2017 21st International Conference on Control Systems and Computer Science

Survey of Standardized Protocols for the Internet of Things

Iulia Florea, Razvan Rughinis, Laura Ruse, Dan Dragomir,
Computer Science and Engineering Department
University Politehnica of Bucharest
Bucharest, Romania
Email: iulia.florea@cti.pub.ro, {razvan.rughinis, laura.ruse, dan.dragomir}@cs.pub.ro

***Abstract*—Internet of Things is a growing technology, developing in multiple fields, from smart homes to health and industries. From a technological perspective, it enables the development of new protocols and scenarios, since the standard network protocol suite cannot face the growing number of connected devices and data transmitted. This paper aims to present several standardized protocols, at different networking levels, developed especially for embedded devices with low memory, low processing power and low data rate. We also propose a smart home scenario that uses only standardized IoT protocols.**

Keywords—IoT, CoAP, MQTT, mDNS, DNS-SD, 802.15.4, LoRaWAN, 6LoWPAN, smart home, sensor nodes, robot assistant.

I. Introduction

Recently, the Internet of Things (IoT) has become an important research field. It can be described as a communication system where any device can be connected to the Internet and be able to identify itself to other objects. It can be applied in different domains, from personal use, such as smart homes or wearable devices to fields such as urban environment monitoring, health care, industrial automation or emergencies.

Generally, the IoT devices have low memory, reduced battery capacity, reduced processing capabilities and vulnerable radio conditions. The standard TCP/IP stack is not suitable for this environment, so working groups have started to adapt the existing protocols to new versions for IoT.

An addressing scheme, such as IPv6 should be taken into consideration, since there are billions of interconnected nodes. Multiple working groups already started to standardize IoT specific protocols, such as 6LoWPAN (RFC 4944 and RFC 6282), IEEE802.15.4 and ZigBee describe ways of enabling IPv6 in constrained environments. Other requirements refer to security and privacy, since the number of Denial of Service attacks has recently increased.

At the application layer, a common way of retrieving and requesting data is using Web architecture, and more specific, HTTP. This uses URIs as resource identifiers and it is based on REST architecture to publish information. For embedded devices, there is a Constrained RESTful Environments (CoRE) IETF working group, that aims to develop RESTful protocols, compatible with HTTP for resource-constrained devices. They specified CoAP, an application-layer protocol for IoT.

In this paper, we present a survey of the most used standardized protocols for Internet of Things. We investigate application layer protocols (CoAP, MQTT), service discovery protocols (mDNS, DNS-SD, uBonjour) and infrastructure protocols (IEEE 802.15.4, 6LoWPAN, LoRaWAN). We present different ways to integrate different application-layer protocols such as MQTT, CoAP and HTTP. Finally, we propose a smart home system with wireless sensor nodes and robot assistants that use standardized IoT protocols (IEEE 802.15.4, 6LoWPAN, CoAP).

The paper is organized as follows. The second section presents the application-layer protocols, the third the service discovery methods, the fourth the interaction between protocols and the fifth presents a smart home scenario, that uses only standardized protocols. Finally, some conclusions are summarized.

II. Application Layer Protocols

The application layer provides the services that the users need. These protocols handle information between gateways in local networks and Internet, update users with latest obtained data and carry commands from applications to end devices.

A. CoAP

Nowadays, applications depend on the Web architecture, using HTTP for getting information or updates. This is based on REST (Representational State Transfer) architectural style, that makes resources available through URIs. HTTP is rather complex for small IoT devices, so a new protocol based on REST was specified by IETF CoRE working group. This protocol is called Constrained Application Protocol (CoAP), a web transfer protocol that exchanges data between servers and clients [1].

It uses UDP instead of TCP at the transport layer and defines a "message layer" to deal with retransmissions. Fig.1.

2379-0482/17 $31.00 © 2017 IEEE
DOI 10.1109/CSCS.2017.33

presents the CoAP header, which has a length that varies from 4 to maximum 16 bytes [2]. The first two bits, "Ver" represents the version of CoAP, set to 1 for now; it defines the current version and other values are reserved for the future. The next field, "T" is the type of Transaction. It is a 2-bit integer and its possible values are 0, for confirmable messages, 1 for non-confirmable, 2 for acknowledgement and 3 for reset packets. "OC" is the Option Count, referring to the number of options set in the packet. "Code" in an 8-bit integer, split into a 3-bit class and a 5-bit detail. The class can indicate a request (0), a success response (2), a client error response (4) or a server error response (5). A code of 0.00 means an empty message. The "Message ID" is used to detect duplicates and to match packets of type Acknowledgement/Reset to packets of type Confirmable / Non-Confirmable. The "Token" values are used to pair requests to responses. This can be followed by zero or more "Options". If there is also a "Payload", then a prefix marker is present (0xFF), to separate the two last fields [3].

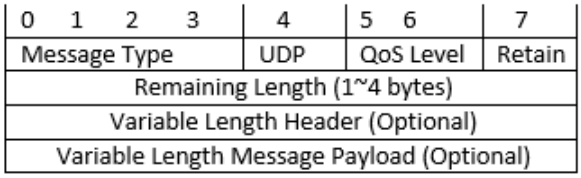

Fig. 1. CoAP Header

Some of the most important features provided by CoAP are resource observation, referring to subscriptions to interesting published data, block-wise resource transport (exchanging transceiver data with partial updates, to avoid overhead), security, together with DTLS, resource discovery, based on well-known URIs [2]. One of the most important advantages is that it can interact with HTTP, through proxies. It is possible to build intermediates that run CoAP on one side and HTTP on the other. Since there are equivalent methods, requests and response codes, it is straightforward to statically map between the two protocols. They both use URIs for identification: "coap://" and "http://". Reverse proxies can also be enabled, that make standard HTTP clients able to transparently access CoAP servers. Is is also possible to map a single HTTP request to a multicast CoAP request. The responses are aggregated into a single HTTP response and sent to the client [1].

Regarding the CoAP options, some of the most important are Block, Observe and Discovery. The first one is used in case of larger packets. Generally, CoAP messages from light or temperature sensors imply small payloads, but in case of updating firmware, larger packets are necessary. CoAP does not rely on IP fragmentation and it uses the Block option. This means that multiple packets are sent in request-response pairs. It enables a server to be completely stateless and handle each transfer separately. Observe is an option sent by clients in request. Instead of sending multiple GET messages to obtain information on a resource, they can specify this option and receive asynchronous messages every time it changes. In machine-to-machine environments, the devices should be able to discover resources, in a similar way to HTTP "/index/" pages. For this /.well-known/scheme" page is available. Resource descriptions are also standardized and they can be found at "/well-knows/core" URIs and can be accessed through GET packets [1].

B. MQTT

Message Queue Telemetry Transport (MQTT) is a messaging protocol that aims to connect embedded devices with applications and middleware, standardized in 2003, by OASIS [4]. It is built on top of the TCP layer and it is suitable for low resource devices. It consists of three elements, subscriber, publisher and broker. The publisher is the one that sends data and it forwards through the Broker through subscribers.

The broker also acts as a security mechanism, being able to authorize both entities. Since it uses little resources, it is commonly used in machine-to-machine messaging, in health care, monitoring or Facebook notifications [2].

The header of an MQTT message has a variable length, between 1 to 4 bytes. The first two bits are fixed, then the "Message Type" can get one of the following values: CONNECT(1), CONNACK(2), PUBLISH(3), SUBCRIBE(8) or others. The next field is the "DUP" flag. If it is set, then the message is a duplicate and the server may have received it earlier. There are three levels of "QoS" for PUBLISH messages that can be found in the header. The "Retain" field tells the broker to keep the message and send it to subscribers as a first message [2].

Every client, published or subscriber, sends a "CONNECT" message to the Broker to establish a connection. In order to keep the connection alive, the client must send periodical messages to the broker, either data or ping. In the "CONNECT" message, the client can send a "will" message, that would be published by the broker in case it doesn't get any keep alive messages from the client. They can send "PUBLISH" messages, containing a topic and a message, or they can send "SUBSCRIBE" messages to receive packets. Also, the clients can send "UNSUBSCRIBE" to stop receiving messages on a certain topic and they acknowledge every packet they send [5].

There are two major specifications, MQTT and MQTT-SN. The later was defined for sensor networks and was optimized for low bandwidth, battery operated devices and high link failures. There are several differences between the two implementations. First of all, in MQTT-SN the "CONNECT" message is split between one mandatory and two optional messages, used to transfer the "Will" topic and message to the server. In the "PUBLISH" message, the topic names were replaced with IDs. Before this, registration messages were sent to get, from the server, a unique ID for a specific topic. In this case, the clients should also be informed in advance by the corresponding IDs. Also, pre-defined topics or short names can be added, to skip registration. These are two byte long and they are known in advance by both clients and publishers. A discovery procedure was added for clients to get the network address of an operating gateway that communicates with the broker. "Will" topics and message are persistent in case of MQTT-SN, and the clients can modify them during a session.

If the devices are in sleep mode, the messages directed to them are stored at the server and sent when they wake up [6].

C. CoAP versus MQTT

Thangavel et al. [7] perform a comparison between the two protocols running in the same environment. The middleware is extensible, having support for existing and future application protocols, it provides a common API to access different functionalities and it is adaptive, so in the future it will be able to choose a running protocol based on the constraints. In the first test, the influence of packet loss on delay was considered. At a loss rate lower that 20%, MQTT has little delay, but in case of higher rates, CoAP performs better. This happens because CoAP runs over UDP, has its own retransmission scheme and avoids the overhead implied by TCP retransmissions. The second test computes the total amount of data transferred in case of different values for packet loss. In case of CoAP there is less data re-sent and the higher values are obtained in MQTT when the QoS level 2 is set, because it requires a four-way handshake. Another test was performed to compare the overhead for different message sizes. When the packet loss is low, CoAP generates less overhead, regardless the message size. In case of higher rates and larger packets, MQTT generates less overhead. This can be explained by the fact that there is a higher risk of losing large packets when using UDP than in case of TCP [7].

III. Service Discovery Protocols

In the Internet, the most extended discovery architecture is based on Domain Name Server (DNS), but this is not a suitable option for the Internet of Things, because IoT devices join or leave networks more often. So, two extensions have been developed, called DNS-SD (Service Discovery) and mDNS (multicast DNS).

The challenges that they must face, related to Internet of Things, are scalability, since there is an estimation that, in 2020, there will be almost 50 billion devices connected, dynamism, for wearable smart devices, that can easily switch connections, devices in sleep mode, leading to a limitation for queries addressed to endpoints and payload size, as less data is transferred.

Local directories should be extended to give information on other domains where resources are available. Also, directories should support multicast, for queries such as “turn off all lights in the room”, where all the endpoints are treated as one. The access to directories should be done in an already known manner and a common description should be used for services and attributes [8].

A. mDNS and DNS-SD

mDNS (RFC6762) [9] is a service that can perform the task of a DNS server. In a local network, every client has a cache where it keeps the pairing between names and addresses. Every time a device wants to get name information, it sends a multicast query and waits for a response. The targeted machine sends a multicast response in the network and all the devices that receive it save the pairing in their local cache. It has the advantage that there is no need for a dedicated server and it also adapts easily to changes in the network [2].

DNS-SD (RFC6763) [10] is a solution proposed by IETF ZeroConf WG that re-uses and extends the capabilities of the DNS. It uses the same types of queries (AAA, PTR) and enables locating and publishing services in a network [9]. For instance, clients that want to get printer services in the network use DNS-SD. They have firstly to get the names of the devices that provide the required service, then they use mDNS to get the address. It is essential to first find the hostname, because the IP addressed may change. The main drawback of these two protocols is the need to keep cache entries in devices with low memory. Bonjour and Avahi are two well-known implementations based on DNS.

B. uBonjour

uBonjour is a lightweight service that combines mDNS and DNS-SD for addressing and discovering services available in a network. It provides standardized discovery and self-configuration without any hardcoded addresses, both necessary in IoT. It implements the standards defined by DNS protocols and systems are able to discover each other without any gateway. When an IoT node requests for a service, it is added in the database and they are deleted when getting a “service unavailable” message [11].

Resolving hostnames is based on mDNS, meaning that a multicast request is sent. The device responses with an A record, containing the address. This message will be broadcasted to all listening devices. In case of services, a multicast PTR record is sent. If the service name is found, then a broadcast message containing the address and the port is sent. Otherwise, a timeout is triggered [11].

In order to publish a service, an IoT node has to send four standard DNS records and each application has to register an address, a hostname and a port. To remove a service, the device has to send a PTR record having the TTL set to 0. Updating a service implies re-sending the four records with new data. Eight service registration are supported per device [11].

Further optimizations can be obtained by minimizing data traffic, by using two methods. The first one, Known Answer Suppression. If a node already has an older response to a query and it wants to get newer information, it sends a query that contains not only the question, but also the data it already has. The responder doesn't send anything if the correct information is already in the query. Otherwise, it sends updates. The other is Duplicate Question Suppression. If a host wants to send a query and it sees that another host is sending the same request, then it should wait for the broadcast response [12].

Another optimization is One-Way Traffic, which puts a device in passive mode. It publishes services periodically and responds to incoming requests. So, it avoids active resolving of hostnames and parsing queries from other devices.

IV. Infrastructure Protocols

At the physical and media access layer, the standardized protocols are designed for different needs. Part of them are

specialized for local networks, which require low distances and low power. They are addressed especially to smart buildings or homes. Others can offer a high range, being able to supply the needs of smart cities or industrial environments.

A. IEEE 802.15.4

The IEEE 802.15.4 [13] standard was especially designed for low-power, short-range and low-bit rate embedded devices. The IEEE 802.15 working groups aims to keep the hardware costs low, to spread the protocol compatibility among sensors. The protocol describes the physical layer and the media access control sublayer [14].

The physical layer is responsible with activation and deactivation of the radio transceiver, receiving and transmitting data and selecting a channel and listening on it. At this layer, there are two supported frequencies. The low-band (868/915 MHz), that uses binary phase shift key modulation and the high band (2.4GHz) that uses offset quadrature phase shift keying modulation. There are 27 available channels, 11 in low band and 16 in high band. The raw bit rates are 20-40kbps in low band and 250kbps in high band. By using these modulations and spreading techniques, the transmissions are robust and noise resistant [15].

At the MAC layer, CSMA/CA is enabled, together with optional time slot structure and security. Every time a device wants to send data, the medium is checked by the PHY layer. If it is occupied, the transmission is postponed for a certain period of time [14].

The transmission units are called Physical Protocol Data Unit for Layer 1 and MAC Protocol Data Unit. The PPDU header has two fields, then the payload is added. The SYNC field is used for clock synchronization and the PHY field contains the length of the payload. In the MAC frame, the first two bytes indicate the type of the frame [14]. There are four types of frames defined in the standard. The first ones, beacon frames are used by coordinators to describe the way the channels can be accessed. Then, there are data frames and acknowledgements sent as responses for control and data packets. The last, control frames are used for network management, such as associations or disassociations [15]. The next flag, the sequence number, is used by acknowledgements and refers to the received frame. Then, there are addressing pieces of information, related to source, destination addresses and security. The last two bytes represent the checksum, used for data integrity.

The devices are classified into two categories: full function (FFD) and reduced function devices (RFD). A RFD can communicate only with an FFD, while an FFD can communicate with both types. The latter have, generally, the role of coordinators. They keep a routing table and are responsible with network creation and maintenance. The standard topologies in 802.15.4 include star, with a central FFD and several RFDs that communicate with it, peer-to-peer (mesh), where there is a coordinator, several FFDs and RFDs and cluster, having a coordinator, a FFD and several RFDs [2].

An interesting situation happens when 802.11 and 802.15.4 devices run in the same area and they use the 2.4 GHz frequency. Two situations were analyzed. The first case is the one when 802.11b transmissions interfere with 802.15.4 communications. The degradation is higher if the frequencies are not shifted with 7MHz. Also, there are more errors for larger packets. In the second case, wireless 802.11b/g transmissions are interfered. Here, there are errors when the WiFi packets are larger than 600bytes and the offset between central frequencies is 2MHz [15].

Security is provided at the MAC layer, using AES-128 with CCM mode of operation. It supports 8 levels of security, where 0 means unsecured, levels 1 to 3 provide only integrity and authentication, while levels 4 to 7 also provide data confidentiality [13].

B. 6LoWPAN

6LowPAN is developed by an IETF working group especially for small networks and embedded devices interconnected by IEEE 802.15.4 in RFC4944 [16]. It maintains an IPv6 network, but with compressed headers. A new layer is added, between the network and data link, responsible with fragmentation, reassembly, header compression and data link layer routing for multi hop [16].

The first byte of the encapsulation header identifies the next header. The first three bits are used for indication, while the others have different purposes, depending on the header type. If they are 00x, then this is not a 6LoWPAN frame and are used if case of protocols co-existence. 010 means Uncompressed or HC1 compressed and the type of the address can be determined by the last 5 bits. 10x means that the next header is a mesh header and the next 5 bits are used for routing. 11x is a fragmentation header [16].

HC1 is the main compression technique defined in RFC4944 [16]. It is used for packets containing link layer addresses and it removes the common fields, such as Version, TC, Flow label, it doesn't send the link layer addresses (which can be computed from 802.15.4 header), and the standard prefix length. The next header field is limited to TCP, UDP and ICMP [16]. RFC 6282 adds two new compression techniques, LOWPAN_IPHC and LOWPAN_NHC. The first one uses 13 bits for compression. It removes the Version field and compresses Traffic Class and Flow Label into 2 bits. Hop Limit has 2 bits allocated, assuming that is set to either 1, 64 or 255. The global IPv6 addresses are also transformed and can be determined using Source Address Compression (SAC) and Destination Address Compression (DAC) bits, together with Source Address Mode (SAM) [17].

The standard does not specify any security mechanisms and it uses the MAC layer options in the 802.15.4 protocol [16].

C. LoRaWAN

Since a part of the technologies developed for IoT focus on nearby devices, LoRaWAN is a solution designed for large distances. It is suitable for smart cities or smart agriculture projects, where applications need to send little amount of data over large distances. The protocol consists of two parts, LoRA, which specifies the physical layer able to create long communication links and LoRaWAN, which defines the system architecture for the network [18].

At the physical level, it uses chirp spread spectrum, a sinusoidal signal whose frequency increases and decreases over time. It has the same features that FSK modulation provides, but it gives the advantage of larger distances [18].

Regarding the network architecture, there are four types of devices: nodes, gateways, network server and applications server. The nodes are not associated to a specific gateway, they send data which can be received by multiple concentrators. Each of them will forward the received packets to a network server in the cloud and is responsible with filtering duplicate packets, performing security checks and send acknowledgements through the optimal gateway. The nodes are asynchronous and they wake up whenever they have data to send or when they are scheduled. They do not need to keep an internal clock synchronized with the network, giving the advantage of larger battery lifetime. LoRa is based on spread-spectrum modulation, so the signals are orthogonal on each other when using different spreading factors. The gateways have a multi-modem transceiver incorporated, so they can listen to data on multiple channels at once, and so be able to adapt to a large number of nodes. The devices close to the gateways do not switch to the lowest data rate, they shift to higher values to fill up the space, send faster and leave more space for the others to transmit [18].

There are three types of devices, Class A (battery powered sensors), Class B (battery powered actuators) and Class C (main powered actuators). Class A devices are the most energy efficient, they can send data and then have two short receiving windows, when they wait for messages. If the server doesn't respond in that period of time, then it has to wait until it gets another message from the device. Class B devices have an extra configurable time slot when they can receive information. In order to wake up, the gateway sends a synchronization beacon. Class C devices can get data anytime, except the period of time when they are transmitting [18].

LoRaWAN integrates two layers of security, one at the network and the other at the application layer. The first one ensures authentication, while the other encrypts data so the network operator cannot access the user's application data. Both use AES with a key length of 128 bits [18].

V. Integration between application protocols

At a large scale, in IoT, devices can be divided into two categories: resource-rich, the ones that support the TCP/IP stack and constrained devices. The first are the ones that support applications developed on top of CoAP, MQTT, REST, AMQP and other. Furthermore, microcontroller-based appliances, for instance, should have the capability to communicate with “smart” devices.

At application level, there are several solutions implemented. One of them is Ponte, developed by Eclipe IoT group. Its aim is to create a bridge between CoAP, MQTT and HTTP. It exposes a REST compatible API and it is able to convert between several data formats, such as JSON, XML or Byson. It works as a gateway between CoAP and HTTP, which use the same data format and as a broker for MQTT.

Another Eclipse project is Franca, build for the automotive industry, but can be extended to IoT environments. Its aim is to integrate software from different suppliers, using various frameworks, platforms and Inter Process Communication tools. It has the role of a hub, translating the code into another language. It contains interface description languages and an editor, code generation mechanisms, specification of dynamic behavior between clients and servers and rapid interface prototyping.

For interface definition, Eclipse also provides Vorto, a tool that keeps meta information models and provides code generation. Device manufactures can create a repository, using a modeling framework provided by Eclipse, where they can add information on provided functionalities. A developer can access the repositories, invoke the code generation tool and create the requests.

Light-weight M2M is a device management protocol, which provides a unified way of managing devices remotely. The current implementation is based on CoAP and uses DTLS for security. It defines an architecture using REST objects. Even though it may apply to Wireless Sensor Networks or Cellular devices, for now it is only compatible with IP [19].

Another approach is virtualization. An IoT-Virtual Network is created, where all devices are included, even the resource constrained ones. It can be established on top of layer 3 for objected connected to the Internet, or over layer 2, for constrained sensors. Inside, applications and services see only the logical layer. This can be used when partitioning a wireless sensor network. If the devices are maintained by different administrators, they can be divided into multiple virtual networks and only part of them will be accessible. This technique can also be used for aggregating separate sensor networks. In this case, they can be connected using a Layer 3 tunnel and enabling secure communication. Also, a WSN can be extended with unconstrained devices, such as servers in cloud that gather data [20].

VI. Use case: Smart Home

An interesting deployment for the Internet of Things is a smart home scenario that besides the home monitoring system also include a smart robot assistant. This system has the main purpose to improve the quality of life by assisting the user in his daily life.

The sensing capabilities of the home monitoring system can be represented by IoT enabled low-power wireless sensor nodes. The nodes use standardized WSN protocols and connect to the residence’s gateway. Using the IoT paradigm allows us to decouple the sensing capabilities of the system from the processing component. The processing can thus be done on-site, for privacy conscious users, or in the cloud, for added flexibility, represented by the almost unlimited storage and processing capabilities and interaction with other services.

Having an Internet mediated system allows various options for interacting with the environment and closing the feedback loop. Processed data can be equally easy to access from on-site servers or from the cloud by various assistant apps. This could go as far as having robot assistants roaming the house that help

users in their daily lives [21]. The system can learn user habits based on sensed data and dispatch the robot assistants to assist with various tasks: waking up in the morning, presenting the weather and traffic on work days, showing missed house events in the evening and selecting cooking recipes for dinner.

The robot assistants are themselves IoT enabled devices, which means they can serve the user remotely, if desired, or can interact directly with other IoT devices in the house (e.g. sensor nodes, appliances, user gadgets). In the cloud based system, the assistants could still function even if Internet connectivity is limited, by directly interacting with other devices in the house, instead of going to the processing hub for information. This situation will provide less functionality than the full system, but will still allow users to make basic queries to the system.

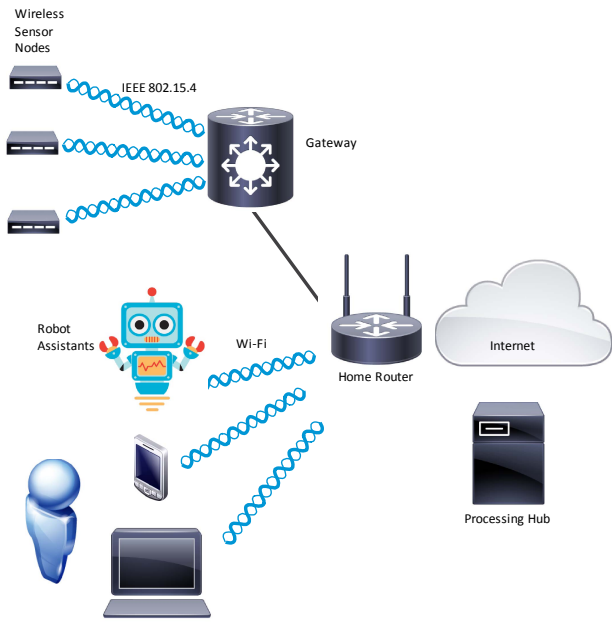

Fig. 2. Smart Home System

The main components of the smart home system (sensor nodes, gateway, robot assistants, processing hub) use standardized protocols to communicate with each other. All system components are addressable through IPv6, over the Internet.

The sensor nodes and gateway run a networking stack that includes IEEE 802.15.4, 6LoWPAN and CoAP. These standardized protocols have been especially designed for resource-constrained devices, as specified in Section II and IV. The sensor nodes are interrogated by the processing hub through CoAP requests and send the collected data through CoAP replies.

In addition, the gateway is able to translate from 6LoWPAN to IPv6 and vice versa, in order to enable the communication between the sensor nodes and the server. This allows the integration between 6LoWPAN and IPv6 networks.

The robot assistants run a full TCP/IP stack and communicate directly with Internet servers in order to obtain information for the user. They interrogate the processing hub through HTTPS and receive data in JSON format.

However, assistants are also capable of communicating with the sensor nodes by sending CoAP requests. This is done for obtaining raw data very fast, for example when a robot assistant wants to find out the position of the user.

Therefore, the proposed smart home system integrates cutting-edge technology such as robot assistants and wireless sensor nodes, and uses only standardized protocols for the communication between its components.

VII. Conclusions

The idea of Internet of Things is rapidly growing and becoming part of modern human life. Its aim is to improve the life quality, by automation, connecting devices, applications and publish more information fast.

This survey provides a brief overview of several standardized technologies developed especially for embedded devices and constrained environments. We have presented application protocols and a comparison between two of them, service discovery and the way DNS was extended for networks that change fast and physical layer protocols developed for low-range and long transmissions.

Also, an IoT scenario where these protocols are used together with TCP/IP technologies was provided. We proposed a smart home scenario based on sensor nodes and robot assistants that uses only standardized protocols to enable the communication between different components.

References

[1] Z. Shelby, K. Hartke, and C. Bormann, “RFC7252 - The constrained application protocol (CoAP)”, *Internet Engineering Task Force (IETF)*, 2014.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications.”, *IEEE Communications Surveys & Tutorials*, vol. 17(4), pp. 2347-2376, 2015.

[3] C. Bormann, A. P. Castellani and Z. Shelby, “Coap: An application protocol for billions of tiny internet nodes”, *IEEE Internet Computing*, vol. 16(2), pp. 62-67, 2012.

[4] D. Locke, “Mq telemetry transport (mqtt) v3. 1 protocol specification”, *IBM developerWorks Technical Library*, 2010.

[5] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-s a publish/subscribe protocol for wireless sensor networks” in *Communication Systems Software and Middleware and Workshops*, pp. 791–798, 2008.

[6] A. Stanford-Clark, and H. L. Truong. "MQTT for sensor networks (MQTT-S) protocol specification." *International Business Machines Corporation version 1*, 2008.

[7] D. Thangavel, X. Ma, A. Valera, H. X. Tan and C. K. Y. Tan, “Performance evaluation of MQTT and CoAP via a common middleware” in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), IEEE Ninth International Conference*, p. 1-6, 2014.

[8] A. J. Jara, P. Martinez-Julia and A. Skarmeta, “Light-weight multicast DNS and DNS-SD (lmDNS-SD): IPv6-based resource and service discovery for the Web of Things” in *Innovative mobile and internet services in ubiquitous computing (IMIS),* pp. 731-738, 2012.

[9] S. Cheshire and M. Krochmal. "RFC 6762: Multicast DNS." *Internet Engineering Task Force (IETF)*, 2013.

[10] S. Cheshire and M. Krochmal, “Rfc 6763, dns-based service discovery”, *Internet Engineering Task Force*, 2013.

[11] R. Klauck and K. Michael "Bonjour contiki: A case study of a DNS-based discovery service for the internet of things." *International Conference on Ad-Hoc Networks and Wireless*, Springer Berlin Heidelberg, 2012.

[12] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. V. D. Abeele,E. D. Poorter, P. Demeester et. al. “IETF standardization in the field of the internet of things (IoT): a survey”, *Journal of Sensor and Actuator Networks*, vol. 2(2), pp. 235-287, 2013.

[13] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “RFC 4944. Transmission of IPv6 packets over IEEE”, 802(4), 2007.

[14] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella, “Performance study of IEEE 802.15. 4 using measurements and simulations” in *Wireless communications and networking conference*, 2006. pp. 487-492.

[15] S. M. Sajjad and M. Yousaf, “Security analysis of IEEE 802.15.4 MAC in the context of Internet of Things (IoT)” in *Conference on Information Assurance and Cyber Security (CIACS)*, 2014, pp. 9–14.

[16] N. Kushalanger, G. Montenegro and C. Schumacher, “IETF RFC 4919, 6LoWPAN: overview, assumptions, problem statement, and goals”, 2007

[17] J. Hui and P. Thubert, “Rfc 6282 compression format for ipv6 datagrams over ieee 802.15. 4-based networks”, 2011.

[18] LoRa Allicance, “LoRaWAN. What is it? A technical overview of LoRa and LoRaWAN Institution”, 2015.

[19] H. M. Oen, "Interoperability at the Application Layer in the Internet of Things", Master's Thesis, NTNU, 2015.

[20] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, “Internet of things virtual networks: Bringing network virtualization to resource-constrained devices” in Green Computing and Communications (GreenCom),” 2012, pp. 293-300.

[21] A.M. Stanescu, A. Nita, M.A. Moisescu and I.S. Sacala, "From industrial robotics towards intelligent robotic systems," 2008 4th International IEEE Conference Intelligent Systems, Varna, 2008, pp. 6-73-6-77.