Monitoring of IoT Devices Using SNMP

Technical Report, FIT VUT

Petr Matoušek, Patrik Krajč

Technical Report no. FIT-TR-2021-01
Faculty of Information Technology, Brno University of Technology

Last modified: June 22, 2021

Contents

Abstract

Internet of Things is a network composed of various physical devices like sensors, actuators and smart gadgets connected to a network. IoT devices are usually implemented with constrained hardware and software possibilities that prevents full communication over TCP/IP stack. Thus, to monitor IoT devices, we cannot use standard network monitoring and management tools.

This report summarizes our experiments with IoT monitoring using SNMP proxy agent that accesses monitoring data by parsing MQTT messages or reading IoT gateway log files. It also presents MIB templates for IoT MIB objects that are suitable for monitoring common IoT devices like climate sensors, smart outlets, etc.

Chapter 1

Introduction

Today, the number of IoT devices is rapidly increasing. Many times, these devices do not allow you to implement a full TCP stack, and there are problems in monitoring these devices.

In this document, we will focus on the method of monitoring. We will look at the SNMP protocol and tools supporting this protocol, such as mib2c, which will allow us to implement an SNMP agent and how to deploy it in an IoT network.

1.1 Structure of the Report

1.2 Acknowledgement

This work is supported by Brno University of Technology project "Application of AI methods to cyber security and control systems"(2020–2022), no. FIT-S-20-6293.

Chapter 2

Topology

In that chapter we will focus on the devices connected in the IoT network, and their communication. First of all, I would like to mention that the target element in communication is the home assistant, which also represents the MQTT broker.

Devices connected in the network can be divided into three groups. The first group of devices are devices that communicate directly with the home assistant over the MQTT protocol.The second group of devices are devices that do not communicate directly with the home assistant. This group of devices communicates via wireless communication over the ZigBee protocol to gateway. In order to be able to establish communication between the home assistant and this group, we need another device to be able to forward information from the sensors to the home assistant, and these devices represent our third group, the mentoined gateway. The picture 2.1 shows how the sensors are connected to our IoT network. A complete description of the network is in Annex A.

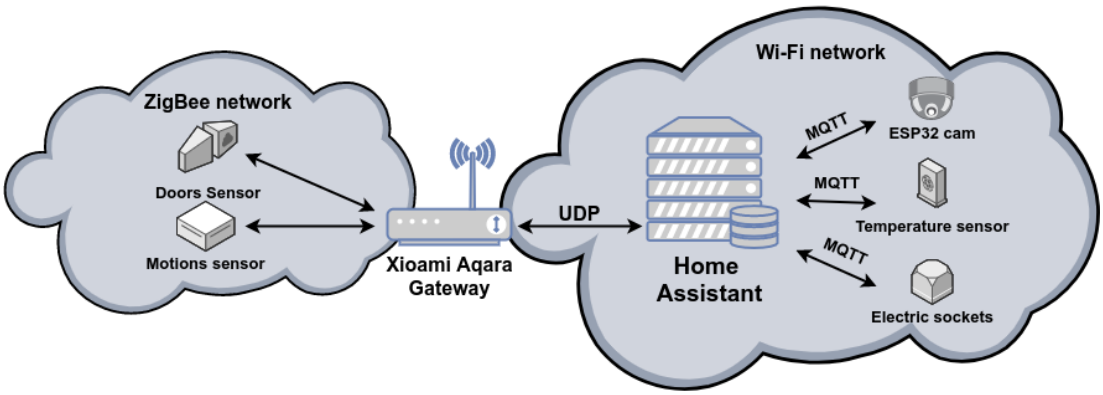


Figure 2.1: IoT topology

Chapter 3

Devices

In this chapter we will focus on devices connected in our IoT network. We will take a closer look at the method of communication of various types of devices, the transmitted data and the assignment of a record of devices to the SNMP MIB database.

3.1 Xiaomi aqara Gateway and sensors

The first device we will focus on is the Xiaomi aqara gateway. We will imagine its role in the IoT network together with other related sensors, such as a magnetic and motion sensor.

Xiaomi aqara gateway represent device which one is used to integrate devices used to communication technology ZigBee. The gateway is the interface between the sensors and the home assistant. Its task is to collect information from directly connected sensors and distribute it to the home assistant.

3.1.1 Message format

In our IoT network, xiaomi aqara gateway collect information from magnetic and motion sensors.

The messages between the gateway are in JSON format and consists of several items based on which it is able to distinguish which device is in a specific message.

- cmd - The item contains information about the used method. There are several methods like *read/write_ ack* method which one is used to confirm received messages. *Write* method is used when the home assistant communicates with the gateway and sets some sensor attribute. *Report* method is used when some sensor change his state and informs the home assistant. The last method is used to inform the home as-

sistant about the current availability of the device. These keep alive messages use method called *heartbeat*.

- model - It contains information about what device it is, in this case it is a gateway, for magnetic sensors is it *sensor_ magnet.aq2* and for motion sensors is it *sensor_ motion.aq2*.
- sid - This item is used to identify sensors. It contains the identifier of the specific sensor under which it is registered on the gateway.
- short_id - It serves as a message counter for specific types of communications.
- data - This item consist information from sensors transmitted to home assistent. Item contains all the resources of the sensors. Resources such as *voltage* that reports the current battery status, *status* (open / closed), *illumination* and *RGB* color of light.

3.1.2 Communication analysis

From the communication between the gateway and the home assistant, we can divide the communication into two types. The first type of communication directly between the gateway and the home assistant. The second type of communication between the gateway and the home assistant, with the difference that information are transmitted from devices that are connected to a gateway.

In order to distinguish the communication between the gateway and the home assistant and the communication of a sensor from the ZigBee network, it is possible to use the model item, because this item contains information about what type of device transmits information in the data item.

3.1.3 Communication protocol

In the previous chapter, we described the format of the message. In this section we will show examples of communications. We will show unicast and multicast communication and in which cases they are used.

Multicast communication uses the address 224.0.0.50, and uses application port 4321. This type of communication is used in the following cases:

- Registration - this type of communication starts with a new sensor connected to the network. The sensor sends a message with one entry, a command with a value of whois.
- Status Reporting - Sensors send a message to a multicast address by informing all devices in the group of their current status. The message format is the same as when forwarding information from sensors, except

that the command contains the value of report. From the gateway side, the communication is a bit different, the gateway sends keep alive messages that contain the heartbeat command and the data items contain the ip address of the device.

```
Registration request:
{
  "cmd":"whois"
}

Registration respond:
{
  "cmd":"iam",
  "ip":"192.168.11.211" ,
  "port":"9898,
  "model":"gateway"
}

Status reporting:
{
  "cmd":"heartbeat",
  "model":"gateway",
  "sid":"04cf8cb09255",
  "short_id":"0",
  "token":"58KR9mvOLQ07m4RD",
  "data":"{\"ip\":\"192.168.11.211\"}"
}
```

If we want to limit the number of devices that can access the gateway, we can create a 16-character string with the Mi home application, which will be used to authenticate the devices that will want to communicate with the gateway. If we do not want to affect access to the gateway, we will set this string to an empty string and allow everyone to access the gateway. The communication used to authenticate the devices is encrypted using AES - CBC 128.

An example of unicast communication could be, when a home assistant requests gateways, all the identifiers of all connected sensors. The request only contains a command with a value of get_id_list. The gateway must acknowledge receipt of the request with the get_id_list_ack command, and specify a sequence of identifiers in the data entry.

```
Request:
{
  "cmd":"get_id_list"
}
```

```
Respond:
{
  "cmd":"get_id_list_ack"
  "sid":"04cf8cb09255"
  "token":"58KR9mvOLQ07m4RD",
  "data":"[\"sid1\",\"sid2\",\"sid3\"]"
}
```

3.1.4 Sensors

As mentioned above, we will use the entry model to distinguish communication. For magnetic sensor, this entry takes the value *sensor_ magnet.aq2* nad motion sensor takes *sensor_ motion.aq2*.

This entry, but does not identify a specific sensor. To be able to uniquely determine which sensor it is, we need an ID item, which is a unique identifier of the device connected directly to the gateway.

Once we are able to identify what type of device it is, and uniquely determine which device it is, we can focus on what information we can get from these devices. Information from sensors are transmitted in a data entry that encloses a sequence of entries for a specific sensor.

In addition to forwarding messages from individual sensors, the gateway informs the home assistant about its availability. This type of keep-alive message is transmitted with the heart-beat command, and contains only one entry in the data entry, the IP address of the device.

In the case of magnetic sensor, the data entry contains a sequence of items:

- Voltage - current device battery status
- Status - current state of the object, closed or open

A motion sensor contains the following items:

- Voltage - current device battery status
- Lux - current luminance value

Information about individual sensors is grouped into SNMP tables. Each sensor type represents one table. Below are examples of items in the relevant tables for magnetic sensor and motion sensor.

```
aqaraGatewayTableEntry ::= SEQUENCE {
    gatewayIndex         Integer32,
    gatewaySID           DisplayString,
    gatewayModel         DisplayString,
```

```
    gatewayShortID       DisplayString,
    gatewayToken         DisplayString,
    gatewayipAddr        DisplayString
}

magneticSensorAQ2TableEntry ::= SEQUENCE {
    magSensorIndex     Integer32,
    magSensorSID       DisplayString,
    magSensorModel     DisplayString,
    magSensorShortID   DisplayString,
    magSensorCommand   DisplayString,
    magSensorVoltage   Integer32,
    magSensorStatus    Boolean
}

motionSensorAQ2TableEntry ::= SEQUENCE {
    motionSensorIndex     Integer32,
    motionSensorSID       DisplayString,
    motionSensorModel     DisplayString,
    motionSensorShortID   DisplayString,
    motionSensorCommand   DisplayString,
    motionSensorVoltage   Integer32,
    motionSensorLux       Integer32
}
```

3.2 MQTT broker and sensors

Home assistant has an MQTT broker installed. An MQTT broker is a component against which individual devices communicating through the MQTT protocol must register.They can register resources that they want to share with devices in IoT network, or request some specific resources and the MQTT broker will provide them.

Devices communicate via the MQTT protocol using the tasmota firmware. This firmware allows relative flexibility in the configuration of the sensors. Allows to configure their attributes and select the type of device.

In our network topology there are several kind of devices using MQTT protocol. Specifically, these are electrical outlets BlitzWolf SHP6, camera ESP32-cam.

Electrical outlets periodically send two types of messages that contain information about the current device and resources status. Both types of messages are a topic within MQTT communication. These are specifically topics called *sensor_name/tele/State* and *sensor_name/tele/Sensor*.

The topic for monitoring device status includes the following items:

- Time - actual time
- Uptime - up time of device
- UptimeSec - up time of device in seconds
- Heap - current device heap
- Sleepmode - how the device switches to sleep mode
- Sleep -
- LoadAvg
- MqttCoung -
- Power - describe actual device state
- Wifi - This entry is a sequence of entries that describe the access point to which it is connected. These are items such as the SSID, BSSId, Chanel, RSSI, Signal, LinkCount, Downtime

The topic for monitoring device resources includes the following items:

- Time - current time
- Energy - this entry is a sequence of entries, which include information about resources. The individual items are described below.
- TotalStartTime - date and time the device was added to service
- Total - total energy consumed
- Yesterday - energy consumed for yesterday
- Today - energy consumed for today
- Period
- Power - current consumed power
- ApparentPower
- ReactivePower
- Factor
- Voltage
- Current

As in the previous case, all types of sensors are included in the SNMP tables. The items stored in the table for electrical outlets are shown below.

Chapter 4

Data acquisition

In this chapter, we'll show you how to get sensor information from Home Assistant. We will show two ways, using SQLite, and MariaDB.

The data acquisition scheme is shown in the figure 4.1. The picture shows the SNMP agent, which we have not yet mentioned, the following chapteris devoted to the SNMP agent. As can be seen from the picture, we have a database that contains records that are processed by the SNMP agent. Thus, the first step in this schema is to allow the SNMP agent to retrieve information stored in the home assistant database which we'll show in this chapter.

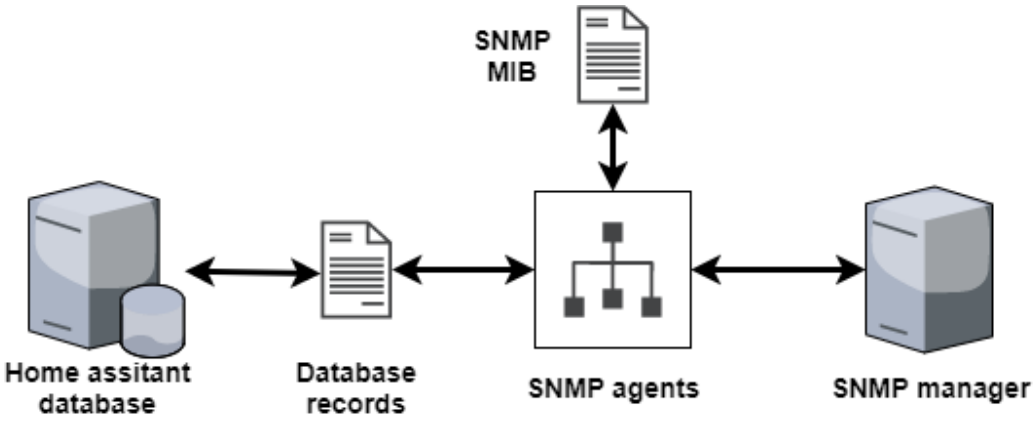


Figure 4.1: Data acquisition scheme

4.1 SQLite in home assistant

As we mentioned, a home assistant is involved in our IoT network. Home assistant is used to collect all information from the IoT network. It stores this information in a DB called *home-assistant_ v2.db*.

The database consists of three tables, but we will only be interested in one, the table *states*. In order to uniquely identify specific devices, I need to use a composite key, this key consists of the domain and entity identifier entries. For example, a domain could be a device type, in our case it could be *sensor_ magnet.aq2*.

In order to provide the agent with the required information, we created a script that creates a file for each type of sensor with all the items that we want to query via the SNMP protocol.

This script contains one function called *init_ resources* whose task is to create a data file for a specific type of sensor. It has three input arguments:

- Domain - represents the sensor type
- Attributes - sensor attributes specified for the SNMP agent
- File path - path with the file name where the data will be saved

Below is an example of creating data for an agent. Specifically, it is a gate. First we define the domain, because the domain is the type of device, our domain is the gateway. Next, we need to define what attributes of the given types of sensors interest us. As you may have noticed, it is also necessary to define between the attributes whether the attribute is immersed in a sequence, such as the ip attribute, which is part of the data item. The last thing we need to define is the output file. Finally, we just call the init_resources function with arguments.

```
GATEWAY_DOMAIN='gateway'
GATEWAY_ATTRIBUTES="cmd model sid short_id token data.ip"
GATEWAY_FILE_PATH="./gateway.conf"
init_resources "$GATEWAY_DOMAIN" "${GATEWAY_ATTRIBUTES}" "${GATEWAY_FILE_PATH}"
```

4.2 MariaDB in home assistant

In order to obtain data from the home assistant, we can also use the client-server method using a database engine, in our case it is the MariaDB database. But first we need to set the way the data is stored.[1].

We will use this method when creating an SNMP agent. We can communicate directly with the home assistant database, and we do not need a script to store the data in a file from which we can process the data.

The way we will process the SNMP agent data will be mentioned in the chapter 5.

[1] https://www.home-assistant.io/integrations/recorder/

Chapter 5

SNMP agent

In order to be able to use the SNMP protocol, we need to create an agent on the device that will serve to monitor the required objects. Our objects represent different types of sensors connected in the IoT network.

In previous chapter, we seen how we are able to obtain information about sensors from the home assistant database.

In this chapter, we will focus on how to create an agent for each type of sensor, and how to obtain information for MIB objects for it.

Before we start generating a template for the SNMP agent, we must have a MIB database created, because the generation of the template is based on this information. We have to copy our created database to the folder where all MIBs are stored, an example is shown below. In case the MIB is not visible, we have to export it, and we can verify the existence of it with the snmptranslate command, all these commands are shown below.

```
sudo cp IOT-MIB.txt /usr/local/share/snmp/mibs/
export MIBS=+IOT-MIB
snmptranslate -Tp -IR agentxIOT
```

5.1 Creating SNMP agent

In that section we will show how we can create an SNMP agent that will manage the obtained resources. We will create the agent in python, and use the pyagentx library. The required packages can be obtained by the following commands on the CentOS operating system.

```
sudo yum install python
sudo yum install python-pip
pip install --upgrade pip
pip install pyagentx
```

We will show the structure of the program. We can divide it into two parts, the agent and the part that prepares the data for the agent.

We will create a class that will represent an agent, this class is derived from the *pyagentx.Agent* class. In this class we need to register SNMP objects with their IOD and at the class that will process the object data for the agent. An example of an agent is shown below. The agent registers one SNMP table for the magnetic sensor. The IOD used is intended for the experimental SNMP branch. When we view the SNMP MIB translation, we can notice that the OID matches exactly one table.

```
+--agentxIOT(999)
   |
   +--Tables(1)
      |
      +--magneticSensorTable(1)
         |
         +--magneticSensorTableEntry(1)
            |  Index: magneticSensorIndex
            |
            +-- -R-- Integer32 magneticSensorIndex(1)
            +-- -R-- String    magneticSensorState(2)
            |        Textual Convention: DisplayString
            |        Size: 0..4
            +-- -R-- Integer32 magneticSensorOpenSince(3)
            +-- -R-- Integer32 magneticSensorBattery(4)
            +-- -R-- String    magneticSensorFriednlyName(5)
            |        Textual Convention: DisplayString
            |        Size: 0..64
            +-- -R-- String    magneticSensorIdentifier(6)
            |        Textual Convention: DisplayString
            |        Size: 0..64
            +-- -R-- String    magneticSensorLastChanged(7)
            |        Textual Convention: DisplayString
            |        Size: 0..64
            +-- -R-- String    magneticSensorLastUpdate(8)
                     Textual Convention: DisplayString
                     Size: 0..64
```

```
class IoTAgent(pyagentx.Agent):
    def setup(self):
        self.register('1.3.6.1.3.999.1.1', magneticSensorTable)
```

5.2 Creating SNMP subagents

In this part we will show the basic principle of creating sub agents that will process individual types of sensors. First we will create a class that will be derived from the pyagent.Updater class. This derivation forces the class we created to implement the update method. This method will be key for us along with the *updateTableEntries* method, which we will explain below.

As an example of creating a table for IoT devices, we chose a magnetic sensor, with class name *magneticSensorTable*. We will work with several objects, specifically the *databaseObjectList*, *databesObject* and *database* objects.

- databaseObject - this object consist all information from database, attributes, identifier, last update time, last changed time. It also contains methods by which we can query the status and attributes of a given sensor. These methods are for integer format, and string format.
- databaseObjectList - the task of this class is to keep all sensors of one type together.Another task it performs is to check whether there has been a change of state on an object.
- database - this object is used to query information about individual sensors. It performs another task, this task is that we need to find out all the names of sensors, of the specified type. We must define in each class a prefix that represents the name, a substring, of a particular sensor type that is used to find all sensors of that type.

Once we have imagined all the objects we will need, we can show a small demonstration of the implementation, and the class diagram that is shown 5.1.

As we can see, the class contains a sensor prefix variable, this variable is mentioned above as a substring that we use to find all sensors of a given type.

The update method is iteratively called by the agent, and with each call it obtains all available sensors and inserts them into the list of objects. Next, the updateTableEntries method is called, which updates the SNMP table values based on the list of objects.

```
class magneticSensorTable(pyagentx.Updater):
  list_objects = databaseObjectList()
  db = database()
  sensor_prefix = 'binary_sensor.door_window_sensor'

  def update(self):
    self.db.connect_to_db()
    sensors = self.db.get_sensors_id(self.sensor_prefix)
    for sensor in sensors:
      db_object = self.db.get_attributes_and_last_update(sensor[0])
      self.list_objects.updateObject(db_object)
    self.updateTableEntries()

  def updateTableEntries(self):
    cnt = 1
    for sensor in self.list_objects.list:
      idx = str(cnt)
      self.set_INTEGER('1.1.' + idx, cnt)
      self.set_OCTETSTRING('1.2.' + idx, sensor.get_str_state())
      self.set_INTEGER('1.3.' + idx, sensor.get_int_attribute('Open since'))
      self.set_INTEGER('1.4.' + idx, sensor.get_int_attribute('battery_level'))
      self.set_OCTETSTRING('1.5.' + idx, sensor.get_str_attribute('friendly_name'))
      self.set_OCTETSTRING('1.6.' + idx, sensor.entity_id)
      self.set_OCTETSTRING('1.7.' + idx, sensor.last_changed)
      self.set_OCTETSTRING('1.8.' + idx, sensor.last_updated)
      cnt += 1
    self.db.close_db_connection()
```

Oncewe have an subagent created, it's time to try query inidvidual resources. For example, we can use snmptable tool. In this way, we can query the entire contents of the table, which will also show us information with column names. The result is shown in the figure .

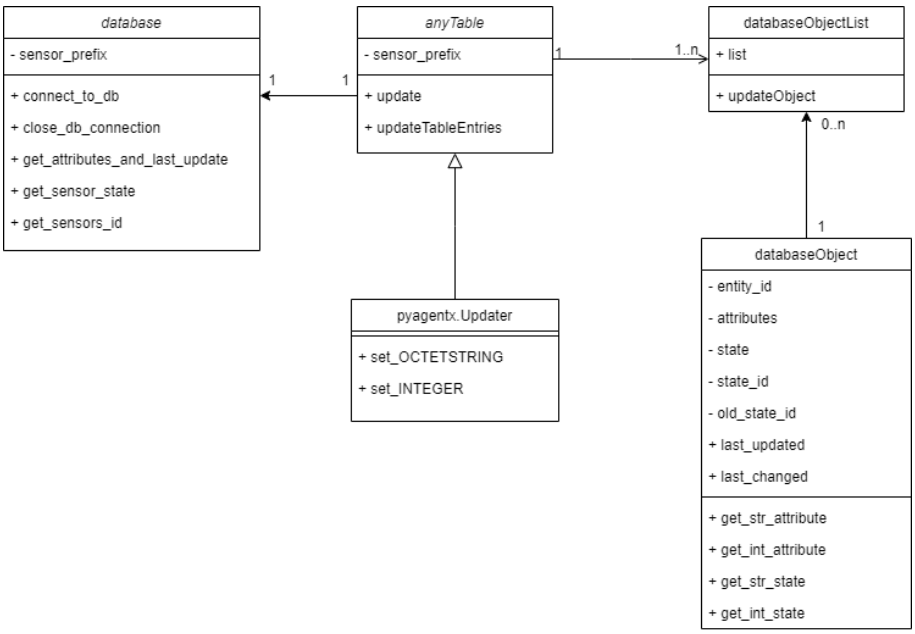


Figure 5.1: Subagent class diagram

```
SNMP table: IOT-MIB::magneticSensorAQ2Table

 index magSensorSID       magSensorModel magSensorShortID magSensorCommand                    magSensorVoltage magSensorStatus
  1.49       "alfa" "magnetic.sensor.aq2"         "s1s2s3"         "report" Wrong Type (should be INTEGER): "105"          closed
  1.50       "beta" "magnetic.sensor.aq2"         "s1a2s3"         "report" Wrong Type (should be INTEGER): "106"          closed
  1.51       "gama" "magnetic.sensor.aq2"         "s1b2s3"         "report" Wrong Type (should be INTEGER): "107"          closed
```

Figure 5.2: Caption

Chapter 6

Nagios installation and configuration

To make our solution portable, we decided to run the monitoring system on a virtual machine. We will use the virtualization software VirtualBox[1], and we chose Ubuntu 18.04[2] as the operating system.

6.1 Virtual machine

To log in, we created the user nagios with the password also nagios.

This step is specified only if the virtual machine is running outside the company network or when you don't have permissions connect to server.

After successfully installing the virtualbox, and installing the operating system, we will need to configure the VPN client. We need to download the configuration file and run openvpn client with root permissions.

```
wget https://www.fit.vut.cz/units/cvt/net/FIT.ovpn
sudo openvpn --config FIT.ovpn
```

6.2 Nagios installation

As a monitoring tool, we chose Nagios core 4.4.5, this tool is freely available, and provides a wide range of tools, and various plugins. In this section, we'll show how to install it on Ubuntu 18.04, and configure it to be able to get information from an agent that gets information from a home assistant.

First, we will update the Ubuntu repository and install some packages dependencies for the Nagios installation.

[1] https://www.virtualbox.org/
[2] https://ubuntu.com/

```
sudo apt update
sudo apt install -y autoconf bc gawk dc build-essential gettext
    gcc libc6 make wget unzip apache2 php libapache2-mod-php7.2
    libgd-dev libmcrypt-dev make libssl-dev snmp libnet-snmp-perl
```

As soon as we have the packages downloaded and installed, we need to download the source files for Nagios core 4.4.5, and extract them to a folder.

```
tar xzf nagios-4.4.5.tar.gz
wget https://github.com/NagiosEnterprises/nagioscore/archive/nagios-4.4.5.tar.gz
cd nagioscore-nagios-4.4.5/
```

First, compile Nagios source code and define the Apache virtual host configuration for Nagios.

```
sudo ./configure --with-httpd-conf=/etc/apache2/sites-enabled
sudo make all
```

Create the Nagios user and group, and add the 'www-data' Apache user to the nagios group.

```
sudo make install-groups-users
sudo usermod -a -G nagios www-data
```

We will now install Nagios along with the daemon and command mode, which we will work with later.

```
sudo make install
sudo make install-daemoninit
sudo make install-commandmode
```

Run script for sample configuration.

```
sudo make install-config
```

Install apache configuration for Nagios, activate mode for rewrite and cgi modules and restart apache service.

```
sudo make install-webconf
sudo a2enmod rewrite cgi
systemctl restart apache2
```

We should have installed the Nagios kernel, now we need to add basic user authentication. We will create the user nagiosadmin, and as the password we will choose the same as the username, nagiosadmin.

```
sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

The next step will be the administration of the firewall, where we will enable all the services necessary for the operation of Nagios core.

```
sudo ufw allow Apache
sudo ufw reload
```

The last step in completing the Nagios core installation is to add the basic plugins.

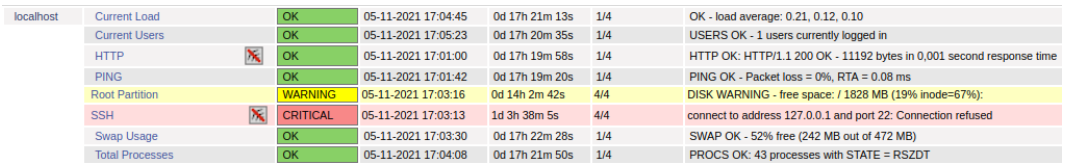
```
sudo apt install nagios-plugins nagios-nrpe-plugin
```

To have Nagios and plugins in one place, we need to copy them, we can use the sequence of commands below.

```
cd /usr/local/nagios/
sudo mkdir plugins
sudo cp /usr/lib/nagios/plugins/* ./plugins/
```

To verify that our installation process was successful, we will log in to the application available at https://localhost/nagios. After successful login, in the left menu we select the item hosts, in which we can see the current status of our virtual machine.

We will also check the availability of downloaded plugins. In the left menu we will now select the item services, which will show us the current status of services such as http, ping, and the like on the picture 6.1.



localhost	Current Load	OK	05-11-2021 17:04:45	0d 17h 21m 13s	1/4	OK - load average: 0.21, 0.12, 0.10
	Current Users	OK	05-11-2021 17:05:23	0d 17h 20m 35s	1/4	USERS OK - 1 users currently logged in
	HTTP	OK	05-11-2021 17:01:00	0d 17h 19m 58s	1/4	HTTP OK: HTTP/1.1 200 OK - 11192 bytes in 0,001 second response time
	PING	OK	05-11-2021 17:01:42	0d 17h 19m 20s	1/4	PING OK - Packet loss = 0%, RTA = 0.08 ms
	Root Partition	WARNING	05-11-2021 17:03:16	0d 14h 2m 42s	4/4	DISK WARNING - free space: / 1828 MB (19% inode=67%):
	SSH	CRITICAL	05-11-2021 17:03:13	1d 3h 38m 5s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	05-11-2021 17:03:30	0d 17h 22m 28s	1/4	SWAP OK - 52% free (242 MB out of 472 MB)
	Total Processes	OK	05-11-2021 17:04:08	0d 17h 21m 50s	1/4	PROCS OK: 43 processes with STATE = RSZDT

Figure 6.1: Localhost services

6.3 Nagios configuration

In that section, we'll show you how to add a new device to your Nagios system and how to monitor its services. We need to modify the *nagios.cfg* configuration file, which is stored in */usr/local/nagios/etc/* the directory. In the configuration file we can directly define the directory from which the configuration for Nagios will be loaded, or only one file. All the information we want to find out is stored on one server, so we will only add the path to the file like file for localhost.

```
cfg_file=/usr/local/nagios/etc/objects/localhost.cfg
cfg_file=/usr/local/nagios/etc/objects/hassio.cfg
```

We will now discuss how to properly fill in the configuration file for the new device. First we need to define what device it is, host name, alias and IP address. If we did not use use linux-server, we would have to fill in other mandatory fields[3]. An example is shown below.

```
define host{
    use         linux-server
    host_name   example
    alias       example
    address     192.168.0.1
}
```

Once we have added a new device, it's time to assign it some services that we will monitor. We will show the addition of a new service on the PING service.

As with the definition of a new device, some items must be filled in, we can use for example the use general-service setting. Next, we need to identify the device to which the service applies, add a comment on what the service should do, and finally add the settings that will perform the service.

Now let's look at the structure behind check_command.

- check_ping - this section defines which plugin should be called when calling this service. If we look in the plugins directory, we will find the check_ping program there.
- !100.,20% - char "!" is used as a delimiter for the arguments that you use to define the command that we show in section 6.4.

[3]https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/objectdefinitions.html

```
define service{
    use                     generic-service
    host_name               example
    service_description     Ping example
    check_command           check_ping!100.0,20%!500.0,60%
}
```

We will use the commands below to verify the configuration and restart the Nagios service.

```
/user/local/nagios/bin/nagios -v /usr/local/nagis/etc/nagios.cfg
service nagios restart
```

6.4 Plugin for SNMP table

There is nothing in the basic plugins to help us monitor an agent on a remote device. We will use a freely available plugin called check_snmp_table[4]. After downloading, unzip the plugin, rename it so that it does not have the .pl extension, and copy it among other plugins.

In order to run a new plugin, we need to install new packages, which we install into the following sequence of commands.

```
sudo perl -MCPAN -e 'install XML::Simple'
sudo perl -MCPAN -e 'install Monitoring::Plugin'
```

Due to the dependency between packages, we need to rename them in the check_snmp_table file *Nagios::Plugin* to *Monitoring::Plugin*. In order to verify the functionality of the plugin, we therefore need to create an XML file, which we will show in section 6.5.

In order to use this plugin, we must first define a command to call it. Commands are created in a file */usr/local/nagios/etc/objects/commands.cfg*. The command consists of two parts, its name and method of execution. The command name will be used when creating a new service. The first argument defines the destination address, and the second argument will contain our arguments that we will add when defining the service, specifically the path to the XML file.

```
define command{
    command_name check_snmp_table
    command_line $USER1$/check_snmp_table -H $HOSTADDRESS$ $ARG1$
}
```

[4] check-snmp-table.ommeluse.de/podwiki.pl?page=check_snmp_table_download

6.5 Import SNMP table to Nagios

Before we start adding our SNMP tables, we will create a directory in which we will store XML files that will define our monitoring parameters.

```
mkdir /usr/local/nagios/etc/objects/tables
```

Below is an example of an XML file that is intended for a motion sensor, its task is to monitor the current battery status of individual sensors. We will gradually analyze what the individual items in the XML file mean.

The XML file is divided into two main elements, general and status. The first general element contains elements that define basic information about the table, and the status element contains tests that will check the status of sensors.

```
<?xml version='1.0'?>
<config>
  <general>
    <description>MotionSensor battery</description>
    <name>1</name>
    <type>static</type>
    <base_oid>1.3.6.1.3.999.1.2.1</base_oid>
    <index>4</index>
    <method>AND</method>
  </general>
  <status>
    <test>
      <item>motionSensorBattery</item>
      <check_oid>5</check_oid>
      <critical>15:</critical>
    </test>
   </status>
</config>
```

- description - a description that will be displayed along with the message.
- name - sub OID that we can use as an identifier that we will display when reporting the message with current status.
- type - its define type for IOD, in dynamic state we can use for example regex to define OID.
- base_oid - OID for SNMP table.
- index - define sub identifier for table entry.

- method - define how to sum up all return value from tests.
- test - tests in the status element, this element can contain several tests, if we use a larger number of tests to evaluate the state of the object, we must define how to evaluate these tests, using the method AND, OR, MIN. Each test can also contain sub tests, with the same rule as the element status.
- item - this element describe structure of sub OID
- check_oid - element whose value will be checked.
- critical - in this element we define the value from which it will be considered like critical. Last char in our critical element is : which describe, that value lower then 15 is critical, othewise char . is used for greater values.
- warning - in this element we define the value from which it will be considered like warning.
- string - comparing string value in sub OID, reporting is same like critical or warning.

We have almost everything ready so that we can obtain information from the SNMP agent and display it in the Nagios software. The last step is to create the services in the configuration file as we mentioned in section 6.3. Below is an example that we used to monitor the battery status of the motion sensor. Where path is */usr/local/nagios/etc/objects/table/*.

```
define service {
    use                     generic-service
    host_name               hassio
    service_description     Motion sensor - battery
    check_command           check_snmp_table!-c public -f /path/file.xml
}
```

6.6 Monitoring

To monitor the status of sensors from the home assistant, we have created several usage examples. An example is the monitoring of the battery status of individual sensors, which was used in the motion and magnetic sensor, whose outputs are shown in the pictures.

With the magnetic sensor, we have information on how long it is in the open state, how suddenly the duration is longer than ten minutes, it will be reported as a warning.

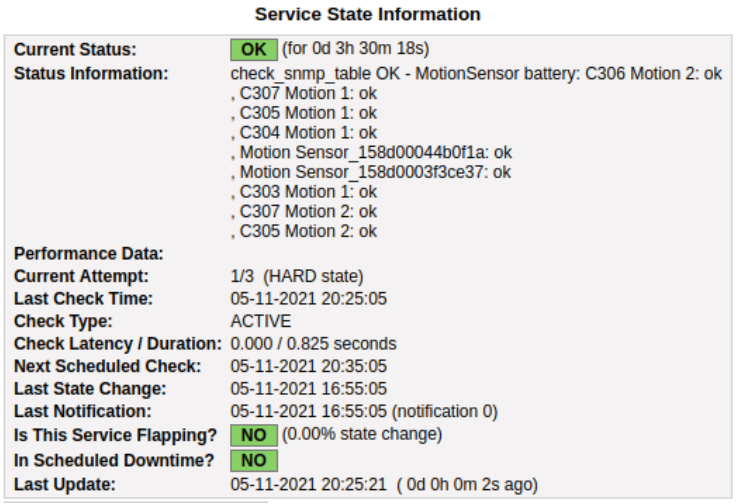
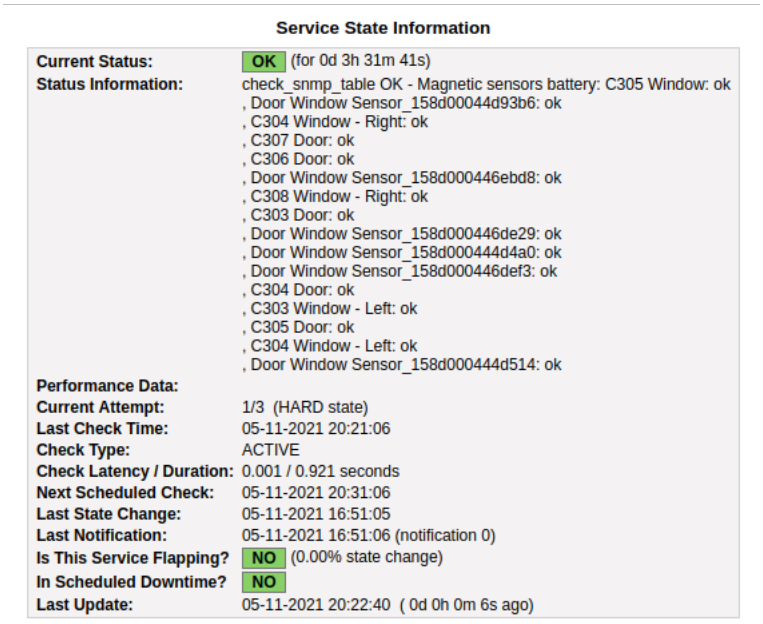


Figure 6.2: Nagios - motion sensor battery state

With the motion sensor, we have information about the status or whether motion was detected, and we decided to add this information to Nagios.

Another sensor that we monitor with Nagios is a smart socket SHP6, this sensor connects to Wi-Fi, and I monitor the signal strength, this parameter can be significant, so we added it, and checks whether the signal level does not fall below fifty percent. Together with this parameter, we monitor the current voltage in the socket, at which we monitor whether the voltage value does not reach over 250 volts.



Service State Information

Current Status:	OK (for 0d 3h 31m 41s)
Status Information:	check_snmp_table OK - Magnetic sensors battery: C305 Window: ok , Door Window Sensor_158d00044d93b6: ok , C304 Window - Right: ok , C307 Door: ok , C306 Door: ok , Door Window Sensor_158d000446ebd8: ok , C308 Window - Right: ok , C303 Door: ok , Door Window Sensor_158d000446de29: ok , Door Window Sensor_158d000444d4a0: ok , Door Window Sensor_158d000446def3: ok , C304 Door: ok , C303 Window - Left: ok , C305 Door: ok , C304 Window - Left: ok , Door Window Sensor_158d000444d514: ok
Performance Data:	
Current Attempt:	1/3 (HARD state)
Last Check Time:	05-11-2021 20:21:06
Check Type:	ACTIVE
Check Latency / Duration:	0.001 / 0.921 seconds
Next Scheduled Check:	05-11-2021 20:31:06
Last State Change:	05-11-2021 16:51:05
Last Notification:	05-11-2021 16:51:06 (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	05-11-2021 20:22:40 (0d 0h 0m 6s ago)

Figure 6.3: Nagios - magnetic sensor battery state

Bibliography

Appendix A

Floor plan

The picture A.1 shows the floor plan where the IoT network is located. Attached to this image is a table A that contains information about the distribution of sensors in the rooms.

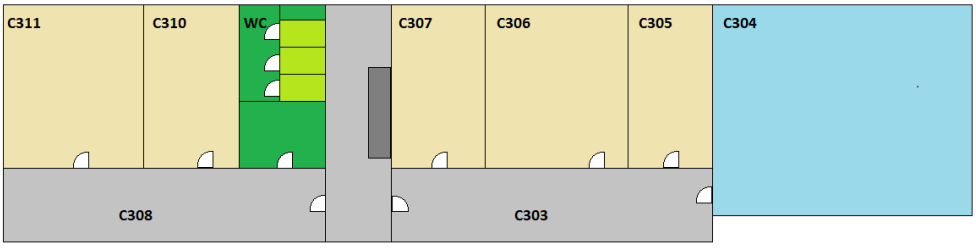


Figure A.1: Floor plan

Room	Friendly name	Protocol	Destination	Destination Addr
C303	C303 Motion 1	Wifi/ZigBee	Gateway C304	192.168.11.211
C304	C304 Motion 1	Wifi/ZigBee	Gateway C304	192.168.11.211
C304	C304 Motion 2	Wifi/ZigBee	Gateway C304	192.168.11.211
C305	C305 Motion 1	Wifi/ZigBee	Gateway C304	192.168.11.211
C305	C305 Motion 2	Wifi/ZigBee	Gateway C304	192.168.11.211
C306	C306 Motion 1	Wifi/ZigBee	Gateway C304	192.168.11.211
C306	C306 Motion 2	Wifi/ZigBee	Gateway C304	192.168.11.211
C307	C307 Motion 1	Wifi/ZigBee	Gateway C307	192.168.11.212
C307	C307 Motion 2	Wifi/ZigBee	Gateway C307	192.168.11.212
C304	C304 Door	Wifi/ZigBee	Gateway C304	192.168.11.211
C305	C305 Door	Wifi/ZigBee	Gateway C304	192.168.11.211
C306	C306 Door	Wifi/ZigBee	Gateway C304	192.168.11.211
C307	C307 Door	Wifi/ZigBee	Gateway C307	192.168.11.212
C304	C304 Window - Left	Wifi/ZigBee	Gateway C304	192.168.11.211
C304	C304 Window - Right	Wifi/ZigBee	Gateway C304	192.168.11.211
C305	C305 Window	Wifi/ZigBee	Gateway C304	192.168.11.211
C306	C306 Window	Wifi/ZigBee	Gateway C304	192.168.11.211
C307	C307 Window	Wifi/ZigBee	Gateway C307	192.168.11.212
C308	C308 Window - Right	Wifi/ZigBee	Gateway C308	192.168.11.213
C306	SHP6_kavovar_delongi	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_kavovar_saeco	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_konvica	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_ladnicka	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_nes_ap1	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_nes_ap1	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_koutensky	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_letavay	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_lichtner	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_mucka	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_pluskal	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_sperka	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_pc_vecerka	Wifi/MQTT	MQTT Broker	147.229.13.92
C306	SHP6_ventilator	Wifi/MQTT	MQTT Broker	147.229.13.92
C304	Gateway C304	Wifi/UDP	Home assistant	192.168.11.10
C307	Gateway C307	Wifi/UDP	Home assistant	192.168.11.10
C308	Gateway C308	Wifi/UDP	Home assistant	192.168.11.10
C310	Gateway C310	Wifi/UDP	Home assistant	192.168.11.10
C306	C306 Air Quality Monitor 1	Wifi/UDP	Home assistant	192.168.11.201
C306	C306 Air Quality Monitor 2	Wifi/UDP	Home assistant	192.168.11.202
C306	C306 Air Quality Monitor 3	Wifi/UDP	Home assistant	192.168.11.203
C306	C306 Air Quality Monitor 4	Wifi/UDP	Home assistant	192.168.11.204

Table A.1: Distribution of sensors