WISE: MQTT-based Protocol for IP Device Provisioning and Abstraction in IoT Solutions

Istvan Papp[1, 2, *], Roman Pavlovic[2], Marija Antic[1, 2]
[1]*Faculty of Technical Sciences, University of Novi Sad,*
Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia
[2]*OBLO Living LLC,*
Narodnog fronta 21a, 21000 Novi Sad, Serbia
istvan.papp@rt-rk.uns.ac.rs

***Abstract*—Although numerous consumer devices use Transmission Control Protocol/Internet Protocol (TCP/IP) protocol stack to connect and communicate over the Internet, their integration into a single Internet of Things (IoT) solution represents a challenge, primarily due to the lack of the standardized interoperability framework on the application layer. Devices produced by different manufacturers cannot operate together out of the box, thus raising the cost of system setup and maintenance. In this paper, a novel protocol is proposed that aims to bridge this gap. It is based on Message Queuing Telemetry Transport (MQTT) protocol and allows the seamless integration of different kinds of IP devices into the connected system. The proposed protocol is complete as it covers the aspects of device discovery and association in the IoT network. It provides mechanisms for IoT network maintenance and defines the abstract device model and communication patterns to enable system-wide device interoperability. The other goal of the protocol is to be portable to resource-constrained platforms. To validate the proposed protocol, it was integrated into the existing smart home hub, and for testing and validation purposes, prototype devices were developed.**

***Index Terms*—Internet of Things; Protocol; Interoperability.**

I. INTRODUCTION

Complex IoT solutions, such as smart homes, fitness and health tracking systems, or assisted living systems, usually comprise of a number of heterogeneous devices. To achieve the desired system behaviour, these devices need to work together, i.e., to form the network and exchange the application-level data [1] without any additional help. This is not the case today. There is already a number of specialized wireless IoT protocol stacks, which are adopted by consumer electronics manufacturers, and which provide the device abstraction on the application layer, define the uniform representation of device capabilities, and the standardized format of messages exchanged between the devices [2]–[4]. The standardized communication and messaging approach allows developers to focus on the system-level logic and provide advanced features, such as automation rules or context-aware reactions. From the user perspective, the existing standards allow the interoperability of devices produced by different manufacturers and lower the cost of the system setup.

However, the widely used IoT protocols are often optimized for low power consumption, providing data rates that are not sufficient for devices, such as cameras, smart speakers, or media players. Therefore, there is also a multitude of smart devices available on the market, which use the Transmission Control Protocol/Internet protocol (TCP/IP) communication stack for achieving higher bandwidth. Additionally, some device developers choose to use IP due to the availability of existing home or corporate infrastructure, communication modules, and their familiarity with the technology. Mainly, such solutions rely on the WiFi connection to communicate with the rest of the network, as it requires no network cabling and provides the mobility of the end device. We will refer to these devices as IP devices.

Although the IP communication stack is well known, the semantics related to IoT is missing: there is no uniform device abstraction defined, and that impacts both the device manufacturers and the end users. Namely, all manufacturers have to define and implement their own messaging Application Programming Interfaces (APIs) and device abstractions, which results in highly specialized and proprietary solutions [5]: some manufacturers focus only on cameras and video surveillance, others on air conditioning, multimedia, alarm systems, etc. As a consequence, users have to install and use multiple systems of limited functionality in parallel, which cannot work together to provide a unique user experience.

The technology presented in this paper aims to bridge this gap by providing a scalable, universal protocol, covering the entire life cycle of a connected device.

The protocol can be used for the interaction between an IP end-point device (referred to as IP device) and an IoT hub (referred as hub). That relation can be even replicated to reach a multilevel hierarchy, as an end-device can also act as a hub for subordinate end-devices. We will refer to the proposed protocol as WISE. This protocol goes beyond the level of being just a communication protocol. Besides

Manuscript received 31 December, 2020; accepted 30 March, 2021.
This research has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under Grant No. 451-03-68/2020-14/200156 (Project title: "Innovative scientific and artistic research from the FTS (activity) domain").

addressing the obvious pain points of device vendors and integrators (design, interoperability), it also covers the common features of similar IoT technologies: device network provisioning, device inclusion into (or exclusion from) the IoT network, IoT network management, the interaction model, and the capabilities of IP devices. It also aims at low power consumption and small communication overhead, so it can be suitable for battery-powered devices. WISE defines the hub in a way that it can reside both in the local network and in the cloud. Scalability and information protection were built into the foundations of WISE. That results in a protocol that is:

– Complete - it addresses all usability aspects of a device, not only communication.

– Lightweight - it can run both on bare-metal, resource-constrained microcontrollers, as well as on desktop/server platforms. It imposes almost no translation overhead compared to Zigbee/Z-Wave and Bluetooth Low Energy (BLE) as the data model is similar to them, therefore the bridge devices can be very lightweight.

– Extensible - the data model is always backward compatible and can be extended to support future devices and their new services. The data model resembles generics of the data models used in 802.15.4-based protocols.

– Scalable - scalability support is built into the protocol to enable forming large-scale IoT systems.

– Versatile - it is designed having on mind future technologies and needs: it works well with IPv6, NB-IoT/5G.

– Efficient - power efficiency and data-bandwidth efficiency are supported in a way that battery powered WISE devices will sleep as much as possible without ruining the user experience, preserving the battery at the same time.

– Secure - the protocol uses secure MQTT and secure HyperText Transfer Protocol (HTTP) to ensure data encryption.

– Easily integrated - use of open standards and technologies to promote adoption by device vendors.

The proposed protocol is field-tested: it is integrated into the existing smart home hub [6] and prototype devices. For the device vendors, a Software Development Kit (SDK) is created to allow developers to easily create WISE compatible devices.

II. RELATED WORKS

Various IP-based IoT devices have been proposed by different authors, such as the indoor air quality detector [7], temperature control system [8], smart plug [9], infusion monitoring system [10], or various multisensors [11]. As some of these devices require mobility, they must be battery-powered, and their battery lifespan is an issue that needs to be addressed accordingly. While thermoelectric energy harvesting was used to build autonomous WiFi sensors for a heating system [12], it has also been shown that multiple years of battery lifetime are possible for WiFi sensors, in the case of periodic data transmission at high data rates [13]. This corresponds with the principle of device hibernation already supported by the specialized IoT technologies [2]–[4].

To make an IP device a functional part of an IoT solution, the application level logic needs to be created, which allows the interoperability of different endpoints [14]–[17]. With this in mind, the design of the home automation system has been proposed [18], which combines WiFi actuators and Zigbee sensors with multiple master nodes supporting both communication technologies. On the other hand, some of the authors propose methods for device discovery in the local network and simple control using MQTT messages [19], while others build the cloud-based WiFi sensor management system for home automation, based on Constrained Application Protocol (CoAP) protocol [20]. However, these authors focus mainly on the device discovery procedure and network association, while only the basics of device abstraction are covered.

The protocol used to advertise device capabilities must consider the implementation in a very constrained environment (low memory, processing and power consumption) [21], [22]. Some authors opt to use User Datagram Protocol (UDP) for the communication between the WiFi sensors [23] to reduce the size of the messages exchanged, but the proposed solution models device functionalities by using code words of the predefined length, and is therefore not easily extensible. While the UPnP protocol provides mechanisms that could serve as a base for the more flexible IP device abstraction, its communication overhead is not suitable for IoT applications. It has been shown that MQTT and CoAP are more suitable for IoT purposes [24]. A significant effort has been made in [25] to provide service abstraction using oneM2M reference architecture and control system via MQTT messages. However, device hibernation is not considered, while the proposed MQTT topic scheme depends on the IP addresses of devices in the system and is not suitable for the cases in which the IP address changes dynamically. Additionally, oneM2M requires an interworking proxy to connect with the legacy IoT products and technologies [26]. Other authors suggest using GraphQL to abstract device functionalities [27], or propose a data-driven agent-based abstraction layer, which aims to minimize the overhead of representing the information generated by devices and services [28].

As mentioned, many IoT devices use 802.15.4-based communication standards. To allow those devices to be exposed to the Internet, a bridge device is required that mediates between the two worlds. In the past, several protocols were proposed that would bring the 802.15.4 world closer to the Internet: IPv6, 6LoWPAN, and its successor Thread achieved only a modest success and did not jeopardize the domination of Zigbee and Z-Wave in home automation. The latest attempt is Connected Home over IP (CHIP) [29] initiative started in late 2019. Although CHIP is supported by major companies, it is still in the definition and early development phase, without clear indications of the expected timeline.

As previously pointed out, the existing protocols focus on the interoperability of different devices, but do not address the problems of IoT network setup and maintenance. The

proposed WISE protocol aims to provide the complete solution and covers both the networking aspects and the device life cycle. We believe the completeness and scalability of WISE protocol is a significant enabler for wide adoption.

In the rest of this paper, the WISE protocol stack is introduced, and the supported message exchange patterns are defined. Later, details regarding device association and network maintenance are provided, as well as the proposed device abstraction. Finally, the existing smart home hub is extended to support the proposed technology, and the prototype implementation of the WISE device is created for testing purposes.

III. WISE OVERVIEW

As other specialized IoT technologies, the proposed WISE protocol should cover the aspects of logical IoT network establishment and maintenance, message routing, and device abstraction. Since it is designed to enable the integration of IP devices into the IoT system, the communication must rely on the protocols of the TCP/IP stack, and WISE adds a layer on top of them (Fig. 1). Depending on the architecture of the system with WISE, Simple Service Discovery Protocol (SSDP) or HTTP can be used for device discovery and association. Network monitoring and communication between parties in the system is based on MQTT protocol, with the topic structure and message payloads as defined in the sections that follow. The sensitive payload is accordingly encrypted: MQTT uses Transport Layer Security (TLS), while HTTP uses Secure Sockets Layer (SSL).

Logically, the network implementing WISE protocol has a topology of a star, i.e., WISE puts the IoT hub in the centre of the logical IoT network, and the hub acts as the WISE network controller. The role of the IoT hub is to implement advanced logic, such as if-this-then-that relationships between changes in the system, grouping of devices by their location or type, system configuration, perform functional processing, etc. In addition, it serves as the interface between client applications and the end devices.

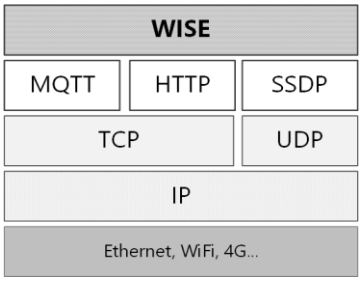

Fig. 1. WISE protocol stack.

The IoT hub can reside in the same local network as the IP devices it controls, as depicted in Fig. 2. In this case, besides all aforementioned roles, it can also serve as the gateway between different local IoT networking technologies. In the case of the WISE network, the hub also runs the MQTT broker. The main advantage of this approach is the fact that the Wide Area Network (WAN) connection is not required for the normal system operation, as long as the Local Area Network (LAN) network is operational.

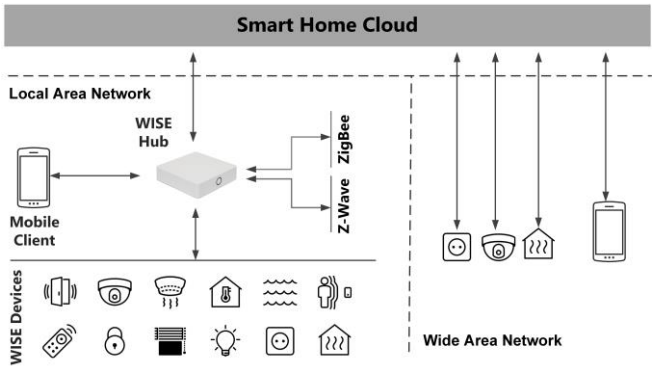

Fig. 2. WISE IoT system with the hub in the local network.

On the other hand, the IoT hub can be placed in the IoT cloud, as shown in Fig. 3. In this case, it is necessary to move all of the control logic to a virtual hub process in the cloud, while the MQTT broker instance can be shared by all virtual hubs. The WAN connection is required for the system to operate, but it is not required that all connected devices are at the same location.

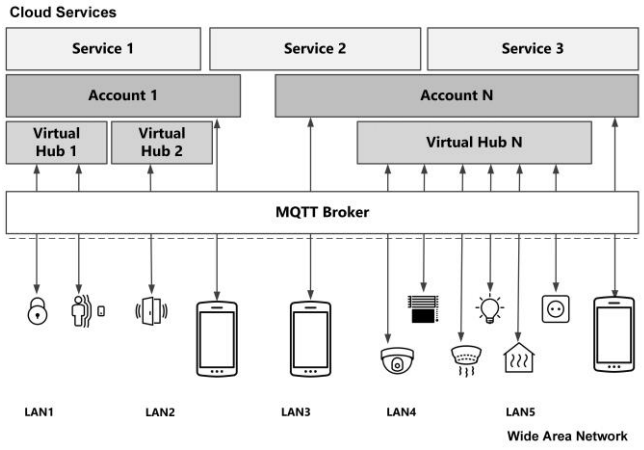

Fig. 3. WISE IoT system with the hub in the cloud.

IV. WISE MESSAGING MODEL

The messaging between the devices in the WISE IoT system is based on MQTT protocol. The WISE layer defines the MQTT topic structure and message exchange patterns within the system.

A. Topic Structure

The topics used in the WISE network must comply with the predefined structure, which allows identifying the parties performing the communication and their functionalities. The base topic model consists of three parts: WISE identifier *W*, service identifier *S*, and the message type m_t, as represented in Fig. 4. These elements must be in the exact order, $W/S/m_t$, separated by the topic level separator.

The WISE identifier *W* uniquely addresses a device or hub in the system, and it is mandatory. It consists of the following:

– *Domain ID*: represents the unique name of the IoT system provider in question, e.g., the hub and IP device manufacturer.

– *Home ID*: identifies a group of IP devices that interact only among each other and a hub, and form the IoT system controlled by a particular user.
– *Object role*: identifies the type of party participating in the exchange (e.g., hub, IP device, etc.).
– *Object ID*: the unique identifier of the particular instance of the device with the defined role.

Service identifier *S* is an optional identifier used to address different functionalities or services of an object. It carries the following information (which will be explained in Section VIII):

– *Service group*: represents an identifier of a group of device functionalities (e.g., temperature, light).
– *Service type*: identifies the type of functionality (e.g., light intensity, light color, etc.).

The message type is the last part of the topic, and it is a mandatory identifier of one of the possible message types: request, response, event, or status. Different message types are used to enable both synchronous and asynchronous communication between the parties in the data exchange, as explained in the next section.

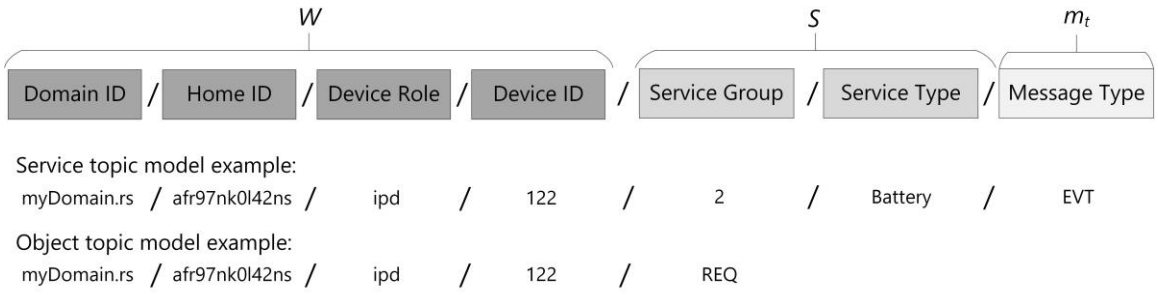

Fig. 4. WISE protocol topic structure.

B. Message Exchange

The parties in the system can exchange the data either synchronously or asynchronously. The synchronous communication is used when one of the parties in the exchange attempts to retrieve data or wants to execute a command and be informed about its execution status. On the other hand, the asynchronous message exchange is used to inform one or multiple subscribers to the MQTT topic about the change in the system.

The latest version (version 5) of the MQTT protocol specification does support the synchronous data exchange in the form of requests and responses, but the older versions of the protocol (3.1.1 and earlier) did not support it, and most of the implementations are not yet compatible with the latest standard. Therefore, WISE protocol defines its own request/response pattern. The synchronous message exchange begins by the node *A* sending a message to the request topic of node *B*. This can be either the WISE object request topic W_B/req, or the service topic $W_B/S/req$. The WISE identifier W_A of the message sender should be specified in the payload of the MQTT message in the sender field (Fig. 5). The receiving party *B* then responds to the response topic of the sender, W_A/rsp.

The information about the changes in the system is transmitted asynchronously, using the event topic of the sender, either the object event topic *W/evt* or the service event topic *W/S/evt*, which the interested parties are subscribed to. For example, the change of the light intensity of a bulb can be reported by an event sent to the topic associated with that particular service of that particular bulb. Additionally, for network monitoring purposes, the special status topic is defined, *W/sts*. This topic is used to asynchronously transmit information about the status of the device (online or not).

C. Message Payloads

Although the MQTT protocol does not impose any limitations regarding the content type of the message, payload, WISE protocol will require that message payloads are formatted as JSON, as this is the native format for cloud applications, and satisfies the low overhead requirement, while maintaining human readability. The payloads of various message types are represented in Fig. 5.

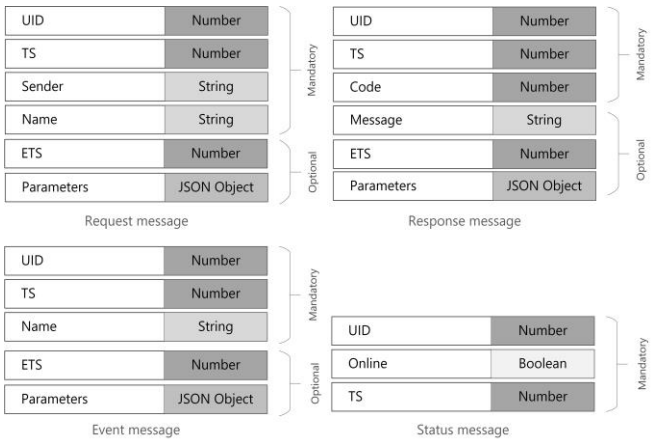

Fig. 5. WISE message payloads.

The field UID represents the unique identifier of the MQTT message exchange session. While every event message is assigned a new UID, in case of the synchronous data exchange, both the request and the response message carry the same UID. The field TS represents the timestamp of the message generation, while the ETS is the expiration timestamp. The receiving side should drop the message if it is received or set to process after the expiration timestamp.

The sender field of the request message is used to identify the response topic, as explained in Section IV-B. The

mandatory name field determines the actual semantics of the request and determines how the optional message parameters are further processed.

The response messages must contain the response code and may contain an optional text message explaining the result. In addition, optional parameters are allowed to carry the data queried by the request.

The event name specifies the change, which occurred in the system (device status changed, device property changed, etc.), while the optional parameters carry the additional information required to process the event by the subscribers (e.g., the new status, or the new value of the property). Status messages are similar to event messages, but they have the dedicated topic and only carry the information about the device online status.

V. Device Association and De-Association

Although an IP device may be connected to the Internet, it does not become a fully functional member of an IoT system, until it joins the logical network of a hub. The process during which a device joins the logical network of a hub is called "association", while the inverse process is referred to as the de-association of the device.

A. Association

The association process consists of three stages: discovery, inclusion, and authentication. In the discovery stage, the IP device scans for the available WISE networks. In the inclusion phase, it sends the association request to the hub using the temporary MQTT connection, while in the authentication stage, it establishes the permanent connection with the WISE hub.

From the perspective of the association process, the hub can be in one of the following states:

– *Idle*: This is the initial state, and the hub is not accepting association requests.

– *Advertising*: The hub remains in this state during the discovery and inclusion stages. If the association process is not completed within the configured interval (60–300 seconds), the hub returns to the idle state.

– *Joined*: The inclusion stage is finished, and the hub accepted the IP device to the network. After remaining in this state for a short period of time (3 seconds), the hub returns to the idle state.

On the other hand, the IP device can be in one of the following states:

– *Idle*: This is the initial state, and the IP device is not scanning for networks nor sending association requests.

– *Joining*: The IP device remains in this state during the discovery stage and is trying to locate the hub and to initiate the MQTT message exchange.

– *Accepted*: The IP device discovered the hub of interest and proceeds to the inclusion stage.

– *Rejected*: The IP device has not discovered the hub of interest and does not want to continue with the association process. After the configured association timeout expires, the device will return to the idle state.

– *Joined*: The hub accepted the IP device association request and assigned it an object ID. The device is now included in the network.

1. Discovery stage

The discovery process depends on the location of the hub. If the IP device and the hub are in the same LAN network, SSDP protocol is used to transfer the relevant information.

As depicted in Fig. 6, the association process is initiated by the user, preferably by pressing the dedicated buttons on the device and the hub. This action brings the IP device to the joining state, while the hub enters the advertising state. When switched to the joining state, the IP device must start listening on NOTIFY ALIVE messages, which are sent by the hub. The IP device may also actively scan for the network by sending the M-SEARCH message to the hub and processing the response M-SEARCH messages.

Both NOTIFY ALIVE messages and M-SEARCH responses contain information about the location of the hub description, i.e., the configuration data of interest for establishing the local IoT network. The device description of the hub must be implemented according to the UPnP device schema. The most important part of the description data is the path to the MQTT broker instance, i.e., the broker URL and the port it listens on. The rest of the mandatory data represents the information needed to build the desired MQTT topics: domain ID, home ID, and hub ID.

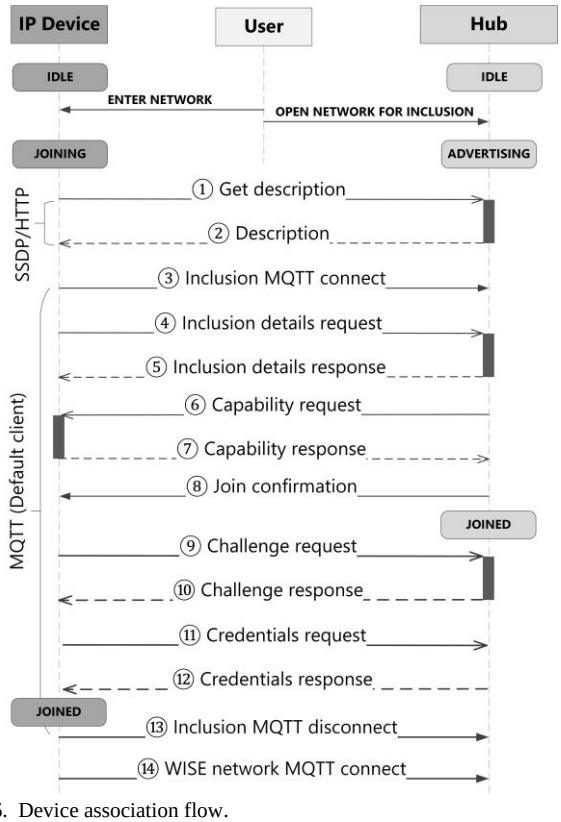

Fig. 6. Device association flow.

If the WISE hub resides in the cloud, HTTP is used to fetch the hub description data from a predefined location. The IP device is brought to the joining state by the user, and it uses the HTTP GET request to obtain the description data in the JSON body of the HTTP response message.

2. Inclusion stage

Once the hub description is fetched by the IP device and the parameters of the MQTT broker are known, MQTT protocol will be used for further communication between the device and the hub. The inclusion MQTT connection is established using the default client name and password. This connection is temporary and closes when the hub returns to the idle state. For the default client, the hub allows only requests to the topic W_H/req, where W_H is the WISE identifier of the hub, as explained in Section V-A. During the inclusion stage, the IP device uses its MAC address as the object ID part of the topic, i.e., it represents itself with the initial WISE identifier W_D^{init}.

First, the device sends the inclusion details request to the hub - Fig. 6. The parameters of the message represent the information about IP device manufacturer, model, version, supported WISE protocol versions, etc. The hub responds by either allowing the inclusion or rejecting it based on one of the reasons (device model/version/manufacturer not supported, version of WISE protocol not supported, inclusion process not started on the hub, etc.). If the inclusion is allowed, the temporary inclusion ID parameter is created to track the rest of the inclusion process for the device in question. The hub may request additional information from the device by sending the capability request to it. Finally, the hub informs the device about the assigned object ID by sending the join confirmation.

3. Authentication challenge stage

In this stage, the IP device fetches the MQTT credentials assigned to it by the hub. These credentials will be used to establish a WISE network MQTT connection between the two parties and enable the normal operation of the device. This process consists of two steps. First, the IP device obtains the challenge code from the hub. Then, it creates the authentication code by performing the 128-AES encryption of the obtained challenge code, using the combination of the assigned inclusion ID and object ID as a key. If the encrypted authentication code corresponds to the information about the device that initiated the association procedure, the hub generates a client ID and the password for the IP device, which will be used to establish the device MQTT session.

B. De-Association

The de-association process is initiated by the hub and does not require confirmation from the IP device. The MQTT connection with the device selected for exclusion from the network is stopped, and the hub clears all details related to the IP device, including its inclusion ID, device ID, and device information. The device ID may be reused in the future and assigned to other devices.

VI. WISE NETWORK MANAGEMENT

Since WISE protocol is based on MQTT, it relies on the TCP connection to operate. However, since the maintenance of the permanent TCP connection requires that the communicating parties send keep-alive messages periodically, it can have a negative impact on the IP device battery life time. Therefore, WISE protocol allows device hibernation and distinguishes between active devices and sleeping devices.

1. Active devices

These are the devices that maintain a permanent MQTT connection with the hub. This operation mode is suitable for actuators with the permanent power supply. Active devices will try to re-initiate the connection in the case of connection loss.

When the hub detects the loss of device connection, it starts sending SSDP advertisements (NOTIFY ALIVE messages), while the device starts listening to them. These messages contain the description of the hub, just like the messages in the discovery stage of association. If the ID of the advertised hub matches the one the device was connected to previously, the device should re-instantiate the MQTT connection using the credentials already obtained during the authentication challenge stage.

If the hub resides in the cloud, the active device should request the hub description from the predefined location and connect to the MQTT broker on the provided URI and port. Note that the location of the broker may be updated if the cause of the connection loss is the failure of the broker.

When disconnecting from the WISE network, the active device must first send a status message to its status topic, announcing its intention to go offline. After that, it should close the MQTT connection.

2. Sleeping devices

The sleeping devices do not maintain a permanent MQTT connection with the hub. When switching to hibernation, they must disconnect from the MQTT broker. Unlike the active devices, they should not send the status message first. They must define the sleeping interval, during which they remain in the hibernation mode. Once this interval expires, they must switch to the active state and maintain it for at least t_{active} = 10 s. The device must reset the active period timer after every request from the hub, i.e., it must remain active for at least t_{active} after the last request.

During the hibernation period, sleeping devices cannot be controlled, unless the user manually wakes them up by pressing a dedicated button. This is the reason the sleeping mode is usually avoided for actuators and used predominantly for sensor devices.

VII. DEVICE ABSTRACTION

The description of the IP device functionality represents a hierarchical structure of service groups, services and properties (Fig. 7).

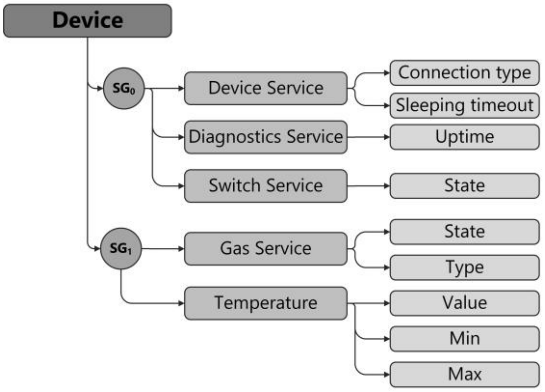

Fig. 7. Device object structure.

The device abstraction resembles the best of Zigbee, Z-Wave, and Bluetooth Low Energy models, thus allowing very simple translation between data models. The property represents the physical or logical state of the device, while sets of related properties form services. Finally, subsets of device services can be organized into service groups, each group identified by the service group identifier SG_i, $i \epsilon N_0$.

Every device must support *Device Service* and *Diagnostics Service*, and these services must belong to the service group SG_0. *Device Service* provides the information about the sleeping interval, device name, firmware version, WISE protocol version supported by the device, etc. On the other hand, *Diagnostics Service* allows accessing the information about device uptime, quality of the WiFi connection, etc. Besides the mandatory services, the IP device must support at least one additional service. Every service is defined by the type and the list of properties and commands it supports. Each property is defined by its name, type (number, text, boolean, range, etc.), current value, and read-write flag. Commands are defined by the name and the list of input parameters.

Every IP device must handle the following requests on the device object request topic W_D/req:

– *GetGroupList*: Used to fetch the list of service groups the device supports.

– *GetDeviceInfo*: Used to fetch the complete device object, holding the information about the device manufacturer, model, and the service group list.

– *GetState*: Used to fetch all changes of the device properties that occurred since the time specified in the request parameters.

– *SetTime*: Used to update the current time of the IP device to the value specified in the request parameters.

– *AdjustTime*: Used to adjust the current time of the IP device by adding the difference specified in the request parameters.

– *VerifyIdentity*: Used to validate the product details provided by the IP device.

– *GetServiceList*: Used to get the list of all services within a service group specified in the request.

– *GetService*: Used to fetch the complete service object with all its properties and commands.

Every IP device must handle the following requests on the service request topics $W_D/S_i/req$:

– *GetPropertyList*: Used to fetch the list of properties supported by the particular service specified in the topic.

– *GetPropertyValue*: Used to fetch the value of the particular property of the service specified in the topic.

– *SetPropertyValue*: Used to set the property to the desired value.

– *GetCommandList*: Used to fetch the list of all commands the service supports.

– *ExecuteCommand*: Used to execute the command specified in the request parameters.

For every change of a property of service S_i, the IP device must emit the *PropertyChanged* event to the service topic $W_D/S_i/evt$: In addition, information about firmware upgrade and device reboot should be sent to the device topic W_D/evt within *FirmwareUpgraded* and *DeviceRebooted* event messages, respectively.

VIII. IMPLEMENTATION

The proposed protocol was implemented within the existing smart home hub that already serves as the gateway between Zigbee, Z-Wave, and IP protocols [6], [16], [17]. The hub supports both WiFi and Ethernet connections, and it already uses MQTT protocol to communicate with the cloud and mobile applications. It is based on a WLAN chipset module running Linux, with 64 MB of RAM memory. For efficiency, the software is developed using C and C++ programming languages.

Internally, the hub software consists of three components: Home Manager, Message Broker, and System Manager. The Home Manager module acts as an interface to different communication technologies and provides the advanced hub logic. A bus is a software component that allows communication using a given communication protocol. The Home Manager carries the high-level logic that has to reside on the premise, like automation scripts and rules support, as well as various device control algorithms. The System Manager is responsible for system level maintenance tasks, firmware upgrade and monitoring of the hub's performance. Finally, the Message Broker component is responsible for MQTT communication within the system - between internal software components, with the cloud and with WISE devices. This Message Broker component is extended to support the WISE topics as defined in previous sections, while the WISE hub support is added to the Home Manager component (Fig. 8).

The supported service types are designed to cover the smart home scenario, and are listed in Table I.

TABLE I. SUPPORTED SERVICE TYPES.

Service Type	Description
device	Mandatory device service, containing information about device network connection.
diagnostics	Mandatory diagnostics service, containing information about device operational state.
battery	Optional service tracking battery level.
switch	Used by switches, plugs, etc.
dim	Used by dimmers and bulbs.
levelControl	Used by devices that allow control of a level, but the more specific service does not exist (e.g., volume or dim service).
color	Used by bulbs.
colorTemperature	Used by bulbs.
temperature	Used by temperature sensors.
humidity	Used by humidity sensors.
gas	Used by various gas sensors.
motion	Used by motion sensors.
contact	Used by contact sensors.
flood	Used by water leakage sensors.
smoke	Used by smoke sensors.
powerMetering	Used by metering plugs.
thermostat	Used by thermostats.
shutter	Used by rollers and shutters.
alarm	Used by alarm devices.
volume	Used by speaker devices, set top box, etc.

The other party in the communication is the WISE device. To ease the development of WISE devices, the WISE client SDK was created. The SDK consists of the MQTT messenger, SSDP client, utility classes, and the IP device core implementation (Fig. 9). Utility module

implements wrapper classes for C/C++ standard libraries (threading and synchronization mechanisms, timers, JSON parsing, exception handling, etc.). IP device core implements the network management, association process, event reporting, and required WISE API methods as listed in Section VII.

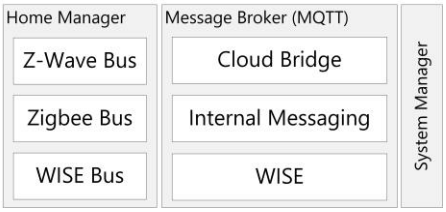

Fig. 8. Software architecture of the smart home hub.

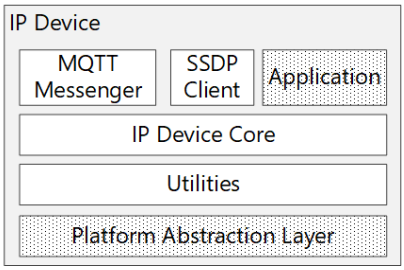

Fig. 9. IP device internal architecture.

Besides the SDK, for a complete device functionality, the application logic has to be developed, which will give the actual function to the device. To support portability, the WISE device SDK relies on a Platform Abstraction layer that allows porting to various platforms. As of today, the SDK has been ported to Linux and to FreeRTOS platforms.

IX. EVALUATION

For evaluation, the following components were used:

– WISE hub running on the existing smart home hub due to the convenience of cloud connection and client applications used for testing.

– A WISE device prototype based on a Linux platform with 1 GB of RAM, 900 MHz, and 4 cores, using temperature sensors and LEDs (to simulate the switch behaviour, i.e., digital output). Besides the mandatory *Device* and *Diagnostics* services, the *Temperature* and *Switch* service were also implemented. This platform is used for performance measurement and SDK verification.

– The other device based on the ESP32 WiFi microcontroller running at 240 MHz, with 4 MB of flash memory and 512 kB of RAM, using FreeRTOS. Only mandatory services were implemented on this platform, and as such used for resource usage measurement and functional verification.

For the hub performance testing, JMeter was used. To test the hub implementation, a total of 30 device association requests were generated at the same time, and the processing time was measured. Instead of real WISE devices, device software models were used. The testing results for relevant device association commands from Fig. 6 are presented in Table II, where T_{avg}, T_{min}, and T_{max} represent the average, minimum, and maximum processing time, respectfully.

TABLE II. DEVICE ASSOCIATION TESTING RESULTS.

Message	T_{avg} [ms]	T_{min} [ms]	T_{max} [ms]	S_{avg} [B]
Inclusion request	52	13	915	86
Capability request	591	62	1436	94
Join confirmation	110	70	428	124
Challenge request	97	38	334	84
Credentials request	535	320	1625	139

The average MQTT message size is given as S_{avg}. All devices successfully joined the network within a couple of seconds. During this test, the average CPU load of the Message Broker component was 1.4 %, and it used 19.1 MB of RAM memory on average, while the average CPU load of the Home Manager component was 4.7 %, and it used 10.1 MB on average.

After the device software models were connected to the hub, they continued to periodically send the events reporting the temperature changes and switch state changes. A total of 421771 temperature change and 421631 switch state change events were generated during the test run, during the 40 hours long test run. The average event processing time on the hub side was 14 ms, and there were no losses of connection detected and no errors in event processing.

The WISE client SDK implementation was first tested by measuring the performance of different commands the device is required to support.

Each command was executed 1000 times without any errors, and the results are presented in Table III. It can be observed that the commands *SetPropertyValue* and *ExecuteCommand* take longer to execute than the others since they result in the change of the diode state.

TABLE III. IP DEVICE TESTING RESULTS.

Message	T_{avg} [ms]	T_{min} [ms]	T_{max} [ms]	S_{avg} [B]
GetDeviceInfo	23	15	184	3732
GetGroupList	23	14	168	3117
GetServiceList	23	15	338	2689
GetService	20	13	183	690
GetPropertyList	20	12	214	542
GetPropertyValue	16	9	571	68
SetPropertyValue	42	18	219	39
GetCommandList	16	10	214	124
ExecuteCommand	42	22	211	39

Then, stress performance testing was also performed. Commands were generated every 250 ms, during the test run, which lasted 18 hours and 45 minutes. A total of 241583 requests were generated, and the device responded with no errors. No connection losses were detected.

On the microcontroller-based platform, the basic implementation consumes a total of 623 kB of flash memory and 116 kB of RAM memory, including the operating system, platform abstraction layer, and WISE SDK. This leaves plenty of room for the application itself. The tests showed the correct operation of the device, proving that the protocol is convenient even for resource-constrained platforms. During the user tests, the device was responsive as expected at the level of the first prototype.

X. CONCLUSIONS

This paper proposes a WISE protocol stack, designed to

be a complete solution for IP device integration into the IoT system, and yet lightweight enough to be used on resource-constrained platforms. The proposed technology addresses all aspects of network establishment and management typically covered by the IoT technologies and enables the uniform device representation on the application layer. This allows building a universal ecosystem with interoperable devices and related services, at one side leading to a better user experience, while on the other side greatly reducing the required device development effort for device vendors.

The technology elements are proposed in accordance with direct functional requirements, but also by taking into consideration the implementation complexity, resource usage, scalability, and security. The selected technologies are complemented with communication flows defining the behaviour of the communication participants in various scenarios.

The WISE protocol was implemented on hub and device prototypes (the smart home hub and the IP temperature sensor and switch), which were used for testing and validation. Performance and stability tests were also conducted. It was shown that the message processing times are acceptable for realistic usage scenarios. The implementation of a WiFi microcontroller platform showed that the complexity of the proposed protocol is well suited for such resource-constrained platforms. The client SDK allows the easy development of WISE-compatible IP devices. It has been successfully applied to integrate additional devices (Android set top box, home intercom system, WiFi lightbulb, and smart speakers) into the existing smart home solution. The results are not presented here due to paper length limits. In the future, the implementation can be extended to support other service types, suitable for industrial, agricultural, or health monitoring scenarios.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] J. Yun, I.-Y. Ahn, N.-M. Sung, and J. Kim, "A device software platform for consumer electronics based on the Internet of Things", *IEEE Trans. Consumer Electron*., vol. 61, no. 4, Nov. 2015. DOI: 10.1109/TCE.2015.7389813.

[2] D. Gislason, *Zigbee Wireless Networking*. Elsevier Inc., 2008.

[3] C. Paetz, *Z-Wave Basics: Remote Control in Smart Homes*. CreateSpace IPP, Jun. 2013.

[4] K. Townsend, C. Cufi, Akiba, and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. O'Reilly Media, Apr. 2014.

[5] T. Perumal, A. R. Ramli, and C. Y. Leong, "Interoperability framework for smart home systems", *IEEE Trans. Consumer Electron*., vol. 57, no. 4, pp. 1607–1611, Nov. 2011. DOI: 10.1109/TCE.2011.6131132.

[6] I. Lazarevic, M. Sekulic, M. S. Savic, and V. Mihic, "Modular home automation software with uniform cross component interaction based on services", in *Proc. of 2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Sept. 2015, pp. 363–365. DOI: 10.1109/ICCE-Berlin.2015.7391281.

[7] L. Zhao, W. Wu, and S. Li, "Design and implementation of an IoT based indoor air quality detector with multiple communication interfaces", *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9621–9632, Dec. 2019. DOI: 10.1109/JIOT.2019.2930191.

[8] B. L. Hughes, "WiFi and cloud enabled temperature control system", U.S. Patent Appl. US10495346B2, Dec. 2019.

[9] S. Jakovljev, M. Subotić, and I. Papp, "Realisation of a smart plug device based on Wi-Fi technology for use in home automation systems", in *Proc. of 2017 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2017, pp. 327–328. DOI: 10.1109/ICCE.2017.7889340.

[10] N. Shofa, A. Rakhmatsyah, and S. A. Karimah, "Infusion monitoring using WiFi (802.11) through MQTT protocol", in *Proc. of 2017 5th International Conference on Information and Communication Technology (ICoIC7)*, May 2017, pp. 1–7. DOI: 10.1109/ICoICT.2017.8074693.

[11] A. Yoddumnern, R. Chaisricharoen, and T. Yooyativong, "Cloud based WiFi multi-sensor network", *International Journal of Online and Biomedical Engineering*, vol. 14, no. 8, pp. 35–51, Aug. 2018. DOI: 10.3991/ijoe.v14i08.8536.

[12] C. A. Trasviña-Moreno, R. Blasco, R. Casas, and A. Marco, "Autonomous WiFi sensor for heating systems in the Internet of Things", *Hindawi Journal of Sensors*, vol. 2016, article 7235984, pp. 1–14, Feb. 2016. DOI: 10.1155/2016/7235984.

[13] S. Tozlu, M. Senel, W. Mao, and A. Keshavarzian, "WiFi enabled sensors for Internet of Things: A practical approach", *IEEE Communications Magazine*, vol. 50, no. 6, pp. 134–143, Jun. 2012. DOI: 10.1109/MCOM.2012.6211498.

[14] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies", *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb. 2017. DOI: 10.1109/JIOT.2016.2615180.

[15] L. Bracciale, P. Loreti, A. Detti, R. Paolillo, and N. B. Melazzi, "Lightweight named object: An ICN-based abstraction for IoT device programming and management", *IEEE Internet of Things Journal*, vol. 6, no. 3, Jun. 2019. DOI: 10.1109/JIOT.2019.2894969.

[16] V. Moravcevic, M. Tucic, R. Pavlovic, and A. Majdak, "An approach for uniform representation and control of ZigBee devices in home automation software", in *Proc. of 2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Sept. 2015, pp. 237–239. DOI: 10.1109/ICCE-Berlin.2015.7391244.

[17] I. Papp, G. Velikic, N. Lukac, and I. Horvat, "Uniform representation and control of bluetooth low energy devices in home automation software", in *Proc. of 2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Sept. 2015, pp. 366–368. DOI: 10.1109/ICCE-Berlin.2015.7391282.

[18] I. Froiz-Miguez, T. M. Fernandez-Carames, P. Fraga-Lamas, and L. Castedo, "Design, implementation and practical evaluation of an IoT home automation system for fog computing applications based on MQTT and ZigBee-WiFi sensor nodes", *Sensors (Basel)*, vol. 18, no. 8, Aug. 2018. DOI: 10.3390/s18082660.

[19] S.-M. Kim, H.-S. Choi, and W.-S. Rhee, "IoT home gateway for auto-configuration and management of MQTT devices", in *Proc. of 2015 IEEE Conference on Wireless Sensors (ICWiSe)*, Aug. 2015, pp. 12–17. DOI: 10.1109/ICWISE.2015.7380346.

[20] X. Cai, Y. Wang, X. Zhang, and L. Luo, "Design and implementation of a WiFi sensor device management system", in *Proc. of 2014 IEEE World Forum on Internet of Things (WF-IoT)*, Apr. 2014, pp. 10–14. DOI: 10.1109/WF-IoT.2014.6803108.

[21] A. Djama, B. Djamaa, and M. R. Senouci, "TCP/IP and ICN networking technologies for the Internet of Things: A comparative study", in *Proc. of 2019 International Conference on Networking and Advanced Systems (ICNAS)*, Jun. 2019, pp. 1–6. DOI: 10.1109/ICNAS.2019.8807890.

[22] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in IoT networking via TCP/IP architecture", NDN Project NDN-0038, Technical Report, Feb. 2016.

[23] W. Chen, S. Jeong, and H. Jung, "WiFi-Based home IoT communication system", *Journal of Information and Communication Convergence Engineering*, vol. 18, no. 1, pp. 8–15, Mar. 2020. DOI: 10.6109/jicce.2020.18.1.8.

[24] M. Tucic, R. Pavlovic, I. Papp, and D. Saric, "Networking layer for unifying distributed smart home entities", in *Proc. of 2014 22nd Telecommunications Forum Telfor (TELFOR)*, Nov. 2014, pp. 368–371. DOI: 10.1109/TELFOR.2014.7034426.

[25] G. Kim, S. Kang, J. Park, and K. Chung, "An MQTT-based context-aware autonomous system in oneM2M architecture", *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8519–8528, Oct. 2019. DOI: 10.1109/JIOT.2019.2919971.

[26] J. Yun, R. C. Teja, N. Chen, N. Sung, and J. Kim, "Interworking of oneM2M-based IoT systems and legacy systems for consumer products", in *Proc. of 2016 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct. 2016, pp. 423–428. DOI: 10.1109/ICTC.2016.7763511.

[27] R. Khan and A. N. Mian, "Sustainable IoT sensing applications development through graphQL-based abstraction layer", *Electronics*, vol. 9, no. 4, p. 564, Mar. 2020. DOI: 10.3390/electronics9040564.

[28] A. Günter, C. Schwarzer, and M. König, "IAL: An information abstraction layer for IoT middleware", in *Informatik 2020*, Sept. 2020, pp. 1255–1235. DOI: 10.18420/inf2020_114.

[29] CHIP Working Group, "Connected Home over IP", 2019. [Online]. Available: https://github.com/project-chip