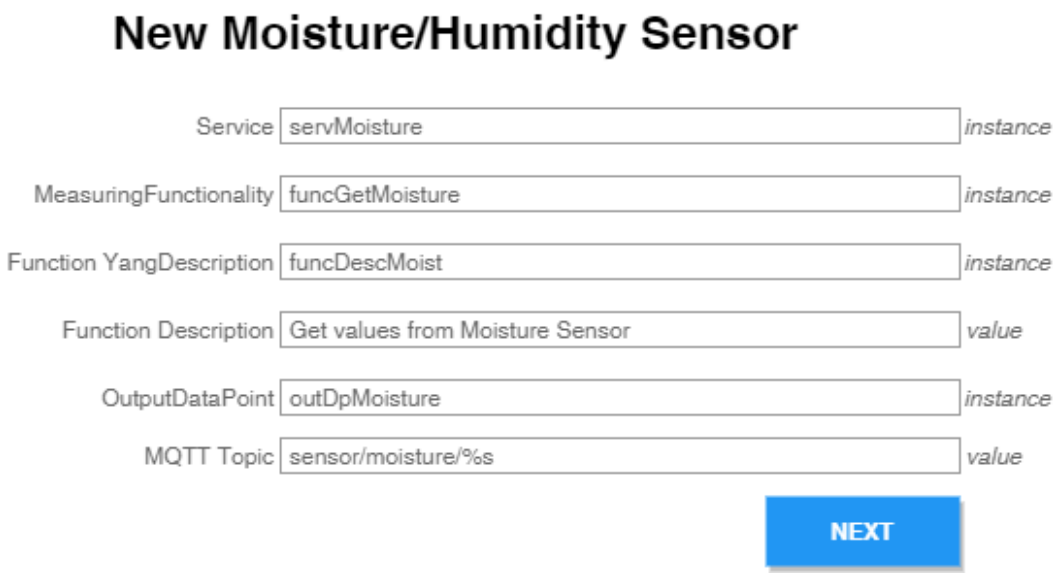


Fig. 9.3: Mockup Design for a new Sensor Function

Markup Language (XML) and SMIng-like [374]. XPath expressions can be used for querying in XML representation. Meanwhile the encoding for the JSON representation was defined in [211] to use YANG in REST-based protocols.

The generated YANG model is shown in Listing 9.1. YANG has some advantages like vocabulary for describing network devices and actuator actions via RPC call. Even though YANG is not suitable to use it directly for device descriptions:

- semantic expression is restricted;
- YANG knows only RPC calls;
- YANG is a data modeling language: it is possible to describe the existence of the actions but not possible to describe how this action should be executed e.g. MQTT subscription;
- Currently, YANG and NETCONF don't have any mechanism for pushing data i.e. of sensors. The IETF is working on this topic [66].
- XPath expressions are not that powerful as semantic web standards for querying data.

While NETCONF is using the XML-representation of YANG for datastore, the JSON representation is used for interfaces.

9.2.4 Compression Evaluation

In Section 8.3.3, we considered two compression technologies for ontology-based device descriptions, namely CBOR and RDF HDT. The compression to CBOR can be made only from the JSON-LD format. The RDF HDT library can compress data provided as RDF triples in Turtle or N-Triples format. However, the oneM2M ontology is provided in RDF XML format. We use converters to transform a device description from one format to another.

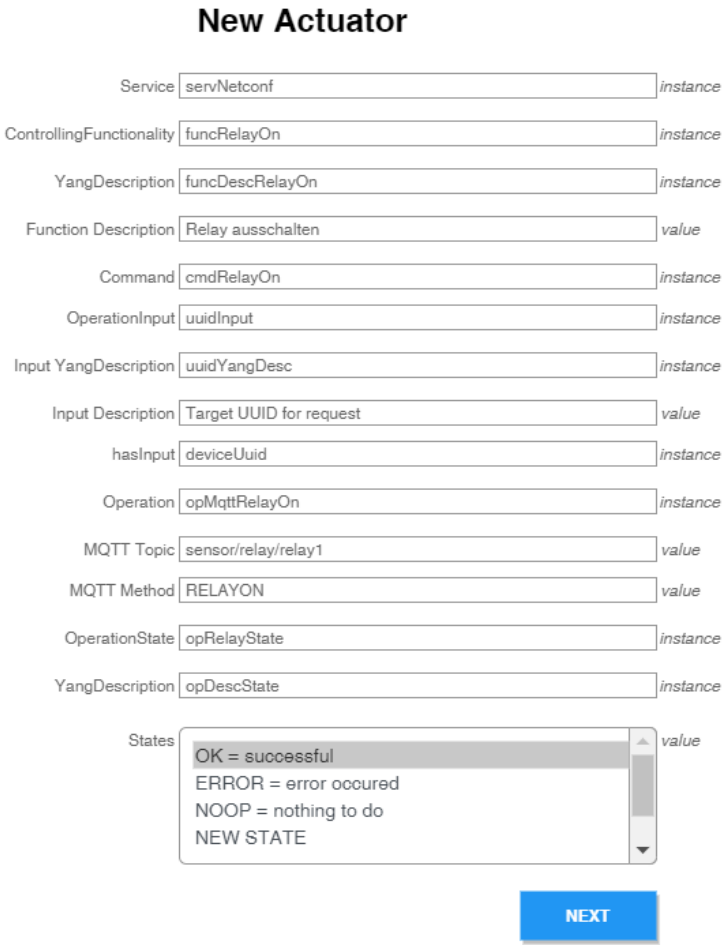


Fig. 9.4: Mockup Design for a new Actuator Function

CBOR Evaluation

For the evaluation of CBOR, we implemented a CBOR2JSON decoder and JSON2CBOR encoder in Python which is based on the CBOR library[62]. The conversion of the ontology file from RDF-XML format to JSON-LD is performed by Protégé [35]. This file is still well-formed for human-reading i.e. includes white spaces and tabs.

The results of the conversion are shown in Figure 9.5. CBOR achieved 18.43% space savings applied to non-optimized JSON-LD. The major drawback of CBOR is the poor compression of strings. Strings up to 23 Bytes length can save one Byte per string. Strings up to 256 Bytes length have no savings, and longer strings require one Byte more per string [152].

Next, we optimized the JSON-LD file by two steps: (1) We introduced context (with namespaces) for compact IRIs using JSON-LD Playground [181]; (2) we removed white spaces and tabs which are unnecessary for machine-reading. The conversion results of the optimized JSON-LD to CBOR representation are shown in the last two columns in Figure 9.5. Obviously, optimization of JSON-LD file pays off because the strings are shortened. CBOR achieves still 14.65% space savings compared to the optimized JSON-LD file.

[62] `https://github.com/agronholm/cbor2`

```
module mqtt-led {
  namespace "http://ipv6lab.beuth-hochschule.de/led";
  prefix led;

  container device {
    description
      "MQTT-Device identified by UUID";
    list device-id {
      key "uuid";
      leaf uuid {
        type string; }
    }
    leaf device-category {
      description
        "Identifies the device category";
      type string; }
  }
  rpc set_color_green {
    description
      "RPC call that sets the LIFX-Led Color to green";
    input {
      leaf uuid {
        description
          "Sends request to device specified by uuid";
        type string; }  }
  }
  rpc switch_off {
    description
      "Switches the LIFX-Led off"; }
}
```

Listing 9.1: Generated YANG model by MYNO

Thus, optimized JSON-LD and CBOR reduce the file size according to the requirements 1, 2 and 3, see Section 8.3.3. The editing of the CBOR file requires more effort to fulfill the requirement 4 because we have to replace a string in a binary encoded file. Therefore, we implemented a function CBORStreamInject in C using key-value pairs in an additional meta file. Based on a key, the position will be found in the file and the value inserted.

Figure 9.6 shows the evaluation results comparing with the RDF-XML syntax. CBOR achieved compressing rate about only 85.34% comparing to the optimized JSON-LD file.

RDF HDT Evaluation

For the compression, we use the HDT C++ Library[63]. This library accepts Turtle or N-Triples syntax of ontology files. Beforehand, we converted our device description from RDF-XML to Turtle using Protégé. Notice, there was no optimization e.g. removing unnecessary white spaces and tabs. Then we converted the optimized JSON-LD file to Turtle using the Easy RDF converter [157]. The compression results are shown in Figure 9.7. Finally, we converted the ontology file to N-Triples using Protégé. The corresponding results are given in Table 9.2. The third row shows the file size after the reverse conversion. The conversion from N-Triples results in the smallest file because of the string syntax. The N-Triples representation uses only quotation marks for strings, and Turtle adds the data type `http://www.w3.org/`

[63] `https://github.com/rdfhdt/hdt-cpp`

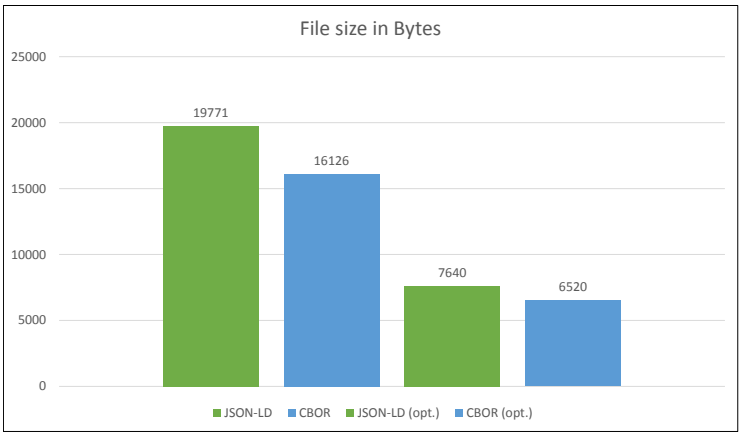


Fig. 9.5: Ontology file size in Bytes for JSON-LD and CBOR

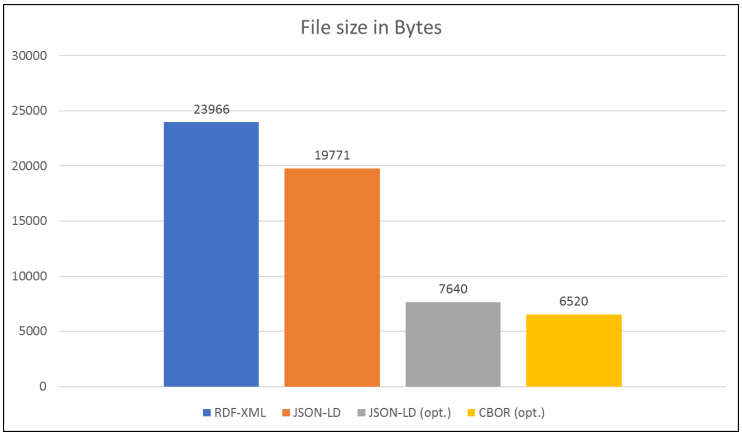


Fig. 9.6: Ontology compression results

`2001/XMLSchema#string` to every string. If this is omitted, any data type is defined for a value.

We observe space savings of 72.68% in Figure 9.7 and 84.06% in Table 9.2 because the input files have a verbose syntax. In both cases of the Turtle compression, the resulting HDT file is nearly identical (5,568 versus 5,513 Bytes). Only the so-called *Dictionary* part is shorter when using optimized Turtle as input. The reason for space savings of only 44.82%, is the optimized Turtle where the IRIs are shortened by prefix, see in the last two columns in Figure 9.7.

HDT addresses our requirements 1, 2 and 4 from Section 8.3.3 regarding small file size and RDF data structure. The requirement 3 about editing the file without prior decompression is only partly fulfilled: it is possible to exchange the string values in the Dictionary part (e.g. change a UUID) but side effects may occur because the meta data in the Header section doesn't match anymore (i.e. dictionarysizeStrings).

Tab. 9.2: Ontology file size in Bytes for N-Triples

N-Triples	27,025
HDT	4,307
HDT to N-Triples	26,848

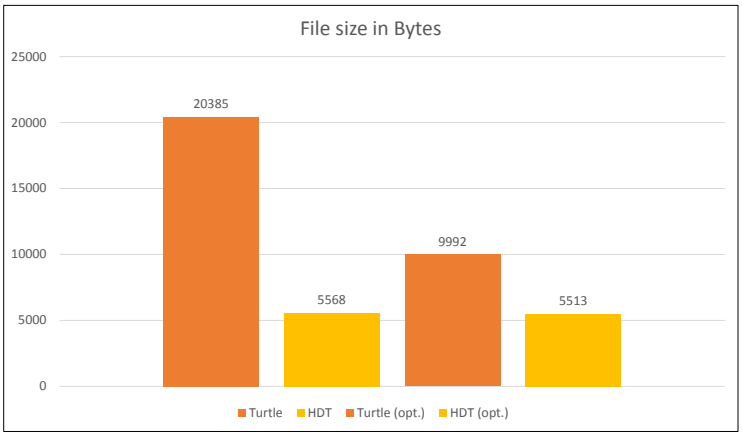


Fig. 9.7: Ontology file size in Bytes for Turtle and RDF HDT

Conclusion

We evaluated two binary representations, CBOR and RDF HDT, for ontology-based device-descriptions. HDT has shown better space savings than CBOR in our use-case. CBOR has a straightforward approach applied to data types without considering the data structure. CBOR is not well suited for the compression of ontologies, since long strings, which are the main part in device descriptions, are not efficiently compressed (less than 15 % savings in our example). A more detailed evaluation of CBOR is given in [152]. The recommendation of this evaluation is: do not exceed the length of 23 bytes for strings; replace strings by integer and literals when possible. But CBOR may still have its strength when sensor data has to be encoded for transmission.

HDT is constructed for ontology files in RDF format and therefore optimized on this structure. RDF HDT achieved better compression results compared to RDF Turtle and N-Triples files. Moreover, HDT files are smaller than CBOR files. Contrary to CBOR, HDT applies native RDF algorithms considering data structure and long strings. However, HDT has still some restrictions to edit the file due to the encoding algorithm: meta data should not be modified in the Header component; string values can be modified in the Dictionary; but it is not possible to edit the Triples component. Overall, RDF HDT is a promising candidate for further evaluation on constrained devices.

9.2.5 RDFLib Evaluation

We reproduced the measurements for RDF4Led [11] engine and tested the RDFLib v4.2 library with the same benchmark data on a Raspberry Pi Zero [372]. The RDF4Led is written in Java 7 and was optimized for RDF processing. The RDFLib is a Python library used in MYNO framework.

The test contains two steps: to fill triples into a RDF graph and then query this graph. The benchmark data is originally from the Waterloo SPARQL Diversity Test Suite (WatDiv) [7]. This benchmark provides a workload of generated 300 WatDiv tests. The input files contains 100.000 RDF triples each. There are 15 files with queries, respectively five of Linear (L), Star (S) and Snowflake (F) complexity as shown in Figure 9.8. Every file consists of 100 queries.

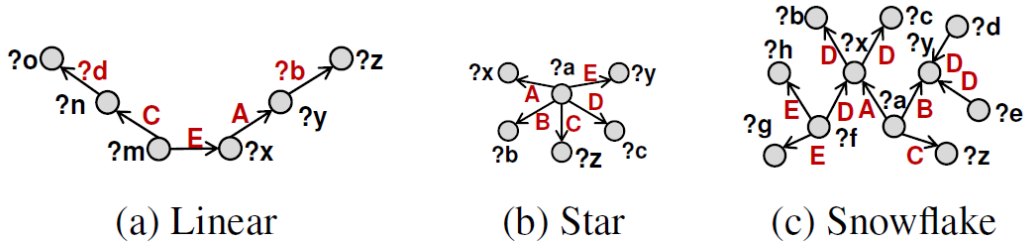


Fig. 9.8: Possible query structures in WatDiv benchmark [7]

As the edge device for tests, the Raspberry Pi Zero w is used. It hat an ARM 11 CPU with 1 GHz single-core, 512MB RAM, and a MicroSD card with 16 GB of class10, 40MB/s. The Raspbian Stretch Lite is running as an operation system.

The input throughput of RDF4Led and RDFLib is shown in Figure 9.9. In this case, higher is better. Notable is the big difference of triples per second of both lines: RDFLib starts very weak at the beginning with about 140 triples/s; RDF4Led with 1,500 triples/s. The performance of RDFLib decreases linearly slightly. The performance of RDF4Led decreases logarithmically converging.

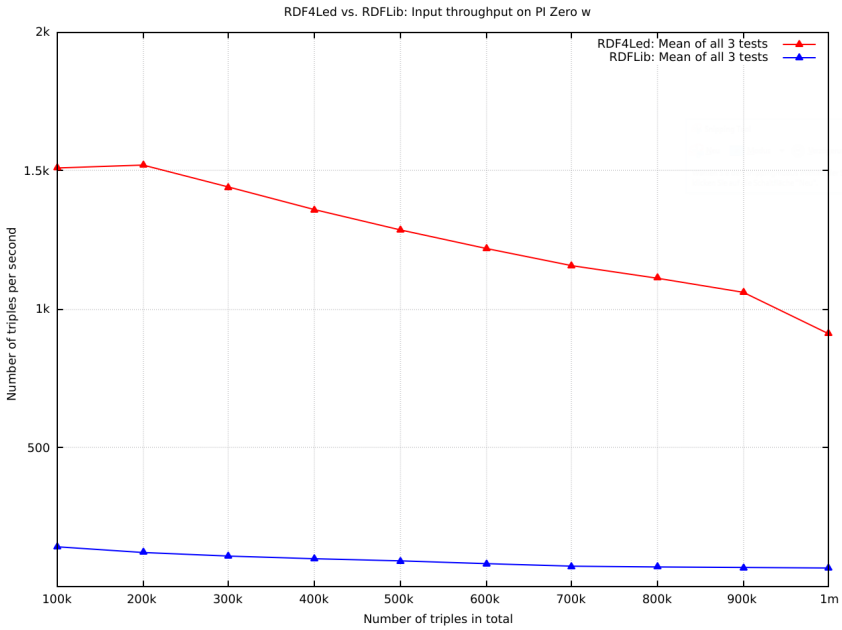
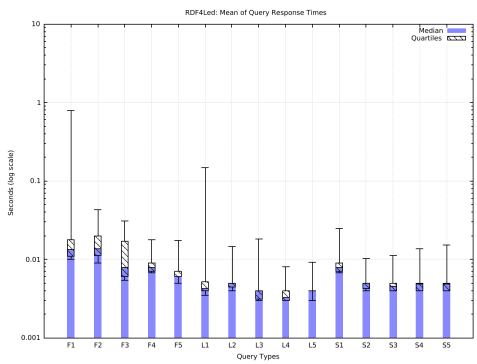


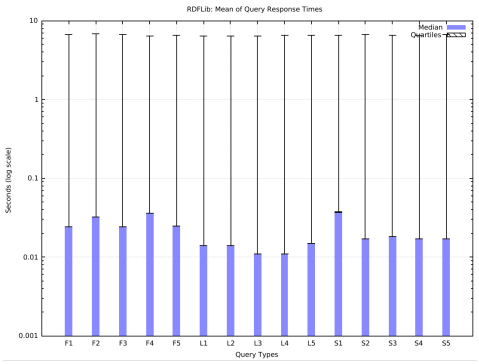
Fig. 9.9: RDF Triples Input-Experiment: RDF4Led and RDFLib in comparison [372]

Figure 9.10 shows the results from the query experiment. In that case, lower is better. The min and max values in Figure 9.10a for 1 million triples are strongly different and have no direct relation to the query templates. Noticeable in comparison in Figure 9.10b is that the max value remains roughly consistent at about 8 seconds. Each initial query in RDFLib took a long time to initialize. Each template with 100 queries was executed individually. Important note: The queries did not produce any query results. The reason could be that for the WatDiv queries the complete 30 million data sets must be available. RDF4Led can handle this amount of data [11].

Concluding, the RDFLib has a lower throughput and is not suitable for a huge amount of data on a constrained device such as a Raspberry Pi Zero. However, RDFLib is



(a) RDF4Led: Median values with 1 Mio. Triples



(b) RDFLib: Median values with 1 Mio. Triples

Fig. 9.10: SPARQL Query Experiment: RDF4Led and RDFLib in comparison [372]

sufficient enough to process the ontology-based device descriptions because they must be processed only once at the bootstrap and have a much lower amount of triples.

9.2.6 Sensor Data Evaluation

The concept of describing sensor data by SSN from Section 5.5 was tested in projects [185, 234]: on a CC2538dk board with 6LoWPAN and on an ESP-32 NodeMCU with WLAN. Both boards sent semantic sensor data based on the SSN ontology. It was feasible in WLAN network but too much data for 6LoWPAN.

Analyzing these examples, some challenges with possible solutions are recognized:

- result and observation instances must have unique names per device and after restart of a device. This a challenge because of distributed nature of devices it is hard to ensure that every device produces unique names.
- result and observation have some static and some dynamic information which are concatenated through the triples (S-P-O). Static information is always the same and therefore redundant, dynamic information is changing from measurement to measurement. These two kinds of information must be separated, because the static information can be set once i.e. during the bootstrapping. Placeholders can be set for concatenation and replaced dynamically to complete a result of observation.
- constrained devices might do not have a clock on board. The time stamp can be set by MQTT broker on message arrival.
- SSN files can have a large size. Different formats must be compared JSON-LD, Turtle, etc.
- location of sensors is an instance of the class *FeatureOfInterest* will not change for static devices. This can be configured once i.e. during the bootstrapping.

Summarizing, the possible solution is to integrate the static information into device description and dynamic information will enrich the static template on arrival with time stamp and unique names for results and observations. A disadvantage is that the device description will grow in size. Following advantages will arise:

- static information will be sent only once
- no redundant data will be transmitted
- dynamic information is consistent because of the same clock and central definition of unique names for results and observations

Sensor Data Stream Processing

When sensor data is sent periodically to an MQTT broker, stream processing concepts can be applied to MYNO framework [379]. Apache Kafka [64] is a common event streaming platform which is open-source and distributed. It is used for streaming analytics, data integration. Kafka broker follows the publish/subscribe paradigm. Kafka can be connected to MQTT by using an MQTT Connector, as shown in Figure 9.11. The connector converts the MQTT message to the Kafka message format. The connector will receive MQTT Topics for sensor data from the NETCONF-MQTT bridge upon arrival of device descriptions.

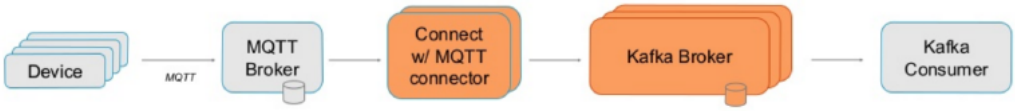


Fig. 9.11: Sensor Data Stream Processing with Kafka

On the other side of Kafka broker, a so-called *Kafka Consumer* receives the streaming data and aggregates it. The format of data can vary depending on the application use case. This collected data can be visualized by a web-based client.

The IoT devices send sensor data periodically to a sink. This process is also called *Event Streaming*. Streaming processing can be used to continuously capture and analyze sensor data from IoT devices. Apache **Kafka** is an open-source distributed event streaming platform.

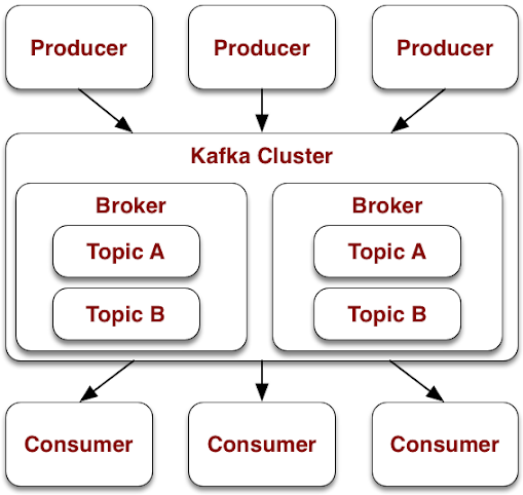


Fig. 9.12: Kafka architecture overview [379]

[64] https://kafka.apache.org/

An overview of Kafka architecture [65] is provided in Figure 9.12. Kafka collects messages and persists them. Messages are data that are published and received (publish-subscribe). Kafka is distributed over several nodes, called *brokers*, in order to scale and to ensure reliability. A Kafka cluster is coordinated by Apache ZooKeeper. Messages are grouped in named containers called *topics*. New messages are appended to the end of a topic and assigned an incremented number. This number is called the *offset*. A topic is replicated to several brokers. A topic is divided into *partitions*. The number of partitions is set when the topic is created, and it determines how the topic scales. The messages of a topic are distributed to the partitions. The offset applies per partition. *Producers* publish news in topics. *Consumers* subscribe to topics and read the incoming messages in the stream.

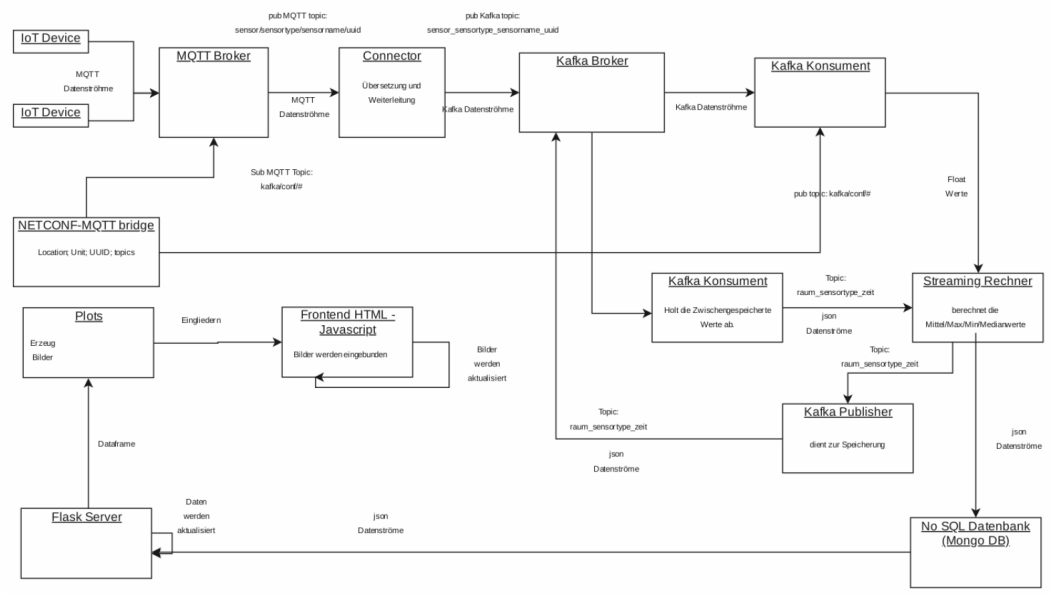


Fig. 9.13: Kafka connected to the MYNO framework [379]

Kafka was connected to MYNO framework for processing of sensor data, see Figure 9.13. A connector between MQTT and Kafka converts the messages to Kafka format. A Kafka consumer receives these messages. This consumer is a web-based application based on Flask Python library. The sensor data is aggregated and stored into NoSQL database MongoDB. The results are rendered in HTML format and AJAX is updating the diagrams on arrival of new data. The implementation with Kafka shows feasibility for stream processing [379].

9.3 MUP Performance Evaluation

Our evaluation testbed was the same as introduced in Section 8.4.1. We used a CC2531 USB-Dongle from Texas Instruments as a sniffer and the sniffing software *whsniff* (v1.3) [414] for traffic capture analysis. Detailed evaluation and optimization on the MUP is also described in [64], [313].

The MQTT client implementation of Contiki-NG defines several buffers which are shown in Table 9.3.

[65] `http://www.soutier.de/blog/2018/08/14/fast-data-intro-kafka`

Tab. 9.3: Parameter settings in Contiki-NG.

Parameter	Value in Bytes	Description
MQTT_TCP_INPUT_BUFF_SIZE	512	size of the TCP input buffer
MQTT_TCP_OUTPUT_BUFF_SIZE	512	size of the TCP output buffer
MQTT_INPUT_BUFF_SIZE	512	buffer for MQTT Input Payload
MAX_TCP_SEGMENT_SIZE	128 (default 32)	customized buffer for Output TCP segments

9.3.1 Device Description Evaluation

As mentioned in Section 8.4.2, on constrained devices such as CC2538dk with 6LoWPAN network, a device description is sent in pieces from a device to the bridge. We evaluated this transport using Tshark and MQTT logging. The ontology file was optimized for transmission: all whitespaces and line breaks are removed. This optimization reduces the file size to 17139 Bytes. This device description was sent in five pieces because the buffer on the CC2538dk is limited and is set to 4096 Bytes. Obviously, the Contiki-NG implementation is optimized for MQTT publish process. Each piece was sent in 33 TCP segments with 128 Bytes. Except the last piece which was smaller, only 1118 Bytes and 9 TCP segments. The transmission time of the whole device description was 20.815 seconds. This time seems to be long but manageable because the discovery process is only once per device.

9.3.2 Firmware Transmission Times

We measured the firmware transmission time for three different configurations shown in Table 9.4. Each configuration changed one parameter compared to the previous one (marked with green color). First, we increased the slice size from $220\,\mathrm{B}$ to $600\,\mathrm{B}$ (MUP 600), next we switched the use of the response topic alias on (MUP 600 + RTA).

Tab. 9.4: MYNO Update Protocol (MUP) configurations

Configuration	Slice size	Number of slices	Response Topic Alias
1. MUP 220	220 B	399 slices	No
2. MUP 600	600 B	146 slices	No
3. MUP 600 + RTA	600 B	146 slices	Yes

Each experiment was repeated three times and showed little deviation. In Table 9.5, the average values are shown. For each of the three configurations, the transmission time of the complete firmware is given (t_{total}) and the transmission time per slice. Further, the total traffic to and from the IoT device was measured.

By increasing the slice size from $220\,\mathrm{B}$ to $600\,\mathrm{B}$, the transmission time was reduced from $146.26\,\mathrm{s}$ to $81.54\,\mathrm{s}$. The main reason for this behavior is that the amount of traffic sent from the Update Server to the device was reduced from $199.77\,\mathrm{kB}$ to

Tab. 9.5: Measured performance metrics of the three evaluated configurations: total duration of the transmission of the entire update file (t_{total}), average duration of the transmission of a single update slice ($\overline{t_{slice}}$), total traffic over the wireless link to the IoT device ($traffic_{in}$) and from the IoT device ($traffic_{out}$).

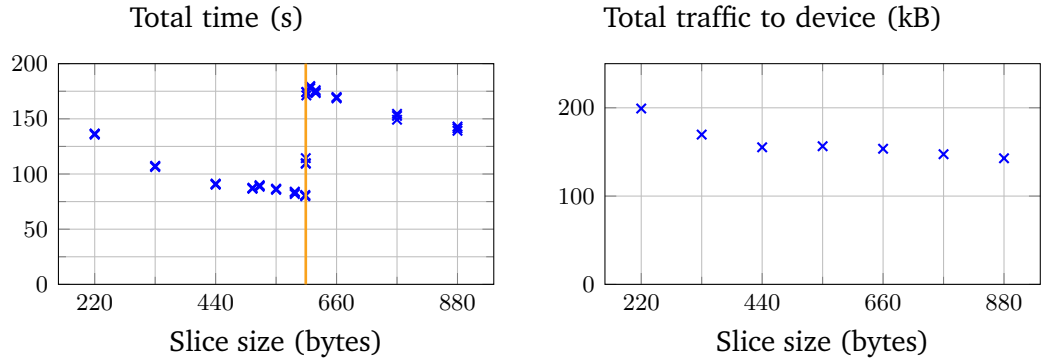
Configuration	t_{total}	$\overline{t_{slice}}$	$traffic_{in}$	$traffic_{out}$
1. MUP 220	146.26 s	0.362 s	199.77 kB	93.91 kB
2. MUP 600	85.91 s	0.575 s	151.42 kB	44.03 kB
3. MUP 600 + RTA	81.54 s	0.545 s	150.97 kB	30.19 kB

150.97 kB. In the experiments, the slice size was increased by 63.33 % but the *average* slice transmission duration was only increased by 37 %. This was caused by the reduced fixed costs due to fewer packets sent, and hence less IP, TCP and MQTT header overhead. The incoming traffic was reduced due to the lowered header overhead, and the outgoing traffic was reduced due to the decreased number of acknowledgements that must be sent.

Further, the usage of a `Topic Alias` for the response topic had also a positive effect mostly on the amount of outgoing traffic, as expected. It was reduced by 31.43 %. In the next subsections, we analyze the impact of the slice size on the transmission time and fragmentation overhead.

9.3.3 Impact of Slice Size

We measured time and traffic for the firmware update with different slice sizes. Figure 9.14a shows the transmission time of the firmware image for different slice sizes. We did measurements for varying slice sizes between 220 B and 880 B. Larger slices caused the IoT device to get stuck during slice transmissions, presumably due to a limited number of fragments supported by Contiki-NG.



(a) Comparison of total time required for the transmission.

(b) Comparison of total traffic to the device.

Fig. 9.14: Comparison of different slice sizes. The updates were transmitted in binary encoding, using MQTT ACKs and a response topic alias.

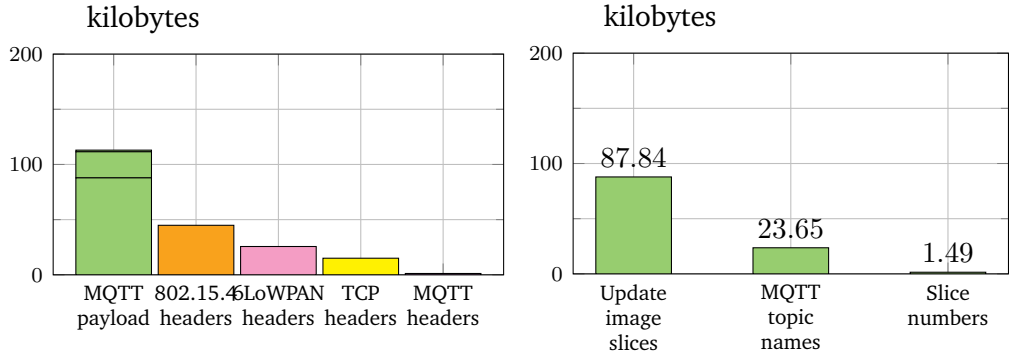
For each slice size, the results of three update runs are shown. They deviated only very slightly from each other. The transmission time lies between 79.86 s and 180 s. Larger slice sizes led to a lower total transmission time up to a threshold size of 603 B (marked with an orange line). The average transmission time measured at

a slice size of 603 B was 80.45 s. The average transmission times measured at the slice sizes of 604 B and 605 B are 111.17 s and 173.22 s, respectively. An inspection of the traffic traces shows that this is due to a delay that appeared when the MQTT message lengths are larger than 607 B. The MQTT message length consists of the slice size plus the bytes used for the slice number (between 2 B and 4 B, depending on the number of digits in the slice number).

Figure 9.14b shows that the amount of total traffic sent to the device is much higher than the 87.8 kB firmware image. For larger slice sizes the amount of total traffic decreased steadily, but only slightly for slice sizes bigger than 440 B. For example, for a slice size of 220 B the firmware image is transmitted in 399 slices which results in 399 MQTT publish messages. For a slice size of 880 B, this number is lowered to only 99 publish messages which drastically reduced the overhead caused by the long MQTT topic names. On the other hand, the overhead due to fragmentation increased which will be discussed in detail in the next section.

9.3.4 Fragmentation Overhead

Since IEEE 802.15.4 allows only a physical layer payload size of 127 B, the image slice was fragmented by the 6LoWPAN router. To illustrate the overhead due to fragmentation, we analyzed the traffic going from the MQTT Broker to the device. The capture files were analyzed using a custom Python program based on the packet parser PyShark (version used: v0.4.2.11). PyShark [283] is a Python wrapper for `tshark`, a command line tool for network analysis that comes bundled with Wireshark.



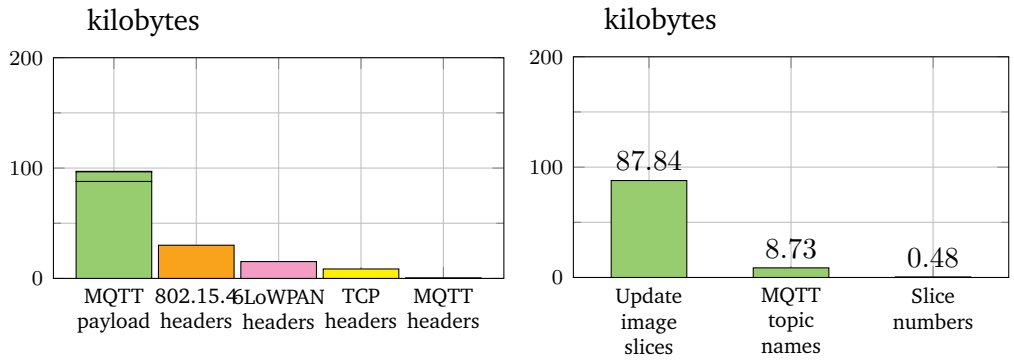
(a) Distribution of data traffic going from broker to device.

(b) Distribution of the MQTT payload going from broker to device.

Fig. 9.15: Analysis of the network traffic captured during the transmission of an update image file with a size of 87.8 kB using a slice size of 220 bytes.

Figures 9.15 and 9.16 give a summary of the amount of data which was sent between broker and device for the transfer of the complete firmware image of 87.8 kB with a slice size of 220 B and 600 B, respectively. In case of slice size 220 B the border router fragmented the message into four fragments of size 120, 96, 96, and 40 B. The 600 B slice size resulted in seven fragments (120, $5 * 96$ and 40 B).

The smaller slice size resulted in a total amount of traffic from broker to device of 199.74 kB, while the traffic was reduced to 151.39 kB for the bigger one. Figures 9.15a and 9.16a show a detailed breakdown of the traffic going from broker to device for the protocols IEEE 802.15.4, 6LoWPAN, TCP and MQTT.



(a) Distribution of data traffic going from broker to device.

(b) Distribution of MQTT payload going from broker to device.

Fig. 9.16: Analysis of the network traffic captured during the transmission of an update image file with a size of 87.8 kB using a slice size of 600 bytes.

The MQTT payload is analyzed further in Figures 9.15b and 9.16b. The MQTT topic names caused considerable additional traffic overhead. The bigger slice size reduced this overhead from $23.65\,\mathrm{kB}$ to $8.73\,\mathrm{kB}$. Since fewer MQTT publish messages were necessary, about $15\,\mathrm{kB}$ were saved just for the MQTT topic names.

Smaller slice sizes than $220\,\mathrm{B}$ were not tested. It can be expected that they would only result in worse traffic and time values due to increased header overhead and time spent waiting for ACKs. We could only expect an improvement if the slice size was decreased so much that the slices would not need to be fragmented further by the 6LoWPAN router, since fragmentation caused a performance penalty [357, p. 59], [188]. However, this was not possible at the time of writing, since the topic name alone already caused at least two fragments to be created and topic aliases could not be implemented for the update slice topic because of the lacking Contiki-NG support.

9.3.5 Acknowledgment Traffic

The *Stop-and-Wait* protocol made the implementation robust, but it was also a performance bottleneck. Figure 9.17 shows the acknowledgement traffic from the device to the broker for a slice size of $220\,\mathrm{B}$ where 399 slices had to be acknowledged. This resulted in $93.91\,\mathrm{kB}$ total acknowledgement traffic. Since Contiki-NG supports the `Topic Alias` feature for outgoing messages, we switched it on. This optimization reduced the traffic to $56.54\,\mathrm{kB}$. Additionally, we increased the value for the Contiki-NG parameter MAX_TCP_SEGMENT_SIZE to 128 for outgoing messages on the device. The acknowledgement was then sent in only one TCP segment instead of two, lowering the segmentation and packet header overhead.

Since we wanted to evaluate the overhead introduced by the *Stop-and-Wait* protocol, we implemented a MUP version where a sleep was used between the slice publish messages instead of acknowledgements. The sleep time was determined experimentally to give the device enough time to process a slice completely and be ready for the next one. For slice size 220, a sleep time of $0.3\,\mathrm{s}$ was appropriate. This allowed sending the firmware update without any acknowledgements on MQTT level and there remained only the acknowledgements traffic on the TCP level. This reduced the amount of traffic from the device to the broker to $25.7\,\mathrm{kB}$.

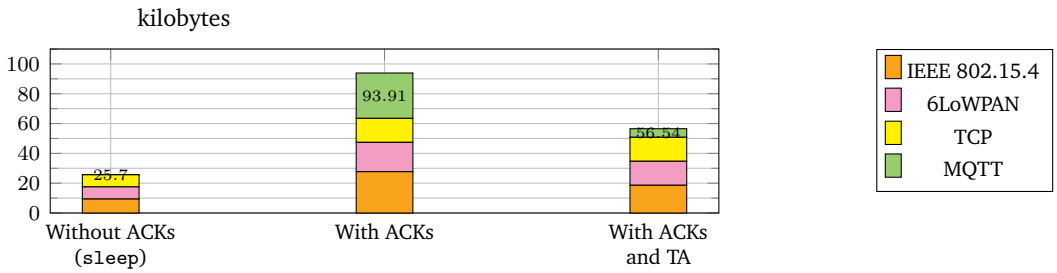


Fig. 9.17: Comparison of Acknowledge Traffic from Device to MQTT Broker for the Slice Size of 220 Bytes (Total of 399 Slices) with and without Topic Alias (TA) in Response.

9.3.6 Discussion of MQTT Implementation Issues

There are possibilities for optimization regarding constrained devices in the MQTT protocol which will be discussed in this section.

Slice Size

The experiments have confirmed that a bigger slice size reduces the protocol header overhead, since fewer MQTT publish messages are sent. However, there is an upper limit for the slice size, because the communication stack is optimized for the constrained device and uses static communication buffers. While in our test environment we achieved good performance results with a slice size of 600 B, this value obviously depends on the given hardware and software, and has to be re-evaluated for other settings.

A `Maximum Packet Size` parameter was introduced in MQTT v5. The device may set this parameter, but the MQTT broker will not inform the publisher client. Hence, the MQTT clients (publisher and subscriber) have to agree on the same packet size in advance. Instead, our prototype implementation transfers the firmware via slices which is comparable with the block-wise transfer [44] already specified in the CoAP protocol for transferring multiple blocks of information in so-called multiple request-response pairs. For the better support of constrained devices, we propose to add a similar feature to the next MQTT version. Instead of the publisher client, the MQTT broker should be responsible for slicing to the maximum packet size specified by the device.

MQTT Quality of Service

While the *Stop-and-Wait* protocol is a robust solution, it implicates overhead as shown in the detailed traffic analysis. Alternatively, Quality of Service (QoS) 1 or 2 in the MQTT protocol could be used for delivery assurance ("at least once" or "exactly once").

Additionally, in MQTT v5 a new property `Receive Maximum` is defined to control the number of unacknowledged PUBLISH packets the clients receive. In combination, this would delegate the burden of the *Stop-and-Wait* protocol down to the MQTT layer and reduce the amount of traffic. Unfortunately, the current Contiki-NG v4.5 supports QoS 1 and 2 only for *outgoing* messages [241]. For incoming messages

this is still an open issue in Contiki-NG which reflects that over the air firmware updates in the IoT is still not appropriately supported. At least, we could use QoS 1 to transmit the device description from device to broker without additional acknowledges.

MQTT Topic Alias

To use the `Topic Alias`, the clients (Update Server and IoT device) must specify that they wish to use MQTT v5 when connecting to the broker. Since MQTT v5 support is included in the newest development version of Contiki-NG, the usage of topic aliases for the messages published by the IoT device (i.e., the slice acknowledgments) has been implemented in the optimized version of MUP.

However, it was not possible to implement the usage of topic aliases for the update slice messages at the time of writing. First, the Update Server is a web application implemented in Python based on a framework called Flask. Flask offers an extension for integrating an MQTT client into a web application called Flask-MQTT [106]. This extension is a thin wrapper around the Eclipse Paho MQTT Client [271] which does not support MQTT v5 yet. It would need to be replaced by another MQTT client implementations that can be used in Python applications and already supports MQTT v5. The `gmqtt` implementation had the same problem [117], but recently fixed it in v0.6.7 [118].

Second, the MQTT broker implementation Mosquitto behaves in an unexpected way: It does not use topic aliases in outgoing messages to the subscribers, even when a topic alias was set by the publisher. Instead, it always performs a translation of incoming topic aliases back to the full topic name. Therefore, a topic alias set for the update slice topic never reaches the IoT device.

In the MQTT v5 specification, the `Topic Alias Maximum` property is defined as „the highest value that the client will accept as a `Topic Alias` sent by the Server“ [243, p. 37], which clearly implies that topic aliases were intended to be sent from the broker („server“) to subscribers („clients“). However, the specification does not clearly state that the broker MUST or SHOULD send topic aliases to subscribers when a `Topic Alias Maximum` is set. It only states that the broker must not send topic aliases to subscribers when the `Topic Alias Maximum` is not set or set to zero. Therefore, the Mosquitto broker implementation does not directly violate the specification. Still, the unexpected behavior was reported in the project's issue tracker [240] and may be fixed in a future release. This may allow update slice topic aliases to be implemented in the future.

9.3.7 MUP Conclusion

Providing firmware updates for IoT devices is one of the central questions to deal with IoT security issues. We present MUP, a scalable and secure firmware update protocol for constrained IoT devices over MQTT. The MUP protocol does not rely on TLS.

MUP follows the two-phase approach also used in update frameworks like TUF [323] and UpKit [204]. The benefit of this approach is that the energy-intensive transfer of the firmware image is only initiated by the device when the freshness of the firmware is proven. The measurements with the prototype implementation show that the

transmission of a firmware image of 87.8 kB can be done within 81.54 s, close to the results of UpKit [204]. This proves that the MUP approach is feasible within an MQTT-based IoT scenario. Further, the update protocol was easily integrated in our MYNO architecture which shows the flexibility of MYNO's semantic approach.

While the proposed update protocol could easily be integrated with an MQTT based IoT scenario, the implementation showed some missing points in the MQTT v5 specification. While CoAP supports block-wise transmission, MQTT lacks this feature. In the MUP prototype implementation, the firmware update image must be sent in *slices* because of constraints in network bandwidth and memory on the device. Therefore, it was necessary to implement a *Stop-and-Wait* protocol on the application layer, since the MQTT broker in the testbed did not support any streaming capability to the IoT devices. We analyzed the impacts of slice size and fragmentation. Both have a considerable impact on the amount of data which have to be transferred and the firmware transmission time.

An improvement of the MUP update protocol will be the extension of the key roll-over of the public vendor key using DNSSec/DANE [82].

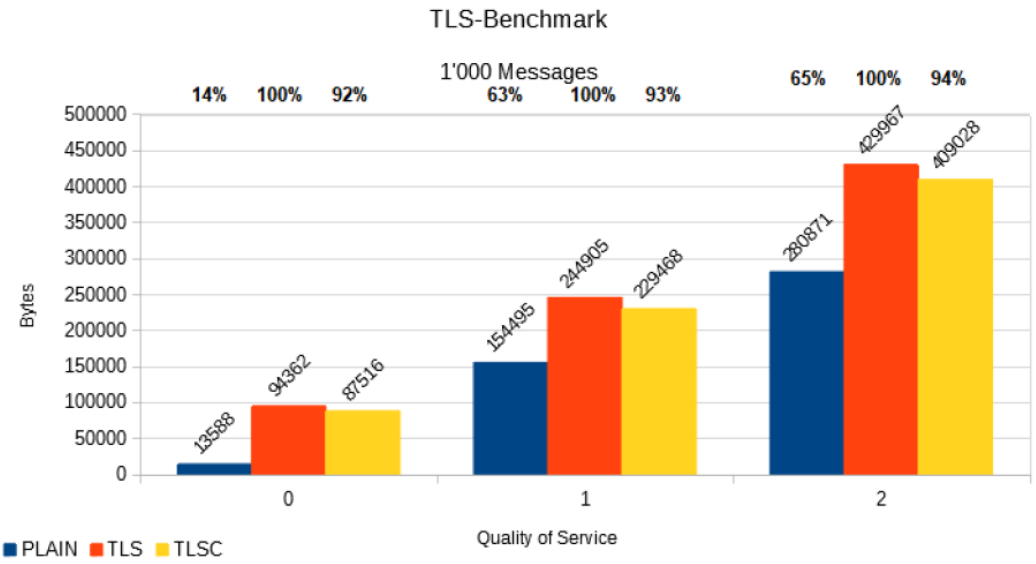


Fig. 9.18: TLS Benchmark: MQTT-Null-Message with 1000 Samples via Mosquitto (lower is better) [371]

9.4 TLS Benchmark

We analyzed security at the Arduino Nano 33 IoT board [13] and the Arduino Cloud platform [371]. The Arduino Nano 33 IoT has a crypto chip ATECC608A onboard which is a co-processor with hardware-based key storage and can store up to 16 protected keys (or certificates or data). Asymmetric cryptography with elliptic curve is implemented on the chip. Further, SHA-256 and HMAC hashing is available. The AES-128 is available for encryption of data. The Arduino board communicates with the Arduino Cloud using the MQTT protocol and the TLS secured connection. We comprehensively studied the security chip functions to understand how they work. However, a benchmark could not be directly performed on the Arduino board.

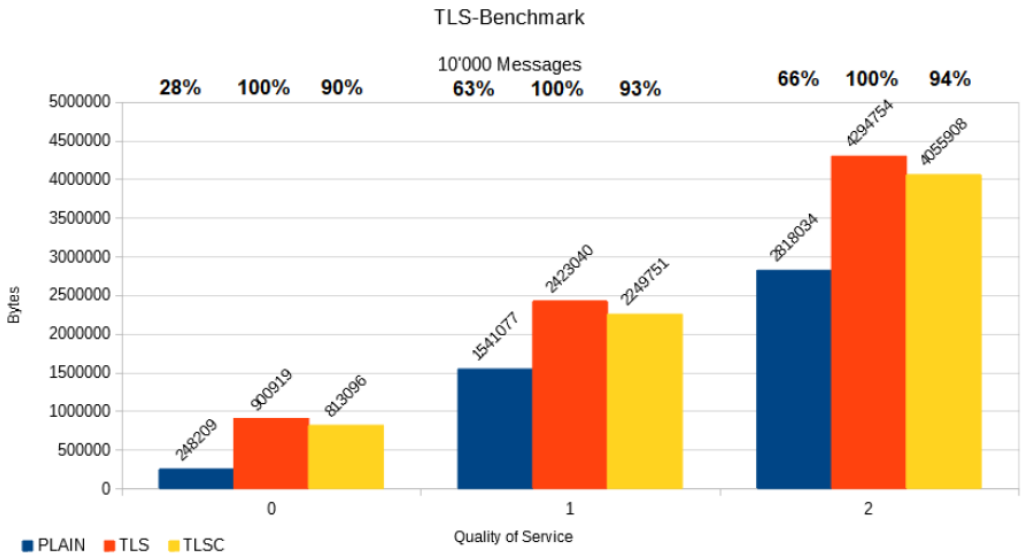


Fig. 9.19: TLS Benchmark: MQTT-Null-Message with 10000 Samples via Mosquitto (lower is better) [371]

We measured the overhead of a TLS connection using Mosquitto v1.6.7 broker and Mosquitto tools mosquitto_pub (Publisher) und mosquitto_sub (Subscriber). The used client and server certificates (TLS v1.2) are based on elliptic curves and have been signed by our own CA which conform to the specifications of the Arduino solution. This CA is used as a trust anchor in the Mosquitto configuration. The experiments were performed with three configurations of ECC certificates: none (plain), Server-only (TLS), Server & Client (TLSC). Additionally, three Quality of Services (QoS) of the MQTT protocol were tested.

The results in Figures 9.18 and 9.19 show that the overhead increases linear to the quality of service in MQTT because more messages will be transported. But the relative increase does not change significantly with 1000 or 10000 samples. For further experiments, only 1000 messages were sufficient.

The results in Figure 9.20 show the higher overhead for subscriber messages than for publisher messages. This could be explained by acknowledging of subscriber messages. The results in Figure 9.21 show no significant difference between plain message and TLS-secured messages.

Summarizing, TLS should be used if a board has a crypto chip on board. The overhead is negligible. The overhead in MQTT QoS 0 is lower because more messages in QoS 1 and 2 must be transferred.

9.5 Performance Evaluation with Precision Agriculture

Potentially, the MQTT broker is scalable[66] but the NETCONF-MQTT bridge behind it must be fast enough to process device descriptions and RPC calls. The project "WSN for Precision Agriculture on a windowsill" tests scalability of the MYNO framework

[66] https://www.hivemq.com/blog/mqtt-broker-scalability-tests/

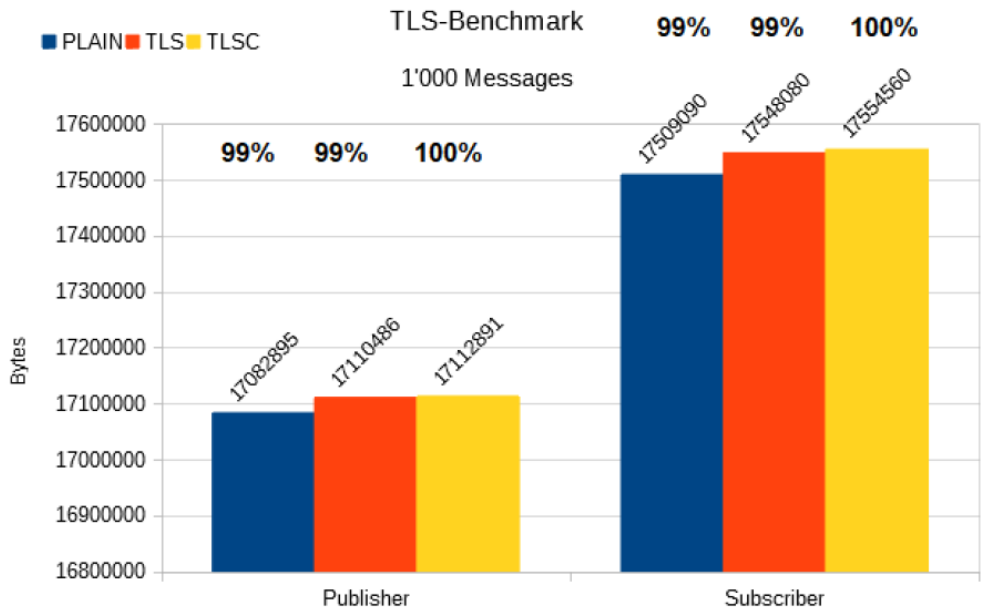


Fig. 9.20: TLS Benchmark: 16378 Bytes MQTT-Message with 1000 Samples via Mosquitto. Comparison between Publisher and Subscriber. (lower is better) [371]

with 10 devices, see Figure 9.22 for system architecture and Figure 9.23 for the used IoT devices. This project is a prototype implementation for IoT-based Precision Agriculture in a greenhouse or on a farm. Precision Agriculture [171, 219, 62] is an ongoing research field because it can help to achieve better harvest.

The requirements on our agriculture devices are: sensing environment data (air and soil); controlling irrigation; event configuration and notification at thresholds; automation of controlling functions (if-then condition).

9.5.1 Testbed

The prototype contains an edge component, a Raspberry Pi 3B, and 10 microcontroller boards which monitor 10 plants on the edge network. A single plant is representing a greenhouse or a field. The Raspberry Pi 3B has a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, BCM43438 WLAN on board. A 16 GB Micro SD card in the slot is serving the persistent memory.

A WLAN hotspot is installed in the room as an access point for Raspberry Pi and devices. They are not connected to the internet. The Raspberry Pi has running: MQTT broker from Mosquitto, the NETCONF-MQTT bridge and the web-based NETCONF Client as well as a Virtual Device.

The microcontroller boards are based on the ESP32 NodeMCU Module[67] from AZ-Delivery. They have a low-cost ESP-WROOM-32 processor with WLAN 802.11 b/g/n, 160MHz Tensilica L108 32 bit Dual-Core CPU, 512 KB SRAM and 16 MB flash memory. Every EPS32 board was extended through a breadboard equipped with sensors and actuators. The following sensors are wired with the breadboard:

[67] `https://www.az-delivery.de/products/esp32-developmentboard`

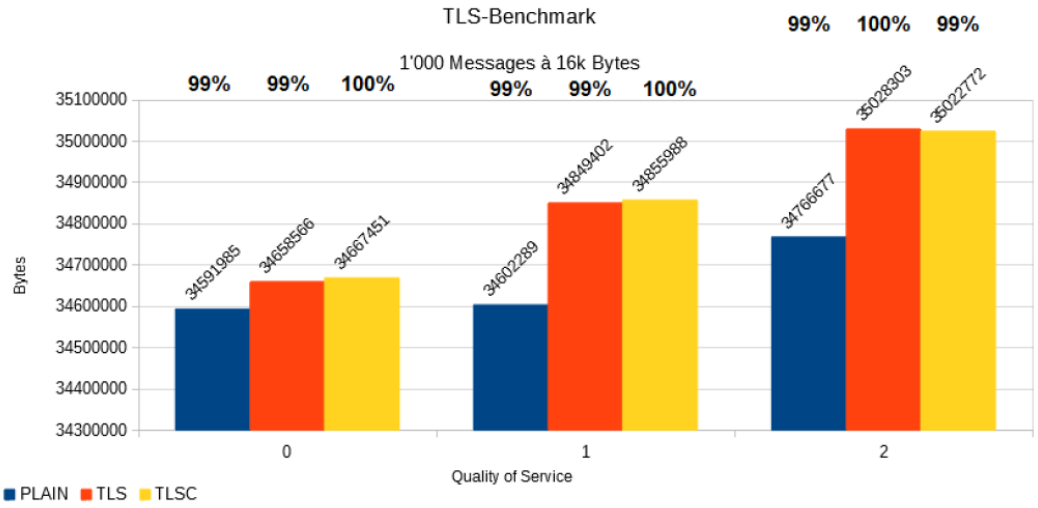


Fig. 9.21: TLS Benchmark: 16408 Byte-MQTT-Message with 1000 Samples via Mosquitto (lower is better) [371]

- capacitive soil moisture sensor v1.2 [68] determines the dielectric constant of the soil which is an indicator for dry or wet soil;
- GY-302 BH1750 light sensor[69] measures intensity of visible light in lux;
- three sensors, namely temperature, humidity and air pressure, are combined in a GY-BME280[70] module which measures air condition;
- raindrops sensor [71] measures the conductivity of the surface with the help of electrical voltage and switches at one adjustable threshold. The more raindrops are on the sensor the higher it gets the analog output value.

The following actuators are deployed on the breadboard:

- KY-016 RGB LED module[72] can be used for state signaling like a traffic light;
- 1-relais 5V KY-019 module[73] controls the pump;
- 5V mini water pump with external power supply (2 AA batteries) and watering pipe is controlled through the relais.

The power supply for a EPS32 board is ensured through a powerbank connected over the micro USB port. The breadboard wiring for ESP32 is shown in Figure 9.24.

A virtual device is a component which can be started optionally on the edge. The main task of such device is the aggregation of the devices on the edge. The virtual device subscribes to the bootstrapping topics and analyzes the device descriptions to collect controlling and measuring functions as well as configuration and automation functions. The virtual devices publishes its own device description to the MQTT

[68] https://www.az-delivery.de/products/bodenfeuchte-sensor-modul-v1-2
[69] https://www.az-delivery.de/products/gy-302-bh1750-lichtsensor-lichtstaerke-modul-fuer-arduino-und-raspberry
[70] https://www.az-delivery.de/products/gy-bme280
[71] https://www.az-delivery.de/products/regen-sensor-modul
[72] https://www.az-delivery.de/products/led-rgb-modul
[73] https://www.az-delivery.de/products/relais-modul

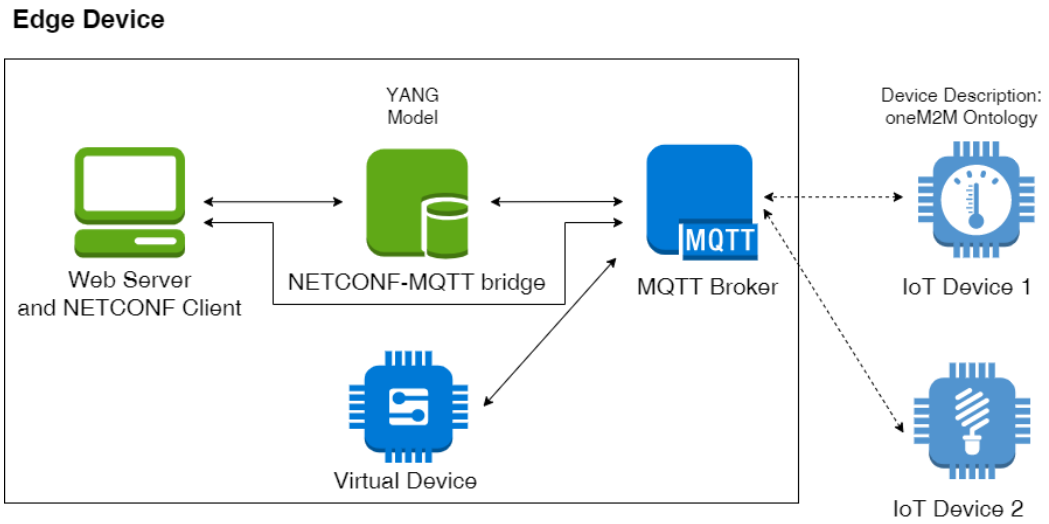


Fig. 9.22: MYNO System Architecture for Precision Agriculture

Fig. 9.23: Testbed for the Precision Agriculture Project

broker and therefore appears as a managed device in the bridge and on the Web Client GUI.

9.5.2 Device Description

A device description must contain all capabilities which are provided by such agriculture device. The following controlling functions are defined:

1. switch the RGD LED on and off;
2. switch the RGB LED with a given RGB color;
3. turn the water pump on and off;

For sensors, the device description was extended by reusing the OM-2 ontology [298] to provide units of measurements. The values in units are not directly provided by

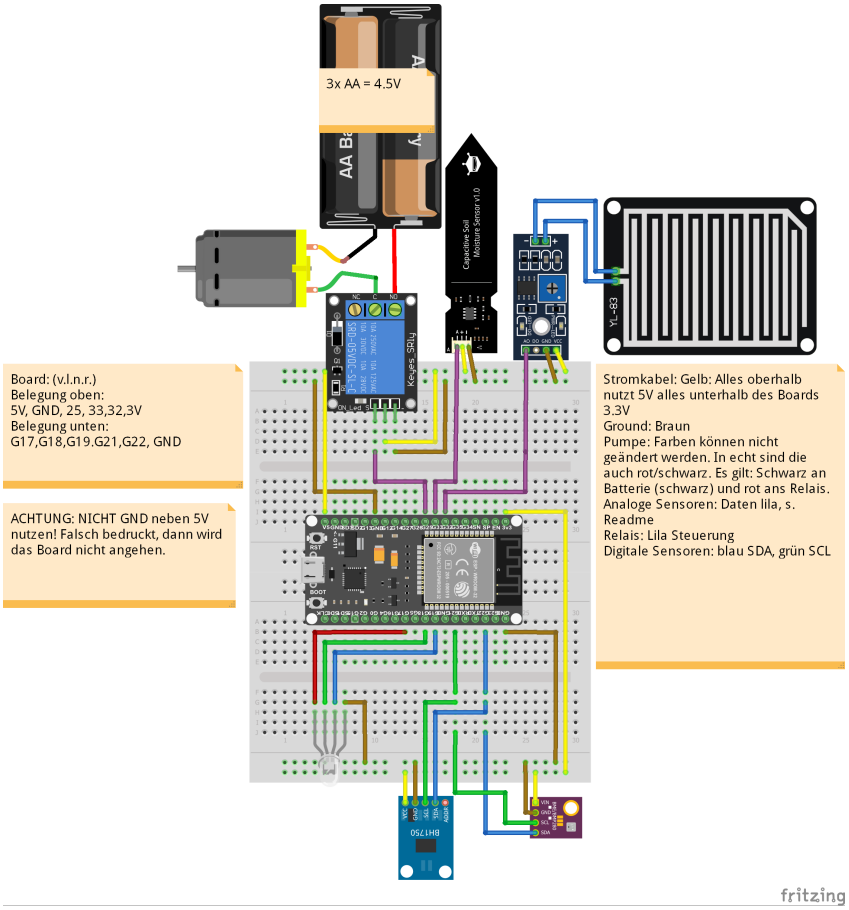


Fig. 9.24: ESP-32 NodeMCU breadboard for Precision Agriculture

the sensors but calculated using formulas from the measured sensor values. The following sensor measurements are defined in the device description:

1. soil moisture in percent;
2. brightness in lux;
3. air humidity in percent;
4. air pressure in hectopascal;
5. air temperature in degree Celsius;
6. raindrops detection in percent;

For event configuration and notification, the device description was extended by a new class for configuration functions. Such configuration defines a threshold value for a sensor as well as an interval and duration for an event notification. Additionally, a name and a CRUD operation for this configuration must be defined. The difference to the controlling function is not only in the parameters which are always the same but also an MQTT Topic for publication of events like sensor values. The device description is reusing the TIME ontology [74] to provide ontology classes for interval and duration. The configuration functions are defined for two critical sensor measurements: soil moisture and air temperature. For example, events should be published every 10 seconds during the next 60 seconds when the soil moisture

is under 30 percent. The overview of the parameters for so-called configuration functions:

- configuration/event name as a string;
- configuration operator has predefined values: >, <, ==, >=, <=;
- threshold for sensor value;
- interval value in seconds starts with 1;
- duration value in seconds starts with 1;
- CRUD operation for event configuration;

The automation function (if-then condition) is defined in the device description as a combination of a configuration function and a controlling function instead of event. For example, if the soil is dry then turn the water pump on or switch the RGB LED to red. Such automation functions can be used for event-based processing on a device instead of the event-based processing on the edge or cloud.

The screenshot of the web-based NETCONF client for this project is shown in Figure 9.25. The configuration of thresholds can trigger events, shown in yellow fields.

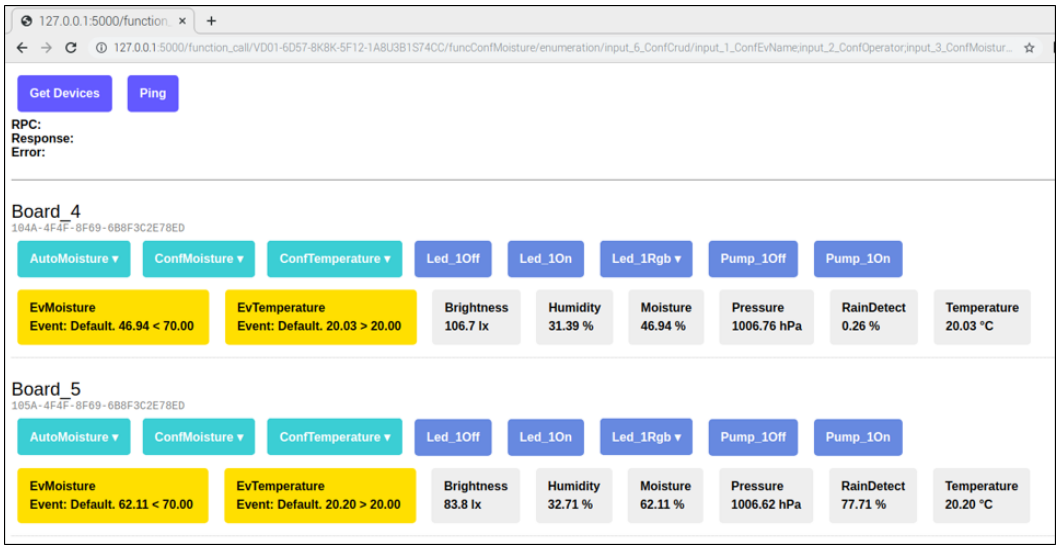


Fig. 9.25: Web-Client for Precision Agriculture

9.5.3 Performance Evaluation

The evaluation criteria are defined according to the guides of Jain [172].

1. Goals of the experiment and system boundaries
The goal of the experiment is to show that the MYNO framework is scalable and robust when running several IoT devices on the edge. The performance evaluation considers only messages with device descriptions and sensor values. The actuator, configuration and automation messages are not considered because their occurrence is marginal.

2. System services and possible outcomes
The system services are represented by the following MQTT messages in MYNO:

- device descriptions;
- sensor values;

As possible outcome, it must be ensured that all published messages arrive at their subscribers and are processed correctly.

3. Performance metrics
The monitoring tool *vmstat* will be used for performance measurement on the Raspberry Pi. Following performance metrics will be measured then:

- CPU load;
- RAM usage;

The `tshark`, a command line tool for network analysis that comes bundled with Wireshark[74] tool will be used for the network traffic analysis and delivers the following metrics:

- time for transmission of a device description;
- time for transmission of sensor messages;
- percentage of sensor messages which get lost or are retransmitted;

For the processing of semantic descriptions by the chosen RDFLib library, following metrics are important:

- time for processing of a device description in the bridge and virtual device;

The energy consumption will be measured by the live time of the full-loaded power-banks.

4. System and workload parameters
The system parameters for Raspberry Pi 3B are:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU;
- 1GB RAM;
- 16 GB micro SD card;
- BCM43438 WLAN a/b/g/n;

The ESP32 NodeMCU Module has following system parameters:

- 160MHz Tensilica L108 32 bit Dual-Core CPU;
- 512 KB SRAM;
- 16 MB flash memory;
- WLAN 802.11 b/g/n;

There are following workload parameters:

1. number of connected devices: up to 10 devices are available
2. number of sensor messages (each device has 6 sensors): published periodically

[74] `https://www.wireshark.org/`

All system components communicate through WLAN on 2.4 GHz basis. The power supply for a EPS32 board is ensured through a powerbank connected over the micro USB-B 5V port. There are two powerbank types used:

- Schwaiger LPB220 533 powerbank with capacity of 2200 mAh (4 items with one port)
- Conrad Tapfer portable charger 6000 LED - Model No. 1001CTF with capacity of 6000 mAh (3 items with two ports)

A further parameter is the periodical sleep state of microcontrollers which is used in order to reduce the power consumption.

5. Factors and their values
The factors are a varying number of connected devices, amount of sensor messages and the duration of the sleeping state between the messages. The values for the number of devices are 1, 3, 6, and 10. The amount of sensor messages results from the number of devices and how often the sensor messages are sent. The duration of sleeping state is the time between the messages defined in the experiments.

6. Evaluation techniques
The evaluation technique is measurement of performance metrics.

7. Workload
For workload, the number of running devices and sensor messages can be configured. Notice, some sensor values cannot be measured very often (e.g. soil moisture sensor) because of their measurement method. The sensor manufacturer define a minimum period between two measurements which must be considered.

8. Design of the experiments
The experiment is divided into three sub-experiments:

1. Device Description Experiment
2. Sensor Messages Experiment
3. Energy Experiment

The factors are different in the experiments.

1. Device Description Experiment
During the bootstrap for one device, the metrics for device descriptions will be measured:

- time for transmission (`tshark`);
- time for processing in the bridge and virtual device;
- CPU and RAM usage in the Raspberry Pi;

Then, the amount of devices will be increased up to 3, 6 and 10 devices. Additional devices will be switched on one after another.

Expectation: CPU and RAM usage will increase minimally but the time for processing and transmission does not change.

2. Sensor Messages Experiment
The first run will be done by 1 device which is running with the recommended interval between measurements suggested by sensor vendors. The interval between

sensor messages is shown in Table 9.6. This is the starting reference measurement. During this first run, the time for transmission of only one sensor (brightness) will be evaluated. The sensor messages are comparable because such MQTT messages consist only of a topic and a sensor value.

Tab. 9.6: Recommended sensing intervals

Sensor	Interval in sec.	MQTT Topic
Soil Moisture	600	`sensor/moisture/moisture_1/UUID`
Raindrops	300	`sensor/rain/rain_1/UUID`
Brightness	60	`sensor/brightness/brightness_1/UUID`
Air Temperature	60	`sensor/temperature/temperature_1/UUID`
Air Pressure	60	`sensor/pressure/pressure_1/UUID`
Air Humidity	60	`sensor/humidity/humidity_1/UUID`

Then, the amount of messages will be increased by increasing the number of devices up to 3, 6 and 10 devices. CPU load and RAM usage on the Raspberry Pi will be measured. The sensor messages will be logged by Mosquitto client.

The running time duration is 1 hour per experiment. Every experiment will be repeated 3 times. Total time result in 12 hours.

Expectation: CPU load and RAM usage on the Raspberry Pi will increase minimally when the message amount increases. Retransmissions and also the higher transmission time are expected because of interferences when several devices are sending.

3. Energy Experiment
The following sub-experiments can run at the same time. Two devices are running with the fully loaded power bank, the recommended sensing interval and without sleeping states. They are running until the power bank is unloaded. The timestamps are logged by Mosquitto client. Two other devices are running with the fully loaded power bank and with sleeping state between the messages for 30 seconds.

Expectation: The sleeping state extends the live time of the powerbank.

9. Data analysis and interpretation
Data will be analyzed for each experiment separately. The performance metrics will be compared and interpreted. Average and median values will be calculated for repeated runs. Anomalies will be analyzed. The log of the components will be analyzed e.g. for runtime errors. All messages must arrive at the subscriber and be processed correctly. If this is not the case, the reasons must be found.

10. Results presentation
Measured results will be depicted as diagrams or tables.

9.5.4 Performance Results

CPU Load
The devices were switched on one after another with 60 seconds pause which is

one sensor interval. The CPU load of 1, 3, 6, and 10 devices is shown in Figures 9.26, 9.27, 9.28, 9.29. The CPU load goes high after receiving a device description but after the short processing, the CPU goes down. The green line shows the Time spent running kernel code, and the blue line shows the time spent running non-kernel code. Thus, obviously RDFLib performs some kernel tasks. There are also some short peaks which show that the processing of device descriptions is a challenging task.

RAM Usage
The RAM usage increases only slightly, independently how many devices are connected, as shown in Figures 9.30, 9.31, 9.32. Cache and buffer are rising linear but only a little, see Figures 9.33, 9.34 for three devices example.

Device Description Evaluation
We measured the time for transmission of device descriptions by Tshark. The results are shown in Table 9.7. There are some outliers: the max value for three devices and the min and the median values for six devices. Anywhere else, the average and other values are close together.

Tab. 9.7: Time for the transmission of a device description in seconds (rounded)

Amount of Devices	AVG	MAX	MIN	MEDIAN
1 Device (3 values)	0.154827	0.198501	0.076399	0.189580
3 Devices (9 values)	0.213094	0.809358	0.072936	0.178104
6 Devices (18 values)	0.118933	0.226045	0.025865	0.093394
10 Devices (30 values)	0.145933	0.243770	0.072548	0.141123

The Table 9.8 shows the time for processing of a device description where the most part is the processing by the RDFLib library. The processing for just one device description takes long comparing with the other results. But looking closer, the max value is much higher than the median. In fact, the max value is always needed for the first device description after restart of the bridge. All following device descriptions will be processed much faster. Obviously, there is some initializing work by the RDFLib before the first RDF querying.

Tab. 9.8: Time for the RDFLib processing of a device description in the bridge in seconds (rounded)

Amount of Devices	AVG	MAX	MIN	MEDIAN
1 Device (3 values)	12.556165	12.778631	12.437580	12.452284
3 Devices (9 values)	8.551245	14.338671	5.430500	5.922269
6 Devices (18 values)	8.116861	14.419649	5.372922	7.516252
10 Devices (30 values)	7.564038	13.978273	5.624919	7.284434

Sensor Messages
The time was measured for transmission of temperature sensor messages by TShark. It is lying between 0.000074375 and 0.000127449 seconds. Because of such low values, further evaluation of results is omitted.

There were no temperature sensor messages lost. The percentage of temperature sensor messages which are retransmitted:

- 1 Device: 89 messages arrived, 2 spurious retransmissions

- 3 Devices: 261 messages arrived, 5 spurious retransmissions
- 6 Devices: 521 messages arrived, 11 spurious retransmissions
- 10 Devices: 842 messages arrived, 26 spurious retransmissions

The amount of such spurious retransmissions increase linearly by the increased number of devices. It is not clear why these retransmissions appear.

Energy Efficiency
We did some experiments with power banks with a sleep state and without it. The Schwaiger LPB220 533 powerbanks have a capacity of 2200 mAh. They were fully loaded. The sensor intervals used from Table 9.6. The powerbanks were running 16h 36m 47s (996 temperature sensor messages) and 18h 11m 42s (1090 temperature sensor messages). Then, we activated the sleep state of 30 seconds on the boards. Due to limitations of the power bank, it was not possible to use longer sleep states. The boards shut down a part of the hardware, then they wake up, measure, publish values and sleep 30 seconds. This time, the time life of the powerbanks lasts much longer, namely 1d 17h 51m and 1d 15h 38m.

Energy efficiency is more important for energy-constrained devices i.e. powered by batteries. Therefore, we optimized some processes in the MYNO framework to support energy efficiency. For example, in bootstrapping before sending a device description, the NETCONF-MQTT checks whether the device is already registered. Unnecessary transport of a device description can be avoided. Additionally, the compression of a device description reduces transport energy cost. And the device description is usually sent only once. Also, the image update process was optimized to save energy.

9.5.5 Conclusion

The processing of the device description is a demanding task. It takes 5 to 14 seconds for RDFLib processing. However, only the first device description takes the maximum of time, all following descriptions are processed much faster. Obviously, there are some initial loading of RDFLib library. The CPU load increases up to 60 - 100 percent but also stays constant for 10 devices. The RAM usage increases only a little . Further, we could see that the sleep state extends the live time of the energy supply.

9.6 Conclusion

Overall, we made many experiments with the MYNO framework to evaluate its approach. First, we proofed the concept and feasibility of the MYNO framework according to different criteria. The comparison with the NETCONF specification, RFC 6241, shows that the most operations and messages are implemented, except configuration editing and notification. It is not possible to edit the semantic device descriptions on the constrained devices due to low computational power. For this reason, notifications about configuration changes are not necessary. The proof against RFC 7547 which addresses requirements for management of networks with constrained devices shows that the most requirements with high priority are fulfilled by the MYNO framework: e.g. self-configuration capability, support of multiple device classes, consistency of data models.

The MYNO framework was tested with heterogeneous devices such as CC2538dk from Texas Instruments, Arduino Yún Rev 3, and ESP-32 NodeMCU, and IP-based networks such as 6LoWPAN and WLAN.

We evaluated devices descriptions from two perspectives: correctness and quality. For the correctness evaluation the OOPS tool [279] was used. As a result, only few issues regarding ontology modeling were found. The quality of device descriptions was analyzed regarding the choice of the oneM2M ontology as the base ontology and its extensibility. Some aspects of device could be modeled in another way but overall, the chosen ontology fulfilled the requirements. We proposed a simplified user interface for creation of device descriptions and discussed why YANG is not appropriate instead of ontology with the Semantic Web Standard.

Reducing the overhead which is produced by the Semantic Web Standards, we evaluated CBOR and RDF HDT for optimization of ontology-based device descriptions for use on constrained devices. The evaluation shows that CBOR is not suitable for long strings and RDF HDT is a promising candidate but is still a W3C Member Submission. Finally, we used optimized JSON-LD format for the syntax of device descriptions.

We measured the performance of the Python RDFLib library which we used for processing of device descriptions. The Python Library has a lower throughput comparing with the RDF4Led [11] but it is sufficient enough for the processing of device descriptions during the bootstrap.

The streaming and collection of sensor data was accessed. The first approach was to describe the sensor data with semantic annotation using RDF triples. The second approach showed the sensor data stream processing with Apache Kafka. Both approaches have advantages and disadvantages and should be considered depending on the application requirements.

The MYNO Update Protocol (MUP) for firmware update was evaluated on constrained devices CC2538dk and 6LoWPAN. The MYNO update process is focused on freshness and authenticity of the firmware. The evaluation shows that it is challenging but feasible to bring the firmware updates to constrained devices using MQTT. As a new requirement for the next MQTT version, we propose to add a slicing feature for the better support of constrained devices. The MQTT broker should slice data to the maximum packet size specified by the device and transfer it slice-by-slice.

A TLS benchmark was performed to investigate the overhead caused by encryption. The overhead was negligible. Even constrained devices such as Arduino Nano 33 IoT board with a built-in crypto-chip can use TLS with MQTT.

For the use case precision agriculture, a scalability project was built with 10 boards at the edge of the network. The ESP-32 NodeMCU boards, connected by WLAN, were equipped with six sensors and two actuators. A performance evaluation shows that the processing of ontology-based descriptions on a Raspberry Pi 3B with the RDFLib is a challenging task regarding computational power. Nevertheless, it is feasible because it must be done only once per device upon the discovery process.

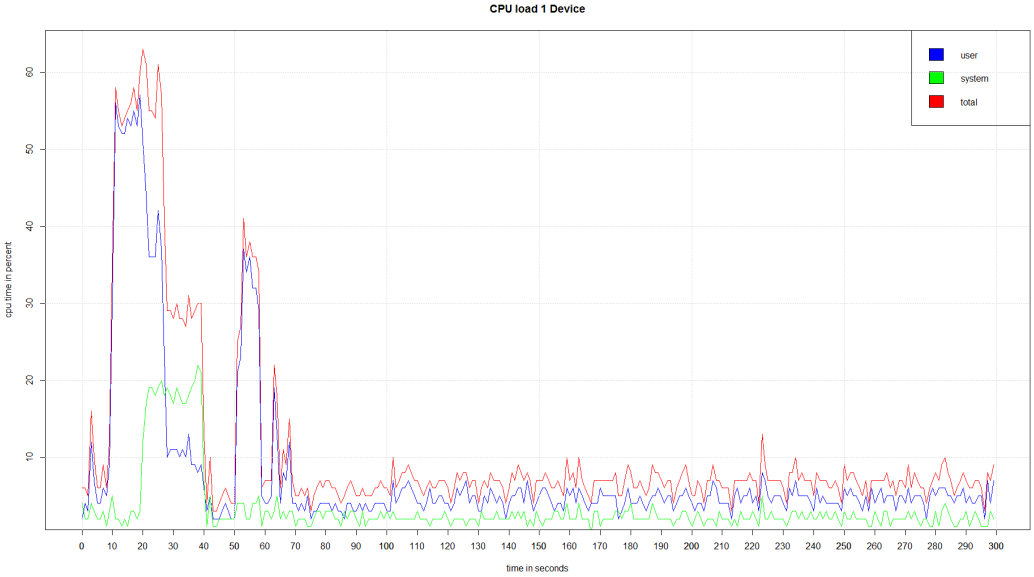


Fig. 9.26: CPU Load of 1 Device

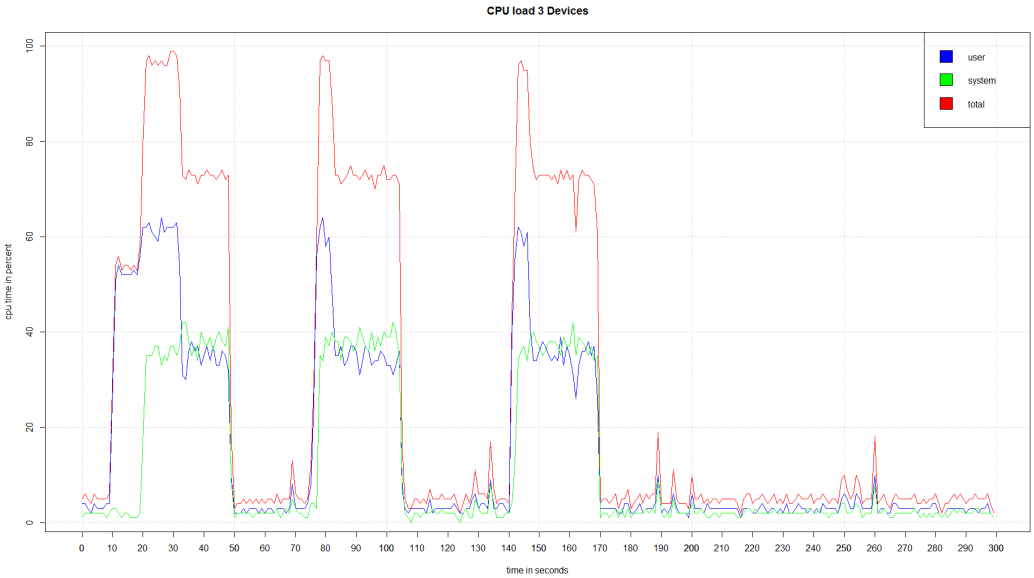


Fig. 9.27: CPU Load of 3 Devices

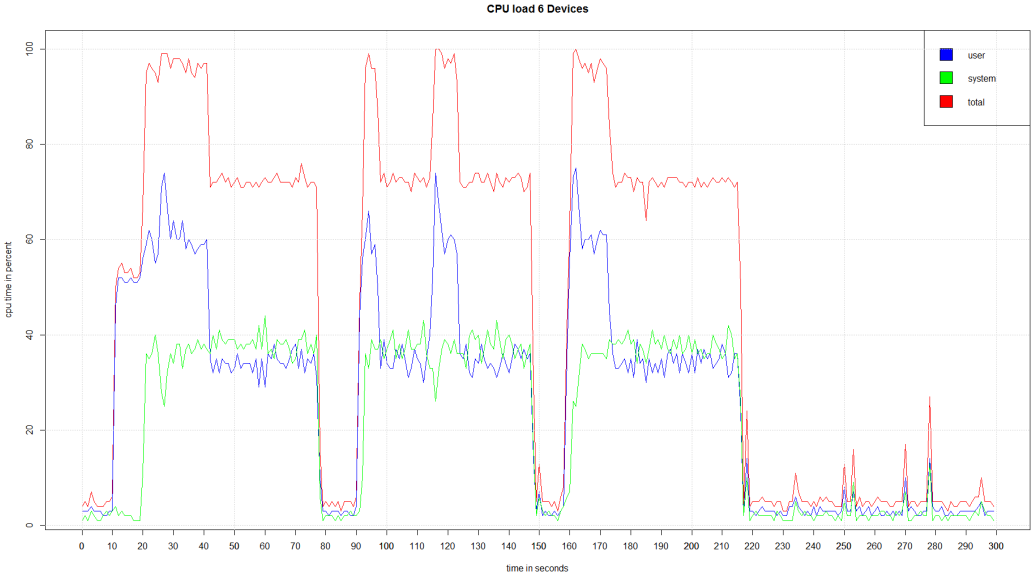


Fig. 9.28: CPU Load of 6 Devices

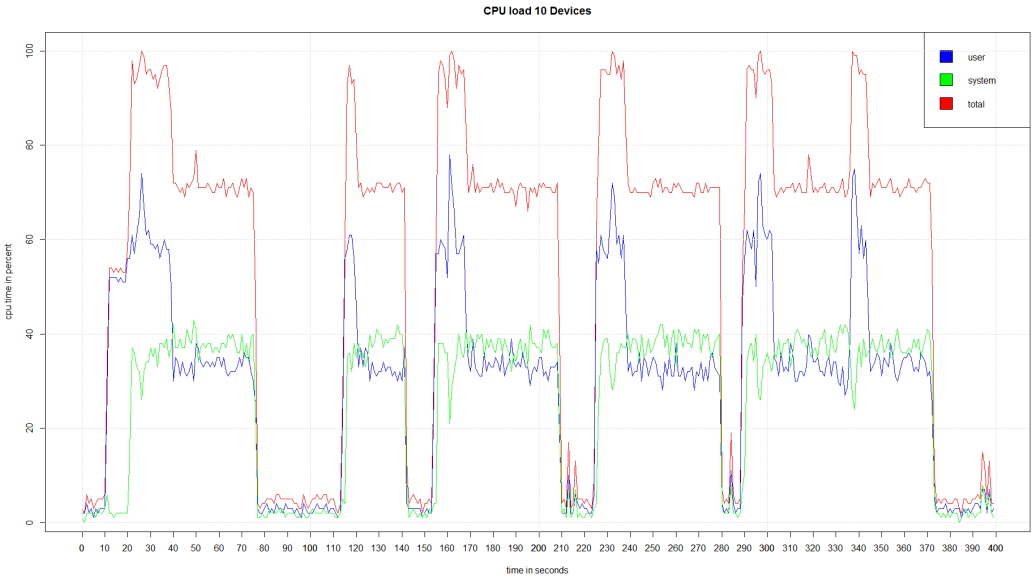


Fig. 9.29: CPU Load of 10 Devices

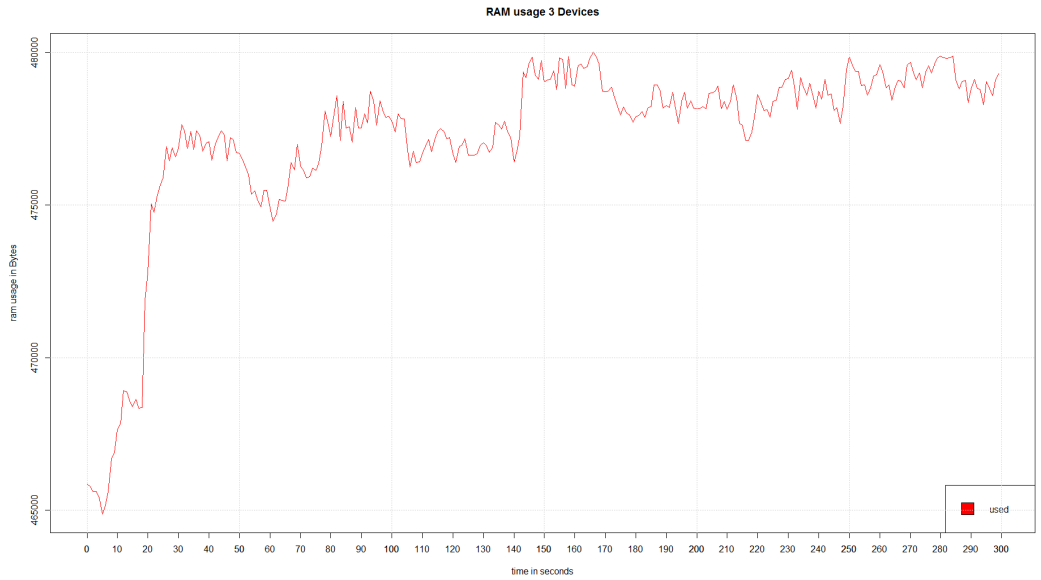


Fig. 9.30: RAM Usage of 3 Devices, Used Memory

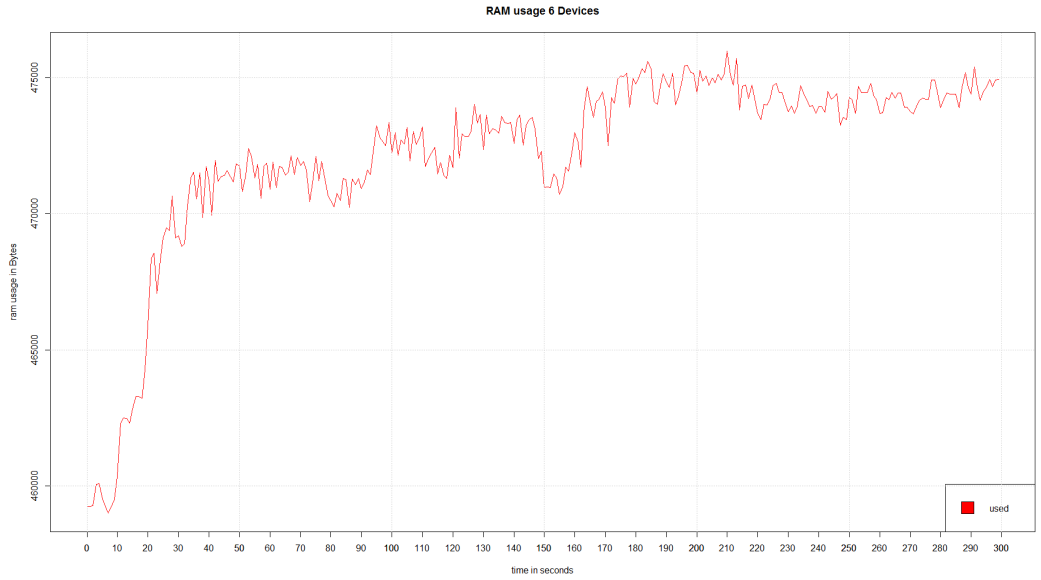


Fig. 9.31: RAM Usage of 6 Devices, Used Memory

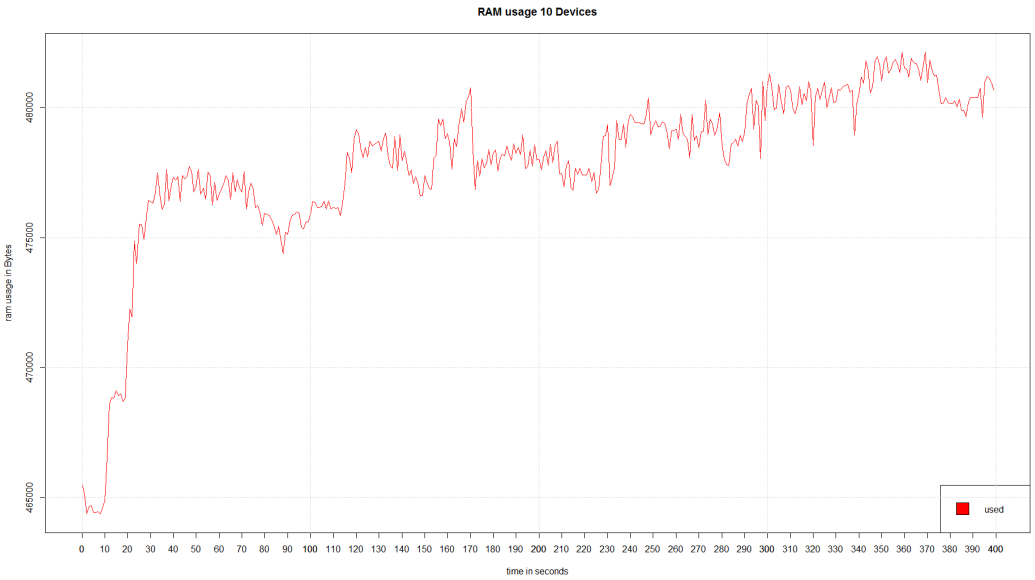


Fig. 9.32: RAM Usage of 10 Devices, Used Memory

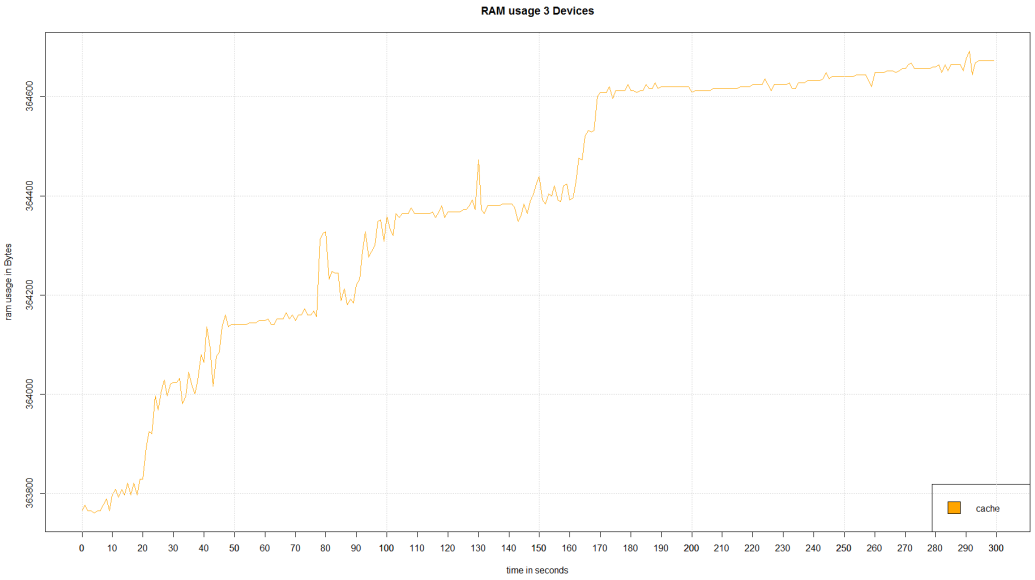


Fig. 9.33: RAM Usage of 3 Devices, Cache

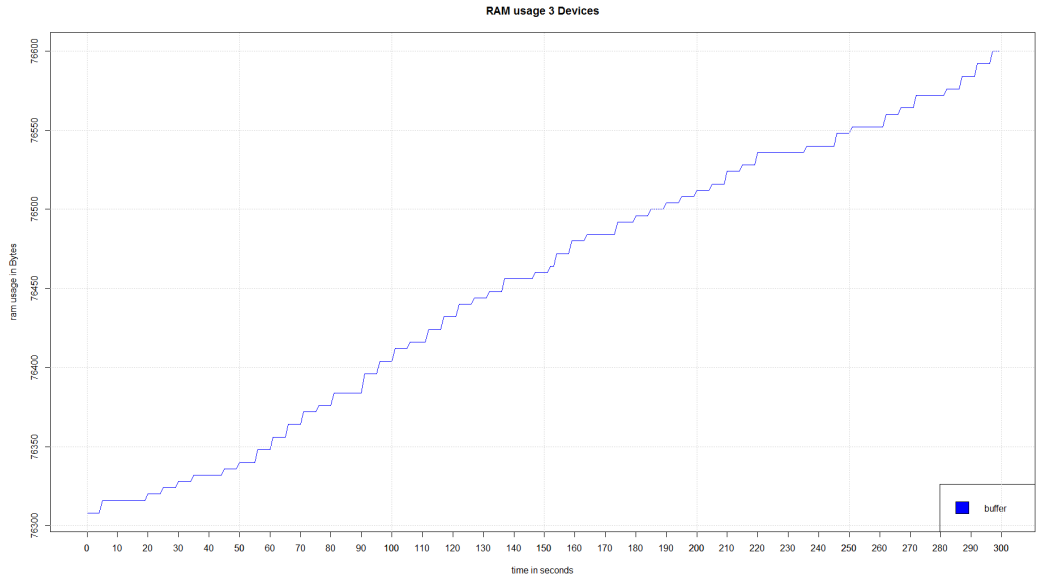


Fig. 9.34: RAM Usage of 3 Devices, Buffer

Conclusion and Future Work

10

> *It is always something. Good, Fast, Cheap: Pick any two (you can't have all three).*
>
> **— RFC 1925, Nr. 7**
> (The Twelve Networking Truths)

We conclude this thesis according to the RFC 1925 [53]: "in protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away".

This thesis proposes a framework for network configuration management of heterogeneous, constrained IoT devices by using semantic descriptions for interoperability. The NETCONF protocol is used as the network configuration protocol. The MQTT protocol was applied by IoT devices to connect to a network. The oneM2M Base ontology was selected among IoT ontologies. The ontology-based device descriptions define the device capabilities and can be used by a Virtual Device for aggregation at the edge of the network. Compression mechanisms were evaluated in order to reduce the overhead produced by semantics. The firmware update was identified as a central security task in the network configuration management. The MYNO Update Protocol (MUP) was introduced and evaluated on constrained devices. The TLS overhead was measured and can be negligible.

We implemented and tested microcontroller boards such as CC2538dk, ESP-32 NodeMCU and Arduino Yún with sensors (e.g. temperature, humidity, light, motion, and smoke) and actuators (e.g. LEDs, relays). The performance study with 10 ESP-32 boards of Class 2 with 6 sensors (temperature, humidity, light, air pressure, soil moisture, rain drops sensor) and two actuators (RGB LEDs, pumps) and WLAN demonstrated the feasibility of the MYNO framework.

We proofed the concept and feasibility of the MYNO framework according to different criteria. The comparison with the NETCONF specification, RFC 6241 [93], shows that the most operations and messages are implemented, except configuration editing and notification. It is not possible to edit the semantic device descriptions on the constrained devices due to low computational power. For this reason, notifications about configuration changes are not necessary. The proof against RFC 7547 [94] which addresses requirements for management of networks with constrained devices shows that the most requirements with high priority are fulfilled by the MYNO framework: e.g. self-configuration capability, support of multiple device classes, consistency of data models.

The ontology-driven device description and the MYNO source code including the implementation of the NETCONF-MQTT bridge, the Update Server and the device application are available as open-source [312].

10.1 Discussion of Results

The semantics-based approach was chosen for the bridge between NETCONF and MQTT protocols. The semantic device descriptions can be used to support the interoperability in network management on the IoT. Back to the introduction with interoperability model for the IoT, Figure 1.3, the results on MYNO can be summarized for the single levels:

- Level 1: Physical Interoperability: achieved connectivity through IEEE 802.15.4 and WLAN.
- Level 2: Network and Transport Interoperability: achieved through IP-based networks and protocols like IPv4/IPv6, 6LoWPAN and TCP.
- Level 3: Integration Interoperability: achieved through application protocols MQTT and NETCONF.
- Level 4: Data Interoperability: achieved through RDF and OWL in ontology-based device descriptions and semantic sensor data, YANG models for NETCONF protocol.

Basically, the MYNO framework could be used in all IP-based networks because the relying protocol MQTT is transported by TCP over IP. The IETF 6lo working group describes the use cases for IPv6 over constrained node networks in [154] and specifies IPv6 over: Bluetooth Low Energy in the RFC 7668 [256], ITU-T G.9959 networks in the RFC 7428 [46], Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE) in RFC 8105 [220]. This group argues that IPv6 is an unified way to address the things in the IoT.

The NETCONF-MQTT bridge offers an NETCONF server interface and responds the generated YANG data model which could be consumed by any NETCONF client, for example from Yumaworks[75]. The defined RPC calls should be supported without restrictions but it is not possible to edit or delete a YANG configuration because constrained devices cannot parse its own device description. Next, containers for sensor data with MQTT topics are defined in the YANG model. Therefore, a client should be able to subscribe to these topics. In our case the web-based NETCONF client uses an additional MQTT library for displaying sensor data. Otherwise, sensor data could be consumed by the bridge and sent as notifications to the NETCONF client. However, such push notifications [65] are recently defined and not supported by our Python library. Thus, completeness cannot be fulfilled for the NETCONF protocol.

The data transmission over MQTT in a WLAN network worked well, as the performance evaluation in Section 9.5 showed. However, it is challenging to transmit several kilobytes of data in 6LoWPAN networks. The device descriptions during the bootstrap process and the firmware update file in the MUP protocol must be sent in *slices* because of constraints in network bandwidth and memory on the device. Therefore, we propose block-wise transmission for the MQTT protocol like in CoAP protocol [44]. The evaluation of the MQTT v5.0 showed that some new features are still not supported by the implementations (e.g. MQTT Topic Alias).

[75] `https://www.yumaworks.com/tools/netconf-client/`

The data interoperability for IoT devices was achieved through semantic device descriptions. The additional value of the ontology-based approach is the underlying model which represents data meaning and self-descriptive, machine readable and re-usable. These advantages are used by the bridge and especially by the concept of Virtual Device for aggregation of device capabilities. IETF identified the advantages of semantics and started to work on Semantic Definition Format (SDF) [193] for data and interaction models in the IoT.

The device descriptions are based on the oneM2M Base ontology. Generally, the underlying ontology could be exchanged (e.g. through W3C Thing Description [184]) and only the SPARQL queries must be adjusted. A unified ontology would be desirable for description of device capabilities. There are some recent efforts, for example, iotschema.org [166] and One Data Model[76]. The schema.org vocabulary is founded and used by the big four search engines. Websites can use this vocabulary for search engine optimization. The iotschema.org is an extension for the IoT and defines an IoT vocabulary which can be used for the M2M interaction. One Data Model was initiated by Bluetooth, OCF, OMA SpecWorks and Zigbee, and uses the SDF format.

The semantic approach has its roots in the earlier proposed Ad-hoc Semantic Internet Protocol (ASIP) [318] and Micro-Ontology Context-Aware Protocol (MOCAP) [316]. The definition of semantic annotation in Section 5.5 is close to our identified dimensions for sensor data in [318]: topics, types, times, locations, sender, receivers, approvers. The sender is identified by an UUID and receivers are determined by the MQTT subscription. An approver could be the MQTT broker which could confirm the provenance of a message. However, the implementation of semantic sensor data showed that semantics brings the overhead and most of data is redundant, see Section 9.2.6. Thus, static semantic sensor data should be transferred only once, and all following data should contain only dynamic data, e.g. sensor values and time stamps.

The compression of the device descriptions was evaluated in Section 9.2.4 by applying CBOR and RDF HDT. The evaluation showed that CBOR is not suitable for long strings and RDF HDT is a promising candidate but is still a W3C Member Submission. Finally, we used optimized JSON-LD format for the syntax of device descriptions. We proposed to use a micro-ontology in [316]. This is a snippet of an ontology (e.g. only instance definitions). However, the processing software (the bridge in our case) must hold the whole ontology (e.g. class definitions).

One of the security tasks of network management is the distribution of firmware updates. The MYNO Update Protocol (MUP) was developed and evaluated on constrained devices CC2538dk and 6LoWPAN. The MYNO update process is focused on freshness and authenticity of the firmware. The evaluation shows that it is challenging but feasible to bring the firmware updates to constrained devices using MQTT.

For the performance and scalability evaluation of MYNO framework, we setup the testbed with 10 ESP-32 NodeMCU boards at the edge of the network. The ESP-32 NodeMCU boards, connected by WLAN, were equipped with six sensors and two actuators. The performance evaluation shows that the processing of ontology-based descriptions on a Raspberry Pi 3B with the RDFLib Python library is a challenging

[76] urlhttps://onedm.org/

task regarding computational power. Nevertheless, it is feasible because it must be done only once per device upon discovery process.

Overall, this framework goes in the right direction as the current developments in the standardization organizations show. Examples are the Web of Things (WoT) [389] and the Thing Description from W3C; the CoAP Management Interface (CORE-CONF) [403], and Software Updates for Internet of Things (SUIT) [239] from IETF.

The used protocols and libraries are still actively maintained and were lifted to new versions: MQTT 5.0, RDFLib 5.0, Contiki-NG 4.6, JSON-LD 1.1, YANG 1.1. This shows that the approach in this thesis is an emerging field.

10.2 Future Work

For productive deployment of the MYNO framework, some improvements on dependability, fault tolerance and security will make the framework robust. First, a definition for dependability is provided:

> We understand the dependability of a system to be the property of not assuming any inadmissible states (functional safety) and ensuring that the specified function is performed reliably. (translation from [89])

If an IoT device is failed, it is mostly manageable because there are many of them. However, they can be monitored by a periodic state request. The MQTT broker and the NETCONF-MQTT bridge in the MYNO framework have to provide dependability and fault tolerance. In general the broker does not store messages. However, retained messages, persistent connections and QoS levels can result in messages being stored temporarily on the broker/server. The MQTT Broker is the single point of failure. The solution is clustering which is not a part of the MQTT specification. A commercial broker like HiveMQ [149] implements its own solution. A MQTT broker cluster is a distributed system that represents one logical MQTT broker.

The NETCONF-MQTT bridge is also a single point of failure. The bridge stores the generated YANG model as a file. However, it does not load this configuration during the restart. Thanks to bootstrap process, the IoT devices resend their device descriptions if they notice that they are unknown in the network. For robust functionality, the configuration and the RPC mappings should be stored and loaded during the start in case if the bridge fails. The Virtual Device should also store and load the aggregated device description. A recovery procedure for possible network changes must be defined in case of failure.

The initial pre-configuration of devices (e.g. WLAN access and an IP-address of a MQTT broker) can be improved by using Bluetooth or a Micro-USB cable. Additional security mechanisms (e.g. TLS) can be pre-configured in this way. If TLS should be used, the MYNO components must be adjusted in order to support it.

There are also some minor considerations for the future work. The oneM2M Base ontology for device descriptions was extended in a generic way. To make these descriptions domain-specific, other ontology (e.g. SAREF) could be used instead. Then, the NETCONF-MQTT bridge must be enabled to process other ontology.

During implementation, we experienced some challenges with software libraries and hardware. Software libraries for MQTT client and broker require broader support for MQTT features, e.g. for MQTT Topic Alias and Quality of Service. This would lead to better performance on constrained devices. The chosen NETCONF server and client libraries do not support the notification feature. Another library is required or the existing library must be extended. The hardware components, namely ESP-32 boards, sensors and actuators, were sometimes volatile in their function. There was no obvious reason, however, low quality or loose wired contact could be a consideration. A printed circuit board would reduce such effects. The energy supply for IoT devices (e.g. in precision agriculture use case) could be improved through a small solar panel.

Finally, sensor data should be collected, stored and analyzed. As shown in [379], Kafka could be used for stream processing.

However, the MYNO framework was running stable on a Raspberry Pi 3B for some weeks and it is a lean solution for the edge computing.

Appendix

Listing 1: Optimized Device Description for Precision Agriculture with 4 Use Cases: Sensors, Actuators, Configurations, Events.

```
{   "@context":
     {"owl": "http://www.w3.org/2002/07/owl#",
     "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
     "xsd": "http://www.w3.org/2001/XMLSchema#",
     "base": "http://yang-netconf-mqtt#",
     "onem2m": "http://www.onem2m.org/ontology/Base_Ontology/
        base_ontology#",
     "om-2": "http://www.ontology-of-units-of-measure.org/resource
        /om-2/",
     "time": "http://www.w3.org/2006/time#"},

   "@graph": [
     {"@id": "onem2m:Command", "@type": "owl:Class"},
     {"@id": "onem2m:ControllingFunctionality",
         "@type": "owl:Class"},
     {"@id": "onem2m:Device", "@type": "owl:Class"},
     {"@id": "onem2m:MeasuringFunctionality",
         "@type": "owl:Class"},
     {"@id": "onem2m:Operation", "@type": "owl:Class"},
     {"@id": "onem2m:OperationInput", "@type": "owl:Class"},
     {"@id": "onem2m:OperationOutput", "@type": "owl:Class"},
     {"@id": "onem2m:OperationState", "@type": "owl:Class"},
     {"@id": "onem2m:OutputDataPoint", "@type": "owl:Class"},
     {"@id": "onem2m:Service", "@type": "owl:Class"},
     {"@id": "onem2m:ThingProperty", "@type": "owl:Class"},
     {"@id": "onem2m:Variable", "@type": "owl:Class"},

     {"@id": "onem2m:cmdPump_1Off",
         "@type": ["owl:NamedIndividual", "onem2m:Command"],
         "onem2m:hasInput": {"@id": "base:uuidInput"}},
     {"@id": "onem2m:cmdPump_1On",
         "@type": ["owl:NamedIndividual", "onem2m:Command"],
         "onem2m:hasInput": {"@id": "base:uuidInput"}},

     {"@id": "onem2m:exposesCommand",
         "@type": "owl:ObjectProperty"},
     {"@id": "onem2m:exposesFunctionality",
         "@type": "owl:AnnotationProperty"},

     {"@id": "onem2m:funcDescHumidity",
         "@type": ["owl:NamedIndividual", "base:YangDescription"],
```

```
        "onem2m:hasValue": "Get humidity from sensor"},
    {"@id": "onem2m:funcDescMoisture",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "Get moisture from sensor"},
    {"@id": "onem2m:funcDescPressure",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "Get pressure from sensor"},
    {"@id": "onem2m:funcDescPump_1Off",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "turn pump 1 off"},
    {"@id": "onem2m:funcDescPump_1On",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "turn pump 1 on"},
    {"@id": "onem2m:funcDescRainDetect",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "Get rain detection signal from sensor
            "},
    {"@id": "onem2m:funcDescTemperature",
        "@type": ["owl:NamedIndividual", "base:YangDescription"],
        "onem2m:hasValue": "Get Temperature from sensor"},

    {"@id": "onem2m:funcGetHumidity",
        "@type": ["owl:NamedIndividual",
                "onem2m:MeasuringFunctionality"],
        "onem2m:hasThingProperty":
                {"@id": "onem2m:funcDescHumidity"} },
    {"@id": "onem2m:funcGetMoisture",
             "@type": ["owl:NamedIndividual",
                "onem2m:MeasuringFunctionality"],
        "onem2m:hasThingProperty":
            {"@id": "onem2m:funcDescMoisture"} },
    {"@id": "onem2m:funcGetPressure",
        "@type": ["owl:NamedIndividual",
            "onem2m:MeasuringFunctionality"],
        "onem2m:hasThingProperty":
            {"@id": "onem2m:funcDescPressure" } },
    {"@id": "onem2m:funcGetRainDetect",
        "@type": ["owl:NamedIndividual",
            "onem2m:MeasuringFunctionality"],
        "onem2m:hasThingProperty":
            {"@id": "onem2m:funcDescRainDetect"} },
    {"@id": "onem2m:funcGetTemperature",
        "@type": ["owl:NamedIndividual",
            "onem2m:MeasuringFunctionality"],
        "onem2m:hasThingProperty":
            {"@id": "onem2m:funcDescTemperature"}},
    {"@id": "onem2m:funcPump_1Off",
        "@type": ["owl:NamedIndividual",
            "onem2m:ControllingFunctionality" ],
        "onem2m:hasCommand": {"@id": "onem2m:cmdPump_1Off"},
        "onem2m:hasThingProperty":
            {"@id": "onem2m:funcDescPump_1Off"} },
    {"@id": "onem2m:funcPump_1On",
        "@type": ["owl:NamedIndividual",
            "onem2m:ControllingFunctionality" ],
        "onem2m:hasCommand": { "@id": "onem2m:cmdPump_1On"},
```

```
95          "onem2m:hasThingProperty": {"@id": "onem2m:
                funcDescPump_1On"} },
96
97      {"@id": "onem2m:hasCommand", "@type": "owl:ObjectProperty" },
98      {"@id": "onem2m:hasDataRestriction_maxInclusive",
99          "@type": "owl:AnnotationProperty"},
100     {"@id": "onem2m:hasDataRestriction_minInclusive",
101         "@type": "owl:AnnotationProperty"},
102     {"@id": "onem2m:hasDataRestriction_pattern",
103         "@type": "owl:DatatypeProperty"},
104     {"@id": "onem2m:hasDataType",
105         "@type": "owl:AnnotationProperty"},
106     {"@id": "onem2m:hasFunctionality",
107         "@type": "owl:AnnotationProperty"},
108     {"@id": "onem2m:hasInput", "@type": "owl:ObjectProperty",
109         "rdfs:range": {"@id": "onem2m:ThingProperty"} },
110     {"@id": "onem2m:hasOperation",
111         "@type": "owl:ObjectProperty"},
112     {"@id": "onem2m:hasOperationState",
113         "@type": "owl:ObjectProperty"},
114     {"@id": "onem2m:hasOutput", "@type": "owl:ObjectProperty"},
115     {"@id": "onem2m:hasOutputDataPoint",
116         "@type": "owl:ObjectProperty"},
117     {"@id": "onem2m:hasService", "@type": "owl:ObjectProperty"},
118     {"@id": "onem2m:hasSubService",
119         "@type": "owl:ObjectProperty"},
120     {"@id": "onem2m:hasSubStructure",
121         "@type": "owl:AnnotationProperty"},
122     {"@id": "onem2m:hasThingProperty",
123         "@type": "owl:ObjectProperty"},
124     {"@id": "onem2m:hasValue","@type": "owl:DatatypeProperty"},
125
126     {"@id": "onem2m:opMqttPump_1Off",
127         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
128         "onem2m:exposesCommand": {"@id": "onem2m:cmdPump_1Off"},
129         "onem2m:hasInput": {"@id": "base:uuidInput"},
130         "onem2m:hasOperationState": {"@id": "base:opState"},
131         "base:mqttMethod": "OFF",
132         "base:mqttTopic": "actuator/pump/pump_1/%s"},
133     {"@id": "onem2m:opMqttPump_1On",
134         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
135         "onem2m:exposesCommand": {"@id": "onem2m:cmdPump_1On"},
136         "onem2m:hasInput": {"@id": "base:uuidInput"},
137         "onem2m:hasOperationState": {"@id": "base:opState"},
138         "base:mqttMethod": "ON",
139         "base:mqttTopic": "actuator/pump/pump_1/%s"},
140
141     {"@id": "onem2m:outDpHumidity",
142         "@type": ["owl:NamedIndividual",
143             "onem2m:OutputDataPoint"],
144         "om-2:hasUnit": {"@id": "om-2:percent"},
145         "base:mqttTopic": "sensor/humidity/humidity_1/%s"},
146     {"@id": "onem2m:outDpMoisture",
147         "@type": ["owl:NamedIndividual",
148             "onem2m:OutputDataPoint"],
149         "om-2:hasUnit": {"@id": "om-2:percent"},
```

```
150         "base:mqttTopic": "sensor/moisture/moisture_1/%s"},
151     {"@id": "onem2m:outDpPressure",
152         "@type": ["owl:NamedIndividual",
153             "onem2m:OutputDataPoint"],
154         "om-2:hasUnit": {"@id": "om-2:hectopascal"},
155         "base:mqttTopic": "sensor/pressure/pressure_1/%s"},
156     {"@id": "onem2m:outDpRainDetect",
157         "@type": ["owl:NamedIndividual",
158             "onem2m:OutputDataPoint"],
159         "om-2:hasUnit": {"@id": "om-2:percent"},
160         "base:mqttTopic": "sensor/rain/rain_1/%s"},
161     {"@id": "onem2m:outDpTemperature",
162         "@type": ["owl:NamedIndividual",
163             "onem2m:OutputDataPoint"],
164         "om-2:hasUnit": {"@id": "om-2:degreeCelsius"},
165         "base:mqttTopic": "sensor/temperature/temperature_1/%s"},
166
167     {"@id": "onem2m:servHumidity",
168         "@type": ["owl:NamedIndividual", "onem2m:Service"],
169         "onem2m:exposesFunctionality":
170             {"@id": "onem2m:funcGetHumidity"},
171         "onem2m:hasOutputDataPoint":
172             {"@id": "onem2m:outDpHumidity"}},
173     {"@id": "onem2m:servMoisture",
174         "@type": ["owl:NamedIndividual", "onem2m:Service"],
175         "onem2m:exposesFunctionality":
176             {"@id": "onem2m:funcGetMoisture"},
177         "onem2m:hasOutputDataPoint":
178             {"@id": "onem2m:outDpMoisture"}},
179     {"@id": "onem2m:servPressure",
180         "@type": ["owl:NamedIndividual", "onem2m:Service"],
181         "onem2m:exposesFunctionality":
182             {"@id": "onem2m:funcGetPressure"},
183         "onem2m:hasOutputDataPoint":
184             {"@id": "onem2m:outDpPressure"} },
185     {"@id": "onem2m:servRainDetect",
186         "@type": ["owl:NamedIndividual", "onem2m:Service"],
187         "onem2m:exposesFunctionality":
188             {"@id": "onem2m:funcGetRainDetect"},
189         "onem2m:hasOutputDataPoint":
190             {"@id": "onem2m:outDpRainDetect"} },
191     {"@id": "onem2m:servTemperature",
192         "@type": ["owl:NamedIndividual", "onem2m:Service"],
193         "onem2m:exposesFunctionality":
194             {"@id": "onem2m:funcGetTemperature"},
195         "onem2m:hasOutputDataPoint":
196             {"@id": "onem2m:outDpTemperature"} },
197
198     {"@id": "om-2:degreeCelsius", "om-2:symbol": "C"},
199     {"@id": "om-2:hasUnit","@type": "owl:AnnotationProperty"},
200     {"@id": "om-2:hectopascal", "om-2:symbol": "hPa"},
201     {"@id": "om-2:lux", "om-2:symbol": "lx"},
202     {"@id": "om-2:percent","om-2:symbol": "%%"},
203     {"@id": "om-2:symbol","@type": "owl:AnnotationProperty"},
204
205     {"@id": "http://yang-netconf-mqtt","@type": "owl:Ontology",
```

```
206         "owl:imports": [
207           {"@id": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"},
208           {"@id": "time:2016"},
209           {"@id": "http://www.onem2m.org/ontology/Base_Ontology/
                base_ontology-v0_9_0"},
210           {"@id": "http://www.ontology-of-units-of-measure.org/
                resource/om-2/"} ] },
211
212     {"@id": "base:AutomationFunctionality",
213         "@type": "owl:Class","rdfs:subClassOf":
214             {"@id": "onem2m:Functionality"} },
215     {"@id": "base:ConfigurationFunctionality",
216         "@type": "owl:Class",
217         "rdfs:subClassOf": {"@id": "onem2m:Functionality"} },
218     {"@id": "base:EventFunctionality",
219         "@type": "owl:Class",
220         "rdfs:subClassOf": {"@id": "onem2m:Functionality"} },
221
222     {"@id": "base:YangDescription", "@type": "owl:Class",
223         "rdfs:subClassOf": {"@id": "onem2m:ThingProperty"} },
224     {"@id": "base:autoMoistureYangDesc",
225         "@type": ["owl:NamedIndividual", "base:YangDescription"],
226         "onem2m:hasValue": "automation for moisture"},
227     {"@id": "base:cmdAutoMoisture",
228         "@type": ["owl:NamedIndividual", "onem2m:Command"],
229         "onem2m:hasInput": [{"@id": "base:inputAutoMoisture"},
230                 {"@id": "base:uuidInput"}]},
231     {"@id": "base:cmdConfMoisture",
232         "@type": ["owl:NamedIndividual", "onem2m:Command"],
233         "onem2m:hasInput": [
234           {"@id": "base:input_1_ConfEvName"},
235           {"@id": "base:input_2_ConfOperator"},
236           {"@id": "base:input_3_ConfMoisture"},
237           {"@id": "base:input_4_ConfInterval"},
238           {"@id": "base:input_5_ConfDuration"},
239           {"@id": "base:input_6_ConfCrud"},
240           {"@id": "base:uuidInput"} ] },
241     {"@id": "base:cmdConfTemperature",
242         "@type": ["owl:NamedIndividual", "onem2m:Command"],
243         "onem2m:hasInput": [
244           {"@id": "base:input_1_ConfEvName"},
245           {"@id": "base:input_2_ConfOperator"},
246           {"@id": "base:input_3_ConfTemperature"},
247           {"@id": "base:input_4_ConfInterval"},
248           {"@id": "base:input_5_ConfDuration"},
249           {"@id": "base:input_6_ConfCrud"},
250           {"@id": "base:uuidInput"} ] },
251     {"@id": "base:cmdLed_1Off",
252         "@type": ["owl:NamedIndividual", "onem2m:Command"],
253         "onem2m:hasInput": {"@id": "base:uuidInput"} },
254     {"@id": "base:cmdLed_1On",
255         "@type": ["owl:NamedIndividual", "onem2m:Command"],
256         "onem2m:hasInput": {"@id": "base:uuidInput"} },
257     {"@id": "base:cmdLed_1Rgb",
258         "@type": ["owl:NamedIndividual", "onem2m:Command"],
259         "onem2m:hasInput": [{"@id": "base:rgbinput"},
```

```
260                 {"@id": "base:uuidInput"} ] },
261
262     {"@id": "base:confOperatorYangDesc",
263         "@type": ["owl:NamedIndividual", "base:YangDescription"],
264         "onem2m:hasValue": "describes configuration operators"},
265     {"@id": "base:crudYangDesc",
266         "@type": ["owl:NamedIndividual", "base:YangDescription"],
267         "onem2m:hasValue": "select CRUD operations for events"},
268     {"@id": "base:deviceCategory",
269         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
270         "onem2m:hasValue": "%sBoardName"},
271     {"@id": "base:deviceDesc",
272         "@type": ["owl:NamedIndividual", "base:YangDescription"],
273         "onem2m:hasValue": "MQTT-Device identified by UUID"},
274     {"@id": "base:deviceUuid",
275         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
276         "onem2m:hasDataType": {"@id": "xsd:string"},
277         "onem2m:hasValue": "%s"},
278     {"@id": "base:durYangDesc",
279         "@type": ["owl:NamedIndividual", "base:YangDescription"],
280         "onem2m:hasValue": "duration for event configuration"},
281
282     {"@id": "base:funcAutoMoisture",
283         "@type": ["owl:NamedIndividual",
284             "base:AutomationFunctionality"],
285         "onem2m:hasCommand": {"@id": "base:cmdAutoMoisture"},
286         "onem2m:hasThingProperty":
287             {"@id": "base:funcDescAutoMoisture"} },
288     {"@id": "base:funcConfMoisture",
289         "@type": ["owl:NamedIndividual",
290             "base:ConfigurationFunctionality"],
291         "onem2m:hasCommand": {"@id": "base:cmdConfMoisture"},
292         "onem2m:hasThingProperty":
293             {"@id": "base:funcDescConfMoisture"} },
294     {"@id": "base:funcConfTemperature",
295         "@type": ["owl:NamedIndividual",
296             "base:ConfigurationFunctionality" ],
297         "onem2m:hasCommand": {"@id": "base:cmdConfTemperature"},
298         "onem2m:hasThingProperty":
299             {"@id": "base:funcDescConfTemperature"} },
300
301     {"@id": "base:funcDescAutoMoisture",
302         "@type": ["owl:NamedIndividual", "base:YangDescription"],
303         "onem2m:hasValue": "automation rule for moisture sensor
            "},
304     {"@id": "base:funcDescBrightness",
305         "@type": ["owl:NamedIndividual","base:YangDescription"],
306         "onem2m:hasValue": "Get brightness from sensor"},
307     {"@id": "base:funcDescConfMoisture",
308         "@type": ["owl:NamedIndividual", "base:YangDescription"],
309         "onem2m:hasValue": "configure moisture sensor for events
            "},
310     {"@id": "base:funcDescConfTemperature",
311         "@type": ["owl:NamedIndividual", "base:YangDescription"],
312         "onem2m:hasValue": "configure event for temperature
            sensor"},
```

```
313     {"@id": "base:funcDescEvMoisture",
314         "@type": ["owl:NamedIndividual", "base:YangDescription"],
315         "onem2m:hasValue": "event function for moisture sensor"},
316     {"@id": "base:funcDescEvTemperature",
317         "@type": ["owl:NamedIndividual", "base:YangDescription"],
318         "onem2m:hasValue": "event function for temperature sensor
                "},
319     {"@id": "base:funcDescLed_1Off",
320         "@type": ["owl:NamedIndividual", "base:YangDescription"],
321         "onem2m:hasValue": "turn led 1 off"},
322     {"@id": "base:funcDescLed_1On",
323         "@type": ["owl:NamedIndividual", "base:YangDescription"],
324         "onem2m:hasValue": "turn led 1 on"},
325     {"@id": "base:funcDescLed_1Rgb",
326         "@type": ["owl:NamedIndividual", "base:YangDescription"],
327         "onem2m:hasValue": "set RGB values for led 1"},
328
329     {"@id": "base:funcEvMoisture",
330         "@type": ["owl:NamedIndividual",
331             "base:EventFunctionality"],
332         "onem2m:hasThingProperty":
333             {"@id": "base:funcDescEvMoisture"} },
334     {"@id": "base:funcEvTemperature",
335         "@type": ["owl:NamedIndividual",
336             "base:EventFunctionality"],
337         "onem2m:hasThingProperty":
338             {"@id": "base:funcDescEvTemperature"} },
339     {"@id": "base:funcGetBrightness",
340         "@type": ["owl:NamedIndividual",
341             "onem2m:MeasuringFunctionality"],
342         "onem2m:hasThingProperty":
343             {"@id": "base:funcDescBrightness"} },
344     {"@id": "base:funcLed_1Off",
345         "@type": ["owl:NamedIndividual",
346             "onem2m:ControllingFunctionality"],
347         "onem2m:hasCommand": {"@id": "base:cmdLed_1Off"},
348         "onem2m:hasThingProperty":
349             {"@id": "base:funcDescLed_1Off"} },
350     {"@id": "base:funcLed_1On",
351         "@type": ["owl:NamedIndividual",
352             "onem2m:ControllingFunctionality"],
353         "onem2m:hasCommand": {"@id": "base:cmdLed_1On"},
354         "onem2m:hasThingProperty":
355             {"@id": "base:funcDescLed_1On"} },
356     {"@id": "base:funcLed_1Rgb",
357         "@type": ["owl:NamedIndividual",
358             "onem2m:ControllingFunctionality"],
359         "onem2m:hasCommand": {"@id": "base:cmdLed_1Rgb" },
360         "onem2m:hasThingProperty": {"@id": "base:funcDescLed_1Rgb
                "} },
361
362     {"@id": "base:inputAutoMoisture",
363         "@type": ["owl:NamedIndividual",
364             "onem2m:OperationInput"],
365         "onem2m:hasThingProperty":
366             {"@id": "base:autoMoistureYangDesc"} },
```

```
{"@id": "base:inputBlue",
    "@type": ["owl:NamedIndividual", "onem2m:Variable"],
    "onem2m:hasDataRestriction_maxInclusive":
        {"@type": "xsd:int", "@value": "255"},
    "onem2m:hasDataRestriction_minInclusive":
        {"@type": "xsd:int", "@value": "0"} },
{"@id": "base:inputGreen",
    "@type": ["owl:NamedIndividual", "onem2m:Variable"],
    "onem2m:hasDataRestriction_maxInclusive":
        {"@type": "xsd:int", "@value": "255"},
    "onem2m:hasDataRestriction_minInclusive":
        {"@type": "xsd:int", "@value": "0"} },
{"@id": "base:inputRed",
    "@type": ["owl:NamedIndividual", "onem2m:Variable"],
    "onem2m:hasDataRestriction_maxInclusive":
        {"@type": "xsd:int", "@value": "255"},
    "onem2m:hasDataRestriction_minInclusive":
        {"@type": "xsd:int", "@value": "0"} },

{"@id": "base:input_1_ConfEvName",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput"],
    "onem2m:hasInput": {"@id": "base:propConfEvName"},
    "onem2m:hasThingProperty":
        {"@id": "base:temperatureEvYangDesc"}},
{"@id": "base:input_2_ConfOperator",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput"],
    "onem2m:hasDataRestriction_pattern":
        ["<", "<=", "==", ">", ">="],
    "onem2m:hasThingProperty":
        {"@id": "base:confOperatorYangDesc"}},
{"@id": "base:input_3_ConfMoisture",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput"],
    "onem2m:hasInput": {"@id": "base:propConfMoisture"},
    "onem2m:hasThingProperty":
        {"@id": "base:moistureYangDesc"} },
{"@id": "base:input_3_ConfTemperature",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput"],
    "onem2m:hasInput": {"@id": "base:propConfTemperature"},
    "onem2m:hasThingProperty":
        {"@id": "base:temperatureYangDesc"} },
{"@id": "base:input_4_ConfInterval",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput", "time:Interval"],
    "onem2m:hasInput": {"@id": "base:propConfInterval"},
    "onem2m:hasThingProperty":
        {"@id": "base:intervallYangDesc"}},
{"@id": "base:input_5_ConfDuration",
    "@type": ["owl:NamedIndividual",
        "onem2m:OperationInput", "time:Duration"],
    "onem2m:hasInput": {"@id": "base:propConfDuration"},
    "onem2m:hasThingProperty": {"@id": "base:durYangDesc"} },
{"@id": "base:input_6_ConfCrud",
```

```
423         "@type": ["owl:NamedIndividual",
424             "onem2m:OperationInput"],
425         "onem2m:hasDataRestriction_pattern":
426             ["CREATE", "DELETE", "READ", "UPDATE"],
427         "onem2m:hasThingProperty": {"@id": "base:crudYangDesc"}},
428     {"@id": "base:intervallYangDesc",
429         "@type": ["owl:NamedIndividual", "base:YangDescription"],
430         "onem2m:hasValue": "interval for event configuration"},
431     {"@id": "base:moistureEvYangDesc",
432         "@type": ["owl:NamedIndividual", "base:YangDescription"],
433         "onem2m:hasValue": "event name caused by configuration"},
434     {"@id": "base:moistureYangDesc",
435         "@type": ["owl:NamedIndividual", "base:YangDescription"],
436         "onem2m:hasValue": "threshold values for moisture sensor
              "},
437
438     {"@id": "base:mqttMethod", "@type": "owl:DatatypeProperty"},
439     {"@id": "base:mqttTopic", "@type": "owl:DatatypeProperty"},
440
441     {"@id": "base:myDevice", "@type": ["owl:NamedIndividual", "
          onem2m:Device"],
442         "onem2m:hasFunctionality": [
443           {"@id": "onem2m:funcGetHumidity"},
444           {"@id": "onem2m:funcGetMoisture"},
445           {"@id": "onem2m:funcGetPressure"},
446           {"@id": "onem2m:funcGetRainDetect"},
447           {"@id": "onem2m:funcGetTemperature"},
448           {"@id": "onem2m:funcPump_1Off"},
449           {"@id": "onem2m:funcPump_1On"},
450           {"@id": "base:funcAutoMoisture"},
451           {"@id": "base:funcConfMoisture"},
452           {"@id": "base:funcConfTemperature"},
453           {"@id": "base:funcEvMoisture"},
454           {"@id": "base:funcEvTemperature"},
455           {"@id": "base:funcGetBrightness"},
456           {"@id": "base:funcLed_1Off"},
457           {"@id": "base:funcLed_1On"},
458           {"@id": "base:funcLed_1Rgb"} ],
459         "onem2m:hasService": {"@id": "base:servNetconf"},
460         "onem2m:hasThingProperty":
461             [{"@id": "base:deviceCategory"},
462             {"@id": "base:deviceDesc"},
463             {"@id": "base:deviceUuid"} ] },
464
465     {"@id": "base:opDescState",
466         "@type": ["owl:NamedIndividual", "base:YangDescription"],
467         "onem2m:hasDataRestriction_pattern":
468             ["error", "nothing to do", "successful"] },
469     {"@id": "base:opMqttAutoMoisture",
470         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
471         "onem2m:exposesCommand": {"@id": "base:cmdAutoMoisture"},
472         "onem2m:hasInput":
473             [{"@id": "base:inputAutoMoisture"},
474             {"@id": "base:uuidInput"} ],
475         "onem2m:hasOperationState": {"@id": "base:opState"},
476         "base:mqttMethod": "AUTOMATION",
```

```
477         "base:mqttTopic": "automation/sensor/moisture/moisture_1
                /%s"},
478     {"@id": "base:opMqttConfMoisture",
479         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
480         "onem2m:exposesCommand": {"@id": "base:cmdConfMoisture"},
481         "onem2m:hasInput": [{"@id": "base:input_1_ConfEvName"},
482           {"@id": "base:input_2_ConfOperator"},
483           {"@id": "base:input_3_ConfMoisture"},
484           {"@id": "base:input_4_ConfInterval"},
485           {"@id": "base:input_5_ConfDuration"},
486           {"@id": "base:input_6_ConfCrud"},
487           {"@id": "base:uuidInput"} ],
488         "onem2m:hasOperationState": {"@id": "base:opState"},
489         "base:mqttMethod": "CONFIGEVENT",
490         "base:mqttTopic": "config/sensor/moisture/moisture_1/%s
                "},
491     {"@id": "base:opMqttConfTemperature",
492         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
493         "onem2m:exposesCommand":
494             {"@id": "base:cmdConfTemperature"},
495         "onem2m:hasInput": [
496           {"@id": "base:input_1_ConfEvName"},
497           {"@id": "base:input_2_ConfOperator"},
498           {"@id": "base:input_3_ConfTemperature"},
499           {"@id": "base:input_4_ConfInterval"},
500           {"@id": "base:input_5_ConfDuration"},
501           {"@id": "base:input_6_ConfCrud"},
502           {"@id": "base:uuidInput"} ],
503         "onem2m:hasOperationState": {"@id": "base:opState"},
504         "base:mqttMethod": "CONFIGEVENT",
505         "base:mqttTopic": "config/sensor/temperature/
              temperature_1/%s"},
506     {"@id": "base:opMqttLed_1Off",
507         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
508         "onem2m:exposesCommand": {"@id": "base:cmdLed_1Off"},
509         "onem2m:hasInput": {"@id": "base:uuidInput"},
510         "onem2m:hasOperationState": {"@id": "base:opState"},
511         "base:mqttMethod": "OFF",
512         "base:mqttTopic": "actuator/led/led_1/%s"},
513     {"@id": "base:opMqttLed_1On",
514         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
515         "onem2m:exposesCommand": {"@id": "base:cmdLed_1On"},
516         "onem2m:hasInput": {"@id": "base:uuidInput"},
517         "onem2m:hasOperationState": {"@id": "base:opState"},
518         "base:mqttMethod": "ON",
519         "base:mqttTopic": "actuator/led/led_1/%s"},
520     {"@id": "base:opMqttLed_1Rgb",
521         "@type": ["owl:NamedIndividual", "onem2m:Operation"],
522         "onem2m:exposesCommand": {"@id": "base:cmdLed_1Rgb"},
523         "onem2m:hasInput":
524             [{"@id": "base:rgbinput"},
525             {"@id": "base:uuidInput"} ],
526         "onem2m:hasOperationState": {"@id": "base:opState"},
527         "base:mqttMethod": "RGB",
528         "base:mqttTopic": "actuator/led/led_1/%s"},
529     {"@id": "base:opState",
```

```
530         "@type": ["owl:NamedIndividual",
531             "onem2m:OperationState"],
532         "onem2m:hasDataRestriction_pattern":
533             ["ERROR", "NOOP", "OK"],
534         "onem2m:hasThingProperty": {"@id": "base:opDescState"}},
535
536     {"@id": "base:outDpBrightness",
537         "@type": ["owl:NamedIndividual",
538             "onem2m:OutputDataPoint"],
539         "om-2:hasUnit": {"@id": "om-2:lux"},
540         "base:mqttTopic": "sensor/brightness/brightness_1/%s"},
541     {"@id": "base:outDpEvMoisture",
542         "@type": ["owl:NamedIndividual",
543             "onem2m:OutputDataPoint"],
544         "base:mqttTopic": "event/sensor/moisture/moisture_1/%s"},
545     {"@id": "base:outDpEvTemperature",
546         "@type": ["owl:NamedIndividual",
547             "onem2m:OutputDataPoint"],
548         "base:mqttTopic": "event/sensor/temperature/temperature_1
              /%s"},
549
550     {"@id": "base:propConfDuration",
551         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
552         "onem2m:hasDataType": {"@id": "xsd:int"} },
553     {"@id": "base:propConfEvName",
554         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
555         "onem2m:hasDataType": {"@id": "xsd:string"}},
556     {"@id": "base:propConfInterval",
557         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
558         "onem2m:hasDataType": {"@id": "xsd:int"} },
559     {"@id": "base:propConfMoisture",
560         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
561         "onem2m:hasDataRestriction_maxInclusive": "100",
562         "onem2m:hasDataRestriction_minInclusive": "0",
563         "onem2m:hasDataType": {"@id": "xsd:int"} },
564     {"@id": "base:propConfTemperature",
565         "@type": ["owl:NamedIndividual", "onem2m:ThingProperty"],
566         "onem2m:hasDataRestriction_maxInclusive":
567             {"@type": "xsd:integer", "@value": "100"},
568         "onem2m:hasDataRestriction_minInclusive":
569             {"@type": "xsd:integer", "@value": "-20"},
570         "onem2m:hasDataType": {"@id": "xsd:int"} },
571
572     {"@id": "base:rgbYangDesc",
573         "@type": ["owl:NamedIndividual", "base:YangDescription"],
574         "onem2m:hasValue": "RGB parameter for led_1"},
575     {"@id": "base:rgbinput",
576         "@type": ["owl:NamedIndividual",
577             "onem2m:OperationInput"],
578         "onem2m:hasSubStructure":
579             [{"@id": "base:inputBlue"},
580             {"@id": "base:inputGreen"},
581             {"@id": "base:inputRed"}],
582         "onem2m:hasThingProperty": {"@id": "base:rgbYangDesc"} },
583
584     {"@id": "base:servBrightness",
```

```
585         "@type": ["owl:NamedIndividual", "onem2m:Service"],
586         "onem2m:exposesFunctionality":
587             {"@id": "base:funcGetBrightness"},
588         "onem2m:hasOutputDataPoint":
589             {"@id": "base:outDpBrightness"}},
590     {"@id": "base:servEvMoisture",
591         "@type": ["owl:NamedIndividual", "onem2m:Service"],
592         "onem2m:exposesFunctionality":
593             {"@id": "base:funcEvMoisture"},
594         "onem2m:hasOutputDataPoint":
595             {"@id": "base:outDpEvMoisture"} },
596     {"@id": "base:servEvTemperature",
597         "@type": ["owl:NamedIndividual", "onem2m:Service"],
598         "onem2m:exposesFunctionality":
599             {"@id": "base:funcEvTemperature"},
600         "onem2m:hasOutputDataPoint":
601             {"@id": "base:outDpEvTemperature"} },
602     {"@id": "base:servNetconf",
603         "@type": ["owl:NamedIndividual", "onem2m:Service"],
604         "onem2m:exposesFunctionality": [
605           {"@id": "onem2m:funcGetHumidity"},
606           {"@id": "onem2m:funcGetMoisture"},
607           {"@id": "onem2m:funcGetPressure"},
608           {"@id": "onem2m:funcGetRainDetect"},
609           {"@id": "onem2m:funcGetTemperature"},
610           {"@id": "onem2m:funcPump_1Off"},
611           {"@id": "onem2m:funcPump_1On"},
612           {"@id": "base:funcAutoMoisture"},
613           {"@id": "base:funcConfMoisture"},
614           {"@id": "base:funcConfTemperature"},
615           {"@id": "base:funcEvMoisture"},
616           {"@id": "base:funcEvTemperature"},
617           {"@id": "base:funcGetBrightness"},
618           {"@id": "base:funcLed_1Off"},
619           {"@id": "base:funcLed_1On"},
620           {"@id": "base:funcLed_1Rgb"} ],
621         "onem2m:hasOperation": [
622           {"@id": "onem2m:opMqttPump_1Off"},
623           {"@id": "onem2m:opMqttPump_1On"},
624           {"@id": "base:opMqttAutoMoisture"},
625           {"@id": "base:opMqttConfMoisture"},
626           {"@id": "base:opMqttConfTemperature"},
627           {"@id": "base:opMqttLed_1Off"},
628           {"@id": "base:opMqttLed_1On"},
629           {"@id": "base:opMqttLed_1Rgb"} ],
630         "onem2m:hasSubService": [
631           {"@id": "onem2m:servHumidity"},
632           {"@id": "onem2m:servMoisture"},
633           {"@id": "onem2m:servPressure"},
634           {"@id": "onem2m:servRainDetect"},
635           {"@id": "onem2m:servTemperature"},
636           {"@id": "base:servBrightness"},
637           {"@id": "base:servEvMoisture"},
638           {"@id": "base:servEvTemperature"}] },
639
640     {"@id": "base:temperatureEvYangDesc",
```

```
641             "@type": ["owl:NamedIndividual", "base:YangDescription"],
642             "onem2m:hasValue": "event name caused by configuration"},
643         {"@id": "base:temperatureYangDesc",
644             "@type": ["owl:NamedIndividual", "base:YangDescription"],
645             "onem2m:hasValue": "threshold values for temperature
                    sensor"},
646
647         {"@id": "base:uuidInput",
648             "@type": ["owl:NamedIndividual",
649                 "onem2m:OperationInput"],
650             "onem2m:hasInput": {"@id": "base:deviceUuid"},
651             "onem2m:hasThingProperty": {"@id": "base:uuidYangDesc"}
                    },
652         {"@id": "base:uuidYangDesc",
653             "@type": ["owl:NamedIndividual",
654                 "base:YangDescription"],
655             "onem2m:hasValue": "Target UUID for request"}  ]
656 }
```

List of Figures

List of Tables

Listings

Abbreviations

6LoWPAN IPv6 over Low-Power Wireless Personal Area Network.

AIOTI Alliance for Internet of Things Innovation.

BLE Bluetooth Low-Energy.

CAP Contention Access Period.

CBOR Concise Binary Object Representation.

CFP Contention-Free Period.

CLI Commando Line Interface.

CoAP Constrained Application Protocol.

COSE CBOR Object Signing and Encryption.

CPS Cyber-Physical Systems.

CSMA-CA Carrier Sense Multiple Access with Collision Avoidance.

D2D Device-to-Device.

D2S Device-to-Service.

DAG Directed Acyclic Graph.

DIO DODAG Information Object.

DL Descriptive Logic.

DODAG Destination-Oriented DAG.

DSL Domain Specific Language.

DTLS Datagram Transport Layer Security.

ETSI European Telecommunications Standards Institute.

FCAPS Fault, Configuration, Accounting, Performance, Security.

FFD Full Function Device.

FOL First-Order Logic.

GTS Guaranteed Time Slots.

HDT Header-Dictionary-Triples.

HVAC Heating, Ventilation, Air Conditioning.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IRI Internationalized Resource Identifier.

JSON JavaScript Object Notation.

LCIM Levels of Conceptual Interoperability Model.

LoWPAN Low-Power Wireless Personal Area Network.

LPWAN Low-power Wide Area Networks.

LwM2M Lightweight Machine to Machine.

M2M Machine-to-Machine.

MAC Media Access Control.

MANET Mobile Ad Hoc Network.

MEMS micro-electro-mechanical systems.

MIB Management Information Base.

MQTT Message Queuing Telemetry Transport.

MTU Maximum Transmission Unit.

MUD Manufacturer Usage Descriptions.

MUP MYNO Update Protocol.

MYNO MQTT-YANG-NETCONF-Ontology.

NETCONF Network Configuration Protocol.

NFC Near Field Communication.

OAM Operations, Administration, and Maintenance.

OCF Open Connectivity Foundation.

OCP Objective Code Point.

OF Objective Function.

OGC Open Geospatial Consortium.

OIC Open Internet Consortium.

OMA Open Mobile Alliance.

OPC Open Platform Communications.

OSCORE Object Security for Constrained RESTful Environments.

OSI Open Systems Interconnection.

OSS Open Source Solutions.

OWL Web Ontology Language.

PAN Personal Area Network.

PDA Personal Digital Assistant.

PHY Physical layer.

QoS Quality of Service.

RATS Remote ATtestation ProcedureS.

RDF Resource Description Framework.

RDFS RDF Schema.

REST Representational State Transfer.

RFC Request for Comments.

RFD Reduced Function Device.

RFID Radio Frequency Identifier.

RPC Remote Procedure Call.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

RTS/CTS Request To Send / Clear To Send handshake.

S2S Service-to-Service.

SAREF Smart Applications REFerence.

SDO Standards Development Organization.

SenML Sensor Measurement Lists.

SensorML Sensor Model Language.

SMAE System Management Application Entity.

SMI Structure of Management Information.

SNMP Simple Networking Management Protocol.

SOSA Sensor, Observation, Sample, and Actuator.

SPARQL SPARQL Protocol And RDF Query Language.

SSID Service Set Identifier.

SSN Semantic Sensor Network.

SUIT Software Updates for Internet of Things.

SWE Sensor Web Enablement.

SWoT Semantic Web of Things.

TLS Transport Layer Security.

TSCH Time-Slotted Channel Hopping.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

URN Uniform Resource Name.

UUID Universally Unique Identifier.

W3C World Wide Web Consortium.

WLAN Wireless Local Area Network.

WoT Web of Things.

WPAN Wireless Personal Area Network.

WSN Wireless Sensor Network.

WWW World Wide Web.

YANG Yet Another Next Generation.

Bibliography

[1] Sameh Abdel-Naby, Conor Muldoon, Olga Zlydareva, and Gregory O'Hare. „Agent-Driven Wireless Sensors Cooperation for Limited Resources Allocation". In: *Intelligent sensor networks*. Ed. by Fei Hu and Qi Hao. Boca Raton, FL: CRC Press and Taylor & Francis, 2013, pp. 427–439.

[2] Rachit Agarwal, David Gomez Fernandez, Tarek Elsaleh, et al. „Unified IoT ontology to enable interoperability and federation of testbeds". In: *3rd IEEE World Forum on Internet of Things*. IEEE, 2016, pp. 70–75.

[3] Charu C. Aggarwal, Naveen Ashish, and Amit Sheth. „The Internet of Things: A Survey from the Data-Centric Perspective". In: *Managing and mining sensor data*. Ed. by Charu C. Aggarwal. New York: Springer, 2013, pp. 383–428.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. „Wireless sensor networks: A survey". In: *Computer Networks* 38.4 (2002), pp. 393–422.

[5] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. „Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376.

[6] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. „OM2M: Extensible ETSI-compliant M2M Service Platform with Self-configuration Capability". In: *Procedia Computer Science* 32 (2014), pp. 1079–1086.

[7] Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. „Diversified Stress Testing of RDF Data Management Systems". In: *LNCS 8796 - The Semantic Web – ISWC 2014*. Springer International Publishing, 2014, pp. 197–212.

[8] Amjad F. Alharbi, Bashayer A. Alotaibi, and Fahd S. Alotaibi. „Security of Internet of Things: Challenges, Requirements and Future Directions". In: *International Journal of Information, Control and Computer Sciences: 11.0.* 10.2 (2018).

[9] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. „Internet of Things: A survey on the security of IoT frameworks". In: *Journal of Information Security and Applications* 38 (2018), pp. 8–27.

[10] L. Andersson, H. van Helvoort, R. Bonica, D. Romascanu, and S. Mansfield. *Guidelines for the Use of the "OAM" Acronym in the IETF*. RFC 6291. Internet Engineering Task Force, June 2011.

[11] Anh Le Tuan, Conor Hayes, Marcin Wylot, and Danh Le-Phuoc. „RDF4Led: An RDF engine for Lightweight Edge Devices". In: *Proceedings of the 8th International Conference on the Internet of Things (IoT 2018)*. ACM, 2018.

[16]B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. „What Are Ontologies, and Why Do We Need Them?“ In: *IEEE Intelligent Systems* 1 (1999), pp. 20–26.

[17]E. Baccelli, C. Gündoğan, O. Hahm, et al. „RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT“. In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 4428–4440.

[18]E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt. „RIOT OS: Towards an OS for the Internet of Things“. In: *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 79–80.

[19]Sebastian R. Bader and Maria Maleshkova. „Virtual Representations for an Iterative IoT Deployment“. In: *Companion Proceedings of the The Web Conference*. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, 1887–1892.

[20]M. Badra. *NETCONF over Transport Layer Security (TLS)*. RFC 5539. Internet Engineering Task Force, May 2009.

[21]M. Badra, A. Luchuk, and J. Schoenwaelder. *Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication*. RFC 7589. Internet Engineering Task Force, June 2015.

[22]P. Barnaghi, S. Meissner, M. Presser, and K. Moessner. „Sense and Sens'ability: Semantic Data Modelling for Sensor Networks“. In: *Conference Proceedings of ICT Mobile Summit*. 2009.

[23]Payam Barnaghi, Wei Wang, Cory Henson, and Kerry Taylor. „Semantics for the Internet of Things: early progress and back to the future“. In: *International Journal on Semantic Web and Information Systems* 8.1 (2012), pp. 1–21.

[24]*Base Ontology: oneM2M Technical Specification*. Tech. rep. TS-0012-V3.7.1. Mar. 2018.

[25]Alessandro Bassi, Martin Bauer, Martin Fiedler, et al., eds. *Enabling Things to Talk: Designing IoT solutions with the IoT architectural reference model*. Springer Berlin Heidelberg, 2013.

[26]Maria Bermudez-Edo, Tarek Elsaleh, Payam Barnaghi, and Kerry Taylor. „IoT-Lite: A Lightweight Semantic Model for the Internet of Things and its Use with Dynamic Semantics“. In: *Personal and Ubiquitous Computing* 21.3 (2017), pp. 475–487.

[27]Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Internet Engineering Task Force, Jan. 2005.

[28]Lars Bernhard. *Geodateninfrastruktur: Grundlagen und Anwendungen*. Heidelberg: Wichmann, 2005.

[29]A. Bierman. *Guidelines for Authors and Reviewers of YANG Data Model Documents*. RFC 6087. Internet Engineering Task Force, Jan. 2011.

[30]A Bierman. *Network configuration protocol (NETCONF) base notifications*. RFC 6470. Internet Engineering Task Force, Feb. 2012.

[31]A. Bierman and M. Bjorklund. *Network Configuration Protocol (NETCONF) Access Control Model*. RFC 6536. Internet Engineering Task Force, Mar. 2012.

[32]A. Bierman, M. Bjorklund, and K. Watsen. *RESTCONF Protocol*. RFC 8040. Internet Engineering Task Force, Jan. 2017.

[33]Andy Bierman. *A Guide to NETCONF for SNMP Developers*. IEEE 802 Plenary, San Diego, CA US, 2014.

[34]Ralf Bill. *Grundlagen der Geoinformationssysteme*. Heidelberg: Wichmann, 2010.

[36]H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. *Remote Attestation Procedures Architecture*. Internet-Draft. Internet Engineering Task Force, Feb. 2021.

[37]M. Bjorklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. Internet Engineering Task Force, Aug. 2016.

[38]M. Bjorklund. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. RFC 6020. Internet Engineering Task Force, Oct. 2010.

[39]D. Bogdanovic, B. Claise, and C. Moberg. *YANG Module Classification*. RFC 8199. Internet Engineering Task Force, July 2017.

[40]C. Bormann, M. Ersue, and A. Keranen. *Terminology for Constrained-Node Networks*. RFC 7228. Internet Engineering Task Force, May 2014.

[41]C. Bormann and P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 7049. Internet Engineering Task Force, Oct. 2013.

[42]C. Bormann and P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 8949. Internet Engineering Task Force, Dec. 2020.

[43]C. Bormann, S. Lemay, H. Tschofenig, et al. *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*. RFC 8323. Internet Engineering Task Force, Feb. 2018.

[44]C. Bormann and Z. Shelby. *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. RFC 7959. Internet Engineering Task Force, Aug. 2016.

[45]*SensorML: Model and XML Encoding Standard 2.0, OGC 12-000.* Tech. rep. OGC, 2014.

[46]A. Brandt and J. Buron. *Transmission of IPv6 Packets over ITU-T G.9959 Networks*. RFC 7428. Internet Engineering Task Force, Feb. 2015.

[47]Janez Brank, Marko Grobelnik, and Dunja Mladenic. „A Survey of Ontology Evaluation Techniques“. In: *Proceedings of the Conference on Data Mining and Data Warehouses*. Ljubljana, Slovenia, 2005, pp. 166–170.

[48]Broadband Forum. *TR-069: CPE WAN Management Protocol*. Tech. rep. Mar. 2018.

[49]Arne Broring, Johannes Echterhoff, Simon Jirka, et al. „New generation Sensor Web Enablement“. In: *Sensors* 11.3 (2011), pp. 2652–2699.

[50]Arne Bröring, Patrick Maúe, Krzysztof Janowicz, Daniel Nüst, and Christian Malewski. „Semantically-enabled sensor plug & play for the sensor web“. In: *Sensors* 11.8 (2011), pp. 7568–7605.

[51]Víctor Caballero, Sergi Valbuena, David Vernet, and Agustín Zaballos. „Ontology-Defined Middleware for Internet of Things Architectures“. In: *Sensors* 19.5 (2019).

[52]Christian Cabrera, Andrei Palade, and Siobhán Clarke. „An evaluation of service discovery protocols in the internet of things“. In: *Proceedings of the Symposium on Applied Computing - SAC '17*. New York, USA: ACM Press, 2017, pp. 469–476.

[53]Ross Callon. *The Twelve Networking Truths*. RFC 1925. Internet Engineering Task Force, Apr. 1996.

[54]Gavin Carothers and Eric Prud'hommeaux. *RDF 1.1 Turtle*. W3C Recommendation. W3C, Feb. 2014.

[55]J. Case, D. Harrington, R. Presuhn, and B. Wijnen. *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)*. RFC 3412. Internet Engineering Task Force, Dec. 2002.

[56]J. Case, R. Mundy, D. Partain, and B. Stewart. *Introduction and Applicability Statements for Internet-Standard Management Framework*. RFC 3410. Internet Engineering Task Force, Dec. 2002.

[57]Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall, and James R. Davin. *Simple Network Management Protocol (SNMP)*. RFC 1157. Internet Engineering Task Force, May 1990.

[58]A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. *Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)*. RFC 8075. Internet Engineering Task Force, Feb. 2017.

[60]V.G. Cerf. *IAB recommendations for the development of Internet network management standards*. RFC 1052. Internet Engineering Task Force, Apr. 1988.

[61]S. Chisholm and H. Trevino. *NETCONF Event Notifications*. RFC 5277. Internet Engineering Task Force, July 2008.

[62]Muhammad E. H. Chowdhury, Amith Khandakar, Saba Ahmed, et al. „Design, Construction and Testing of IoT Based Automated Indoor Vertical Hydroponics Farming Test-Bed in Qatar". In: *Sensors* 20.19 (2020).

[63]T. Chown, J. Arkko, A. Brandt, O. Troan, and J. Weil. *IPv6 Home Networking Architecture Principles*. RFC 7368. Internet Engineering Task Force, Oct. 2014.

[64]Vera Clemens. *Improving the MYNO Update Protocol Implementation*. Project Study. University of Potsdam, Institute of Computer Science, 2020.

[65]A. Clemm and E. Voit. *Subscription to YANG Notifications for Datastore Updates*. RFC 8641. Internet Engineering Task Force, Sept. 2019.

[66]Alexander Clemm, Eric Voit, Alberto Prieto, et al. *Subscription to YANG Datastores*. Internet-Draft. Internet Engineering Task Force, Feb. 2018.

[67]Alem Čolaković and Mesud Hadžialić. „Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues". In: *Computer Networks* 144 (2018), pp. 17–39.

[69]Michael Compton, Payam Barnaghi, Luis Bermudez, et al. „The SSN ontology of the W3C semantic sensor network incubator group". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 17 (2012), pp. 25–32.

[72]M. Cotton, B. Leiba, and T. Narten. *Guidelines for Writing an IANA Considerations Section in RFCs*. RFC 8126. Internet Engineering Task Force, June 2017.

[73]*Observations and Measurements (O&M)*. Tech. rep. OGC, 2011.

[74]Simon Cox and Chris Little. *Time Ontology in OWL*. Candidate Recommendation. W3C, Mar. 2020.

[75]D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. RFC 4627. Internet Engineering Task Force, July 2006.

[77] Dalibor Đumić, Sretenka Došlić, Marija Antić and Boško Milić. „Integration of the NETCONF Protocol in the Internet of Things by means of RESTful Web Services“. In: *Proceedings of the 6th International Conference on Electrical, Electronic and Computing Engineering, IcETRAN 2019*. Belgrade: ETRAN Society, 2019.

[79] Andreas Dengel. *Semantische Technologien*. Heidelberg: Spektrum Verlag, 2012.

[80] Denny Vrandečić. „Ontology Evaluation“. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin: Springer, 2009, pp. 293–313.

[81] Denny Vrandečić. „Ontology Evaluation“. PhD thesis. Karlsruhe: Karlsruher Institut für Technologie, 2010.

[82] Daniel Díaz-Sánchez, Andrés Marín-Lopez, Florina Almenárez Mendoza, and Patricia Arias Cabarcos. „DNS/DANE Collision-Based Distributed and Dynamic Authentication for Microservices in IoT“. In: *Sensors* 19.15 (2019).

[83] Henrik Dibowski. „Semantischer Gerätebeschreibungsansatz für einen automatisierten Entwurf von Raumautomationssystemen“. PhD thesis. TU Dresden, 2013.

[84] Henrik Dibowski and Klaus Kabitzsch. „Ontology-Based Device Descriptions and Device Repository for Building Automation Devices“. In: *EURASIP Journal on Embedded Systems* (2011), pp. 1–17.

[85] John Domingue, Dieter Fensel, and James A. Hendler, eds. *Handbook of semantic web technologies*. Springer, 2011.

[86] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Internet Engineering Task Force, Mar. 1997.

[87] Adam Dunkels. „Full TCP/IP for 8-bit architectures“. In: *Proceedings of the 1st international conference on Mobile systems, applications and services*. 2003, pp. 85–98.

[88] S. Duquennoy, A. Elsts, B. A. Nahas, and G. Oikonomo. „TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation“. In: *13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2017, pp. 11–18.

[89] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. De Gruyter Oldenbourg, 2014.

[93] R. Enns, M. Bjorklund, J. Schönwälder, and A. Bierman. *Network Configuration Protocol (NETCONF)*. RFC 6241. Internet Engineering Task Force, June 2011.

[94] M. Ersue, D. Romascanu, J. Schoenwaelder, and U. Herberg. *Management of Networks with Constrained Devices: Problem Statement and Requirements*. RFC 7547. Internet Engineering Task Force, May 2015.

[95] M. Ersue, D. Romascanu, J. Schoenwaelder, and A. Sehgal. *Management of Networks with Constrained Devices: Use Cases*. RFC 7548. Internet Engineering Task Force, May 2015.

[97] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. „The Many Faces of Publish/Subscribe“. In: *ACM Computing Surveys* 35.2 (2003), pp. 114–131.

[99] J. Euzenat and P. Shvaiko. *Ontology Matching*. Berlin: Springer, 2013.

[100] Hossam Mahmoud Ahmad Fahmy. „Protocol Stack of WSNs“. In: *Wireless sensor networks*. Singapore: Springer, 2016, pp. 55–68.

[101]Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutierrez, and Axel Polleres. *Binary RDF Representation for Publication and Exchange (HDT)*. W3C Member Submission. 2011.

[102]Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. „Binary RDF Representation for Publication and Exchange (HDT)“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (2013), 22–41.

[103]Javier D. Fernández, Miguel A. Martínez-Prieto, and Polleres, Axel and Reindorf, Julian. „HDTQ: Managing RDF Datasets in Compressed Space“. In: *15th Extended Semantic Web Conference (ESWC)*. 2018.

[104]I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455. Internet Engineering Task Force, Dec. 2011.

[105]Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Ph.D. Dissertation. Irvine: University of California, 2000.

[108]Frank Matthias Kovatsch. „Scalable Web Technology for the Internet of Things“. Dissertation. ETH Zurich, 2015.

[109]Dustin Frisch, Sven Reißmann, and Christian Pape. „An Over the Air Update Mechanism for ESP8266 Microcontrollers“. In: *Proceedings of the ICSNC, the Twelfth International Conference on Systems and Networks Communications, Athens, Greece*. 2017, pp. 8–12.

[110]*Functional Architecture*. Technical Specification TS-0001-V3.19.0. oneM2M, Dec. 2019.

[111]Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. „Modelling Ontology Evaluation and Validation“. In: *The Semantic Web: Research and Applications*. Ed. by York Sure and John Domingue. Springer Berlin, 2006, pp. 140–154.

[112]O. Garcia-Morchon, S. Kumar, and M. Sethi. *Internet of Things (IoT) Security: State of the Art and Challenges*. RFC 8576. Internet Engineering Task Force, Apr. 2019.

[113]Julien Genestoux and Aaron Parecki. *WebSub*. W3C Recommendation. W3C, Jan. 2018.

[115]Raffaele Giaffreda. „iCore: A Cognitive Management Framework for the Internet of Things“. In: *The Future Internet*. Ed. by Alex Galis. Vol. 7858. Springer, 2013, pp. 350–352.

[116]Birte Glimm and Chimezie Ogbuji. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation. W3C, Mar. 2013.

[119]C. Gomez, M. Kovatsch, H. Tian, and Z. Cao. *Energy-Efficient Features of Internet of Things Protocols*. RFC 8352. Internet Engineering Task Force, Apr. 2018.

[120]Asunción Gómez-Pérez. „Ontology Evaluation“. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Springer Berlin, 2004, pp. 251–273.

[121]Jorge Granjal, Edmundo Monteiro, and Jorge Sa Silva. „Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues“. In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1294–1312.

[122]Bert Greevenbosch, Kepeng Li, and Peter Van der Stok. *Candidate Technologies for COMAN*. Internet-Draft. Internet Engineering Task Force, July 2013.

[124] Michael Grüninger and Mark S Fox. „Methodology for the Design and Evaluation of Ontologies“. In: *IJCAI 1995*. 1995.

[125] Wided Guédria, Yannick Naudet, and David Chen. „Interoperability Maturity Models – Survey and Comparison –“. In: *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*. Ed. by Robert Meersman, Zahir Tari, and Pilar Herrero. Springer, 2008, pp. 273–282.

[126] Ramanathan Guha and Dan Brickley. *RDF Schema 1.1*. W3C Recommendation. W3C, Feb. 2014.

[127] Deepsubhra Guha Roy, Bipasha Mahato, Debashis De, and Rajkumar Buyya. „Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) — MQTT-SN protocols“. In: *Future Generation Computer Systems* 89 (2018), pp. 300–316.

[128] Dominique Guinard. „A Web of Things Application Architecture - Integrating the Real-World into the Web“. Dissertation. ETH Zurich, 2011.

[129] Dominique Guinard and Vlad Trifa. „Towards the Web of Things: Web Mashups for Embedded Devices“. In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*. Madrid, Spain, Apr. 2009.

[130] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. „From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices“. In: *Architecting the Internet of Things*. Ed. by Dieter Uckelmann, Mark Harrison, and Florian Michahelles. Springer Berlin Heidelberg, 2011, pp. 97–129.

[131] Dominique D. Guinard and Vlad M. Trifa. *Building the web of things: With examples in Node.js and Raspberry Pi*. Manning Publications, 2016.

[132] Amelie Gyrard. „Designing cross-domain semantic Web of things applications“. PhD thesis. Télécom ParisTech, 2015.

[133] Amelie Gyrard, Christian Bonnet, and Karima Boudaoud. „Enrich Machine-to-Machine Data with Semantic Web Technologies for Cross-Domain Applications“. In: *World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 559–564.

[134] Amelie Gyrard, Martin Serrano, and Ghislain A. Atemezing. „Semantic web methodologies, best practices and ontology engineering applied to Internet of Things“. In: *IEEE World Forum on Internet of Things*. Milan, Italy: IEEE, 2015, pp. 412–417.

[135] H. C. Pöhls and B. Petschkuhn. „Towards compactly encoded signed IoT messages“. In: *IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2017, pp. 1–6.

[136] H. Ilgner and S. Pienaar. „Implementing a Compact Data Format for Bluetooth and 3G Communication to Monitor Remote Pipelines“. In: *International Conference on Advances in Computing and Communication Engineering (ICACCE)*. 2016, pp. 45–50.

[137] Armin Haller, Krzysztof Janowicz, Simon J.D. Cox, et al. „The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation“. In: *Semantic Web* 4.3 (2018), pp. 1–24.

[138] Armin Haller, Kerry Taylor, Danh Le Phuoc, et al. *Semantic Sensor Network Ontology*. W3C Recommendation. W3C, Oct. 2017.

[139]Hamid Mukhtar, Kim Kang-Myo, Shafique Ahmad Chaudhry, et al. „LNMP - Management architecture for IPv6 based low-power wireless Personal Area Networks (6LoWPAN)“. In: *IEEE Network Operations and Management Symposium*. Apr. 2008, pp. 417–424.

[140]K. Hartke. *Observing Resources in the Constrained Application Protocol (CoAP)*. RFC 7641. Internet Engineering Task Force, Sept. 2015.

[141]Michael Haus, Muhammad Waqas, Aaron Yi Ding, et al. „Security and Privacy in Device-to-Device (D2D) Communication: A Review // Security and Privacy in Device-to-Device (D2D) Communication: A Review“. In: *IEEE Communications Surveys & Tutorials* (2017), p. 1.

[142]Jeff Heflin. *OWL Web Ontology Language Use Cases and Requirements*. Tech. rep.

[143]John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, et al. „Building efficient wireless sensor networks with low-level naming“. In: *Proceedings of the Symposium on Operating System Pronciples (SOSP 2001)*. ACM, Feb. 2001, p. 146.

[144]Wendi Heinzelman, Joanna Kulik, and Hari Balakrishnan. „Adaptive protocols for information dissemination in wireless sensor networks“. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. Seattle, USA, 1999, pp. 174–185.

[145]Robin Heydon. *Bluetooth low energy: The developer's handbook*. Prentice Hall, 2014.

[146]Jason Hill, Robert Szewczyk, Alec Woo, et al. „System Architecture Directions for Networked Sensors“. In: *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. Cambridge, Massachusetts, USA: ACM, 2000, pp. 93–104.

[147]Pascal Hitzler. *Semantic web: Grundlagen*. Springer Berlin, 2008.

[148]Pascal Hitzler, Sebastian Rudolph, and Markus Krötzsch. *Foundations of Semantic Web technologies*. CRC Press, 2010.

[150]Hlomani Hlomani and Deborah Stacey. „Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey“. In: *Semantic Web Journal* 1.5 (2014), pp. 1–11.

[151]P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. Internet Engineering Task Force, Aug. 2012.

[152]Christian Hoffmann. „Effiziente Datendarstellung mittels CBOR für das Internet of Things“. Bachelor Thesis. University of Potsdam, Institute of Computer Science, May 2018.

[153]Jungha Hong, Yong-Geun Hong, Xavier de Foy, et al. *IoT Edge Challenges and Functions*. Internet-Draft. Internet Engineering Task Force, July 2020.

[154]Yong-Geun Hong, Carles Gomez, Abdur Sangi, Take Aanstoot, and Samita Chakrabarti. *IPv6 over Constrained Node Networks (6lo) Applicability & Use cases*. Internet-Draft. Internet Engineering Task Force, Sept. 2019.

[155]Ian Horrocks, Oliver Kutz, and Ulrike Sattler. „The Even More Irresistible SROIQ“. In: *Proceedings of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-06)* 6 (2006), pp. 57–67.

[156]J. Hui and P. Thubert. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282. Internet Engineering Task Force, Sept. 2011.

[158]Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. „MQTT-S — A publish/-subscribe protocol for Wireless Sensor Networks“. In: *3rd International Conference on Communication System Software and Middleware and Workshops*. Bangalore, India: IEEE, 2008, pp. 791–798.

[160]*IEEE 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks*. Tech. rep. Institute of Electrical and Electronics Engineers, 2016.

[161]*Information technology – Internet of Things Reference Architecture (IoT RA): ISO/IEC CD 30141:2016*. Tech. rep. ISO/IEC, 2016.

[162]C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. „Directed diffusion for wireless sensor networking“. In: *IEEE/ACM Transactions on Networking* 11.1 (2003), pp. 2–16.

[163]Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. „Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks“. In.

[165]*IoT LSP Standard Framework Concepts, Release 2.9*. Tech. rep. AIOTI, Oct. 2019.

[167]Isam Ishaq, David Carels, Girum Teklemariam, et al. „IETF Standardization in the Field of the Internet of Things (IoT): A Survey“. In: *Journal of Sensor and Actuator Networks* 2.2 (2013), pp. 235–287.

[168]*ISO/IEC 20922:2016(en), Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1*. Tech. rep. 20922. ISO/IEC, 2016.

[169]*ISO/IEC 7498-4 Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 4: Management framework*. Tech. rep. ISO/IEC JTC 1, 1989.

[170]*IT-Grundschutz-Kompendium, 2. Edition 2019*. Reguvis Bundesanzeiger Verlag and Bundesanzeiger Verlag, 2019.

[171]J. Ma, X. Zhou, S. Li, and Z. Li. „Connecting Agriculture to the Internet of Things through Sensor Networks“. In: *International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. 2011, pp. 184–187.

[172]Raj Jain. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons Inc, 1991.

[173]David P Janes. „Semantic Metastandards Will Unlock IoT Interoperability“. In: *International Internet of Things Summit*. Springer, 2015, pp. 396–402.

[174]Krzysztof Janowicz and Michael Compton. „The Stimulus-Sensor-Observation Ontology Design Pattern and Its Integration into the Semantic Sensor Network Ontology“. In: *Proceedings of the 3rd International Conference on Semantic Sensor Networks - Volume 668*. Shanghai, China, 2010, 64–78.

[175]Krzysztof Janowicz, Armin Haller, Simon J.D. Cox, Danh Le Phuoc, and Maxime Lefrançois. „SOSA: A lightweight ontology for sensors, observations, samples, and actuators“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2018).

[176]Antonio J. Jara, Alex C. Olivieri, Yann Bocchi, et al. „Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence“. In: *International Journal of Web and Grid Services* 10.2/3 (2014), pp. 244–260.

[177]C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann. *Sensor Measurement Lists (SenML)*. RFC 8428. Internet Engineering Task Force, Aug. 2018.

[178]J. Jimenez, H. Tschofenig, and D. Thaler. *Report from the Internet of Things (IoT) Semantic Interoperability (IOTSI) Workshop 2016*. RFC 8477. Internet Engineering Task Force, Oct. 2018.

[179]Wenquan Jin and Do Hyeun Kim. „Design and Implementation of e-Health System Based on Semantic Sensor Network Using IETF YANG“. In: *Sensors* 18.2 (2018).

[180]M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig. *CBOR Web Token (CWT)*. RFC 8392. Internet Engineering Task Force, May 2018.

[182]Tobias Käfer and Andreas Harth. „Rule-based Programming of User Agents for Linked Data“. In: *Linked Data on the Web*. 2018.

[183]Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister. „Next Century Challenges: Mobile Networking for “Smart Dust”“. In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 271–278.

[184]Takuki Kamiya and Sebastian Käbisch. *Web of Things (WoT) Thing Description*. W3C Proposed Reccommendation. W3C, Jan. 2020.

[185]Ekaterina Kapranova. *Generation of sensor data with CC2538dk*. Project Study. University of Potsdam, Institute of Computer Science, 2019.

[186]Holger Karl and Andreas Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[187]Trishank Karthik, Akan Brown, Sebastien Awwad, et al. „Uptane: Securing Software Updates for Automobiles“. In: *International Conference on Embedded Security in Car*. 2016, pp. 1–11.

[188]Christopher A. Kent and Jeffrey C. Mogul. „Fragmentation Considered Harmful“. In: *SIGCOMM Comput. Commun. Rev.* 25.1 (Jan. 1995), 75–87.

[189]Ari Keränen and Carsten Bormann. „Internet of Things: Standards and Guidance from the IETF“. In: *IETF Journal* 11.3 (2016), pp. 20–21.

[191]Kotis Konstantinos and Artem Katasonov. „An ontology for the automated deployment of applications in heterogeneous IoT environments“. In: *Semantic Web Journal (SWJ)* (2012).

[192]Ege Korkan and Michael Koster. *Web of Things (WoT) Binding Templates*. W3C Working Group Note. W3C, Jan. 2020.

[193]Michael Koster and Carsten Bormann. *Semantic Definition Format (SDF) for Data and Interactions of Things*. Internet-Draft. Internet Engineering Task Force, Feb. 2021.

[194]Michael Koster, Ari Keranen, and Jaime Jimenez. *Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)*. Internet-Draft. Internet Engineering Task Force, Sept. 2019.

[195] Kai Kreuzer. „A Pragmatic Approach to Interoperability in the Internet of Things“. In: *IoT Semantic Interoperability Workshop*. San Jose, California, US: Internet Architecture Board (IAB), 2016.

[196] Bhaskar Krishnamachari. *Networking Wireless Sensors*. Cambridge University Press, 2005.

[197] Ioannis Krontiris. „Intrusion Prevention and Detection in Wireless Sensor Networks“. Dissertation. University of Mannheim, 2008.

[198] Krzysztof Janowicz, Simon Scheider, Todd Pehle, and Glen Hart. „Geospatial semantics and linked spatiotemporal data – Past, present, and future“. In: *Semantic Web* 3 (2012), pp. 321–332.

[199] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. „Negotiation-based protocols for disseminating information in wireless sensor networks“. In: *Wireless networks* 8.2/3 (2002), pp. 169–185.

[200] Vinay Kumar and Sudarshan Tiwari. „Routing in IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN): A Survey“. In: *Journal of Computer Networks and Communications* (2012), pp. 1–10.

[201] Agus Kurniawan. *Practical Contiki-NG: Programming for wireless sensor networks*. Berkeley, CA: Apress, 2018.

[202] James F. Kurose and Keith W. Ross. *Computernetzwerke: Der Top-Down-Ansatz*. Pearson Studium, 2014.

[203] N. Kushalnagar, G. Montenegro, and C. Schumacher. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*. RFC 4919. Internet Engineering Task Force, Aug. 2007.

[204] A. Langiu, C. A. Boano, M. Schuß, and K. Römer. „UpKit: An Open-Source, Portable, and Lightweight Update Framework for Constrained IoT Devices“. In: *IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. July 2019, pp. 2101–2112.

[205] Markus Lanthaler and Christian Gütl. „Hydra: A Vocabulary for Hypermedia-Driven Web APIs“. In: *Proceedings of the 6th Workshop on Linked Data on the Web*. 2013.

[206] Markus Lanthaler, Gregg Kellogg, and Manu Sporny. *JSON-LD 1.0*. W3C Recommendation. W3C, Jan. 2014.

[207] Teemu Laukkarinen, Lasse Määttä, Jukka Suhonen, Timo D. Hämäläinen, and Marko Hännikäinen. „Design and Implementation of a Firmware Update Protocol for Resource Constrained Wireless Sensor Networks“. In: *International Journal of Embedded and Real-Time Communication Systems* 2.3 (2011), pp. 50–68.

[208] Paul J. Leach, Michael Mealling, and Rich Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. Internet Engineering Task Force, July 2005.

[209] E. Lear, R. Droms, and D. Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. Internet Engineering Task Force, Mar. 2019.

[210] Engin Leloglu. „A Review of Security Concerns in Internet of Things“. In: *Journal of Computer and Communications* 05.01 (2017), pp. 121–136.

[211]L. Lhotka. *JSON Encoding of Data Modeled with YANG*. RFC 7951. Internet Engineering Task Force, Aug. 2016.

[212]Nai-Wei Lo and Sheng-Hsiang Hsu. „A Secure IoT Firmware Update Framework Based on MQTT Protocol“. In: *International Conference on Information Systems Architecture and Technology*. Springer, 2019, pp. 187–198.

[213]Dave Longley, Pierre-Antoine Champin, and Gregg Kellogg. *JSON-LD 1.1*. W3C Recommendation. W3C, July 2020.

[214]Jorge E. López de Vergara, Antonio Guerrero, Víctor A. Villagrá, and Julio Berrocal. „Ontology-Based Network Management: Study Cases and Lessons Learned“. In: *Journal of Network and Systems Management* 17.3 (2009), pp. 234–254.

[215]Jan Lunze. *Künstliche Intelligenz für Ingenieure: Methoden zur Lösung ingenieurtechnischer Probleme mit Hilfe von Regeln, logischen Formeln und Bayesnetzen*. Oldenbourg Wissenschaftsverlag, 2016.

[216]M. Malik, M. Dutta, and J. Granjal. „A Survey of Key Bootstrapping Protocols Based on Public Key Cryptography in the Internet of Things“. In: *IEEE Access* 7 (2019), pp. 27443–27464.

[217]Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. „TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks“. In: *SIGOPS Operating Systems Review* 36.SI (Dec. 2002), pp. 131–146.

[218]Mahdi Ben Alaya, Samir Medjiah, Thierry Monteil, and Khalil Drira. „Towards Semantic Data Interoperability in oneM2M Standard“. In: *IEEE Communications Magazine* 53 (12) (2015), pp. 35–41.

[219]Ioana Marcu, Carmen Voicu, Ana Maria Claudia Dragulinescu, et al. „Overview of IoT basic platforms for precision agriculture“. In: *International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures*. 2019, pp. 124–137.

[220]P. Mariager, J. Petersen, Z. Shelby, M. Van de Logt, and D. Barthel. *Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)*. RFC 8105. Internet Engineering Task Force, May 2017.

[221]Martin Bauer, Hamza Baqa, Sonia Bilbao, et al. *Semantic IoT Solutions - A Developer Perspective: Semantic Interoperability White Paper*. 2019.

[222]Martin Bauer, Hamza Baqa, Sonia Bilbao, et al. *Towards Semantic Interoperability Standards based on Ontologies*. 2019.

[224]Miguel A. Martínez-Prieto, Mario Arias, and Javier D. Fernández. „Exchange and Consumption of Huge RDF Data“. In: *The Semantic Web: Research and Applications*. Springer, 2012, pp. 437–452.

[226]Ryuichi Matsukura, Toru Kawaguchi, Michael Lagally, et al. *Web of Things (WoT) Architecture*. W3C Proposed Reccommendation. W3C, Jan. 2020.

[227]Keith McCloghrie, David Perkins, and Juergen Schoenwaelder. *Structure of Management Information Version 2 (SMIv2)*. RFC 2578. Internet Engineering Task Force, Apr. 1999.

[228]A. McGibney, A. E. Rodríguez, and S. Rea. „Managing wireless sensor networks within IoT ecosystems“. In: *2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 339–344.

[229]M. Mealling, L. Masinter, T. Hardie, and G. Klyne. *An IETF URN Sub-namespace for Registered Protocol Parameters*. RFC 3553. Internet Engineering Task Force, June 2003.

[230]Rafik Mecibah, Badis Djamaa, Ali Yachir, and Mohamed Aissani. „A Scalable Semantic Resource Discovery Architecture for the Internet of Things“. In: *Advances in computing systems and applications*. Ed. by Oualid Demigha, Badis Djamaa, and Abdenour Amamra. Vol. 50. Cham, Switzerland: Springer, 2019, pp. 37–47.

[232]Mike Botts, George Percivall, Carl Reed, and John Davidson. „OGC Sensor Web Enablement: Overview and High Level Architecture“. In: *GeoSensor Networks: Second International Conference*. Boston, MA, USA: Springer, 2008, pp. 175–190.

[233]Konstantin Mikhaylov, Joni Jamsa, Mika Luimula, Jouni Tervonen, and Ville Autio. „Intelligent Sensor Interfaces and Data Format“. In: *Intelligent Sensor Networks*. Ed. by Fei Hu and Qi Hao. CRC Press and Taylor & Francis, 2013, pp. 55–76.

[234]Florian Mikolajczak. *MYNO-Erweiterung: ESP-32 - SSN - Bootstrapping*. Project Study. University of Potsdam, Institute of Computer Science, 2019.

[235]Eric Miller and Frank Manola. *RDF Primer*. W3C Recommendation. W3C, Feb. 2004.

[236]Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. „A gap analysis of Internet-of-Things platforms“. In: *Computer Communications* 89-90 (2016), pp. 5–16.

[237]G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944. Internet Engineering Task Force, Sept. 2007.

[238]Brendan Moran, Hannes Tschofenig, and Henk Birkholz. *An Information Model for Firmware Updates in IoT Devices*. Internet-Draft. Internet Engineering Task Force, Oct. 2019.

[239]Brendan Moran, Hannes Tschofenig, David Brown, and Milosch Meriac. *A Firmware Update Architecture for Internet of Things*. Internet-Draft. Internet Engineering Task Force, Nov. 2019.

[242]*MQTT Version 3.1.1*. Tech. rep. OASIS, 2014.

[243]*MQTT Version 5.0*. Tech. rep. OASIS, 2019.

[244]Michael Mrissa, Lionel Medini, Jean-Paul Jamont, Nicolas Le Sommer, and Jerome Laplace. „An Avatar Architecture for the Web of Things“. In: *IEEE Internet Computing* 19.2 (2015), pp. 30–38.

[245]T. Mrugalski, M. Siodelski, B. Volz, et al. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. RFC 8415. Internet Engineering Task Force, Nov. 2018.

[246]Geoff Mulligan. „The 6LoWPAN Architecture“. In: *Proceedings of the 4th Workshop on Embedded Networked Sensors*. Cork, Ireland: ACM, 2007, pp. 78–82.

[247]Dae-Hyeok Mun, Minh Le Dinh, and Young-Woo Kwon. „An Assessment of Internet of Things Protocols for Resource-Constrained Applications“. In: *40th Annual Computer Software and Applications Conference*. IEEE, 2016, pp. 555–560.

[249]Gollu Appala Naidu, Jayendra Kumar, Vishnu Garudachedu, and Paidi Raja Ramesh. „6LoWPAN Border Router Implementation for IoT Devices on RaspberryPi“. In: *SSRN Electronic Journal* (2018).

[250] PrithviRaj Narendra, Simon Duquennoy, and Thiemo Voigt. „BLE and IEEE 802.15.4 in the IoT: Evaluation and Interoperability Considerations“. In: *Internet of Things. IoT Infrastructures*. Ed. by Benny Mandler. Springer, 2016, pp. 427–438.

[252] Behailu Negash, Tomi Westerlund, and Hannu Tenhunen. „Towards an interoperable Internet of Things through a web of virtual things at the Fog layer“. In: *Future Generation Computer Systems* 91 (2019), pp. 96–107.

[254] Paulo Neves and Joel Rodrigues. „Internet Protocol over Wireless Sensor Networks, from Myth to Reality“. In: *Journal of Communications* 5.3 (2010).

[255] Huan Nguyen, John Grundy, and Mohamed Almorsy. „Ontology-based Automated Support for Goal-Use case Model Analysis“. In: *Software Quality Journal* 24 (June 2015).

[256] J. Nieminen, T. Savolainen, M. Isomaki, et al. *IPv6 over BLUETOOTH(R) Low Energy*. RFC 7668. Internet Engineering Task Force, Oct. 2015.

[257] Michele Nitti, Virginia Pilloni, Giuseppe Colistra, and Luigi Atzori. „The Virtual Object as a Major Element of the Internet of Things a Survey“. In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1228–1240.

[258] Oscar Novo and Mario Di Francesco. „Semantic Interoperability in the IoT: Extending the Web of Things Architecture“. In: *ACM Trans. Internet Things* 1.1 (Mar. 2020).

[259] Alexander Nowak. „Web-basiertes Management von Geräten im IoT: Erstellung eines Web-Clients für den NETCONF-Server“. Bachelor Thesis. University of Potsdam, Institute of Computer Science, Jan. 2018.

[260] Michael Nowak. *Einbindung von Sensoren und Aktuatoren in die prototypische Implementierung der Verwaltung von IoT Geräten auf Basis von oneM2M Ontologie Beschreibungen*. Project Study. University of Potsdam, Institute of Computer Science, 2018.

[261] Michael Nowak. „Sicherheitsbetrachtung und Absicherung einer MQTT-basierten IoT-Plattform“. Master Thesis. University of Potsdam, Institute of Computer Science, Jan. 2020.

[262] Natalya F. Noy and Deborah L. McGuinness. „Ontology development 101: A guide to creating your first ontology“. In: *Knowledge Systems Laboratory, Stanford University* (2001).

[265] *OMA Device Management Protocol v1.3*. Tech. rep. 1.3. Open Mobile Alliance, May 2016.

[266] *OMA LightweightM2M v1.1.1*. Specification 1.1.1. Open Mobile Alliance, June 2019.

[267] *Ontology based Interworking - TS-0030-V-3.0.2*. Technical Specification TS-0030-V-3.0.2. oneM2M, Apr. 2019.

[269] *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation. W3C OWL Working Group, Dec. 2012.

[270] *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. W3C OWL Working Group, Dec. 2012.

[272] H. Panetto. „Towards a classification framework for interoperability of enterprise applications“. In: *International Journal of Computer Integrated Manufacturing* 20.8 (2007), pp. 727–740.

[274]Daniel Peintner, Kazuaki Nimura, Johannes Hund, and Zoltan Kis. *Web of Things (WoT) Scripting API*. W3C Working Draft. W3C, Oct. 2019.

[275]Charles E. Perkins. *Ad Hoc networking*. Boston: Addison-Wesley, 2004.

[276]*GeoSPARQL - A Geographic Query Language for RDF Data*. Tech. rep. OGC, Sept. 2012.

[277]Dennis Pfisterer, Kay Romer, Daniel Bimschas, et al. „SPITFIRE: Towards a Semantic Web of Things“. In: *IEEE Communications Magazine* 49.11 (2011), pp. 40–48.

[278]G. J. Pottie and W. J. Kaiser. „Wireless Integrated Network Sensors“. In: *Commun. ACM* 43.5 (May 2000), pp. 51–58.

[279]María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. „OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation“. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 10.2 (2014), pp. 7–34.

[280]A. Pras and J. Schoenwaelder. *On the Difference between Information Models and Data Models*. RFC 3444. Internet Engineering Task Force, Jan. 2003.

[281]R. Presuhn. *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*. RFC 3418. Internet Engineering Task Force, Dec. 2002.

[284]Hoan Nguyen Mau Quoc and Danh Le Phuoc. „An elastic and scalable spatiotemporal query processing for linked sensor data“. In: *Semantics, the 11th International Conference*. New York, USA: ACM, 2015, pp. 17–24.

[285]R. Venanzi, B. Kantarci, L. Foschini, and P. Bellavista. „MQTT-Driven Sustainable Node Discovery for Internet of Things-Fog Environments“. In: *IEEE International Conference on Communications (ICC)*. 2018, pp. 1–6.

[286]Joe Raad and Christophe Cruz. „A Survey on Ontology Evaluation Methods“. In: *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*. Lisbonne, Portugal, Nov. 2015.

[287]A. Rahman and E. Dijk. *Group Communication for the Constrained Application Protocol (CoAP)*. RFC 7390. Internet Engineering Task Force, Oct. 2014.

[288]Yves Raimond and Guus Schreiber. *RDF 1.1 Primer*. W3C Note. W3C, June 2014.

[289]R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. „Cyber-Physical Systems: The Next Computing Revolution“. In: *Design Automation Conference*. June 2010, pp. 731–736.

[290]Fano Ramparany and Quyet H. Cao. „A Semantic Approach to IoT Data Aggregation and Interpretation applied to Home Automation“. In: *2016 International Conference on Internet of Things and Applications (IOTA)*. [Piscataway, NJ]: IEEE, 2016.

[291]P. Rawat, K. D. Singh, H. Chaouchi, et al. „Wireless sensor networks: a survey on recent developments and potential synergies“. In: *The Journal of supercomputing* 68.1 (2014), pp. 1–48.

[294]*Recommendation ITU-T X.660 | ISO/IEC 9834-1: OSI networking and system aspects – Naming, Addressing and Registration*. Tech. rep. ITU-T, July 2011.

[295]E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Internet Engineering Task Force, Aug. 2018.

[296] E. Rescorla and B. Korver. *Guidelines for Writing RFC Text on Security Considerations*. RFC 3552. Internet Engineering Task Force, July 2003.

[297] Elena Reshetova and Michael McCool. *Web of Things (WoT) Security and Privacy Guidelines*. W3C Note. W3C, Nov. 2019.

[298] Hajo Rijgersberg, Don Willems, Xin-Ying Ren, Mari Wigham, and Jan Top. *Ontology of units of Measure (OM) 2.0*. Tech. rep. eFoodLab, Dec. 2017.

[300] Thomas Rix, Kai-Oliver Detken, and Marcel Jahnke. „Transformation between XML and CBOR for network load reduction". In: *3rd International Symposium on Wireless Systems (IDAACS-SWS)*. 2016, pp. 106–111.

[301] Maria Ines Robles and Petri Jokela. „Design of a performance measurements platform in lightweight M2M for Internet of Things". In: *IRTF & ISOC Workshop on Research and Applications of Internet Measurements (RAIM)*. 2015.

[302] Remy Rojas, Lionel Médini, and Amélie Cordier. „Toward Constrained Semantic WoT". In: *Seventh International Workshop on the Web of Things*. New York, N.Y.: ACM, 2016, pp. 31–37.

[303] John Romkey. „Toast of the IoT: The 1990 Interop Internet Toaster". In: *IEEE Consumer Electronics Magazine* 6.1 (2017), pp. 116–119.

[304] Sebastian Rudolph. „Foundations of Description Logics". In: *Reasoning Web International Summer School*. Springer, 2011, pp. 76–136.

[305] Stuart J. Russell and Peter Norvig. *Künstliche Intelligenz: Ein moderner Ansatz*. München: Pearson, 2012.

[306] M. Ruta, F. Scioscia, A. Pinto, et al. „Resource Annotation, Dissemination and Discovery in the Semantic Web of Things: A CoAP-Based Framework". In: *IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, pp. 527–534.

[307] Michele Ruta, Floriano Scioscia, Giuseppe Loseto, and Eugenio Di Sciascio. „A logic-based CoAP extension for resource discovery in Semantic Sensor Networks". In: *Proceedings of the 5th International Conference on Semantic Sensor Networks*. Vol. 904. CEUR-WS.org, 2012, pp. 17–32.

[308] S. Andy, B. Rahardjo, and B. Hanindhito. „Attack scenarios and security analysis of MQTT communication protocol in IoT system". In: *4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. 2017, pp. 1–6.

[309] S. K. Datta and C. Bonnet. „A Lightweight Framework for Efficient M2M Device Management in oneM2M Architecture". In: *International Conference on Recent Advances in Internet of Things (RIoT)*. Piscataway, NJ: IEEE, 2015, pp. 1–6.

[311] Christopher M. Sadler and Margaret Martonosi. „Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks". In: *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006.

[313] Kristina Sahlmann, Vera Clemens, Michael Nowak, and Bettina Schnor. „MUP: Simplifying Secure Over-The-Air Update with MQTT for Constrained IoT Devices". In: *Sensors* 21.1 (2021).

[314] Kristina Sahlmann, Alexander Lindemann, and Bettina Schnor. „Binary Representation of Device Descriptions: CBOR versus RDF HDT". In: *Proceedings of the 17. GI/ITG KuVS Fachgespräch Sensornetze: Drahtlose Sensornetze*. Braunschweig, Germany, 2018.

[315] Kristina Sahlmann, Thomas Scheffler, and Bettina Schnor. „Managing IoT device capabilities based on oneM2M ontology descriptions“. In: *Proceedings of the 16. GI/ITG KuVS Fachgespräch Sensornetze*. Hamburg, Germany: HAW Hamburg, 2017, pp. 23–26.

[316] Kristina Sahlmann and Thomas Schwotzer. „MOCAP: Towards the Semantic Web of Things“. In: *Posters&Demos@SEMANTiCS 2015 and DSci15 Workshop*. Vienna, Austria: CEUR Workshop Proceedings, 2015, pp. 59–62.

[317] Kristina Sahlmann and Thomas Schwotzer. „Ontology-based Virtual IoT Devices for Edge Computing“. In: *Proceedings of the 8th International Conference on the Internet of Things*. IOT '18. Santa Barbara, California: ACM, 2018, 15:1–15:7.

[318] Kristina Sahlmann, Thomas Schwotzer, and Bettina Schnor. „The Ad hoc Semantic Internet Protocol (ASIP) for Constrained Devices“. In: *Proceedings of the 15. GI/ITG KuVS Fachgespräch Sensornetze*. Augsburg, Germany, Sept. 2016, pp. 49–52.

[319] Satya Sahoo, Timothy Lebo, and Deborah McGuinness. *PROV-O: The PROV Ontology*. W3C Recommendation. W3C, Apr. 2013.

[320] P. Saint-Andre and J. Klensin. *Uniform Resource Names (URNs)*. RFC 8141. Internet Engineering Task Force, Apr. 2017.

[321] Tara Salman and Raj Jain. „A Survey of Protocols and Standards for the Internet of Things“. In: *arXiv preprint arXiv:1903.11549* (2019).

[322] Musa G. Samaila, Miguel Neto, Diogo A. B. Fernandes, Mário M. Freire, and Pedro R. M. Inácio. „Challenges of securing Internet of Things devices: A survey“. In: *Security and Privacy* 1.2 (2018).

[323] Justin Samuel, Nick Mathewson, Justin Cappos, and Roger Dingledine. „Survivable Key Compromise in Software Update Systems“. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. Chicago, Illinois, USA: Association for Computing Machinery, 2010, 61–72.

[324] L. Sanchez, K. McCloghrie, and J. Saperia. *Requirements for Configuration Management of IP-based Networks*. RFC 3139. Internet Engineering Task Force, June 2001.

[325] J. Schaad. *CBOR Object Signing and Encryption (COSE)*. RFC 8152. Internet Engineering Task Force, July 2017.

[326] Thomas Scheffler. „RESTful Services für 6LoWPAN Networks“. In: *Proceedings of the 14. GI/ITG KuVS Fachgespräch Sensornetze*. Erlangen, 2015.

[327] Thomas Scheffler and Olaf Bonneß. „Manage resource-constrained IoT devices through dynamically generated and deployed YANG models“. In: *Applied Networking Research Workshop (ANRW)*. Prague, Czech Republic: Internet Engineering Task Force, 2017.

[328] Jochen Schiller. *Mobilkommunikation*. Pearson-Studium, 2003.

[329] Paul Schmelzer and Jens-Peter Akelbein. „Evaluation of Hardware Requirements for Device Management of Constrained Nodes Based on the LWM2M Standard“. In: *CERC*. 2019, pp. 103–110.

[330] J. Schoenwaelder. *Common YANG Data Types*. RFC 6991. Internet Engineering Task Force, July 2013.

[331] J. Schoenwaelder. *Overview of the 2002 IAB Network Management Workshop*. RFC 3535. Internet Engineering Task Force, May 2003.

[332]J. Schoenwaelder. *Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules*. RFC 6643. Internet Engineering Task Force, July 2012.

[333]Jürgen Schönwälder. „Internet Management Protocols“. In: *Handbook of network and system administration*. Amsterdam: Elsevier, 2007, pp. 295–328.

[334]Jürgen Schönwälder, Tina Tsou, and Behcet Sarikaya. „Protocol Profiles for Constrained Devices“. In: *Proceedings of the IAB Workshop on Interconnecting Smart Objects with the Internet*. 2011.

[336]Jürgen Schönwälder, Kent Watsen, Mehmet Ersue, and Vladislav Perelman. *Network Configuration Protocol Light (NETCONF Light)*. Internet-Draft. Internet Engineering Task Force, Jan. 2012.

[337]Anuj Sehgal, Vladislav Perelman, Siarhei Kuryla, and Jürgen Schönwälder. „Management of Resource Constrained Devices in the Internet of Things“. In: *IEEE Communications Magazine* 50.12 (2012), pp. 144–149.

[338]L. Seitz, S. Gerdes, G. Selander, M. Mani, and S. Kumar. *Use Cases for Authentication and Authorization in Constrained Environments*. RFC 7744. Internet Engineering Task Force, Jan. 2016.

[339]K. Seklou, P. Kokkinos, N. D. Tselikas, and A. C. Boukouvalas. „Monitoring and management of home appliances with NETCONF and YANG“. In: *Proceedings of the 23rd Pan-Hellenic Conference on Informatics - PCI '19*. ACM Press, 2019, pp. 25–32.

[340]G. Selander, J. Mattsson, F. Palombini, and L. Seitz. *Object Security for Constrained RESTful Environments (OSCORE)*. RFC 8613. Internet Engineering Task Force, July 2019.

[343]Cigdem Sengul and Anthony Kirby. *Message Queuing Telemetry Transport (MQTT)-TLS profile of Authentication and Authorization for Constrained Environments (ACE) Framework*. Internet-Draft. Internet Engineering Task Force, Dec. 2020.

[344]Alexandru Serbanati, Carlo Maria Medaglia, and Ugo Biader Ceipidor. „Building Blocks of the Internet of Things: State of the Art and Beyond“. In: *Deploying RFID*. Rijeka: IntechOpen, 2011. Chap. 20.

[345]*Service Layer Core Protocol - TS-0004-V3.11.2*. Tech. rep. TS-0004-V3.11.2. oneM2M, 2019-05-08.

[346]M. Sethi, J. Arkko, A. Keranen, and H. Back. *Practical Considerations and Implementation Experiences in Securing Smart Object Networks*. RFC 8387. Internet Engineering Task Force, May 2018.

[347]Mohit Sethi, Behcet Sarikaya, and Dan Garcia-Carillo. *Secure IoT Bootstrapping: A Survey*. Internet-Draft. Internet Engineering Task Force, Mar. 2020.

[348]Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. *Capturing the contributions of the semantic web to the IoT: A unifying vision*. 2017.

[349]Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. „IoT-O, a Core-Domain IoT Ontology to Represent Connected Devices Networks“. In: *Knowledge engineering and knowledge management*. Springer International, 2016, pp. 561–576.

[350] Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. „Lowering knowledge: Making constrained devices semantically interoperable“. In: *ISWC 2016, Posters & Demonstrations Track*. CEUR-WS, 2016.

[351] Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. „Reasoning on the edge or in the cloud?“ In: *Internet Technology Letters* 2.1 (2019), p. 51.

[352] P. Shafer. *An Architecture for Network Management Using NETCONF and YANG*. RFC 6244. Internet Engineering Task Force, June 2011.

[353] Cheena Sharma and Dr. Naveen Kumar Gondhi. „Communication Protocol Stack for Constrained IoT Systems“. In: *Proceedings, 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. [Piscataway, New Jersey]: IEEE, 2018, pp. 1–6.

[354] Z. Shelby. *Constrained RESTful Environments (CoRE) Link Format*. RFC 6690. Internet Engineering Task Force, Aug. 2012.

[355] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. *Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*. RFC 6775. Internet Engineering Task Force, Nov. 2012.

[356] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. Internet Engineering Task Force, June 2014.

[357] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded internet*. Chichester, U.K.: John Wiley & Sons, 2009.

[358] Zach Shelby, Michael Koster, Carsten Bormann, Peter Van der Stok, and Christian Amsüss. *CoRE Resource Directory*. Internet-Draft. Work in Progress. Internet Engineering Task Force, Nov. 2020. 84 pp.

[359] Zhengguo Sheng, Shusen Yang, Yifan Yu, et al. „A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities“. In: *IEEE Wireless Communications* 20.6 (2013), pp. 91–98.

[360] Amit P. Sheth, Cory Henson, and Satya S. Sahoo. „Semantic Sensor Web“. In: *IEEE Internet Computing* 12 (4) (2008), pp. 78–83.

[361] Feifei Shi, Qingjuan Li, Tao Zhu, and Huansheng Ning. „A Survey of Data Semantization in Internet of Things“. In: *Sensors* 18.1 (2018).

[362] Yoav Shoham. „Agent-oriented programming“. In: *Artificial Intelligence* 60 (1993), pp. 51–92.

[363] Jonathan de C. Silva, Joel J. P. C. Rodrigues, Jalal Al-Muhtadi, Ricardo A. L. Rabêlo, and Vasco Furtado. „Management Platforms and Protocols for Internet of Things: A Survey“. In: *Sensors* 19.3 (2019).

[365] Meena Singh, M. A. Rajan, V. L. Shivraj, and P. Balamuralidhar. „Secure MQTT for Internet of Things (IoT)“. In: *Fifth International Conference on Communication Systems and Network Technologies (CSNT)*. IEEE, 2015, pp. 746–751.

[367] *SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping*. Technical Specification ETSI TS 103 264 V2.1.1. ETSI, Mar. 2017.

[368] *SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping*. Technical Specification ETSI TS 103 264 V3.1.1. ETSI, Feb. 2020.

[370] Stanford-Clark A., Truong H. L. „MQTT For Sensor Networks (MQTT-SN): Protocol Specification, Version 1.2“. In: *International business machines (IBM) Corporation version* 1.2 (2013).

[371] Wolf-Jörgen Stange. *Interoperability in the Internet of Things: Analyse der Arduino IoT Cloud zur Absicherung des MYNO-Projekts*. Project Study. University of Potsdam, Institute of Computer Science, 2020.

[372] Wolf-Jörgen Stange. *Leistungsanalyse: Ein Vergleich von RDF4Led und RDFLib*. Project Study. University of Potsdam, Institute of Computer Science, 2019.

[374] F. Strauss and J. Schönwälder. *SMIng - Next Generation Structure of Management Information*. RFC 3780. Internet Engineering Task Force, May 2004.

[375] Wei-Tsung Su, Wei-Cheng Chen, and Chao-Chun Chen. „An Extensible and Transparent Thing-to-Thing Security Enhancement for MQTT Protocol in IoT Environment“. In: *GIoTS, Global IoT Summit*. IEEE, 2019, pp. 1–4.

[376] Xiang Su, Hao Zhang, Jukka Riekki, et al. „Connecting IoT Sensors to Knowledge-based Systems by Transforming SenML to RDF“. In: *Procedia Computer Science* 32 (2014), pp. 215–222.

[378] Girum Teklemariam, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester. „Facilitating the creation of IoT applications through conditional observations in CoAP“. In: *EURASIP Journal on Wireless Communications and Networking* 2013.1 (2013), p. 177.

[379] Uchralt Temuulen. *IoT Projekt Kafka Connector*. Project Study. University of Potsdam, Institute of Computer Science, 2020.

[380] Herman J. ter Horst. „Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 3.2-3 (2005), pp. 79–115.

[381] Incorporated Texas Instruments. *CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee/ZigBee IP Applications (Rev. C): User's Guide*. Tech. rep. 2012-2013.

[382] Anurag Thantharate, Cory Beard, and Poonam Kankariya. „CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air“. In: *International Conference on Internet of Things (iThings)*. 2019, pp. 1065–1070.

[385] S. Thomson, T. Narten, and T. Jinmei. *IPv6 Stateless Address Autoconfiguration*. RFC 4862. Internet Engineering Task Force, Sept. 2007.

[386] P. Thubert, E. Nordmark, S. Chakrabarti, and C. Perkins. *Registration Extensions for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Neighbor Discovery*. RFC 8505. Internet Engineering Task Force, Nov. 2018.

[387] Andreas Tolk and James A. Muguira. „The Levels of Conceptual Interoperability Model“. In: *Fall Simulation Interoperability Workshop*. 2003.

[389] Kunihiko Toumura, Toru Kawaguchi, Ryuichi Matsukura, Kazuo Kajimoto, and Michael Lagally. *Web of Things (WoT) Architecture 1.1*. W3C Working Draft. W3C, Nov. 2020.

[390] Mihai Vlad Trifa. „Building Blocks for a Participatory Web of Things: Devices, Infrastructures, and Programming Frameworks“. Dissertation. ETH Zurich, 2011.

[391] Vlad Trifa, Dominique Guinard, and David Carrera. *Web Thing Model*. W3C Member Submission. 2017.

[392] J. Tripathi, J. de Oliveira, and JP. Vasseur. *Performance Evaluation of the Routing Protocol for Low-Power and Lossy Networks (RPL)*. RFC 6687. Internet Engineering Task Force, Oct. 2012.

[393] H. Tschofenig and S. Farrell. *Report from the Internet of Things Software Update (IoTSU) Workshop 2016*. RFC 8240. Internet Engineering Task Force, Sept. 2017.

[394] Hannes Tschofenig and Emmanuel Baccelli. „Cyberphysical Security for the Masses: A Survey of the Internet Protocol Suite for Internet of Things Security". In: *IEEE Security & Privacy* 17.5 (2019), pp. 47–57.

[395] Vlasios Tsiatsis, Alexander Gluhak, Tim Bauge, et al. „The SENSEI Real World Internet Architecture". In: *Towards the future internet*. Amsterdam: IOS Press, 2010, pp. 247–256.

[396] Nicolas Tsiftes, Adam Dunkels, and Thiemo Voigt. „Efficient Sensor Network Reprogramming through Compression of Executable Modules". In: *5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. 2008, pp. 359–367.

[398] Philip Ullrich. „Interoperabilität im IoT - Eine Fallstudie mit MQTT". Bachelor Thesis. University of Potsdam, Institute of Computer Science, Aug. 2017.

[399] Günther v. Görz, Josef Schneeberger, and Ute Schmid, eds. *Handbuch der Künstlichen Intelligenz*. München: Oldenbourg, 2013.

[400] S. Vallin and M. Bjorklund. *A YANG Data Model for Alarm Management*. RFC 8632. Internet Engineering Task Force, Sept. 2019.

[401] Hylke van der Schaaf and Reinhard Herzog. „Mapping the OGC SensorThings API onto the OpenIoT Middleware". In: *Interoperability and open-source solutions for the internet of things*. Vol. 9001. Cham: Springer, 2014, pp. 62–70.

[402] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP The Next Internet*. Burlington, MA: Morgan Kaufmann Publishers/Elsevier, 2010.

[403] Michel Veillette, Peter van der Stok, Alexander Pelov, Andy Bierman, and Ivaylo Petrov. *CoAP Management Interface (CORECONF)*. Internet-Draft. Internet Engineering Task Force, Jan. 2021.

[404] Savita Vijay and M. K. Banga. „Management of IoT Devices in Home Network via Intelligent Home Gateway Using NETCONF". In: *Ubiquitous Communications and Network Computing. Proceedings of UBICNET 2017*. Vol. 218. Cham: Springer, 2018, pp. 196–207.

[405] X. Vilajosana, K. Pister, and T. Watteyne. *Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration*. RFC 8180. Internet Engineering Task Force, May 2017.

[406] Elvis Vogli, Mahdi Ben Alaya, Thierry Monteil, Luigi Alfredo Grieco, and Khalil Drira. „An efficient resource naming for enabling constrained devices in SmartM2M architecture". In: *International Conference on Industrial Technology (ICIT)*. Piscataway, NJ: IEEE, 2015, pp. 1832–1837.

[407]E. Voit, A. Clemm, A. Gonzalez Prieto, E. Nilsen-Nygaard, and A. Tripathy. *Dynamic Subscription to YANG Events and Datastores over NETCONF*. RFC 8640. Internet Engineering Task Force, Sept. 2019.

[408]E. Voit, A. Clemm, A. Gonzalez Prieto, E. Nilsen-Nygaard, and A. Tripathy. *Subscription to YANG Notifications*. RFC 8639. Internet Engineering Task Force, Sept. 2019.

[409]Stefan Wallin and Claes Wikström. „Automating Network and Service Configuration Using NETCONF and YANG“. In: *25th Large Installation System Administration Conference (LISA)*. 2011.

[410]Wenguang Wang, Andreas Tolk, and Weiping Wang. „The Levels of Conceptual Interoperability Model: Applying Systems Engineering Principles to M&S“. In: *Proceedings of the 2009 Spring Simulation Multiconference*. Vol. 168. San Diego, CA, USA, 2009, pp. 1–9.

[411]M. Wasserman. *Using the NETCONF Protocol over Secure Shell (SSH)*. RFC 6242. Internet Engineering Task Force, June 2011.

[413]Mark Weiser. „The Computer for the 21st Century“. In: *Scientific American* 265.3 (1991), pp. 94–104.

[415]T. Winter, P. Thubert, A. Brandt, et al. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. Internet Engineering Task Force, Mar. 2012.

[416]Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes: Compressing and indexing documents and images*. San Francisco, Calif.: Kaufmann, 2007.

[417]A.K.Y. Wong, P. Ray, N. Parameswaran, and J. Strassner. „Ontology mapping for the interoperability problem in network management“. In: *IEEE Journal on Selected Areas in Communications* 23.10 (2005), pp. 2058–2068.

[419]Zhenyu Wu, Yuan Xu, Yunong Yang, et al. „Towards a Semantic Web of Things: A Hybrid Semantic Annotation, Extraction, and Reasoning Framework for Cyber-Physical System“. In: *Sensors* 17.2 (2017).

[420]Hui Xu, Chunzhi Wang, Wei Liu, and Hongwei Chen. „NETCONF-based Integrated Management for Internet of Things using RESTful Web Services“. In: *International Journal of Future Generation Communication and Networking* 5.3 (2012), pp. 73–82.

[421]Liyang Yu. *A Developer's Guide to the Semantic Web*. Springer Berlin Heidelberg, 2014.

[422]Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. „Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check“. In: *IEEE Access* 7 (2019), pp. 71907–71920.

Websites

[12]*Arduino IoT Cloud*. URL: `https://www.arduino.cc/en/IoT/HomePage` (visited on Dec. 10, 2020).

[13]*Arduino Nano 33 IoT*. 2019. URL: `https://store.arduino.cc/arduino-nano-33-iot` (visited on Dec. 10, 2020).

[14]*ATECC608A: Crypto Authentication Device Summary Datasheet*. 2018. URL: `https://content.arduino.cc/assets/microchip_atecc608a_cryptoauthentication_device_summary_datasheet-DS40001977B.pdf` (visited on Dec. 10, 2020).

[15]*AWS IoT*. URL: https://aws.amazon.com/de/iot/ (visited on Dec. 10, 2020).

[35]Stanford Center for Biomedical Informatics Research. *Protégé - Open-source Ontology Editor*. 2016. URL: https://protege.stanford.edu/ (visited on Dec. 10, 2020).

[59]*CC2538 Development Kit*. 2017. URL: https://www.ti.com/tool/CC2538DK (visited on Dec. 10, 2020).

[68]*Collaborative Open Market to Place Objects at your Service*. June 2015. URL: http://www.compose-project.eu/ (visited on Dec. 20, 2020).

[70]*Contiki-NG v4.5*. URL: https://github.com/contiki-ng/contiki-ng (visited on Dec. 10, 2020).

[71]*Contiki v3.0*. URL: https://github.com/contiki-os/contiki (visited on Dec. 10, 2020).

[76]*Crypto-algorithms library*. URL: https://github.com/B-Con/crypto-algorithms (visited on Dec. 10, 2020).

[78]Laura Daniele, Raúl Garcia-Castro, Maxime Lefrançois, and Maria Poveda-Villalon. *SAREF*. June 2019. URL: https://saref.etsi.org/core/ (visited on Feb. 28, 2020).

[90]*Eclipse Mosquitto*. 2021. URL: https://mosquitto.org/ (visited on Jan. 10, 2021).

[91]*Eclipse SmartHome*. URL: https://www.eclipse.org/projects/archives.php (visited on Sept. 20, 2020).

[92]*Eclipse Vorto Language*. 2019. URL: https://github.com/eclipse/vorto/blob/master/docs/vortolang-1.0.md (visited on Dec. 20, 2020).

[96]*ESP-32 MCU Chip*. URL: https://www.espressif.com/en/products/socs/esp32 (visited on Dec. 10, 2020).

[98]European Union Agency For Network And Information Security. *Baseline Security Recommendations for IoT*. 2017. URL: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot (visited on Dec. 10, 202).

[106]*Flask-MQTT Library*. 2018. URL: https://github.com/stlehmann/Flask-MQTT (visited on Dec. 10, 2020).

[107]*Flask Web Development*. 2018. URL: https://flask.palletsprojects.com (visited on Dec. 10, 2020).

[114]*Getting Started With YANG Push*. 2020. URL: https://yumaworks.freshdesk.com/support/solutions/articles/1000294506-getting-started-with-yang-push (visited on Dec. 20, 2020).

[117]*gmqtt issue 96: Topic Alias support?* URL: https://github.com/wialon/gmqtt/issues/96 (visited on Dec. 10, 2020).

[118]*gmqtt: Python async MQTT client implementation*. URL: https://github.com/wialon/gmqtt (visited on Dec. 10, 2020).

[123]Neil Gross. *The Earth Will Don an Electronic Skin*. Aug. 1999. URL: https://www.bloomberg.com/news/articles/1999-08-29/14-the-earth-will-don-an-electronic-skin (visited on Dec. 10, 2020).

[149]*HiveMQ*. 2019. URL: https://www.hivemq.com (visited on Oct. 1, 2019).

[157]Nicholas Humfrey. *EasyRdf Converter*. 2012. URL: https://www.easyrdf.org/converter (visited on Dec. 10, 2020).

[159]*IBM IoT*. URL: https://www.ibm.com/de-de/cloud/internet-of-things (visited on Dec. 10, 2020).

[164]*IoT-Lite Ontology*. 2015. URL: https://www.w3.org/Submission/iot-lite/ (visited on Dec. 20, 2020).

[166]*iotschema.org*. 2019. URL: http://iotschema.org/ (visited on Dec. 20, 2020).

[181]*JSON-LD Playground*. 2013. URL: https://json-ld.org/playground/ (visited on Dec. 10, 2020).

[190]Naveen Khan. *NETCONF by Example*. 2015. URL: https://www.ietf.org/slides/slides-edu-network-configuration-with-netconf-00.pdf (visited on Dec. 20, 2020).

[223]Martín Serrano, Payam Barnaghi, Francois Carrez Philippe Cousin, Ovidiu Vermesan, Peter Friess. *IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps*. 2015. (Visited on Dec. 20, 2020).

[225]Nick Mathewson. *Thandy: Secure Update for Tor | Google Open Source Blog*. 2009. (Visited on Dec. 10, 2020).

[231]*Micro-ecc library*. URL: https://github.com/kmackay/micro-ecc (visited on Dec. 10, 2020).

[240]*Mosquitto issue 1757: MQTT v5 Topic Alias not sent to client?* URL: https://github.com/eclipse/mosquitto/issues/1757 (visited on Dec. 10, 2020).

[241]*MQTT in Contiki-NG*. Accessed on: 10 December 2020. URL: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-MQTT (visited on Dec. 10, 2020).

[248]Paul Murdock, Louay Bassbouss, Martin Bauer, et al. *Semantic Interoperability for the Web of Things*. 2016. URL: https://hal.archives-ouvertes.fr/hal-01362033 (visited on Dec. 20, 2020).

[251]*ncclient: Python library for NETCONF clients*. 2017. URL: https://github.com/ncclient/ncclient (visited on Dec. 10, 2020).

[253]*Netconf Client/Server Library*. 2017. URL: https://github.com/choppsv1/netconf/ (visited on Dec. 10, 2020).

[263]*OGC Sensor Web Enablement: Overview And High Level Architecture: SWE White Paper*. 2013. URL: http://www.opengis.net/doc/wp/swe-high-level-architecture (visited on Mar. 4, 2020).

[264]Jonas Olsson. *6LoWPAN demystified*. 2014. URL: http://www.ti.com/lit/wp/swry013/swry013.pdf (visited on Dec. 20, 2020).

[268]*openHAB*. URL: https://www.openhab.org/ (visited on Oct. 21, 2020).

[271]*Paho MQTT and MQTT-SN Clients*. 2021. URL: https://www.eclipse.org/paho/ (visited on Jan. 10, 2021).

[273]*Paramiko Python SSH module*. 2018. URL: https://github.com/paramiko/paramiko/ (visited on Dec. 10, 2020).

[282]*Pyang Library*. 2017. URL: https://github.com/mbj4668/pyang (visited on Dec. 10, 2020).

[283]*PyShark v0.4.2.11*. URL: https://kiminewt.github.io/pyshark (visited on Dec. 10, 2020).

[292]*RDF Validator*. 2006. URL: https://www.w3.org/RDF/Validator/ (visited on Dec. 10, 2020).

[293]*RDFLib Library*. 2017. URL: https://github.com/RDFLib/rdflib (visited on Dec. 10, 2020).

[299]*RIOT OS*. 2013. URL: https://www.riot-os.org/ (visited on Dec. 10, 2020).

[310]Harald Sack. *Knowledge Engineering with Semantic Web Technologies*. 2015. URL: https://open.hpi.de/courses/semanticweb2015 (visited on Dec. 20, 2020).

[312]Kristina Sahlmann. *MYNO Source Code*. URL: https://github.com/ksahlmann/myno (visited on Feb. 15, 2021).

[335]Jürgen Schönwälder. *Network Configuration Management with NETCONF and YANG*. 2012. URL: https://datatracker.ietf.org/doc/slides-edu-netconf-yang/ (visited on Dec. 10, 2020).

[341]*Semantic Sensor Network Ontology*. 2005. URL: https://www.w3.org/2005/Incubator/ssn/ssnx/ssn (visited on Dec. 20, 2020).

[342]*Semantic Sensor Network XG Final Report*. 2011. URL: https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/ (visited on Dec. 20, 2020).

[364]Simon Duquennoy. *Contiki-NG: The OS for Next Generation IoT Devices*. Apr. 2018. URL: https://youtu.be/4n_mTU76wds (visited on Dec. 10, 2020).

[366]Ian Skerrett. *Why MQTT Has Become the De-Facto IoT Standard*. Oct. 2019. URL: https://dzone.com/articles/why-mqtt-has-become-the-de-facto-iot-standard (visited on Jan. 10, 2021).

[369]Ralph Hodgson; Paul J. Keller; Jack Hodges; Jack Spivak, ed. *QUDT - Quantities, Units, Dimensions and Data Types Ontologies*. Mar. 2014. URL: http://www.qudt.org/ (visited on Dec. 20, 2020).

[373]Alex Stolz. *RDF Translator*. 2013. URL: https://rdf-translator.appspot.com/ (visited on Dec. 10, 2020).

[377]Ronak Sutaria. *Understanding Wireless Routing For IoT Networks*. 2014. URL: https://www.electronicdesign.com/technologies/communications/article/21798975/understanding-wireless-routing-for-iot-networks (visited on Jan. 10, 2021).

[383]*The Contiki IoT OS is the calm in the storm of IoT OSes*. 2017. URL: https://internetofthingsagenda.techtarget.com/feature/The-Contiki-IoT-OS-is-the-calm-in-the-storm-of-IoT-OSes (visited on Dec. 10, 2020).

[384]IoT Analytics: Market Insights for the Internet of Things. *State of IoT Q4 2020 and Outlook 2021*. 2020. URL: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ (visited on Dec. 10, 2020).

[388]*Tor: anonymity online*. URL: https://www.torproject.org/ (visited on Dec. 10, 2020).

[397] *uIP: the IPv6 stack*. 2020. URL: https://contiki-ng.readthedocs.io/en/master/_api/group__uip.html (visited on Dec. 20, 2020).

[412] Volker Weber. *25 Jahre WLAN*. Sept. 2015. URL: https://www.heise.de/newsticker/meldung/25-Jahre-WLAN-2789709.html (visited on Dec. 10, 2020).

[414] *Whsniff v1.3*. URL: https://github.com/homewsn/whsniff (visited on Dec. 10, 2020).

[418] *World's Smallest IPv6 Stack By Cisco, Atmel, SICS*. 2008. URL: https://tech.slashdot.org/story/08/10/15/1839209/worlds-smallest-ipv6-stack-by-cisco-atmel-sics (visited on Oct. 12, 2020).

[423] Evgeny Zolin. *Description Logic Complexity Navigator*. 2013. URL: http://www.cs.man.ac.uk/~ezolin/dl/ (visited on Sept. 30, 2019).

Alphabetical Index

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Potsdam, den 20. März 2021

Diplom-Informatikerin
(Fachhochschule) Kristina
Sahlmann