CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Telecommunications

Master's Thesis

Router IoT Testbed

Nicola Zaru

August 2020
Supervisor: Marek Neruda

Acknowledgement

I would first like to thank my thesis advisor Ing. Bc. Mark Neruda, Ph.D. and Ing. Bc. Lukáš Vojtěch, Ph.D. for their valuable advice during the entire master program.

I would like to recognize the assistance of Ing. Pavel Hnyk and Ing. Tomáš Straka with the useful technical consultation.

I wish to express my deepest gratitude to my family, for the support and for being a constant role model during this journey. This accomplishment would not have been possible without them.

I would also like to acknowledge the support of Mrs. Jana Uhlířová for her valuable advice on the time management and writing methodology and all the people who have contributed to this goal.

I dedicate this achievement to Antonio Zaru and Giovanni Zaru, brothers, traveling companions. Let this be the beginning of a new journey.

Declaration

I declare that I have prepared the submitted work individually with the only help of the supervisor and that I have listed all the information sources used in this paper. I also declare that I have no objection to borrowing or disclosure of my master thesis or part of it, with the approval of the department.

In Prague 10. 08. 2020

. . . .

Abstrakt / Abstract

Tato práce se zaměřuje na návrh a implementaci pracovního prostředí pro testování provozních parametrů přenosového kanálu IoT routeru. Teoretická část práce se zaměřuje na popis hardwaru a softwaru použitého při různých testech. Praktická část popisuje navržený testovací software (GUI), který se používá také k provádění měření s ohledem na přenosový kanál a výkon desky při použití různých technologií IoT.

Klíčová slova: Testbed IoT routeru, propustnost, Internet věcí, GUI, Technologie internetu věcí.

This work focuses on the design and implementation of a workbench environment to test the operational parameters of the transmission channel of the IoT Router. The theoretical part of this thesis focuses on the description on the hardware and software used during the various tests. The practical part describes the designed testbed software (GUI) also used to carry out measurements regarding the transmission channel and the performance of the board when different IoT technologies are used.

Keywords: IoT Router Testbed, throughput, Internet of Things, GUI, IoT technologies.

Contents /

Tables / Figures

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Zaru Nicola**	Personal ID number:	**464463**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Telecommunications Engineering**		
Study program:	**Electronics and Communications**		
Specialisation:	**Communication Systems and Networks**		

II. Master's thesis details

Master's thesis title in English:

IoT Router Testbed

Master's thesis title in Czech:

Testbed IoT routeru

Guidelines:

Design and build a workplace for testing selected operational parameters of the transmission channel for various combinations of available IoT router technologies. Consider transferring data from sensors and actuators with cloud storage. Also consider your own server repository for repeatability of measurement results.

Bibliography / sources:

[1] BRADNER, S.; McQuaid, J. RFC 2544. Benchmarking Methodology for Network Interconnect Devices. Available at: https://tools.ietf.org/html/rfc2544 [on-line]

[2] ITU-T Recommendation Y.1564. Ethernet service activation test methodology. ITU, availabla at: https://www.itu.int/rec/T-REC-Y.1564/en [on-line]

Name and workplace of master's thesis supervisor:

Ing. Marek Neruda, Ph.D., Department of Telecommunications Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **08.01.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Marek Neruda, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student's signature

Chapter 1
Introduction

The concept of the Internet, as we know it, is mutating incredibly fast. Nowadays an enormous amount of data is generated daily from people, animals and objects all around with the goal to improve the quality of life. Billions of devices have access to the internet and are already able to communicate with each other autonomously. In the next few years, even a higher number of objects is expected to join the Internet of Things. With new technologies developed every day and their exponential growth, we are facing new challenges involving security, resource consumption, compatibility and many other. The possibility to combine different technologies to address a problem often represents a big advantage. Such a feature is provided by the IoT Router developed by a team of engineers at the Faculty of Electrical Engineering in Prague and tested during the practical part of this work. To ensure a proper functionality of the network it is connected to, the capabilities and limits of the IoT router need to be explored. Since it is a custom hardware design, a series of tests is carried out with the intent to monitor its networking capabilities, the correct functionality of the different IoT technologies embedded in it (IQRF, BLE, LoRa, NB-IoT and others) and the performance of CPU and RAM when these technologies are used. Different software solutions are selected as testing tools, among which Node-red and vnstat involved in the monitoring of TCP/IP traffic. Moreover a Graphical User Interface is presented, designed as well during this work to complement the rest of the software during the tests. Its purpose is to provide an easy way to build and manage services using the aforementioned IoT technologies. During the theoretical part of this work, the description of the IoT router is provided, along with a brief summary of the supported technologies and their main characteristics. As regards the practical part, different measurements are carried out in a specific order: first the Ethernet and Wi-Fi traffic is explored with respect to the physical limit imposed by the hardware, then the individual technologies are tested along with the analysis of their impact on the CPU, RAM and generated traffic. In the last part of this work, a combination of technologies and two database solutions are implemented. Finally, a summary of the results is provided to have a clearer overview of the capabilities of the IoT Router.

Chapter 2
Software used

This chapter is a general overview of the software and tools used during the practical part of this work. The Graphical User Interface (GUI) specifically designed for this project is described instead in chapter 4.

2.1 Node-RED

Node-RED is an open-source visual programming tool built on top of Node.js platform, mainly used for wiring together hardware devices, APIs and online services [1]. It is based on the logical connections between nodes, where each node has a specific functionality. It represents a great option to manipulate hardware on low-cost devices such as the Raspberry Pi. For this work, the latest versions are used:

- Node-RED version - 1.0.6
- Node.js LTS version - 12.16.2

This software is used to test and control the hardware modules described in the next chapter.

2.2 Iperf

Iperf is a tool for active measurements of the maximum achievable bandwidth on IP networks. Iperf supports tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6). For each test it reports the bandwidth, loss, and other parameters. For the practical part of this work, the version v2.0.9 of the software is used [2].

2.3 Microsoft Azure SQL database

Microsoft Azure SQL database (formerly SQL Azure, SQL Server Data Services, SQL Services, and Windows Azure SQL Database)[3] is a managed cloud database (PaaS) provided as part of Microsoft Azure. Microsoft Azure SQL Database includes built-in intelligence that learns app patterns and adapts to maximize performance, reliability, and data protection. Key capabilities include[3]:

- Continuous learning of your unique app patterns, adaptive performance tuning, and automatic improvements to reliability and data protection
- Scaling as needed, with virtually no app downtime
- Management and monitoring of multitenant apps with isolation benefits of one-customer-per-database
- Leverage open source tools like cheetah, sql-cli, VS Code and Microsoft tools like Visual Studio and SQL Server Management Studio, Azure Management Portal, PowerShell, and REST APIs

- Data protection with encryption, authentication, limiting user access to the appropriate subset of the data, continuous monitoring and auditing to help detect potential threats and provide a record of critical events in case of a breach

For the purposes of this work, the MS Azure SQL database is used to test the connection of the IoT Router to cloud services in order to store data collected from sensors.

2.4 Influx DB

InfluxDB is an open-source time series database (TSDB) developed by InfluxData[4]. It is written in Go[5] and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. It also has support for processing data from Graphite[6]. In this work it is used as a local database solution for the IoT Router.

2.5 vnstat

vnStat (view network statistics) is a network utility for the Linux operating system. It uses a command line interface. vnStat command is a console-based network traffic monitor. It keeps a log of hourly, daily and monthly network traffic for the selected interface(s) but is not a packet sniffer. The traffic information is analyzed from the proc file system, that way vnStat can be used even without root permissions[7]. In this context it is used to monitor the Ethernet traffic generated by the IoT Router. It measures in real-time both the outgoing and incoming traffic in Kbps and number of packets per seconds.

2.6 Wireshark

Wireshark is a free and open-source packet analyzer[8]. It is used for network troubleshooting, analysis, software and communications protocol development, and education. It is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets. It runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows.

Wireshark has a rich feature set which includes the following[9]:

- Deep inspection of hundreds of protocols, with more being added all the time
- Live capture and offline analysis
- Multi-platform: Runs on MS Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, and many others
- Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility
- Rich VoIP analysis
- Read/write many different capture file formats: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network
- Capture files compressed with gzip can be decompressed on the fly
- Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others (depending on the platform)
- Decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2

- Output can be exported to XML, PostScript®, CSV, or plain text

Within this work, the Wirehark software is used to capture UDP packets sent to a local server.

2.7 Picocom

Picocom is a minimal dumb-terminal[10] emulation program. It was designed to serve as a simple, manual, modem configuration, testing, and debugging tool[11]. It has also served as a low-tech `terminal-window` to allow operator intervention in Point to Point Protocol (PPP) connection scripts[12]. In this context, it is used to communicate with hardware modules connected to the IoT Router via serial port. The version 1.7 of the software is used in this work.

Chapter 3
IoT Router

In this chapter, the IoT Router is described together with the IoT technologies that it can support, which are widely used in modern LPWAN/LPN (Low Power Wide Area Network / Low Power Network). It has been designed by a team of engineers at the Czech Technical University in Prague and its networking capabilities and performance will be tested during the practical part of this work.

3.1 Hardware overview

The IoT Router, Fig. 3.1, as per description [13], is a compact mobile device with Ethernet and wireless connectivity. Designed for smart cities, Industrial 4.0 applications and many other situations which rely on wireless sensors technology. This IoT Router combines wireless sensor networks (WSN) and LPWAN technology using well known IoT solutions, such as IQRF, BLE, LTE-NB IoT, LoRa and so on. The heart of the IoT Router is the Raspberry PI Compute Module 3 with the Quad-Core ARM Cortex-A53 BCM2837 chip. The main features are listed below in Table 3.1. Thanks to its modular slot design, it can quickly host various wireless technologies on regional RF bands and wireless communication protocol.

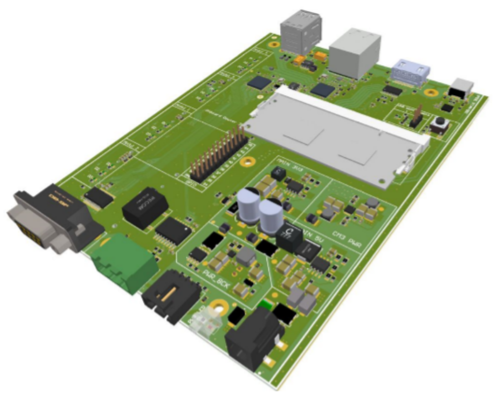


Figure 3.1. Top view of the IoT Router.

Table 3.1. Features of the IoT Router.

Feature	Details
Processor	Quard-Core ARM Cortex-A53 BCM2837, 1,2 GHz CPU
RAM	1 GB
Internet	1 x LAN 10/100Mbps Ethernet port
Serial peripherals	1 x RS232, 1 x RS485, 1 x I2C, 2 x USB-A, 1 x USBmini-A
Add-on modules	3 x miniPCI-e half a 1 x miniPCI-e full
Power	input 7 - 30 V, nominal 12 V 1 A, battery backup
Dimensions	160 x 120 mm

On the back side, Fig. 3.2, the IoT Router is equipped with three miniPCIe-half [14] and one miniPCIe-full slot. Any standard/custom module can therefore be tested and/or implemented in a service, provided that it supports this type of physical connection. In order to test the correct functionality of the hardware and to estimate its networking capabilities, the following modules will be used during the tests:

- NB-IoT BG96
- Bluetooth LE based on BM70BLE01FC2
- IQRF module based on TR-76D
- LoRa module based on SX1272 chip
- Dallas DS18B20 temperature sensor
- Relay module

As shown in Fig. 3.2, some modules have their own MiniPCIe slot on the back side of the IoT Router, while the sensor DS18B20 and the relay are connected to the front side directly to the GPIO header. Besides the modules, there are two more elements: slot number 5 is reserved for a microSD card, useful to expand the internal memory of the IoT Router, but not required for the purposes of this work, while the slot number 6 is designed to host the SIM card, which is necessary to operate with the NB-IoT or LTE module.

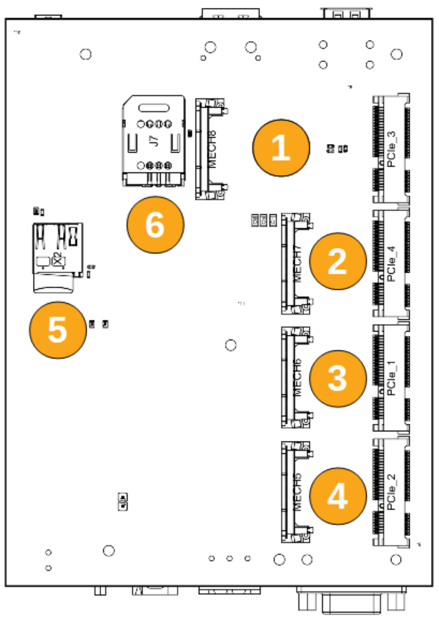


Figure 3.2. Back side of the IoT Router with MiniPCIe slots.

- (1) Slot for the NB-IoT / LoRa module
- (2) Slot for the BLE module
- (3) Empty slot for custom miniPCIe module
- (4) Slot for the IQRF module
- (5) Slot for the SD card
- (6) Slot for the SIM card

3.2 Block scheme

Figure 3.3 provides an overview of the logical connections between the IoT Router and the other components. Detailed hardware schematics are provided by the manual `IoT Router Schematics` [15]

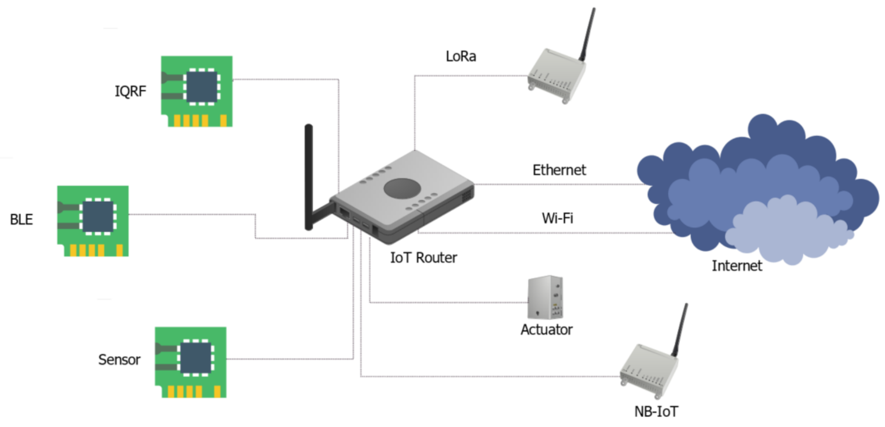


Figure 3.3. Logic connection between the IoT router and the IoT modules.

Despite any of the modules could be used as source of data, for the purposes of this work the main input sources are IQRF, BLE and digital sensors. On the other side, the Ethernet, Wi-Fi NB-IoT, LoRa technologies and the actuator are used as outputs. The IoT Router can also exchange data via RS232, RS485, I2C and USBmini-A. Next part aims to describe more in details what these technologies are and their main features.

3.3 Available IoT technologies

This section provides an overview of the technologies commonly used in wireless and sensor networks, with summary information about the add-on modules with which the IoT Router is equipped. Each of them is described by a picture (if available) and a table containing the main properties. The last part of this section is dedicated to a key feature of many IoT systems: the cloud storage.

3.3.1 NB - IoT

Th IoT Router is equipped with a miniPCIe full-height slot offering two possibilities for LTE communication:

- LTE EC20EA-MiniPCIe
- NB-IoT BG96 MiniPCIe

The Quectel EC20EA [16] is multi-mode LTE module which can provide communication over 4G/LTE networks. It offers a large variety of protocols and backward compatibility with UMTS and GSM/GPRS networks.

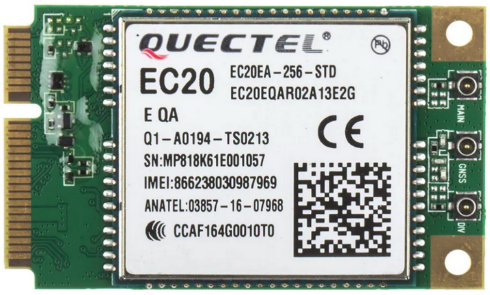


Figure 3.4. EC20EA module[17].

Table 3.2. Features of the EC20EA module.

Element	Features
Name	Quectel EC20EA
Standards	FDD-LTE, DC-HSPA
	HSPA +, HSDPA
	HSUPA, WCDMA
	TD-SCDMA, CDMA
	EDGE and GPRS
Optional	GPS / GLONASSG
Dimensions	51 x 30 x 4.9 mm

The Quectel NB-IoT BG96 is an LTE Cat M1/Cat NB1/EGPRS module [18] offering maximum data rates of 375 kbps downlink and uplink. It features global frequency bands, ultra-low power consumption, and is compatible with Quectel LTE Standard module EG91/EG95, LPWA module BC95-G/BG95, UMTS/HSPA module UG95/UG96 and GSM/GPRS module M95.

Figure 3.5. BG96 module.

Table 3.3. Features of the BG96 module.

Element	Features
Name	Quectel NB-IoT BG96
Standards	Cat.M1 / Cat.NB1
	EGPRS
Cat NB1	Max. 32 kbps (DL)
	Max. 70 kbps (UL)
Consumption	Power Saving: 10 μA
	Idle State: 15 mA
	Sleep State: 1.96/1.1 mA
Dimensions	51 x 30 x 4.9 mm

3.3.2 Bluetooth Low Energy

The module is based on the BM70BLE01FC2 chip providing Bluetooth SIG v5.0 [13]. The chip is certified for FCC, ISED, MIC, KCC, NCC, and SRRC radio standards. The module supports UART interface, 3 channel PWM output, precision temperature sensor (PTS), 12-bit ADC and 18 GPIO.

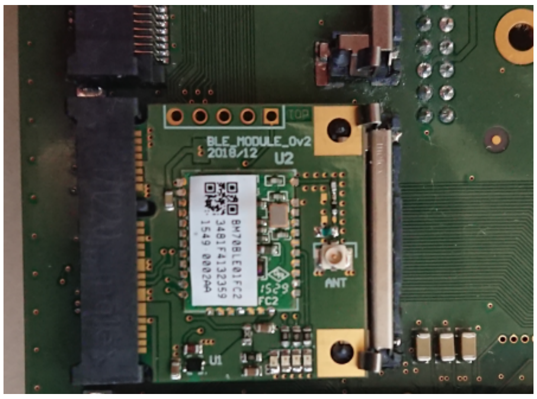


Figure 3.6. BLE module.

Table 3.4. Features of the BLE module.

Element	Features
Model	BM70BLE01FC2
ISM band	2.402 GHz to 2.480 GHz
Channels	40 (0 to 39)
Consumption	15 mA
Key features	Battery status monitoring
	AES128 encryption
	Temperature sensor
Dimensions	26.8 x 30 mm

3.3.3 IQRF

The module is based on the transceiver TR-76D operating in the 868 MHz and 916 MHz bands, i.e., in the unlicensed ISM (industrial, scientific and medical) frequency bands. Its highly integrated design is quick to use. It includes MCUs, RF circuits, serial EEPROM and optional antenna, no additional external components required.

It features extremely low power consumption which makes it suitable for applications running on battery. It gives the best when deployed in a network using MESH topology.

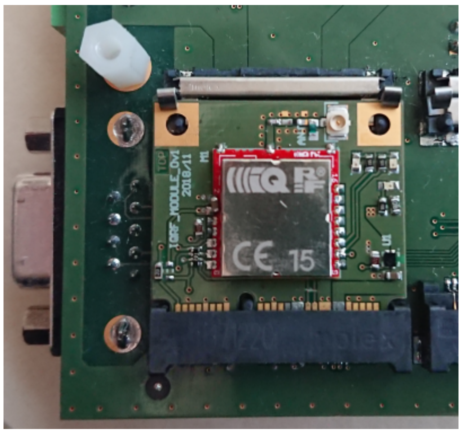


Figure 3.7. IQRF module.

Table 3.5. Features of the IQRF module.

Element	Features
Model	TR - 76D
ISM band	868 MHz
Data transfer rate (radio)	19.8 kbps
TX power	10 dBm (typical)
	Idle State: 15 mA
Key features	GFSK modulation
	AES128 encryption
Dimensions	26.8 mm x 30 mm

3.3.4 LoRa

The selected LRWCCx-MPCIE [19] is from the LoRaWAN family of concentrator cards with a mini PCIe connector using a USB interface for communication. This certified concentrator is based on Semtech SX1301 or SX1308 IC see Fig.2. It can work on linux devices that support USB controller and networking stack (TCP / IP and UDP). The software that takes care of controlling the card is the HAL driver [20], which runs in the user environment (user space). Furthermore, a packet forwarder [21] is needed to send the received data from the sensors to the LoRa network server. Both are available on the LoRa network project, which is Open Source Software. There is also the possibility of uploading Firmware (FW) for the MCU, which allows connection via UART interface. The variant selected for this project is sx1301, 868MHz (European band + outdoor IC).

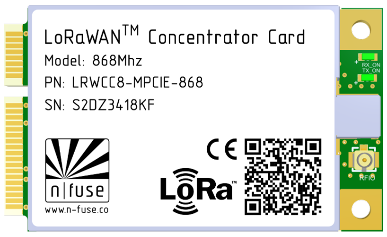


Figure 3.8. LoRaWAN USB module[22].

Table 3.6. Features of the LoRa module.

Element	Features
Model	LRWCCx-MPCIE
ISM band	868 MHz
OUT power	up to 21 dBm
Key features	LoRa® modulation
	standard mini PCIe
Dimensions	51 x 30 x 4.8 mm

3.3.5 Temperature sensor DS18B20

The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. In addition, the DS18B20 can derive power directly from the data line (`parasite power`), eliminating the need for an external power supply [23].

Figure 3.9. Waterproof temperature sensor[24].

Table 3.7. Features of the temperature sensor.

Element	Features
Model	DS18B20
Range	-10°C to +85°C
Accuracy	0.5°C
Key features	Parasitic Power Mode
	1-Wire® Interface
	Digital output
Power supply	3V to 5.5V

3.3.6 Relay module

This relay module is used in the practical part of this work as an example of actuator. It can turn on and off high current loads (fans, lights, etc.) based on the environmental quantities collected by the IoT Router. It can be controlled via GPIO directly from the IoT Router since it provides on-board optical isolation [25].

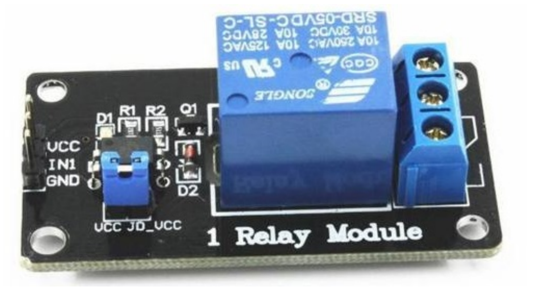


Figure 3.10. Relay module[25].

Table 3.8. Features of the relay module.

Element	Features
Output	AC250V 10A; DC30V 10A
Excitation	15 to 20 mA
Key features	Optical isolation

3.3.7 Cloud storage

Two different databases are used in this project: the MS Azure SQL database which represents the cloud solution and the Influx database which is used as local storage. The Azure SQL database is a fully managed Platform as a Service (PaaS) Database Engine [26] that handles most of the database management functions such as upgrading, patching, backups, and monitoring without user involvement. This database contains the data collected from the DS18B20 temperature sensor. The InfluxDB (as a cloud solution), is a fast, elastic, serverless real-time monitoring platform, dashboarding engine, analytics service and event and metrics processor [27]. Data is organized based on timestamps. For repeatability of the measurements, a local instance on the InfluxDB is used on the IoT router instead of the whole cloud solution.

Table 3.9. Example of temperature entries with the Influx database.

Timestamp	Location	Value
13941981987198924	Office 1	24
13941981900424852	Office 2	25
13941981988928334	Office 1	25

Chapter 4
Design of testbed

This chapter introduces the practical part of this work, the tasks to address and their implementation. The goal is to verify the correct functionality of the IoT Router peripherals, whether it is able to control the modules described in the previous chapter and study their impact on its performance. For the sake of clarity, the tasks to complete are divided into two groups:

- Scenarios involving TCP/IP protocol.
- Bridge between technologies (without TCP/IP).

The first group represents a common situation in IoT systems, where data is collected by a central unit and sent to the public network via TCP/IP protocol. To test how the IoT Router serves this purpose, the following tasks are considered:

- 1) Verify maximum throughput of Ethernet and Wi-Fi interfaces (uplink and downlink) when TCP and UDP traffic is used.
- 2) Test Jitter and total number of datagrams lost in case of UDP traffic.
- 3) Verify correct functionality (no HW errors), maximum throughput of the custom IQRF and BLE modules and their impact on the CPU and RAM.
- 4) Verify the correct functionality of the IoT router as a LoRaWAN gateway.
- 5) Test connection to local and remote databases.
- 6) Verify the correct functionality (no HW errors) and data transmission in uplink and downlink with the NB-IoT module. Verify its impact on the CPU and RAM of the IoT Router. Measure the latency.
- 7) Verify the impact of IQRF and BLE technologies combined in the IoT Router in terms of CPU and RAM usage, CPU temperature and Ethernet traffic.

The second group represents those situations in which the collected data is used to take actions on the field. A classic example is a cooling system driven by the temperature value of a sensor. The data is therefore collected and used locally. Although in real situations these scenarios are often merged together, in this paper they are treated separately to keep things clear. The following tests are planned for this scenario:

- 8) Verify the possibility to collect data from sensors directly connected to the GPIO of the router using a Dallas DS18B20 temperature sensor.
- 9) Verify the possibility to control actuators directly connected to the router using a relay module.
- 10) Test the functionality of the IoT router to bridge data between different IoT technologies (IQRF to BLE and vice versa).

The tasks mentioned above are carried out one by one using the software described in chapter 2.1 and the GUI specifically designed as part of this work. In particular with the Iperf software the tasks 1 and 2 are addressed while task 4 is carried out installing the official software to implement LoRa gateways [20]. The remaining assignments are addressed with the GUI and Node-RED. Next sections describe the GUI and how it is bound to Node-red, then each task is presented in more details along with its implementation and finally the results are summed up.

4.1 Software solution

The software developed for this project is designed as a management tool for IoT services. In this context, a service is defined as a process in which the IoT Router collects or receives data from specific input hardware peripherals, processes it based on user-defined rules and forwards it to selected output peripherals. With the intent to minimize the complexity of the interface, and to release the user from any form of coding, the testbed workplace is designed in the form of a GUI with an intuitive drag & drop interface directly inspired by the Node-RED software [1]. In the next part of this work, the architecture of the proposed software is presented, Fig.4.1, followed by the description of the GUI itself.

4.2 Architecture

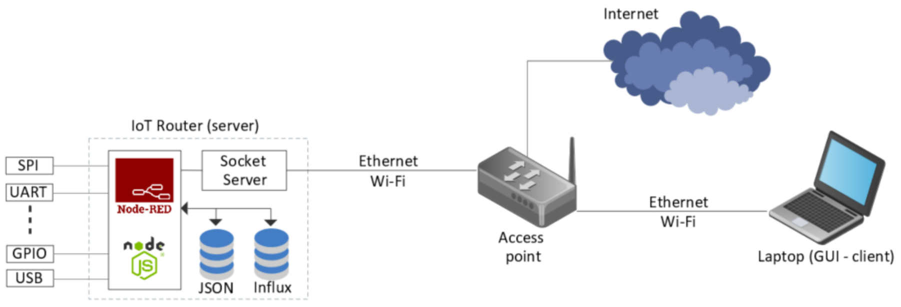


Figure 4.1. Software components.

The software solution consists of three main components:

- Node-RED
- GUI
- Local storage

In this context, the IoT Router is the server and the laptop is the client, which access the GUI locally via the access point. Node-RED communicates with the laptop (GUI) via web sockets. Its main purpose is to execute actions on the hardware based on the messages received from the GUI. Typically these actions consist on exchanging data with the modules, control GPIO pins etc. The GUI on the other hand, has been designed to provide the user with a simple interface to manage the services running on Node-red, i.e. without having to deal with all the nodes and flows deployed in Node-RED, which can start loosing intuitiveness when the flows scale up. The GUI is served from the IoT Router and accessible on the local network from a web browser at the address `iot-router:8080`.

Last but not least, this software solution is provided with two types of local storage. The first one is to keep track of all the settings and parameters set by the user when creating or editing a service. Since Node-RED stores the flows information locally using JSON files, the same approach has been taken regarding the services created with the GUI. The second storage is an Influx database which offers the possibility to store data collected from the IoT Router with modules/sensors. This data can later on be extracted to be analyzed with other platforms (e.g. Grafana) or simply for repeatability of the measurements.

The main advantages of this software are:

- Easy way to control/test the custom hardware modules presented in chapter 3.
- Compatible with many ARM based boards.
- Measurement and evaluation of the CPU and RAM usage with a default interval of five seconds (adjustable within the Node-RED flow called `Performance`).
- Real-time measurement of throughput both for Ethernet and Wi-Fi connections.

4.3 GUI

In this section, the Graphical User Interface is described together with its impact on the board. Generally speaking, it is a web application running on the IoT Router and accessible from the local network to which the IoT Router is connected. As mentioned before, its main purpose is to provide an easy way for the user to collect data from input sources and forward it to selected output peripherals. Along with this, it measures the performance of the board in real time. It is built on top of Node-RED, meaning that the GUI controls the Node-RED flows which in turn control the hardware. Figure 4.2 shows a schematic view of the principle behind the software. The `Socket server` has the purpose to communicate with the GUI and transparently forward the messages already traveling through the flow. Each branch of the flow has a specific functionality (e.g. get data from sensors, send data to a database, edit services etc.). The `Router node` simply decides, based on the rules specified by the user with the GUI, to which branch every message has to go. In this way it is possible to create custom scenarios using different combinations of technologies.

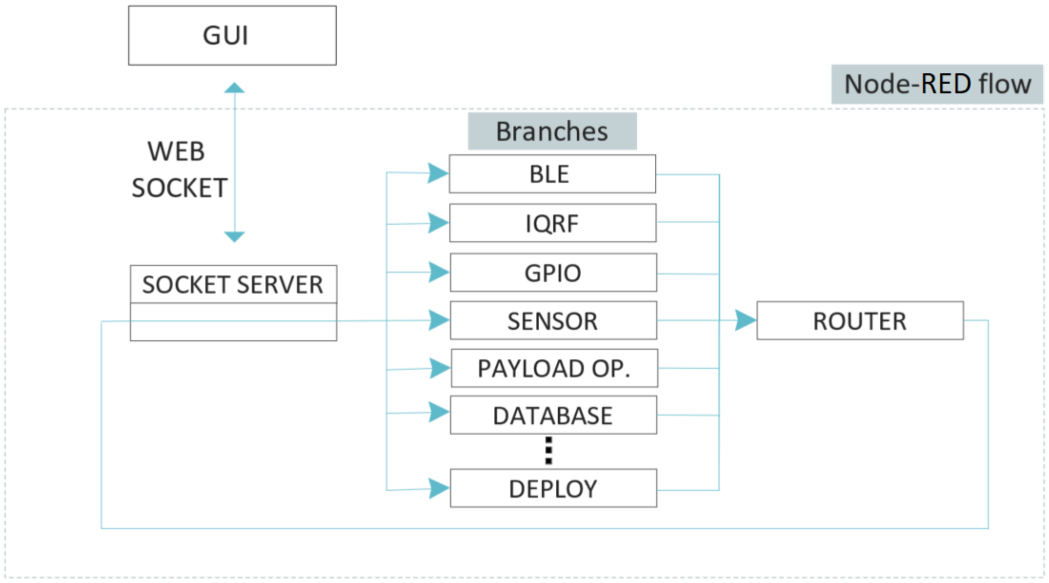


Figure 4.2. Routing of messages between IoT technologies.

The GUI is organized into three main pages:

- Dashboard
- Editor page
- Charts page

4.3.1 Dashboard

The dashboard, Fig. 4.3, is the main page of the web application. On the left side it shows a menu to navigate among the pages, change settings or shut down the board. In the center of the page there are listed the services created by the user (hereinafter the service tags). Each of them contains basic information about its status (red → stopped, green → running) and the amount of messages/sec involved. On the right side, the dashboard provides basic information about the IoT Router and its performance. From top to bottom it shows: actual time, up-time of the board, the number of deployed services (green: active, white: total, yellow: not active), the amount of used RAM (%), the CPU usage (%) and the temperature of the CPU.

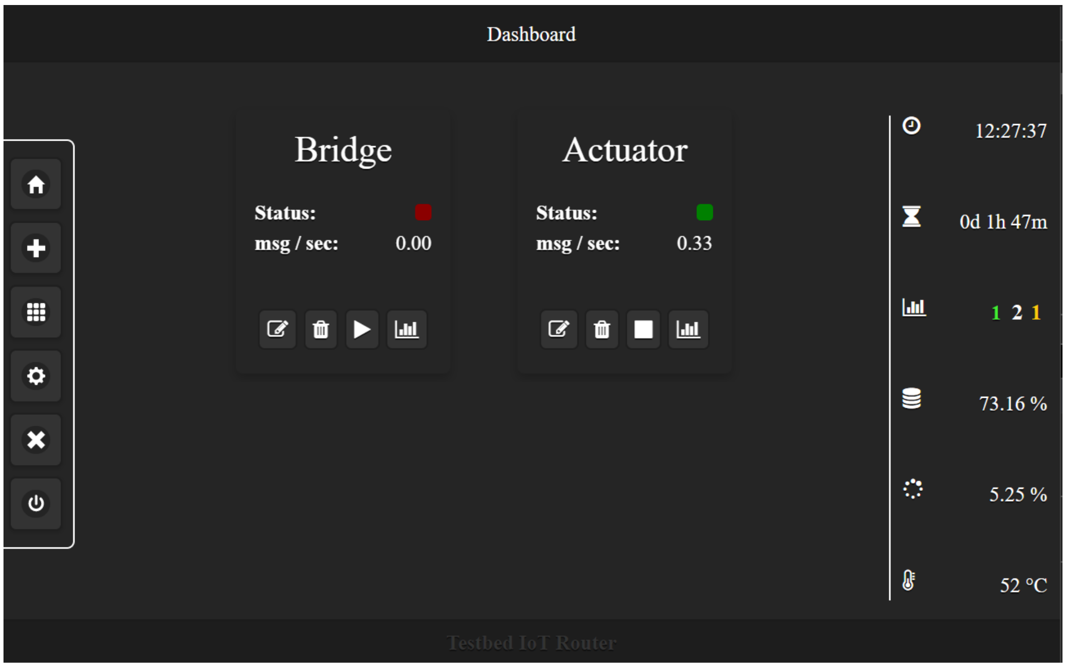
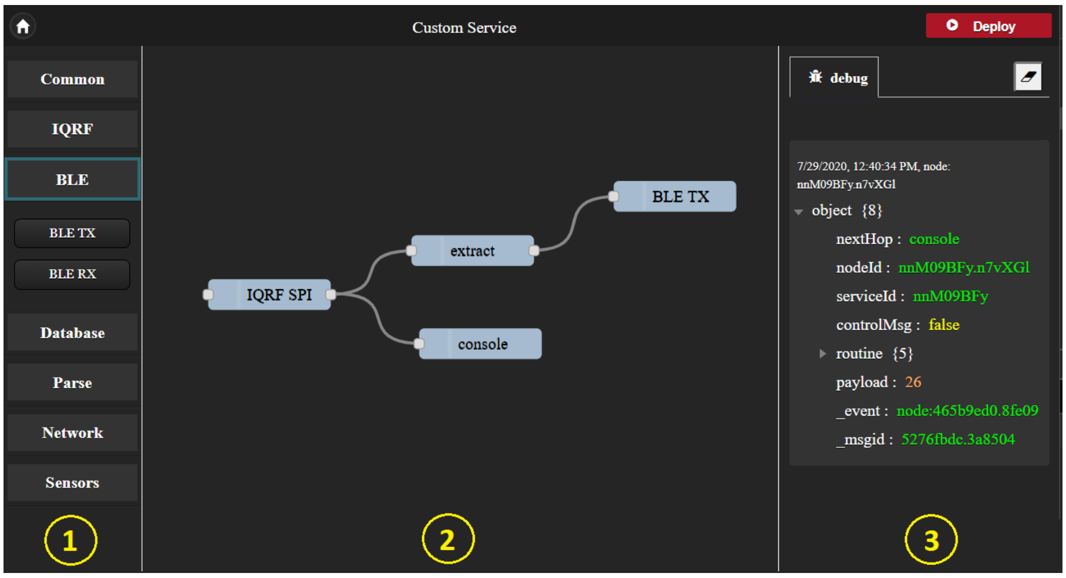


Figure 4.3. Dashboard of the web application.

4.3.2 Editor

The editor page, Fig. 4.4, provides an easy way to create and edit services. It is accessible from the dashboard menu with the `Plus` button or directly from the service tag (edit button). The layout of the editor is directly inspired by the Node-RED editor, in fact, it offers a drag & drop interface that gives the user enough flexibility to control the IoT technologies discussed in chapter 3. On the left side of the editor there is a palette containing the nodes to be dragged to the canvas. Nodes have different functionalities which include sending or receiving data to / from the modules, communicate with databases, log messages etc. They can be connected with each other forming a flow, once the flow is deployed (pressing the red button "Deploy" in the top-right corner) it becomes a service available in the dashboard. On the right side, the editor offers a debug console where messages sent from any node can be inspected. This feature is accessible using the `console` node available in the palette of the editor. It was particularly helpful during the development of the web application but it comes in pretty handy even during the design of new services.



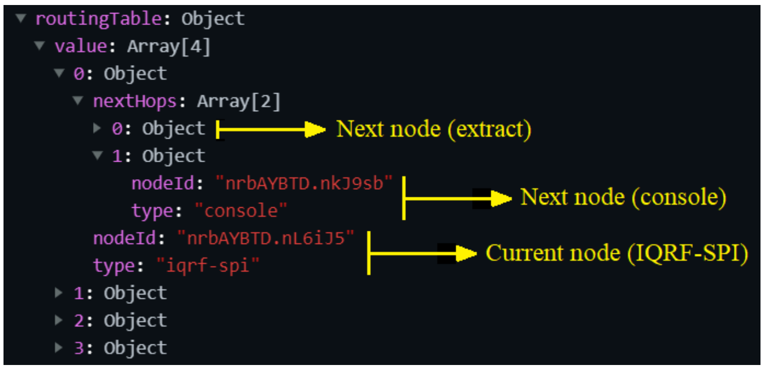
```
7/29/2020, 12:40:34 PM, node:
nnM09BFy.n7vXGl
▾ object {8}
    nextHop : console
    nodeId : nnM09BFy.n7vXGl
    serviceId : nnM09BFy
    controlMsg : false
  ▸ routine {5}
    payload : 26
    _event : node:465b9ed0.8fe09
    _msgid : 5276fbdc.3a8504
```

Figure 4.4. Editor page: node palette (left), canvas (middle), debug console (right).

When a service is deployed, a routing table is generated. For each node in the service, it contains the ID and type of the next node, as shown in Fig. 4.5. This information is used by the `Router` node to forward each message to the proper `Branch`, Fig. 4.2. Each message contains four important properties:

- serviceId: the ID of the service it belongs to.
- nodeId: a unique ID for each node.
- nextHop: the branch to which the message has to go next.
- payload: the payload of the message.

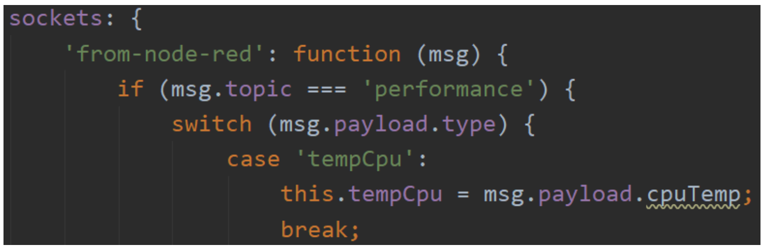
The property `nodeId` and `serviceId` are used by the `Router` node to query the routing table and attach the proper value (the `type` of the next node) to the `nextHop` property. For instance, in Fig. 4.4, two messages are generated from the IQRF SPI node, one with the `nextHop` property set to 'extract' and one with the `nextHop` property set to 'console'.



```
▾ routingTable: Object
  ▾ value: Array[4]
    ▾ 0: Object
      ▾ nextHops: Array[2]
        ▸ 0: Object
        ▾ 1: Object
            nodeId: "nrbAYBTD.nkJ9sb"
            type: "console"
        nodeId: "nrbAYBTD.nL6iJ5"
        type: "iqrf-spi"
    ▸ 1: Object
    ▸ 2: Object
    ▸ 3: Object
```

Figure 4.5. Routing table of the service shown in Fig. 4.4.

When a message is intended for the GUI, such as the CPU temperature, Node-RED emits it the via web socket on the topic 'from-node-red'. The client on the laptop, which is listening on that topic, can process the message, Fig. 4.6.



```
sockets: {
    'from-node-red': function (msg) {
        if (msg.topic === 'performance') {
            switch (msg.payload.type) {
                case 'tempCpu':
                    this.tempCpu = msg.payload.cpuTemp;
                    break;
```

Figure 4.6. Client listener for the Node-RED messages.

4.3.3 Charts

The chart page as well as the editor is accessible from the dashboard menu. It provides statistics about the performance of the board and the throughput associated with the network interfaces both in uplink and downlink. This is achieved in Node-RED with the flows shown in Fig. 4.7. For clarity, it is divided into three sections enclosed by colored frames. The red one is responsible for calculating the CPU usage, CPU temperature, RAM usage and represents each of them by a chart. The green one offers an estimate of the bandwidth available in both directions (upstream and downstream), this idea was taken from another thesis work [28]. The accuracy of this measurement has been verified with the tool speedtest-cli available for Raspbian OS. The last part of the flow (yellow frame) is designed to measure the throughput of the Ethernet and Wlan0 (Wi-Fi) network interfaces in both directions (upstream and downstream).

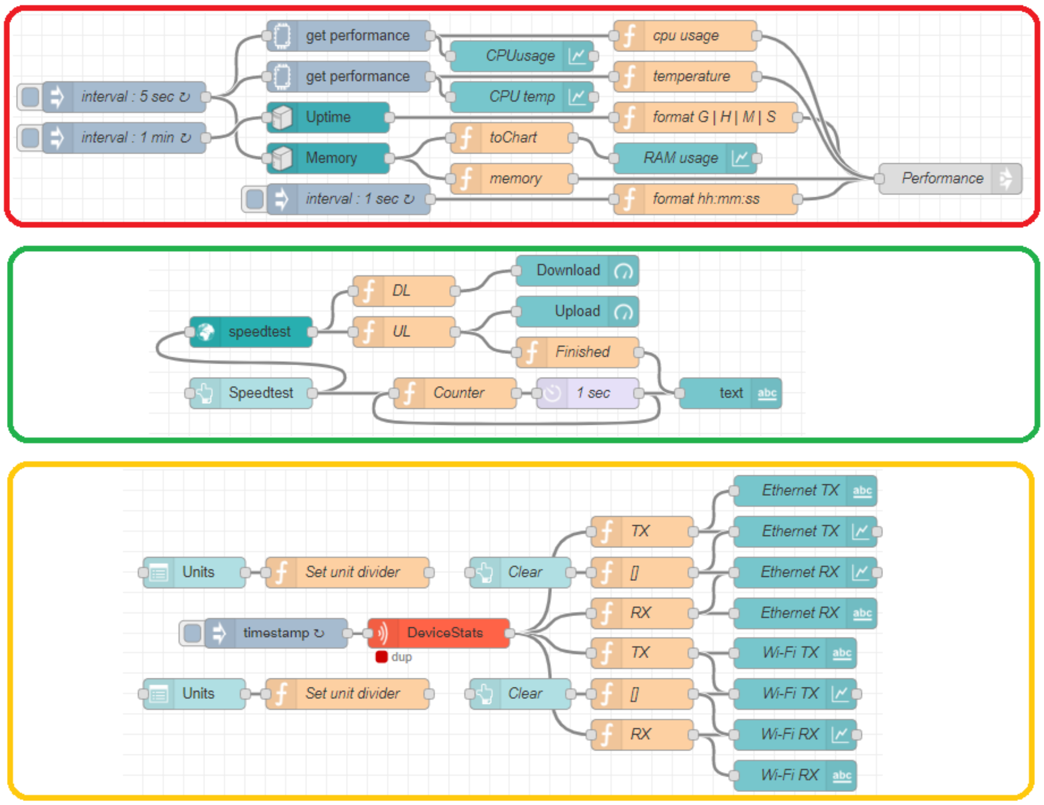


Figure 4.7. Node-RED flows responsible for the charts.

The first group of charts (Fig. 4.8 - top row) shows the CPU usage (%), the temperature of the CPU (°C) and the RAM usage (%). These charts are updated every five seconds and by default display the last 60 seconds of data, although the interval and the time frame can be easily adjusted from the corresponding flow inside Node-RED. The next two rows are dedicated to the throughput of the Ethernet and Wi-Fi link respectively. These values are calculated every second but the data is not displayed immediately, instead it is buffered and the charts are updated every ten seconds (adjustable within Node-RED).

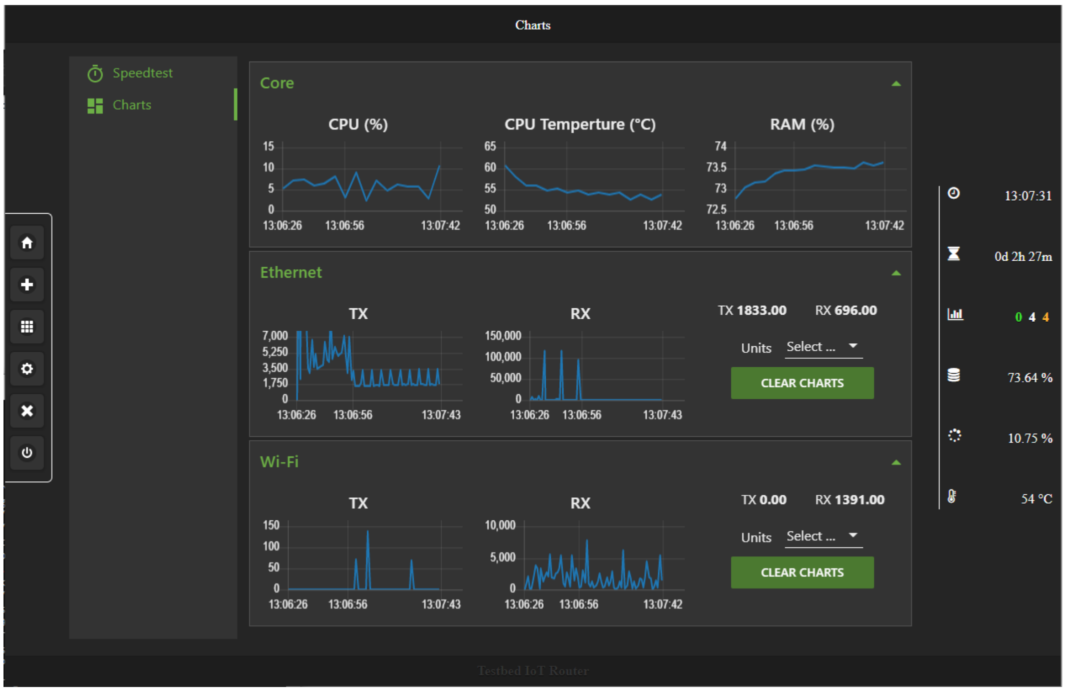


Figure 4.8. Charts page.

The reason behind the time interval of the updates is to lower the impact of the chart widgets on the CPU since the frequent re-rendering of a large number of points is computationally expensive. It also helps to keep the GUI look more fluent. By default the throughput is calculated in bytes/second but it can be adjusted as for the other charts.

4.3.4 Impact on the board

The following paragraphs aim to describe the impact of the GUI and the Node-RED software on the performance of the IoT Router, in particular how much the Ethernet throughput is affected by the socket communication between the GUI and Node-RED. There are three main factors that influence the available bandwidth:

- The charts displayed by the GUI.
- The Node-RED editor.
- The SSH communication with the IoT Router.

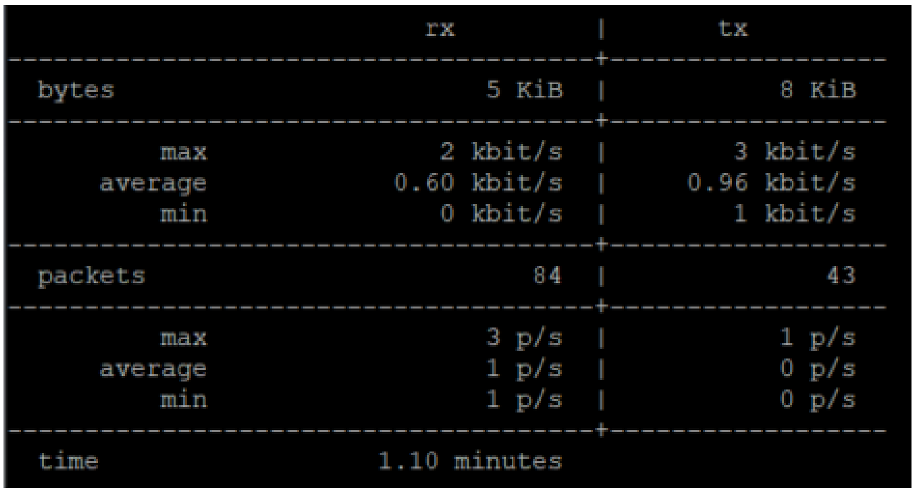
To establish the impact of each factor, the Ethernet traffic is measured using the tool `vnstat` described in Sec. 2.6 under the following conditions:

- Basic case - Node-red running on background, no GUI processes, SSH terminal open on the laptop to receive the real time measurements from vnstat.
- Idle case - Node-red running on background, web page open with the real time charts, SSH terminal open on the laptop to receive the real time measurements from vnstat.
- Editor case - Node-red running on background and Node-red editor page open on the browser, web page open with the real time charts, SSH terminal open on the laptop to receive the real time measurements from vnstat.

For all measurements the command used from the SSH terminal is always the same:

```
sudo vnstat -u -i eth0 --live
```

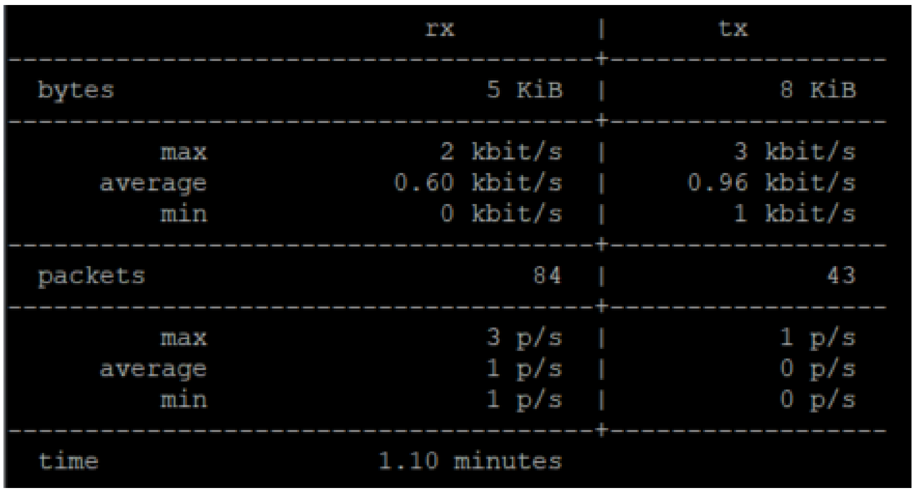
The figure below shows the Ethernet traffic for a period of 70 seconds under the first condition (Basic case). The average throughput for both directions is lower than 1 Kbit/s.



```
                              rx         |         tx
--------------------------------------+------------------
  bytes                       5 KiB  |          8 KiB
--------------------------------------+------------------
          max              2 kbit/s  |       3 kbit/s
      average           0.60 kbit/s  |    0.96 kbit/s
          min              0 kbit/s  |       1 kbit/s
--------------------------------------+------------------
  packets                        84  |             43
--------------------------------------+------------------
          max                 3 p/s  |          1 p/s
      average                 1 p/s  |          0 p/s
          min                 1 p/s  |          0 p/s
--------------------------------------+------------------
  time                 1.10 minutes
```

Figure 4.9. Measurement of the Ethernet traffic for the `basic case`.

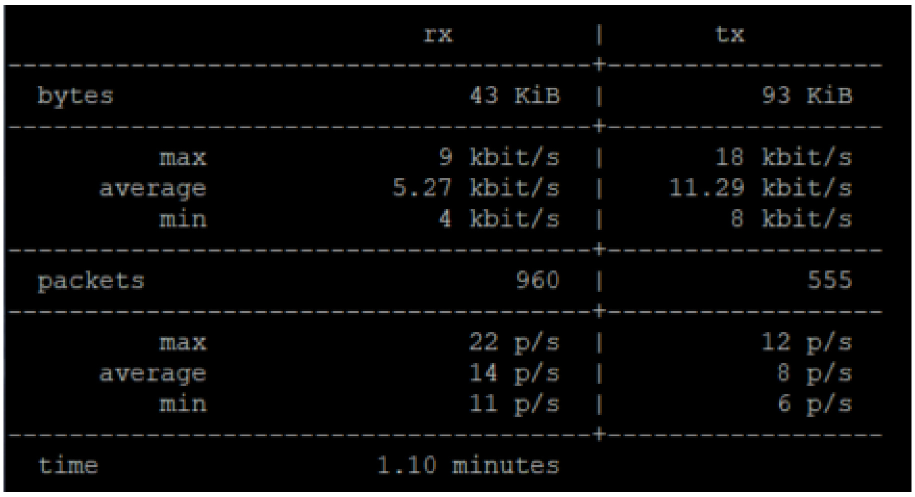
The second test is similar to the previous one with the addition of the GUI chart page open on the web browser. This case is referred from now on as the `idle case` since the GUI is active but no services are running. The traffic is slightly higher as expected, Fig. 4.10. This is due to the periodic exchange of messages required to update the charts. Under this condition, the outgoing traffic is 6.34 Kbit/s (4 packets/sec) while the incoming traffic stands at 2.24 Kbit/s on average (6 packets/sec).



```
                              rx         |         tx
--------------------------------------+------------------
  bytes                       5 KiB  |          8 KiB
--------------------------------------+------------------
          max              2 kbit/s  |       3 kbit/s
      average           0.60 kbit/s  |    0.96 kbit/s
          min              0 kbit/s  |       1 kbit/s
--------------------------------------+------------------
  packets                        84  |             43
--------------------------------------+------------------
          max                 3 p/s  |          1 p/s
      average                 1 p/s  |          0 p/s
          min                 1 p/s  |          0 p/s
--------------------------------------+------------------
  time                 1.10 minutes
```

Figure 4.10. Measurement of the Ethernet traffic for the `idle case`.

The last measurement is again an update of the previous one, this time including the Node-RED editor (not the GUI editor) open in another web page. The Node-RED editor has the higher impact since it continuously updates the status of the nodes in each flow. Since the flows can run on background, there is no reason to keep the Node-RED editor page open during the actual execution of services with the GUI but it is useful to see its impact. On average, the outgoing and incoming traffic respectively is 11.29 Kbit/s and 5.27 Kbit/s.



```
                          rx         |       tx
--------------------------------------+------------------
  bytes                     43 KiB    |          93 KiB
--------------------------------------+------------------
          max              9 kbit/s   |       18 kbit/s
      average           5.27 kbit/s   |    11.29 kbit/s
          min              4 kbit/s   |        8 kbit/s
--------------------------------------+------------------
  packets                     960     |            555
--------------------------------------+------------------
          max              22 p/s     |         12 p/s
      average              14 p/s     |          8 p/s
          min              11 p/s     |          6 p/s
--------------------------------------+------------------
  time                 1.10 minutes
```

Figure 4.11. Measurement of the Ethernet traffic for the `editor case`.

With these considerations, all the upcoming Ethernet traffic measurements are presented as additional traffic to the idle case mentioned above. As regards the CPU usage, the Node-RED editor doesn't affects it considerably as it oscillates around 5%. Its temperature (without heat sink and with the IoT Router inside its metal case) is around 55°C for the idle condition, with a 65% of RAM used. The following sections provide a more detailed description of the tasks described at the beginning of this chapter as well as the necessary information to implement them using the GUI and Node-RED and the auxiliary software.

4.4 Bandwidth measurement

This section is intended to present the hardware and software configuration used to measure the maximum speed achievable by the IoT Router via Ethernet and Wi-Fi interfaces for TCP and UPD data streams. In particular, these tests aim to measures the maximum bandwidth, the jitter and the number of datagrams lost. In order to have accurate measurements during the test, all devices are isolated from the public network.

4.4.1 Software used

This section describes the software setup used during the test. On the laptop is installed Windows 10 while the IoT Router runs on Raspbian with kernel 4.19.66-v7+. On both machines, Iperf v2.0.9, a tool for active measurements of the maximum achievable bandwidth in IP networks, is installed. Iperf supports tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6). For each test it reports the bandwidth, loss, and other parameters. However, the focus of these experiments is on TCP and UDP data streams only.

4.4.2 Ethernet configuration

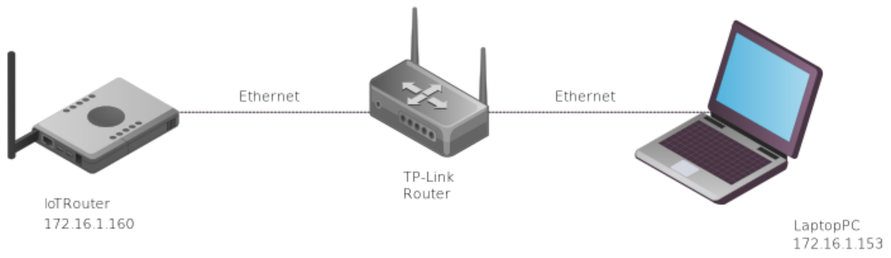


Figure 4.12. Hardware connections for bandwidth measurement via Ethernet cable.

The scheme presented in Fig.4.12 describes how the hardware components are connected. For this scenario, the IoT Router and a laptop are connected to a TP-link router using 2 Ethernet CAT 5E cables supporting 1-Gigabit Ethernet (1Gbps). The table below provides a description of the ports and their IP addresses.

Table 4.1. Ports and IP addresses of the devices used during the test.

Device	Ethernet Port	IP
Laptop HP ZBook 15G3	1 Gbit/s	172.16.1.153
Tp-Link AC1750	2 x 1 Gbit/s	-/-
IoT router	100 Mbit/s	172.16.1.160

Considering this setup, the bottleneck seems to be the 100 Mbit/s Ethernet port of the IoT Router. Therefore, the bandwidth for both upstream and downstream is expected to be lower or equal to 100 Mbit/s.

4.4.3 TCP measurement via Ethernet

The TCP test consists on sending TCP packets between the Iperf terminal on the IoT Router and the Iperf terminal on the laptop in both directions. The first measurement regards the upstream direction, i.e. the IoT Router sends data to the laptop. To start the Iperf server on the laptop, it is sufficient to run the following command in the command line from the directory where Iperf is located:

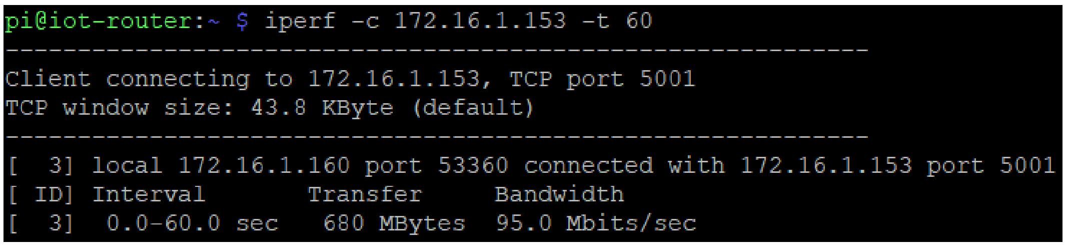
```
iperf -s
```

Once the server is listening, the test can start with the following command from a client terminal of the IoT Router:

```
iperf -c 172.16.1.153 -t 60
```

- Duration: 60 seconds.
- TCP window size: 43.8 KB (default).
- Data transferred: 680 MB.
- Direction: upstream.
- Source IP: 172.16.1.160
- Destination IP: 172.16.1.153
- Achieved bandwidth: 95 Mbit/s

The result, as expected, shows a bandwidth of 95 Mbit/s.



```
pi@iot-router:~ $ iperf -c 172.16.1.153 -t 60
------------------------------------------------------------
Client connecting to 172.16.1.153, TCP port 5001
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.160 port 53360 connected with 172.16.1.153 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   680 MBytes  95.0 Mbits/sec
```

Figure 4.13. Result of TCP upstream test for Ethernet configuration.

The measurement is repeated for the downstream communication (from the laptop to the IoT Router). This time the command to run on the IoT Router is:

```
iperf -s
```

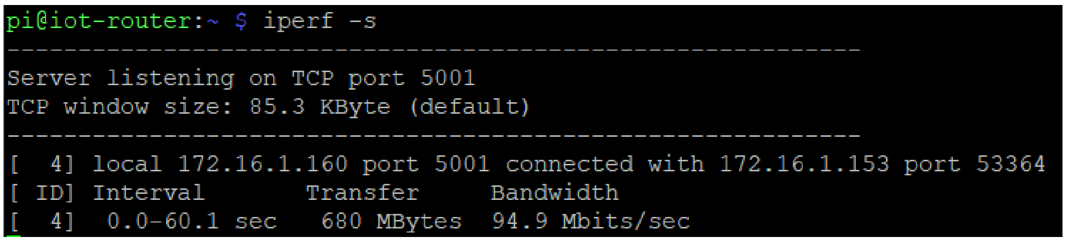
and from the command line of the laptop:

```
iperf -c 172.16.1.160 -t 60
```

Note that now the terminal on the laptop (now the client) is pointing to the IP address of the IoT Router (server).

- Duration: 60 seconds.
- TCP window size: 83.3 KB (default).
- Data transferred: 680 MB.
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.160
- Achieved bandwidth: 94.9 Mbit/s

Similarly as before, the measured bandwidth is 94.9 Mbit/s.



```
pi@iot-router:~ $ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 172.16.1.160 port 5001 connected with 172.16.1.153 port 53364
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-60.1 sec   680 MBytes  94.9 Mbits/sec
```

Figure 4.14. Result of TCP downstream test for Ethernet configuration.

4.4.4 UPD measurement via Ethernet

The procedure to measure UDP data streams is very similar to the one used for TCP. From the laptop the command to run is:

```
iperf -s -u
```

From the IoT Router terminal:

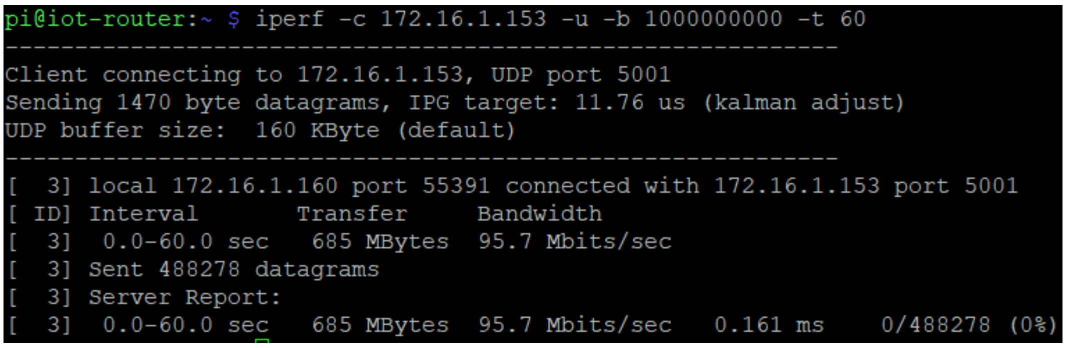
```
iperf -c 172.16.1.153 -u -b 1000000000 -t 60
```

Note the flag -u which refers to the UDP protocol while the flag -b represents the bandwidth the terminal tries to use (1 Gbit/s). This value, higher than the physical limit of 100 Mbit/s, has been chosen on purpose to observe the behavior of the IoT Router.

- Duration: 60 seconds.

- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 685 MB.
- Jitter: 0.161 ms
- Datagrams lost: 0/488278 (0%)
- Direction: upstream.
- Source IP: 172.16.1.160
- Destination IP: 172.16.1.153
- Required bandwidth: 1 Gbit/s
- Achieved bandwidth: 95.7 Mbit/s

As expected, the bandwidth is slightly higher using UDP protocol (95.7 Mbit/s) due to the lack of ACK messages, but still lower than the physical limit of 100 Mbit/s imposed by the IoT Router port.



```
pi@iot-router:~ $ iperf -c 172.16.1.153 -u -b 1000000000 -t 60
------------------------------------------------------------
Client connecting to 172.16.1.153, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11.76 us (kalman adjust)
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.160 port 55391 connected with 172.16.1.153 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   685 MBytes  95.7 Mbits/sec
[  3] Sent 488278 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec   685 MBytes  95.7 Mbits/sec   0.161 ms    0/488278 (0%)
```

Figure 4.15. UDP upstream test with required bandwidth of 1 Gbit/s.

Similarly to the TCP measurement, for the downstream test the commands are swapped. From the IoT Router terminal:

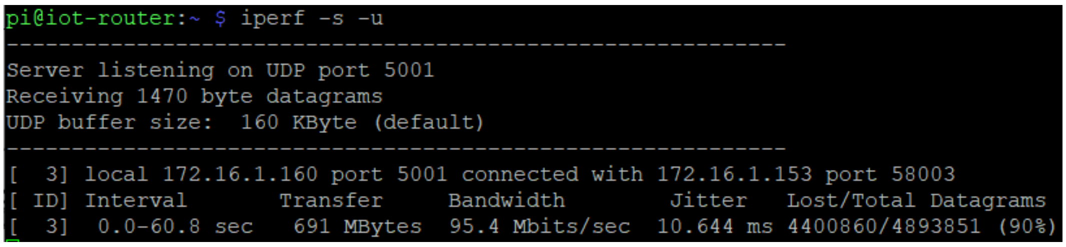
```
iperf -s -u
```

From the laptop:

```
iperf -c 172.16.1.160 -u -b 1000000000 -t 60
```

- Duration: 60 seconds.
- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 691 MB.
- Jitter: 10.644 ms
- Datagrams lost: 4400860/4893851 (90%)
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.160
- Required bandwidth: 1 Gbit/s
- Achieved bandwidth: 95.4 Mbit/s

The achieved bandwidth for downstream seems to be slightly lower, while the jitter is much higher as well as the datagrams lost.

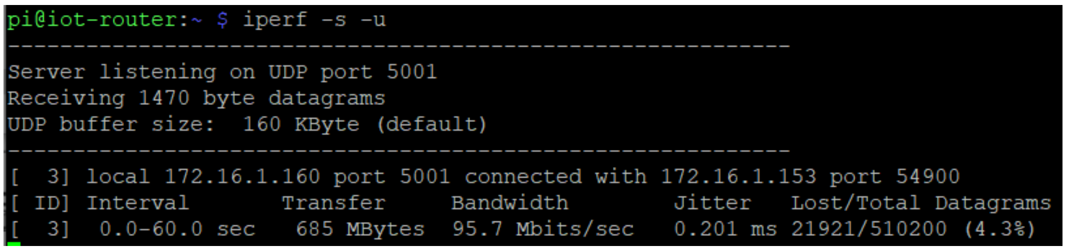


```
pi@iot-router:~ $ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.160 port 5001 connected with 172.16.1.153 port 58003
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-60.8 sec   691 MBytes  95.4 Mbits/sec  10.644 ms 4400860/4893851 (90%)
```

Figure 4.16. UDP downstream test with required bandwidth of 1 Gbit/s.

Repeating the downstream UDP test with different bandwidth flag (100 Mbit/s) the jitter and the number of datagrams lost decrease drastically as expected.

- Duration: 60 seconds.
- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 685 MB.
- Jitter: 0.201 ms
- Datagrams lost: 21921/510200 (4.3%)
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.160
- Required bandwidth: 100 Mbit/s
- Achieved bandwidth: 95.7 Mbit/s



```
pi@iot-router:~ $ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.160 port 5001 connected with 172.16.1.153 port 54900
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-60.0 sec   685 MBytes  95.7 Mbits/sec   0.201 ms 21921/510200  (4.3%)
```

Figure 4.17. UDP downstream test with required bandwidth of 100 Mbit/s.

4.4.5 Wi-Fi configuration

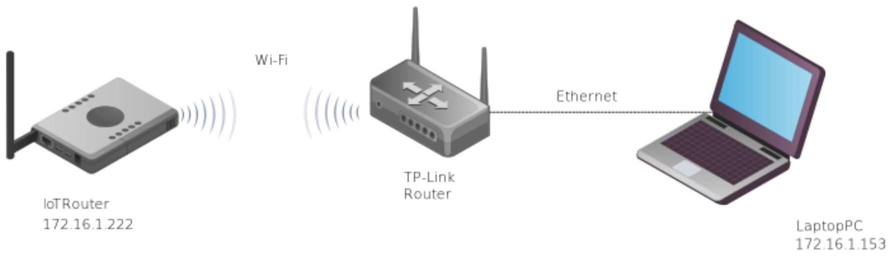


Figure 4.18. Hardware connections for bandwidth measurement via Wi-Fi.

The scheme in Fig. 4.18 is identical to the one adopted for the Ethernet measurements except that this time the IoT router connects to the TP-Link router via 2.4GHz Wi-Fi dongle (TP-LINK model TL-WN725N plugged into the USB port) which has a maximum data-rate of 150 Mbit/s. Therefore, the following tests are expected not to cross this limit.

It is important to note that 150 Mbit/s is a limit imposed by the Wi-Fi dongle, it cannot be excluded that with a different one supporting higher data-rate the IoT

Table 4.2. Ports and IP addresses of the devices used during the test.

Device	Ethernet Port	IP
Laptop HP ZBook 15G3	1 Gbit/s	172.16.1.153
Tp-Link AC1750	1 x 1 Gbit/s	-

Device	Wi-Fi	IP
IoT router	2.4 GHz / 150 Mbit/s	172.16.1.222

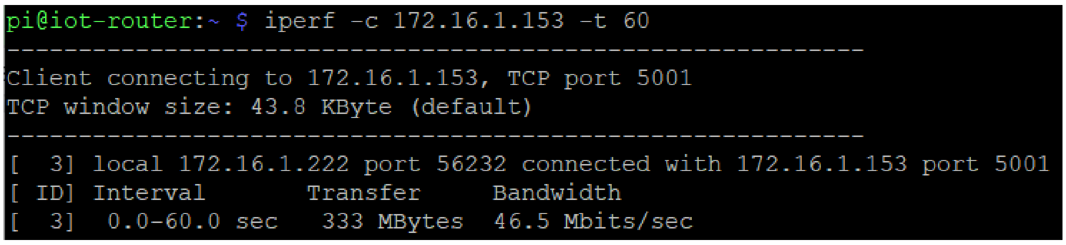
Router could have better performance. These tests assume a distance between the IoT Router and the TP-Link router of around 50 cm. The following sections describe the result of the tests using again the Iperf software but omitting the commands as they are exactly the same ones described earlier.

4.4.6 TCP measurement via Wi-Fi

Upstream measurement:

- Duration: 60 seconds.
- TCP window size: 43.8 KB (default).
- Data transferred: 333 MB.
- Direction: upstream.
- Source IP: 172.16.1.222
- Destination IP: 172.16.1.153
- Achieved bandwidth: 46.5 Mbit/s

The result shows a bandwidth of 46.5 Mbit/s i.e. around one third of the expected value. To establish where the bottleneck is, the Wi-Fi dongle should be tested with laboratory equipment to make sure it can reach the declared value of 150 Mbit/s (not purpose of this thesis, it can be considered in future work). If it can, then the reduction of the data-rate is caused by the IoT Router.



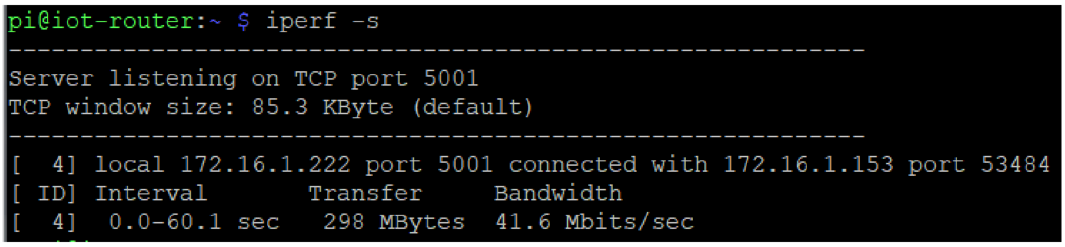
```
pi@iot-router:~ $ iperf -c 172.16.1.153 -t 60
------------------------------------------------------------
Client connecting to 172.16.1.153, TCP port 5001
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.222 port 56232 connected with 172.16.1.153 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   333 MBytes  46.5 Mbits/sec
```

Figure 4.19. Result of TCP upstream test for Wi-Fi configuration.

Similarly for downstream:

- Duration: 60 seconds.
- TCP window size: 85.3 KB (default).
- Data transferred: 298 MB.
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.222
- Achieved bandwidth: 41.6 Mbit/s

The bandwidth appears to be slightly lower than for upstream.



```
pi@iot-router:~ $ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 172.16.1.222 port 5001 connected with 172.16.1.153 port 53484
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-60.1 sec   298 MBytes  41.6 Mbits/sec
```

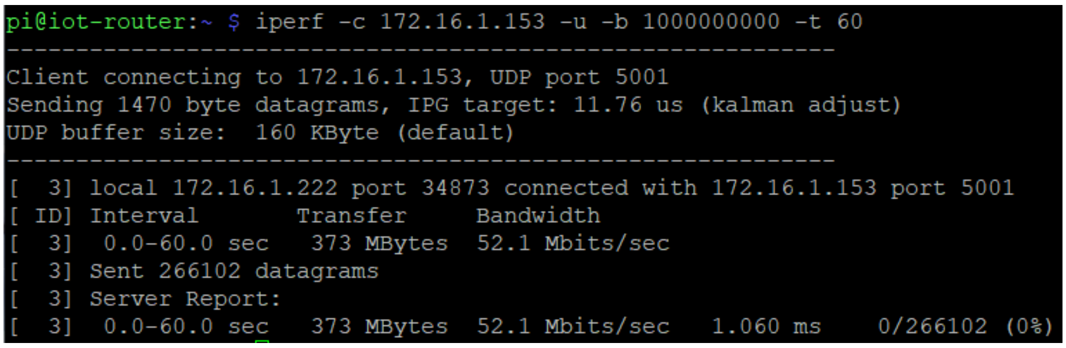
Figure 4.20. Result of TCP downstream test for Wi-Fi configuration.

4.4.7 UDP measurement via Wi-Fi

Upstream measurement:

- Duration: 60 seconds.
- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 373 MB.
- Jitter: 1.060 ms
- Datagrams lost: 0/266102 (0%)
- Direction: upstream.
- Source IP: 172.16.1.222
- Destination IP: 172.16.1.153
- Required bandwidth: 1 Gbit/s
- Achieved bandwidth: 52.1 Mbit/s

In agreement with the UDP measurement for the Ethernet configuration, the achieved bandwidth is higher than TCP.



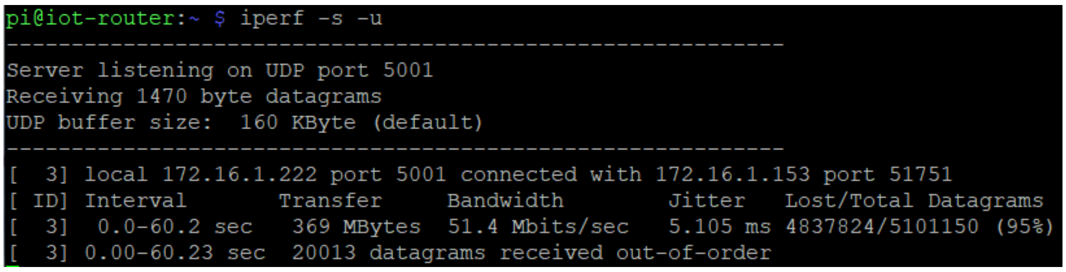
```
pi@iot-router:~ $ iperf -c 172.16.1.153 -u -b 1000000000 -t 60
------------------------------------------------------------
Client connecting to 172.16.1.153, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11.76 us (kalman adjust)
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.222 port 34873 connected with 172.16.1.153 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec   373 MBytes  52.1 Mbits/sec
[  3] Sent 266102 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec   373 MBytes  52.1 Mbits/sec   1.060 ms    0/266102 (0%)
```

Figure 4.21. Result of UDP upstream test for Wi-Fi configuration.

Similarly for downstream:

- Duration: 60 seconds.
- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 369 MB.
- Jitter: 5.105 ms
- Datagrams lost: 4837824/5101150 (95%)
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.222
- Required bandwidth: 1 Gbit/s
- Achieved bandwidth: 51.4 Mbit/s

The downstream bandwidth appears to be similar as for upstream. Note the higher value of jitter and datagrams lost, this is due to the data-rate imposed to the Iperf terminal (1 Gbit/s) which is much higher than the limit.

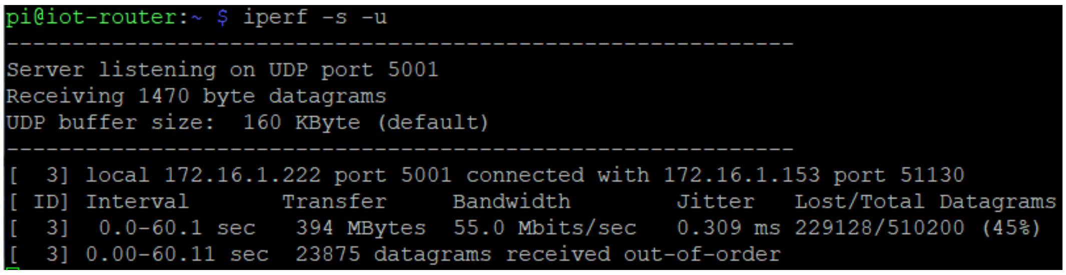


```
pi@iot-router:~ $ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.222 port 5001 connected with 172.16.1.153 port 51751
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-60.2 sec   369 MBytes  51.4 Mbits/sec   5.105 ms 4837824/5101150 (95%)
[  3] 0.00-60.23 sec  20013 datagrams received out-of-order
```

Figure 4.22. Result of UDP downstream test for Wi-Fi configuration with required bandwidth of 1 Gbit/s.

Repeating the downstream test with an imposed bandwidth of 100 Mbit/s both the jitter and the number of datagrams lost decrease as expected.

- Duration: 60 seconds.
- UDP buffer size: 160 KB (default).
- Datagram: 1470 Bytes.
- Data transferred: 394 MB.
- Jitter: 0.309 ms
- Datagrams lost: 229128/510200 (45%)
- Direction: downstream.
- Source IP: 172.16.1.153
- Destination IP: 172.16.1.222
- Required bandwidth: 100 Mbit/s
- Achieved bandwidth: 55.0 Mbit/s



```
pi@iot-router:~ $ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 172.16.1.222 port 5001 connected with 172.16.1.153 port 51130
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-60.1 sec   394 MBytes  55.0 Mbits/sec   0.309 ms 229128/510200 (45%)
[  3] 0.00-60.11 sec  23875 datagrams received out-of-order
```

Figure 4.23. Result of UDP downstream test for Wi-Fi configuration with required bandwidth of 100 Mbit/s.

4.4.8 Results

Following the analysis presented in this section, the available bandwidth for Ethernet interface is fairly close to the physical limit imposed by the 100 Mbit/s port. This limit is suitable for typical IoT applications as the throughput is usually much lower (Kbit/s). As regards the Wi-Fi interface, the results show that just one third of the expected bandwidth was actually available. This might be due to the WI-Fi dongle or due to the IoT Router itself. However, even in this conditions the IoT Router would be able to handle a large IoT traffic. The results are summed up in Table 4.5.

It is important to remember that these results are derived from laboratory measurements, i.e. carried out in isolated environment. In real cases, where the IoT Router

Table 4.3. Measured bandwidth for Ethernet and Wi-Fi interfaces.

Ethernet	TCP	UDP
Uplink	95 Mbit/s	95.7 Mbit/s
Downlink	94.9 Mbit/s	95.4 Mbit/s

Wi-Fi	TCP	UDP
Uplink	46.5 Mbit/s	52.1 Mbit/s
Downlink	41.6 Mbit/s	51.4 Mbit/s

operates connected to the outside network, the available bandwidth can be significantly lower due to limits and bottlenecks of the network itself.

4.5 IQRF simulation

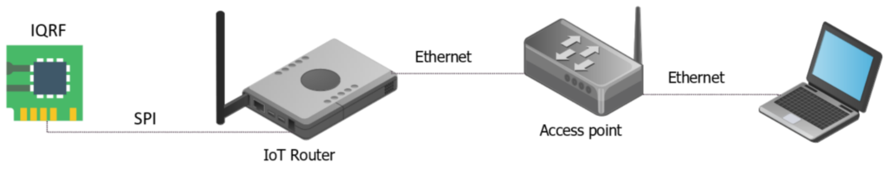


Figure 4.24. Hardware configuration for the IQRF simulation.

The scheme adopted for this simulation is the same as the one described in Fig. 4.12 with IQRF being the only source of traffic. The goal is to understand how the IoT Router would handle the traffic coming from an IQRF network directed to a hypothetical cloud service over the Ethernet interface. This is done with a simulation which generates traffic from the IQRF module connected to the IoT Router (simulating packets coming from a hypothetical network) and sending each packet to the laptop which hosts an Iperf server listening to UDP traffic. The highest data-rate which the transceiver can achieve is limited by two main factors:

- SPI communication
- DPA protocol

IQRF SPI allows packets up to 64 Bytes, with maximum 55 Bytes of data when SPI protocol is used. Due to the acknowledgement system (confirmation message) implemented by the DPA protocol and because of various overheads [29], a single DPA request with 55 bytes of data and a single hop between coordinator and nodes takes around 140 milliseconds to be completed. This means a limit of 7 DPA requests each seconds in the best case scenario (no extra delay/processing required), i.e. 385 Byte/s. To simulate this traffic, it is sufficient to send 7 SPI packets with random data (52B) each second to the IQRF module and send the entire response (52B of data + 3 control bytes, 55B total) over Ethernet. The following paragraphs explain how to implement this simulation within the GUI.

4.5.1 Implementation

To set up this scenario it is necessary to go to the editor page of the GUI and follow these steps:

- 1) Drag the IQRF-SPI and UDP nodes to the canvas. Connect them as shown in Fig. 4.25.

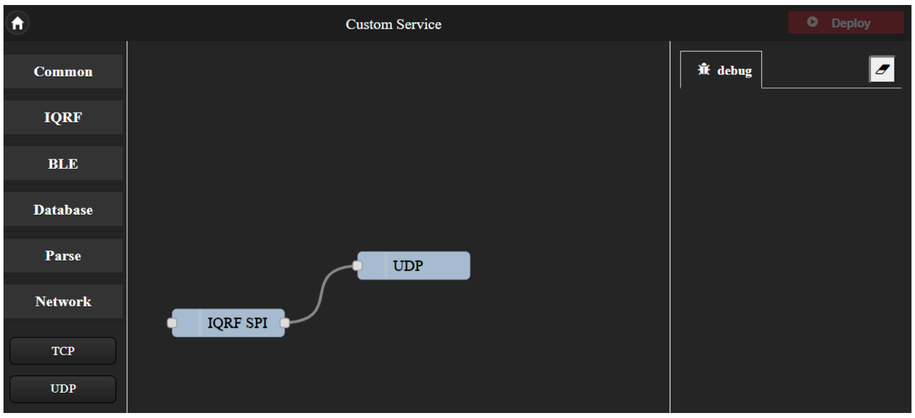


Figure 4.25. Design of IQRF scenario with the GUI editor.

- 2) Double click on the IQRF SPI node and fill the form as described below.

Table 4.4. Parameters of IQRF SPI node.

Field	Value
Serial port	/dev/spidev1.0
pcmd	F0
D-length	0x34
data	52 times 0x00 separated by coma or space
CRCM	0x9B

Note that this command generates a response from the module of 55 bytes in total.

- 3) Click on the + button, Fig. 4.26.
- 4) Select time interval of 140 ms.
- 5) Click the toggle button to activate the command (it is not sent yet).

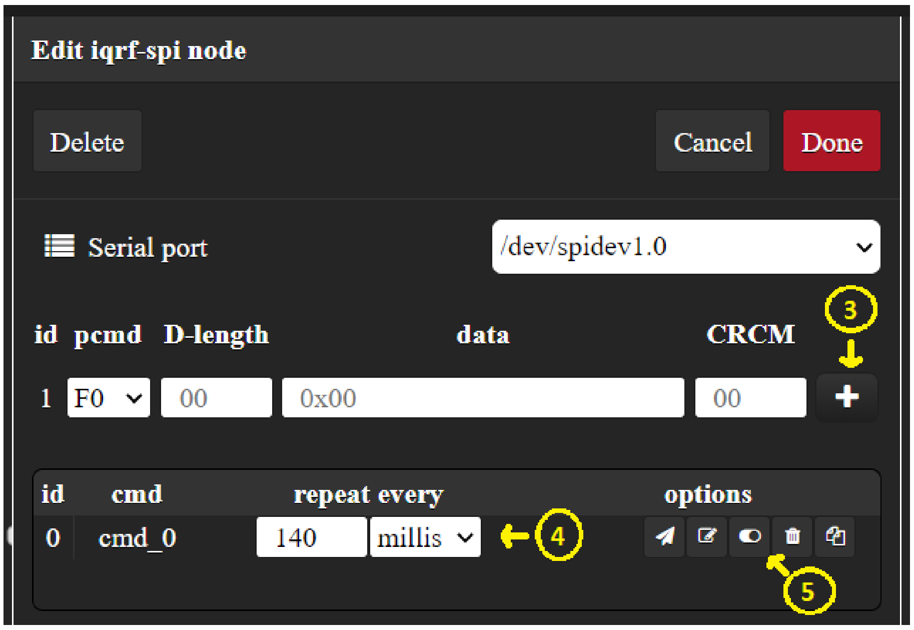


Figure 4.26. Form for the IQRF SPI node with relative parameters.

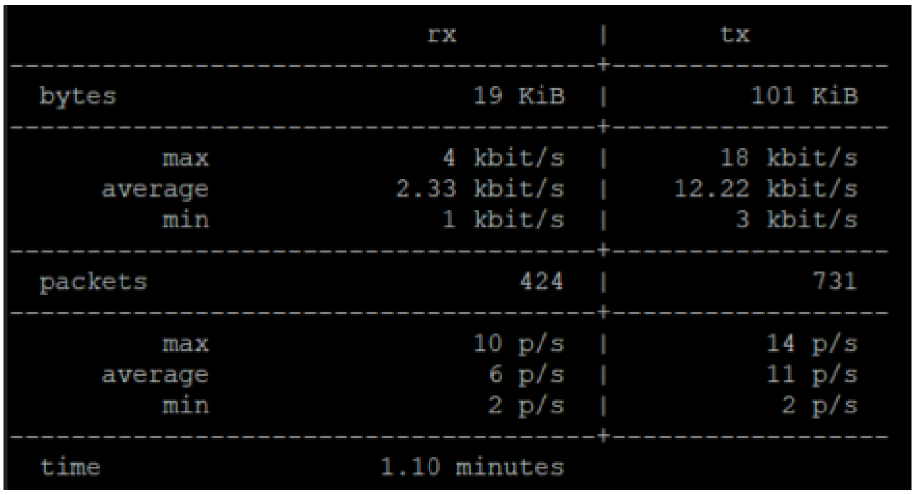
- 6) Click on `Done`.
- 7) Double-click on the UDP node and fill the form as described below.

Table 4.5. Parameters of UDP node.

Field	Value
Host	10.0.0.33
Port	5001

- 8) Click on `Done` and then `Deploy`.
- 9) Go back to the dashboard and click on the `Play` button on the corresponding service.

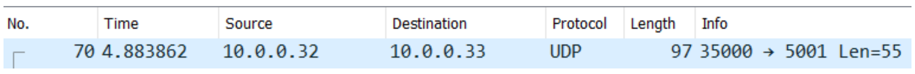
Now the service starts sending the messages activated in step 5 at the specified interval of time to the IQRF transceiver. The packets received from the transceiver are then forwarded to the Iperf UDP server (running on the laptop) on port 5001 as UDP traffic. With respect to the idle condition described in Sec. 4.3.4 the outgoing Ethernet traffic increased by 0.735 Kbyte/sec as shown by the vnstat tool, Fig. 4.27.



```
                                 rx         |       tx
--------------------------------------------+-------------------
  bytes                          19 KiB     |      101 KiB
--------------------------------------------+-------------------
          max                     4 kbit/s  |       18 kbit/s
      average                  2.33 kbit/s  |    12.22 kbit/s
          min                     1 kbit/s  |        3 kbit/s
--------------------------------------------+-------------------
  packets                           424     |          731
--------------------------------------------+-------------------
          max                     10 p/s    |       14 p/s
      average                      6 p/s    |       11 p/s
          min                      2 p/s    |        2 p/s
--------------------------------------------+-------------------
  time                      1.10 minutes
```

Figure 4.27. Ethernet traffic of the IQRF simulation.

Note that during the 60 seconds of measurement, 11 packets/sec on average, 7 of which from the IQRF module and the other 4 supposedly from Node-RED for the GUI charts were transmitted. However, this increment includes the 385 Bytes/s collected from the IQRF module and the overhead introduced by the UDP protocol, whose packets are composed by 97 bytes in total as shown by the frame captured with the Wireshark software, Fig. 4.28.



No.	Time	Source	Destination	Protocol	Length	Info
70	4.883862	10.0.0.32	10.0.0.33	UDP	97	35000 → 5001 Len=55

Figure 4.28. UDP frame captured with Wireshark.

The impact of this process on the IoT Router is visible also graphically on the `Charts` page of the GUI. Figure 4.29 also shows an increment on the CPU usage of 3% with +1°C on the temperature value. The RAM usage went up by 0.5%.

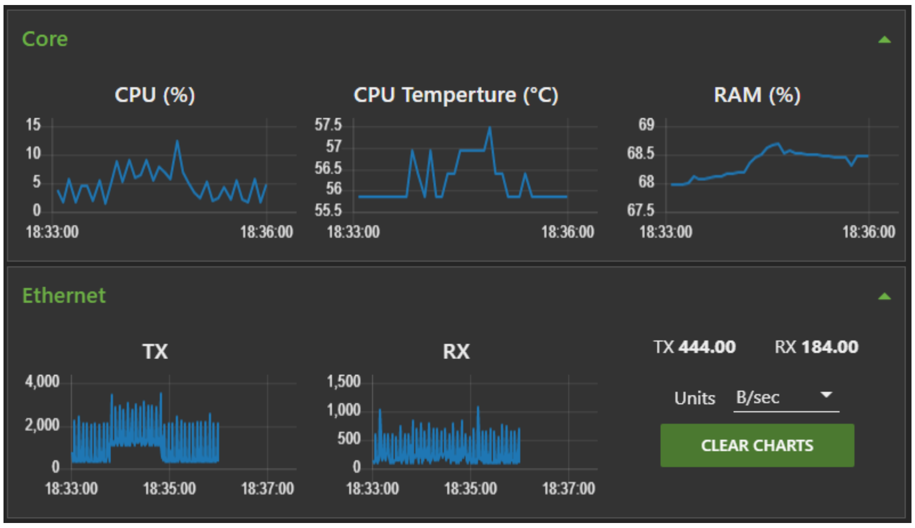


Figure 4.29. Impact of the IQRF simulation on the IoT Router.

The results of this simulation are summed up in the Table 4.61.

Table 4.6. Results of IQRF SPI simulation.

Field	Impact
CPU load(%)	+ 3%
CPU temperature (°C)	+ 1 °C
RAM usage(%)	+ 0.5%
Ethernet TX	+ 0.735 Kbyte/sec
Ethernet RX	+ 11,25 Byte/sec

4.6 BLE simulation

This scenario presents the scheme adopted to test the custom BLE (Bluetooth Low Energy) module and study its impact on the performance of the IoT Router, with the focus on the outgoing traffic from the Ethernet interface. Figure 4.30 shows the architecture adopted for this test. The simulation involves transmitting packets from the mobile phone to the BLE module and forwarding them to the UDP server running on the laptop with the Iperf software. The goal is to measure the performance of the board when high traffic generated from the BLE module is sent over Ethernet. It can be helpful to understand how the IoT Router would handle a hypothetical network of BLE sensors whose data has to be sent to the cloud.

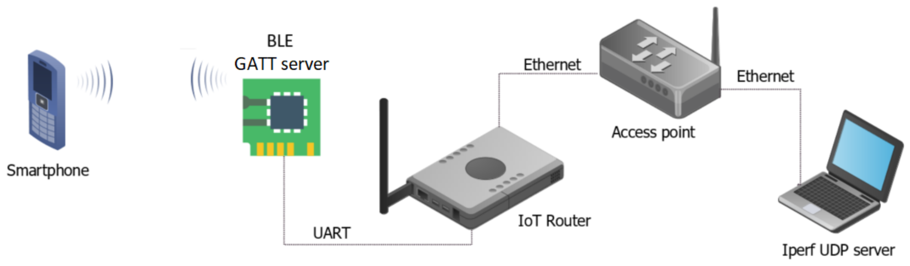


Figure 4.30. Architecture of the BLE scenario.

As a GATT Server, the BM70/RN4870 module (on which the custom BLE module is based) hosts the Microchip Transparent UART Service. The Transparent UART Service provides a simple bidirectional data transfer service. It defines two characteristics for data communication, one for receiving data with the Write property and the other for sending data with the Notify property. The Microchip Transparent UART Service is instantiated as a Primary Service on the BM70/RN4870 BLE module by default [30]. To test the default characteristics the Android application `Serial Bluetooth Terminal` is used [31] as it automatically implements the Microchip RN4870/71, BM70/71 'transparent UART service'. The UUIDs and access properties of the Microchip Transparent UART service and its characteristics are shown in the tables below.

Table 4.7. Service name and UUID of the GATT server.

Service Name	UUID
Microchip Transparent UART	49535343-FE7D-4AE5-8FA9-9FAFD205E455

Table 4.8. Characteristic name and UUID of the GATT server.

Characteristic Name	UUID	Properties
Microchip Transparent UART TX	49535343-1E4D-4BD 9-BA61-23C647249616	Notify Write Write without response
Client Characteristic Descriptor		Read Write
Microchip Transparent UART RX	49535343-8841-43F 4-A8D4-ECBE34729BB3	Write Write without response

The custom BLE module communicates via UART with the IoT Router using a Baud Rate of 115200, 8 data bits, 1 start bit, 1 stop bit and no parity. With this setup the actual UART throughput is 92160 bit/s or 11520 byte/s. It is important to note that this is a limit imposed by the physical connection between module and MCU and it is not dependent on the BLE protocols. The connection between the module and the phone appears to drop frequently when the time interval between transmitted packets is below 25 milliseconds. This might be caused also by overheads of the mobile application. A stable compromise is obtained transmitting packets of 22 bytes each every 30 ms resulting in roughly 660 Bytes/sec which can be reasonable considering a typical IoT application. The simulation implemented for this scenario lasts 60 seconds during which the Ethernet throughput is monitored together with the CPU and RAM performance.

4.6.1 Implementation

Similarly, as for the previous simulation, this scenario is built with the GUI editor. Drag the BLE RX and UDP nodes to the canvas. Connect them as shown in Fig. 4.31.

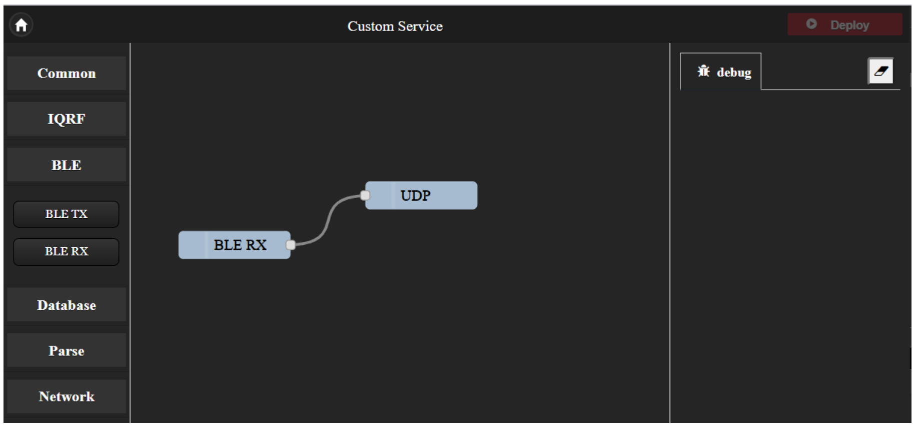


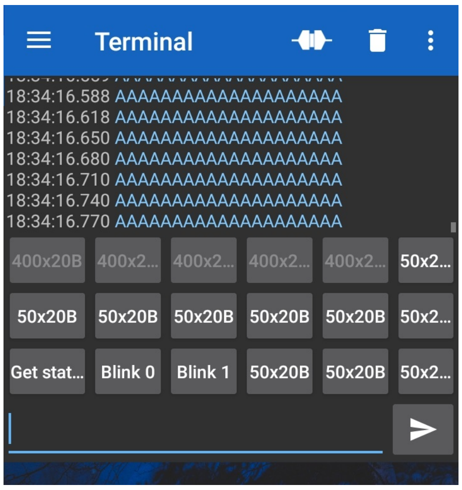
Figure 4.31. Design of BLE scenario with the GUI editor.

Double click on the BLE RX node and click on the App button to put the Bluetooth module into Application mode. The UDP node can be configured as shown in Table 4.9.

Table 4.9. Parameters of the UDP node.

Field	Value
Host	10.0.0.35
Port	5100

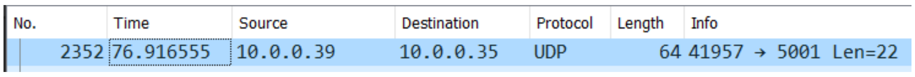
Now all parameters are set. To save the changes click on `Done` and then `Deploy`. The IoT Router is ready to receive data through the Transparent UART Service and send it to the UDP server. To test this process, the easiest way is to connect to the BLE module via a mobile phone. The application chosen for this purpose is called Serial Bluetooth Terminal (Android app). Figure 4.32 shows the terminal of the application transmitting 22 bytes of data (HEX: 0x41, ASCII: `A`) with an interval of roughly 30 ms for 60 seconds. Messages are represented as strings separated by the Carriage Return (\r, last two bytes of data) since Node-RED expects it when reading the serial port.



```
18:34:16.588 AAAAAAAAAAAAAAAAAAAA
18:34:16.618 AAAAAAAAAAAAAAAAAAAA
18:34:16.650 AAAAAAAAAAAAAAAAAAAA
18:34:16.680 AAAAAAAAAAAAAAAAAAAA
18:34:16.710 AAAAAAAAAAAAAAAAAAAA
18:34:16.740 AAAAAAAAAAAAAAAAAAAA
18:34:16.770 AAAAAAAAAAAAAAAAAAAA
```

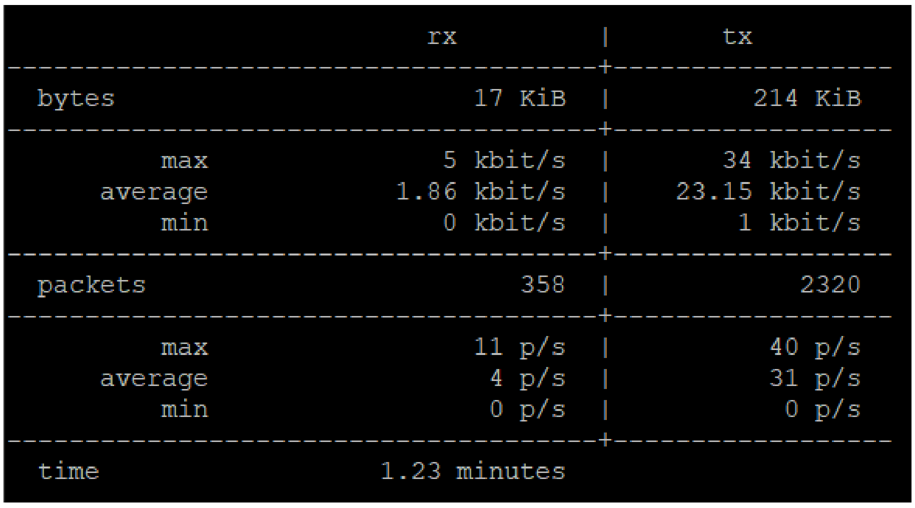
Figure 4.32. Terminal of the BLE android application.

The numerical results of the simulation are presented in Fig. 4.34. Each UDP packet had a length of 64 bytes as shown by the frames captured by the Wireshark software during the simulation, Fig. 4.33. With an average of 31 transmitted packets each second the Ethernet throughput increased by roughly 2.12 Kbyte/s (16.96 Kbit/s more with respect to the idle value of 6.19 Kbit/s described in Sec. 4.3.4). This result is fairly close to the theoretical value of 64 bytes x 31 packets/sec = 1984 Byte/s (15.872 Kbps). The graphical results can be observed in Fig. 4.35. Note an average increment of 5% on the CPU usage with the CPU temperature varying between 52.5°C and 56°C. The RAM usage increased by less than 1%. From this figure it also appears how the outgoing Ethernet traffic assumes a decreasing trend between the beginning and the end of the simulation.



No.	Time	Source	Destination	Protocol	Length	Info
2352	76.916555	10.0.0.39	10.0.0.35	UDP	64	41957 → 5001 Len=22

Figure 4.33. UDP frame captured with Wireshark.



```
                                  rx          |       tx
--------------------------------------+------------------
 bytes                          17 KiB   |      214 KiB
--------------------------------------+------------------
              max              5 kbit/s   |     34 kbit/s
          average           1.86 kbit/s   |  23.15 kbit/s
              min              0 kbit/s   |      1 kbit/s
--------------------------------------+------------------
 packets                           358   |         2320
--------------------------------------+------------------
              max                11 p/s   |       40 p/s
          average                 4 p/s   |       31 p/s
              min                 0 p/s   |        0 p/s
--------------------------------------+------------------
 time                      1.23 minutes
```

Figure 4.34. Numerical results of the BLE simulation.

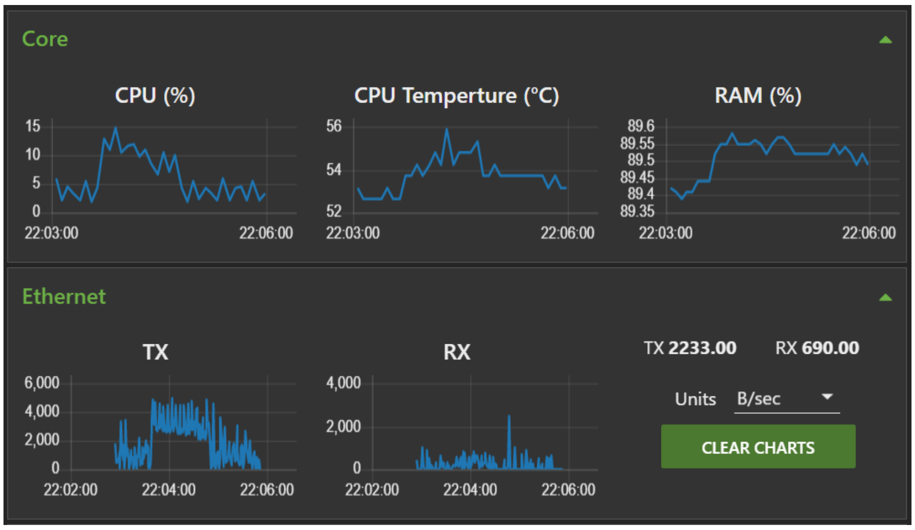


Figure 4.35. Graphical results of the BLE simulation.

The results of this simulation are summed up in the table below.

Table 4.10. Impact of the BLE simulation on the IoT Router.

Field	Impact
CPU load(%)	+ 5%
CPU temperature (°C)	+ 4 °C
RAM usage(%)	+ 0.15%
Ethernet TX	+ 2.12 Kbyte/sec
Ethernet RX	+ 0.68 Kbyte/sec

4.7 LoRa concentrator

This section describes the implementation of a LoRa concentrator using the module shown in Fig. 3.8. The goal is to verify whether the IoT Router is able to work as a LoRa gateway. The implementation includes downloading and installing the appropriate software and verify that the router can receive messages from LoRa nodes deployed in the field and forward them to the cloud `The Things Network` [32]. This procedure is described in more details in the tutorial 'Build a LoRa Gateway with n-fuse mPCIe card' [33] which has been followed step by step, therefore only a summary is reported here. To set up a LoRa gateway, three programs are required. The first is the packet forwarder, running on RPI-CM3, which forwards received RF packets from end nodes to the server via an IP / UDP link. The software can also forward to downlink the packets sent by the server. The second SW is picoGW_hal, it is a driver necessary to communicate with the concentrator via USB or UART interface. The library implements a USB CDC (virtual com port) for communication with the MCU. The third and last is picoGW_mcu, it is a FW for MCU (STM32-F401CD) which is integrated in the concentrator. It also implements a USB CDC protocol for bridging commands from the HOST (RPI-CM3) directly to the SX1301 SPI interface. The HAL driver [20] and the packet forwarder [21] are downloaded with the following commands:

```
mkdir loragateway
cd loragateway
git clone https://github.com/Lora-net/picoGW_packet_forwarder.git
git clone https://github.com/Lora-net/picoGW_hal.git
```

And to install the sources:

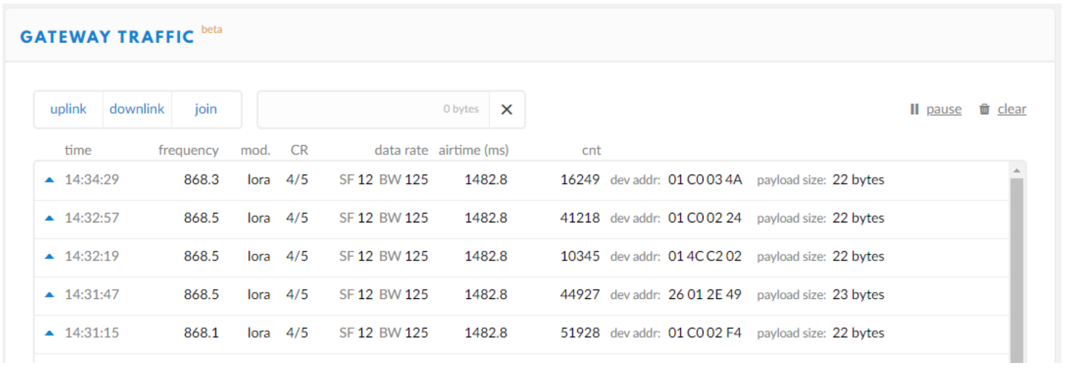
```
cd picoGW_hal
make clean all
cd ..
cd picoGW_packet_forwarder
make clean all
```

Note that the antenna needs to be attached to the module before the module is plugged into the IoT router, otherwise it could be damaged. The configuration (global_conf.json) file has to be properly edited with the unique ID of the concentrator (333833303b005200 in this case) and the parameters described in Table 4.11.

The gateway can now be registered to the cloud (TheThingsNetwork [33]). After starting the packet forwarder, the gateway is up and running and it can receive messages from LoRa nodes in its range. Figure 4.36 shows the messages received by the IoT Router through the LoRa concentrator and forwarded to the cloud. This proves the correct implementation of the IoT Router as a LoRa gateway.

Table 4.11. Parameters of the global_conf.json file.

Field	Value
gateway_ID	333833303b005200
server_address	router.eu.thethings.network
serv_port_up	1700
serv_port_down	1700


GATEWAY TRAFFIC beta

uplink downlink join 0 bytes ✕ pause clear

	time	frequency	mod.	CR	data rate	airtime (ms)	cnt				
▲	14:34:29	868.3	lora	4/5	SF 12 BW 125	1482.8	16249	dev addr:	01 C0 03 4A	payload size:	22 bytes
▲	14:32:57	868.5	lora	4/5	SF 12 BW 125	1482.8	41218	dev addr:	01 C0 02 24	payload size:	22 bytes
▲	14:32:19	868.5	lora	4/5	SF 12 BW 125	1482.8	10345	dev addr:	01 4C C2 02	payload size:	22 bytes
▲	14:31:47	868.5	lora	4/5	SF 12 BW 125	1482.8	44927	dev addr:	26 01 2E 49	payload size:	23 bytes
▲	14:31:15	868.1	lora	4/5	SF 12 BW 125	1482.8	51928	dev addr:	01 C0 02 F4	payload size:	22 bytes

Figure 4.36. LoRa messages received by the gateway.

4.8 Remote & local database

This section describes the implementation of two databases using Node-RED and the GUI. The purpose is to test the capability of the IoT Router to store data locally and remotely. First the Microsoft Azure SQL database is presented as an example of remote solution, then the Influx database follows as representative of a local solution. It is important to note that the MS Azure services are bound to a subscription, in order for this to work an account needs to be created. The following test uses a student account made available by the Czech Technical University in Prague at the time of writing, however this type of subscription has a validity of 30 days. After that period, it won't be possible to use or query the database without upgrading the subscription. Figure 4.37 shows the configuration adopted for the remote database test.

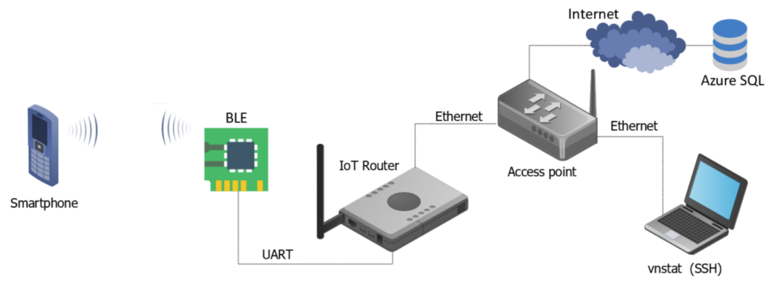


Figure 4.37. Configuration for the remote database test.

The idea is to simulate the reading of a sensor from a BLE network and store it into the MS Azure SQL database using the GUI. The database contains just one table (bluetoothle) with one column called 'temperature' which is enough for the purposes of this test. For simplicity the temperature value is sent from a mobile phone connected to the BLE module, using the android app `Serial Bluetooth Terminal` [31]. This operation is done similarly as described in section 4.6.1. As regards the local solution, the Influx database is used as a data concentrator. The Influx DB is a high-speed read and write time series database which appears to be a good candidate for IoT solutions.

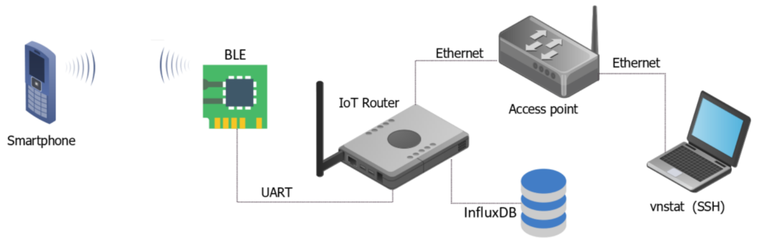


Figure 4.38. Configuration for the local database test.

The figure above shows the scheme adopted for the local database test. Similarly as before, this database has one Measurement (analogous to a SQL table) with one Tag (analogous to a SQL column). Since the local database is running on the board, it is expected to have a higher impact on the CPU and RAM usage compared to the remote solution. On the other side, the execution of queries is expected to be much faster with the local database, partially because it is used without authentication. Note that in real application, for local databases the authentication is usually necessary but for this test, it is not required.

4.8.1 Implementation

The GUI is used to build this simulation. After entering the editor page from the dashboard, the required nodes are dragged on the canvas and connected as shown in Fig. 4.39.

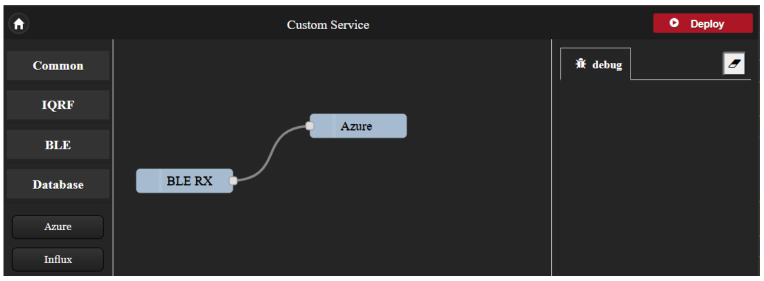
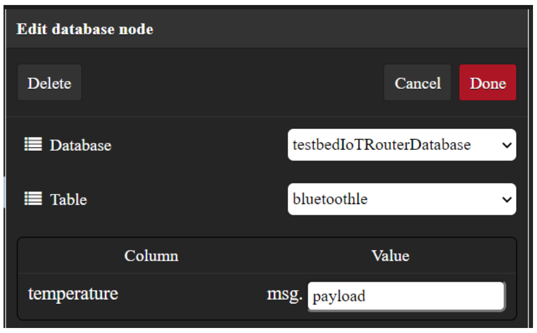


Figure 4.39. Design of the remote database scenario with the GUI.

The Azure node stores the payload of the incoming messages in the database and table specified by the user. To set it up, the form needs to look like below (double click on the node to open the form):


Edit database node

Delete | Cancel | Done

Database	testbedIoTRouterDatabase
Table	bluetoothle

Column	Value
temperature	msg. payload

Figure 4.40. Configuration of the MS Azure database.

For the BLE RX node the only required configuration is to click on the App button inside its form, this will bring the BLE module into the application mode running the default Transparent UART service. In this way, it is possible to connect to it with the phone through the aforementioned android app (see section 4.6.1 for more details). Once these operations are done, the service is deployed with the red button in the top-right corner of the editor. Now it is possible to send values from the phone and have them stored in the selected SQL table. Data can be verified directly from the MS Azure dashboard or through a database management software.

For the local solution the principle is the same, instead of the Azure node the Influxdb node is connected to the BLE RX node. Note that the GUI is not able to handle two BLE RX nodes at the same time. This is because there is just one physical Bluetooth module on the IoT Router, therefore the previous service can be deleted or edited (deleting the Azure node and inserting the Influxdb node, see Fig. 4.41).

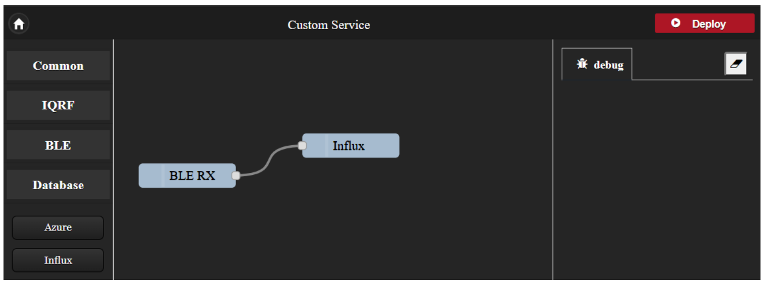


Figure 4.41. Design of the local database scenario with the GUI.

The parameters for the Influxdb node are described in the Table 4.12.

Table 4.12. Parameters of the Influxdb node.

Field	Value
Measurement	temperature
Tag	location
Tag value	Room 2

Once the service is deployed, it is possible to send the temperature values from the mobile phone as done before. The figure below shows a screen-shot of the android app (left) and the temperature value store in the database (right). The local database can be queried from a SSH terminal of the IoT Router as follow:

start the database console:

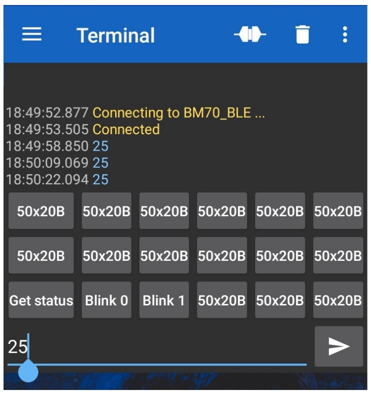
```
influx
```

Select the database to use:

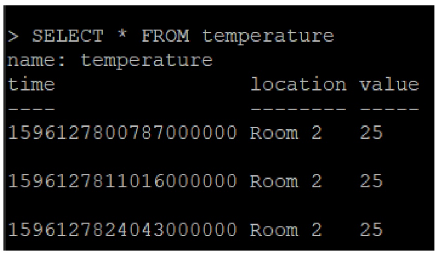
```
use iotrouterdb
```

Query database:

```
SELECT * FROM temperature
```



```
18:49:52.877 Connecting to BM70_BLE ...
18:49:53.505 Connected
18:49:58.850 25
18:50:09.069 25
18:50:22.094 25
```

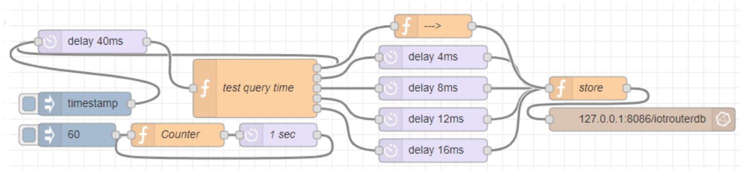


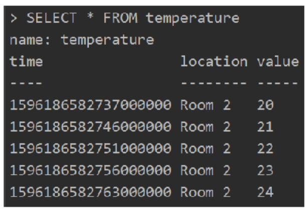
```
> SELECT * FROM temperature
name: temperature
time                location value
----                -------- -----
1596127800787000000 Room 2   25

1596127811016000000 Room 2   25

1596127824043000000 Room 2   25
```

Figure 4.42. Phone view (left), database entries (right).

4.8.2 Performance

The performance of the two solutions appears to be quite different. In the first case, with the Azure node, the execution time of an `INSERT` query with a single temperature value takes on average one second. The reason is mainly because the node[34] which actually implements this operation in Node-RED connects to the database every time before executing a query, then it closes the connection. This solution makes sense when the data is not collected too often (e.g. once per hour), but if the sampling interval decreases or a lot of data is collected at once, then a better approach might be to keep the connection open which would result in a much faster execution time. On the contrary, the influx DB seems to execute queries much faster. The flow shown in Fig. 4.43 (left) inserts numbers from 21 to 24 in loop into the database every 4 ms for 60 seconds. The right side of the picture shows the DB entries, note the timestamp in milliseconds which reflects the execution time of 4/5 ms. The impact of this operation is visible in Fig. 4.44. The CPU usage increased by 30%, the RAM by 1% and the CPU temperature increased by 11°C. Note that this is a test under stress conditions, i.e. to find out if the IoT Router can handle a high number of queries/sec. In real IoT applications this query execution rate is not common and, to improve the performance, one query could be used to store multiple values.





```
> SELECT * FROM temperature
name: temperature
time                location value
----                -------- -----
1596186582737000000 Room 2   20
1596186582746000000 Room 2   21
1596186582751000000 Room 2   22
1596186582756000000 Room 2   23
1596186582763000000 Room 2   24
```

Figure 4.43. Node-RED flow (left), database entries (right).

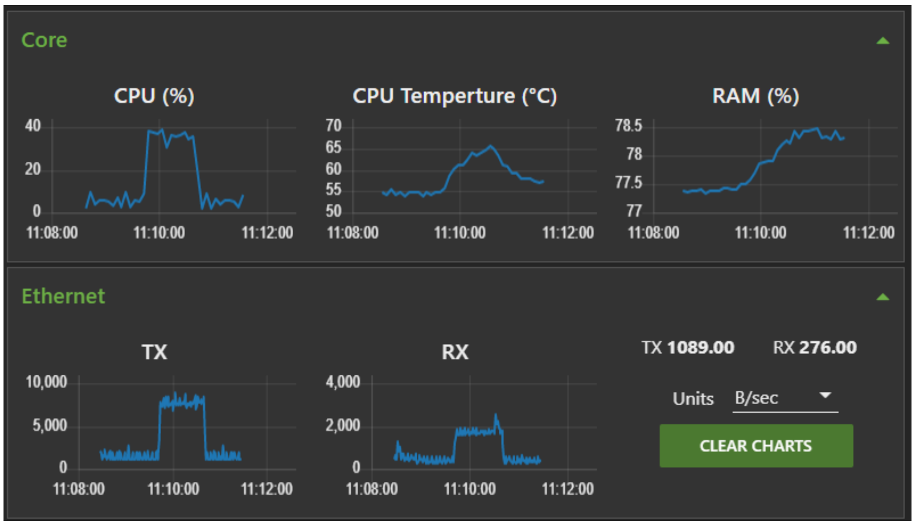


Figure 4.44. IoT Router performance with a large number of queries/sec (Influx DB).

4.9 NB-IoT

This section describes the implementation of the NB-IoT technology with the IoT Router. The goal is to verify the correct functionality of the NB-IoT module and measure the performance of the IoT Router when the module is sending or receiving data. Finally, a measurement of the latency is provided. For this test, the BG96 module is equipped with a T-mobile[35] SIM card with exclusive support for NB-IoT. Figure 4.45 shows the scheme adopted.

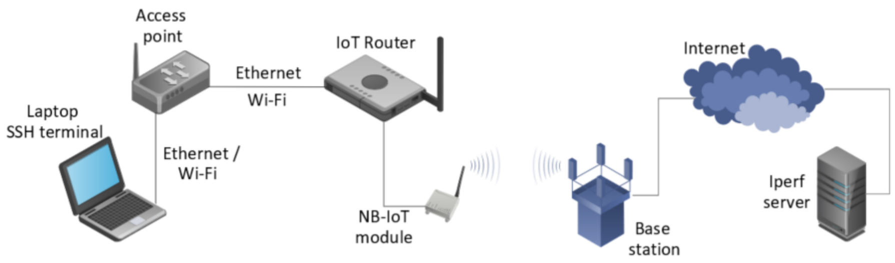


Figure 4.45. Architecture of the NB-IoT simulation.

The LTE Cat NB1 technology, available with the BG96 module[18], has a peak rate[36] of 26 kbit/s in downlink and 16.9 kbit/s in uplink (single-tone). The idea is to simulate traffic in both directions, with the help of a remote server, and see how it affects the CPU and RAM usage of the IoT Router. However, in order to recreate a typical IoT situation, the traffic is not generated continuously. To test the uplink communication, 1460 bytes (11.68 Kbit) of data are sent as UDP traffic to the remote Iperf server `ping.online.net`, few times in a minute. This number of bytes represents the maximum data length which can be transferred to the module at once[37]. Since the UDP packets are sent manually from the SSH terminal of the laptop, a good compromise seems to be sending 4 UDP packets per minute. This is also due to the delay connected with the response of the BG96 module after each message is sent. In real applications these operations would be carried out by a script, providing a better control of the module and, possibly, a higher packet rate. However, this is not required for the purposes of this theses. For the downlink, the traffic is simulated requesting a web page, via HTTP protocol, from a remote server (example.com). The server response is used as an example of downlink traffic. Table 4.14 provides more details about the request. In this way, 11 kbit of data are sent in uplink roughly every 15 seconds and around 4.39 kbit (549 bytes) in downlink with the same interval. The BG96 module is controlled via AT commands[38], which are sent to it using the `picocom` serial terminal[11], available for linux devices[39]. Table 4.13 describes the setup used for the serial communication.

Table 4.13. Settings of the BG96 serial communication.

Parameter	Value
GPIO26 (Enable)	logic 1 (active high)
GPIO27 (Reset)	logic 0 (active low)
Port	/dev/ttyUSB6
Baud rate	115200
Stop bit	1
Parity	none

The following commands are used as a one-time configuration to connect the module to the T-mobile network.

```
AT+CGDCONT = 1,"IP","iot.t-mobile.cz"
AT+CEREG=1
AT+COPS=1,0,"T-Mobile CZ",9
```

These commands set the Access Point Name (APN) to `iot.t-mobile.cz` and manually connect the module to the `T-Mobile CZ` operator. The status of the connection can be queried with the following command.

```
AT+CGATT?
```

If it replies with `+CGATT: 1  OK`, it means it is successfully connected to the network. Now the module is ready. First, the UDP uplink measurement is performed. To open a UDP socket service, another AT command is used.

```
AT+QIOPEN=1,0,"UDP","62.210.18.40",5208,0,0
```

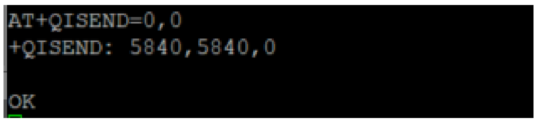
Note that the IP address 62.210.18.40 corresponds to the domain of the Iperf server (ping.online.net), while 5208 is the port it listens to[40]. Once the socket service in open, the UDP messages can be pasted in the terminal after the following command.

```
AT+QISEND=0,1460
```

Note the second parameter, 1460, which sets the size of the data to be transmitted. For this test, a random text of 1460 bytes is used as payload. The text can be pasted in the terminal and sent by pressing the Enter key. The terminal replies with `SEND OK`, but it does not mean the data has been sent to the server successfully. After sending the same UDP packet 4 times, the total amount of transmitted data can be queried with the command below.

```
AT+QISEND=0,0
```

As shown in Fig. 4.46, it amounts to 5840 bytes successfully sent, i.e. 4 x 1460 bytes.



```
AT+QISEND=0,0
+QISEND: 5840,5840,0

OK
```

Figure 4.46. Total transmitted data with the NB-IoT module.

As shown by Fig. 4.47, sending 4 UDP messages per minute doesn't seem to have a noticeable impact on the IoT Router, with less than 1% increment on the CPU and RAM usage.

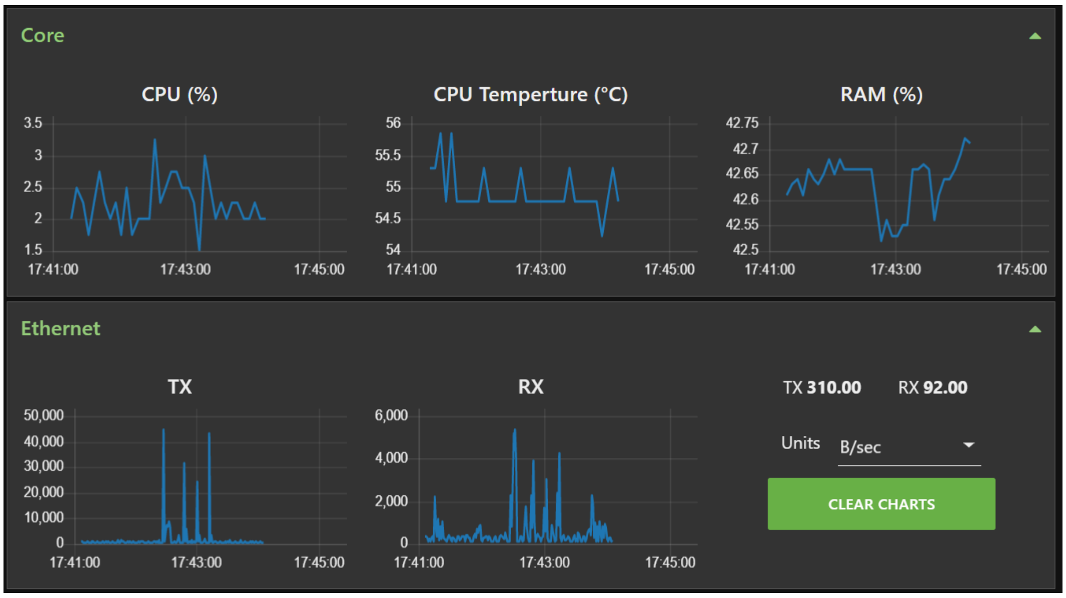


Figure 4.47. Impact of the uplink test on the CPU and RAM usage of the IoT Router.

Note the Ethernet traffic, Fig. 4.47. The peaks are due to the exchange of messages between the laptop and the IoT Router via SSH and TCP protocols. To test the downlink traffic, the procedure is quite similar. A TCP client connection is open with the following command.

```
AT+QIOPEN=1,0,"TCP","93.184.216.34",80,0,0
```

Note the IP address 93.184.216.34, which corresponds to the domain `example.com` listening on port 80. To specify the message payload size, the previous `QISEND` command is used, but with the size of 43 bytes, i.e. the payload string described in Table 4.14.

Table 4.14. Details of the HTTP request.

Parameter	Value
Server IP address	93.184.216.34
Server port	80
Payload (ASCII)	GET / HTTP/1.0\r\nHost: example.com\r\n\r\n
Method	TCP

At this point, the server responds with 549 bytes of data which are read from the module and displayed on the terminal of the laptop with the command below.

```
AT+QIRD=0,1500
```

As for the uplink, this operation is repeated 4 times. Figure 4.48 shows a contained impact on the CPU and RAM usage of the IoT Router. The received traffic however is not clearly visible on the RX chart, because it is mixed with the traffic generated to update the charts.

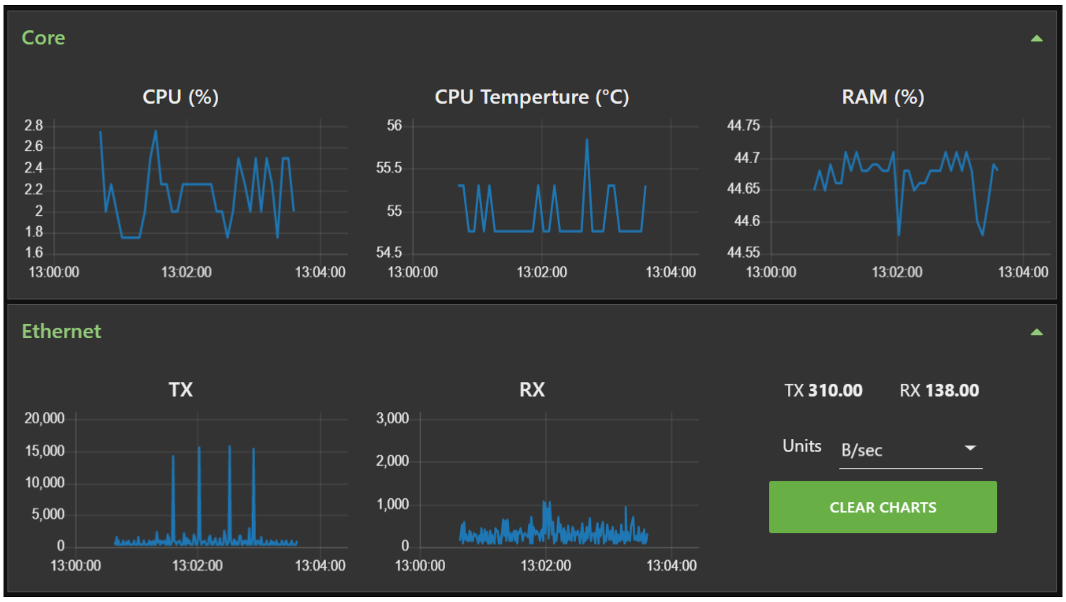
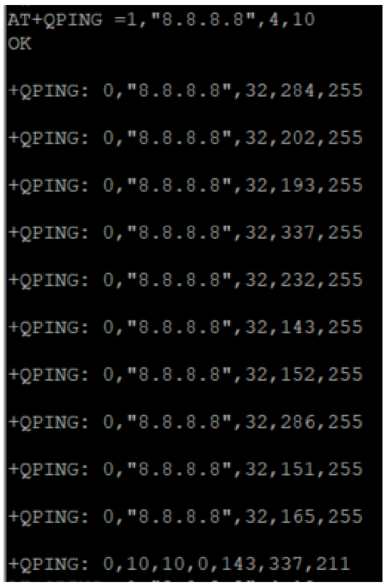


Figure 4.48. Impact of the downlink test on the CPU and RAM usage of the IoT Router.

Finally, the latency measurement with the NB-IoT module is carried out. To estimate the Rount-Trip Time (RTT) of a packet, the following command can be used.

```
AT+QPING =1,"8.8.8.8",4,10
```

This command sends 10 packets with 32 bytes of data to the IP address 8.8.8.8. measuring the RTT of each packet and calculating the average value (ms). To have a reliable estimation, the command is executed 10 times, i.e. 100 packets sent with a final average RTT value of 209.9 ms.



```
AT+QPING =1,"8.8.8.8",4,10
OK

+QPING: 0,"8.8.8.8",32,284,255

+QPING: 0,"8.8.8.8",32,202,255

+QPING: 0,"8.8.8.8",32,193,255

+QPING: 0,"8.8.8.8",32,337,255

+QPING: 0,"8.8.8.8",32,232,255

+QPING: 0,"8.8.8.8",32,143,255

+QPING: 0,"8.8.8.8",32,152,255

+QPING: 0,"8.8.8.8",32,286,255

+QPING: 0,"8.8.8.8",32,151,255

+QPING: 0,"8.8.8.8",32,165,255

+QPING: 0,10,10,0,143,337,211
```

Figure 4.49. Result of a ping test with the NB-IoT module.

Figure 4.49 shows the result of one ping test. Note the last line which shows an average RTT of 211 ms for the previous 10 ping requests. Both in uplink and downlink the packets are sent and received successfully, proving the correct implementation of the NB-IoT technology with the IoT Router.

4.10 IQRF + BLE simulation

Th section describes the implementation of scenario in which different technologies are used at the same time. The goal is to see how the IoT Router performs when data is collected at the same time from the IQRF and the BLE network and sent via Ethernet to a hypothetical cloud represented by the Iperf UDP server running on the laptop. The measurement is carried out based on the considerations made while testing the single technologies, i.e. the following data throughput is used:

- 385 Byte/s for IQRF
- 660 Byte/s for BLE

The figure below represents the architecture adopted for this test. Considering the UDP frames analyzed for the single cases, the actual outgoing traffic for this combined solution is expected to be around 2.83 Kbyte/s higher with respect to the idle condition. The incoming traffic is expected to be almost the same.

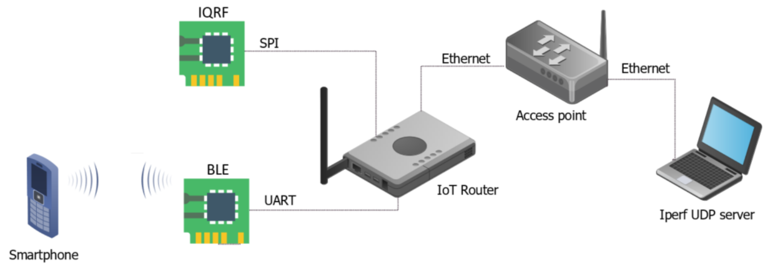


Figure 4.50. Architecture of the IQRF & BLE simulation.

The implementation of this test is done with the GUI using the editor page. The nodes are connected as shown in Fig. 4.50. The configuration of the nodes is omitted as it is the same adopted for the IQRF and BLE individual tests.

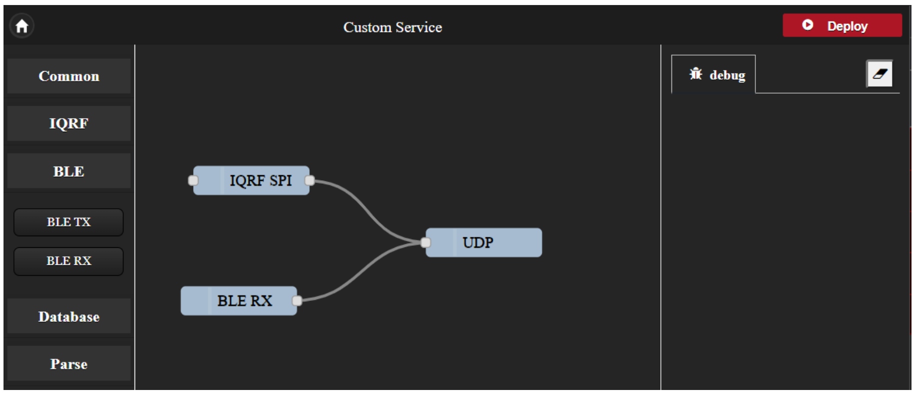
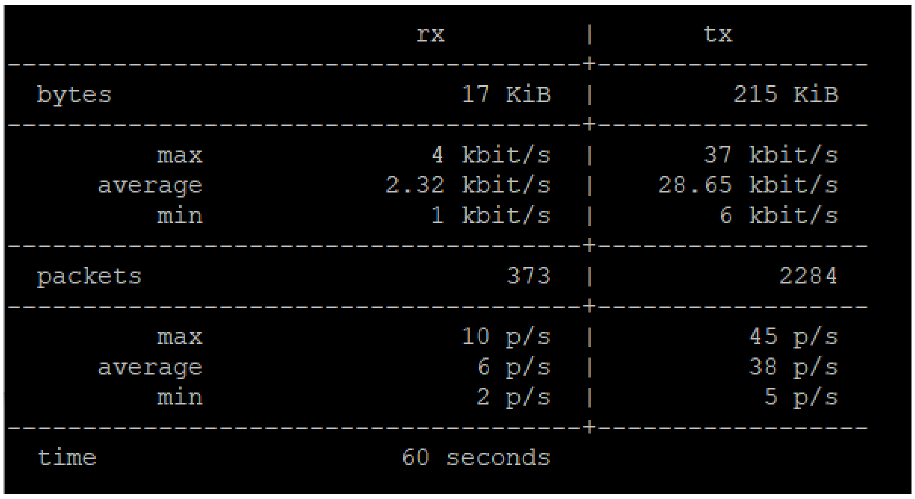


Figure 4.51. Design of the IQRF & BLE simulation.

After deploying the service, the simulation is launched. Fig. 4.53 shows the result after 60 seconds. As expected the Ethernet throughput increased by 2.78 KByte/sec in upstream, including both IQRF and BLE traffic. Note that 38 packets/sec were sent, 31 of which from the BLE module and 7 from the IQRF module.



```
                        rx         |       tx
--------------------------------------+------------------
 bytes                     17 KiB  |       215 KiB
--------------------------------------+------------------
         max             4 kbit/s  |     37 kbit/s
     average          2.32 kbit/s  |  28.65 kbit/s
         min             1 kbit/s  |      6 kbit/s
--------------------------------------+------------------
 packets                      373  |          2284
--------------------------------------+------------------
         max               10 p/s  |        45 p/s
     average                6 p/s  |        38 p/s
         min                2 p/s  |         5 p/s
--------------------------------------+------------------
 time                  60 seconds
```

Figure 4.52. Outgoing Ethernet traffic for the IQRF & BLE simulation.

As regards the CPU, its usage increased by 10% while the temperature touched 60°C before the simulation was stopped. The RAM usage increased by less than 0.5% instead.

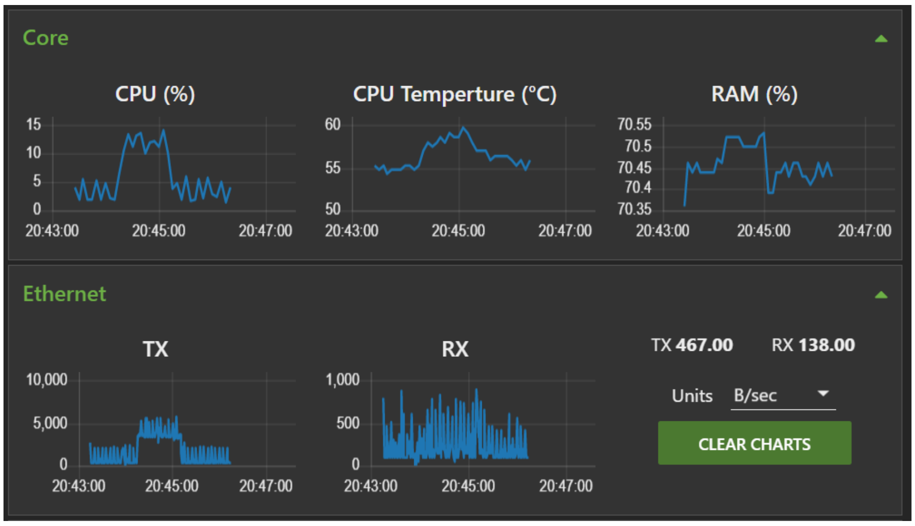


Figure 4.53. Graphical results of the IQRF & BLE simulation.

The results of this simulation are summed up in the table below.

Table 4.15. Impact of the BLE simulation on the IoT Router.

Field	Impact
CPU load(%)	+ 10%
CPU temperature (°C)	+ 5 °C
RAM usage(%)	+ 0.15%
Ethernet TX	+ 2.78 Kbyte/s
Ethernet RX	+ 0.01 Kbyte/s

4.11 Analog & digital sensors

One of the simplest and more common IoT scenarios involves reading data from a wired sensor and send its value over the internet (to the cloud, database, servers etc).

The goal of this simulation is to test the capabilities of the IoT Router to serve this purpose. In particular, the focus is on the performance when several sensors are wired to it. Since the IoT Router doesn't have a built in ADC (Analog to Digital Converter) it can be directly wired only with digital sensors. An example can be a network of Dallas DS18B20 temperature sensors connected together via 1-Wire bus, i.e. using just one GPIO pin of the IoT Router. Despite theoretically a huge number of 1-Wire sensors can be connected to the same pin, the real number is largely limited by the length of the cables and by the power requirements. The GPIO header has 12 pins that can be used to host 1-Wire sensors. Per datasheet the DS18B20 has a maximum current consumption of 1.5mA [23] and the IoT Router can handle up to 50mA per GPIO bank. This means that it could be safe to connect at least 30 sensors to it (assuming a relatively short length of each wire). Note that for this simulation the traffic measurement and the CPU and RAM measurements are not carried out simultaneously. The reason is because the traffic generated is relatively contained and on the chart it wouldn't be clearly distinguishable from the traffic related to the performance of the board. Therefore, the Ethernet throughput is measured first and then the CPU and RAM usage.

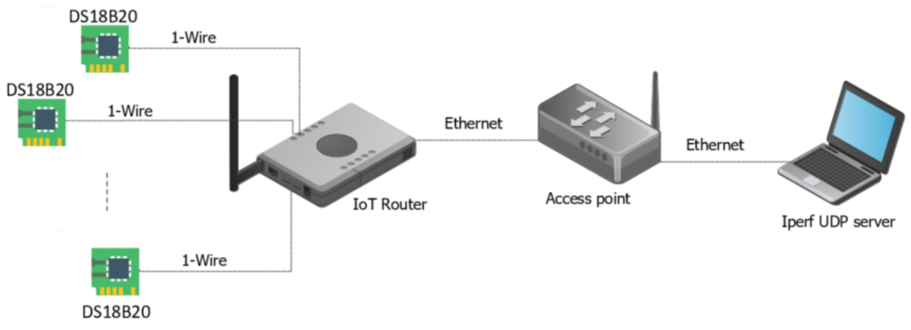


Figure 4.54. Scenario with data collected from digital sensors.

This scenario simulates the reading from 30 DS18B20 sensors every 10 seconds for a minute. This is done to see how the board performs when it has to periodically query many sensors roughly at the same time. Note that sampling every 10 seconds might appear too fast for many IoT applications but it is suitable for this experiment as it shows noticeable results in the period of 60 seconds. Figure 4.54 shows the architecture adopted for this test. As for the other simulations, the data collected from the IoT Router is sent as UDP traffic via Ethernet to a UDP server running on the laptop. The outgoing Ethernet traffic is monitored both with the GUI and with the vnstat tool.

4.11.1 Implementation

To test this scenario with the GUI the following steps are required:

- 1) From the GUI dashboard, click the `plus` button to create a new service.
- 2) Drag 30 DS18B20 nodes and 2 UDP nodes to the editor canvas and connect them as in Fig. 4.55.

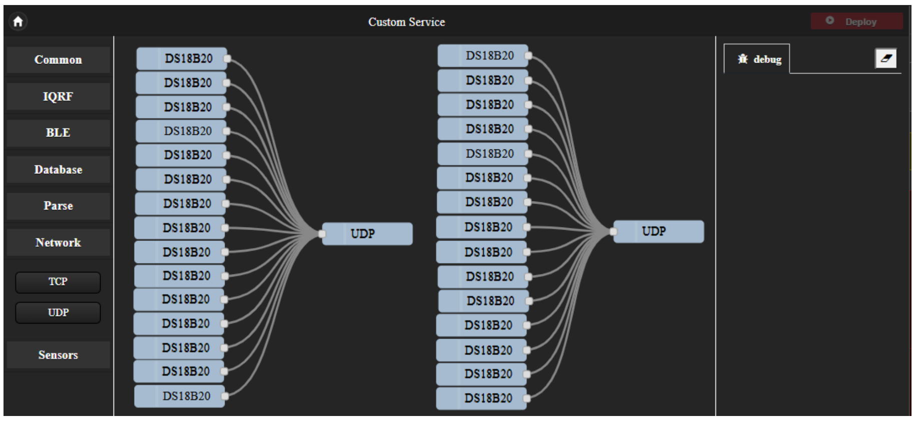


Figure 4.55. Design of DS18B20 scenario with the GUI editor.

- 3) Double click on each DS18B20 node and Click on the + button, Fig. 4.56, leaving the field `Sensor-ID` empty.
- 4) Select time interval of 10 seconds.
- 5) Click the toggle button to activate the command (it is not sent yet).
- 6) Tick the simulation check-box.

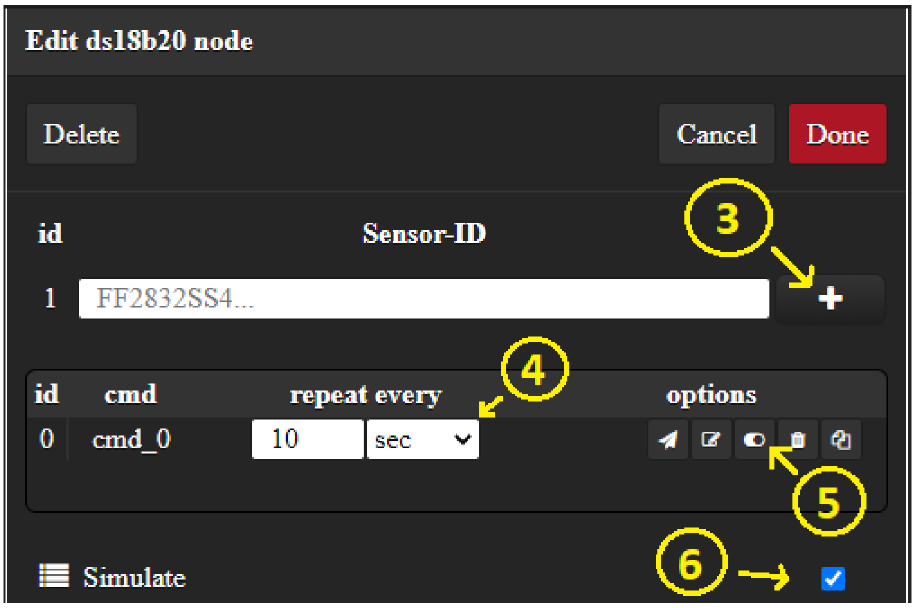


Figure 4.56. Form for the DS18B20 node with relative parameters.

- 7) Click on `Done`.
- 8) Double-click on the UDP node and fill the form as described in Table 4.16.

Table 4.16. Parameters of the UDP node.

Field	Value
Host	10.0.0.33
Port	5001

- 9) Click on `Done` and then `Deploy`. If prompted for a name, select a name (e.g. DS18B20) and click `Continue`.
- 10) Go back to the dashboard and click on the `Play` button on the corresponding service, Fig. 4.57.

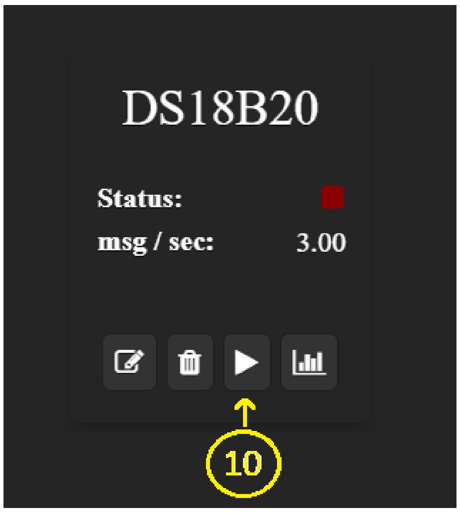
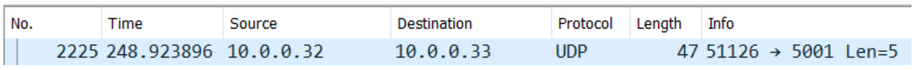


Figure 4.57. Run the DS18B20 simulation.

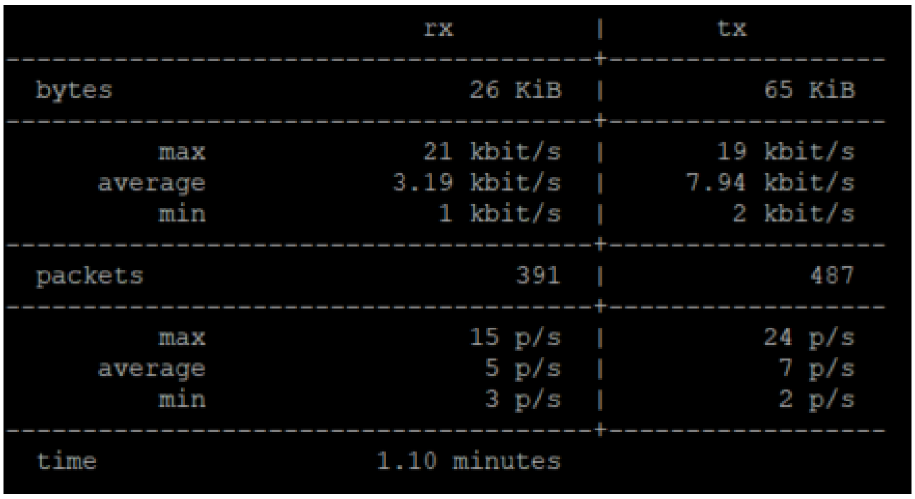
The IoT Router is now simulating the readings of temperature value every 10 seconds from all the sensors. Each reading generates a payload of 5 bytes (number with 2 decimal points converted to string, e.g. 25.12 → 32 35 2E 31 32) therefore a contained impact is expected on the Ethernet throughput and the CPU performance. After an interval of 60 seconds the simulation is stopped (same button) and the result can be observed in the charts page. With a UDP frame of 47 bytes, Fig. 4.58, the expected increase of outgoing traffic is 47 Bytes x 30 packets, i.e. 1410 Bytes.



No.	Time	Source	Destination	Protocol	Length	Info
2225	248.923896	10.0.0.32	10.0.0.33	UDP	47	51126 → 5001 Len=5

Figure 4.58. UDP frame captured with Wireshark.

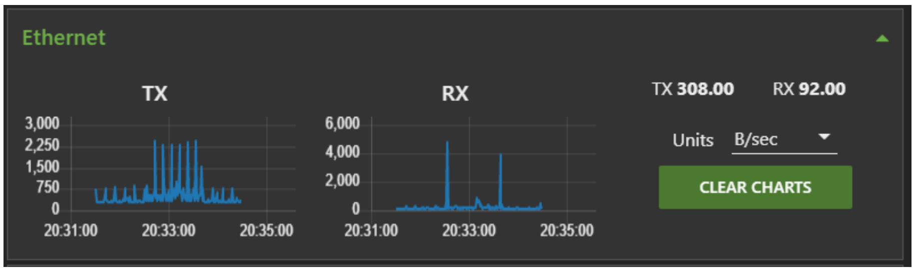
The Fig. 4.59 shows the Ethernet traffic with an average of 7.94 Kbit/s transmitted and 3.19 Kbit/s received. In this case the traffic was not homogeneously spread across time, therefore the average value is lower. However, it is still visible the highest peak of 19 Kbit/s which represents an increment of 1582.5 Bytes/sec with respect to the idle condition, which is not too far from the value mentioned above.



```
                               rx        |          tx
--------------------------------------+------------------
  bytes                        26 KiB   |         65 KiB
--------------------------------------+------------------
          max                21 kbit/s  |      19 kbit/s
      average              3.19 kbit/s  |    7.94 kbit/s
          min                 1 kbit/s  |       2 kbit/s
--------------------------------------+------------------
  packets                         391   |            487
--------------------------------------+------------------
          max                  15 p/s   |         24 p/s
      average                   5 p/s   |          7 p/s
          min                   3 p/s   |          2 p/s
--------------------------------------+------------------
  time                    1.10 minutes
```

Figure 4.59. Ethernet traffic of the DS18B20 simulation.

Fig. 4.60 shows the graphical representation of the result (without CPU and RAM measurement).

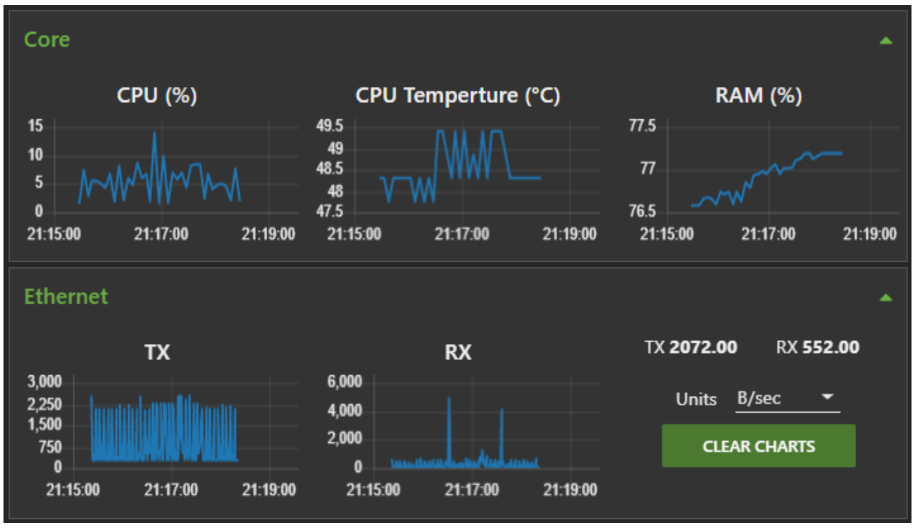


TX 308.00 RX 92.00

Units B/sec

Figure 4.60. Ethernet traffic generated by the DS18B20 simulation (without CPU & RAM measurement).

Each peak in the TX chart represents the transmission of the simulated temperature values. Note that six peaks are visible as expected, plus one smaller peak at the end of the measurement. This smaller peak as well as the two peaks visible in the RX chart are caused by the exchange of messages between the GUI and Node-RED when the simulation starts/stops. The simulation is repeated as before with the addition of the measurements of CPU and RAM. Figure 4.61 shows how this addition changes the Ethernet traffic but it also shows a contained impact on the CPU and RAM usage.



TX 2072.00 RX 552.00

Units B/sec

Figure 4.61. Ethernet traffic generated by the DS18B20 simulation (with CPU & RAM measurement).

The results of this simulation are summed up in the table below.

Table 4.17. Results of DS18B20 simulation.

Field	Impact
CPU load(%)	Negligible
CPU temperature (°C)	+1°C
RAM usage(%)	+0.8%
Ethernet TX	+ 1582.5 Byte/sec
Ethernet RX	+ 118.75 Byte/sec

4.12 Bridge functionality

The following implementation is designed to show how the IoT Router can exchange data between technologies not involving TCP/IP protocol. Figure 4.62 shows the combinations currently supported by the GUI, where data can be collected from a module (e.g. IQRF) and passed to a different one (e.g. BLE). Two tests are carried out in this section: first the bridge between IQRF and BLE is presented, then follows the implementation of actuators driven by sensors data.

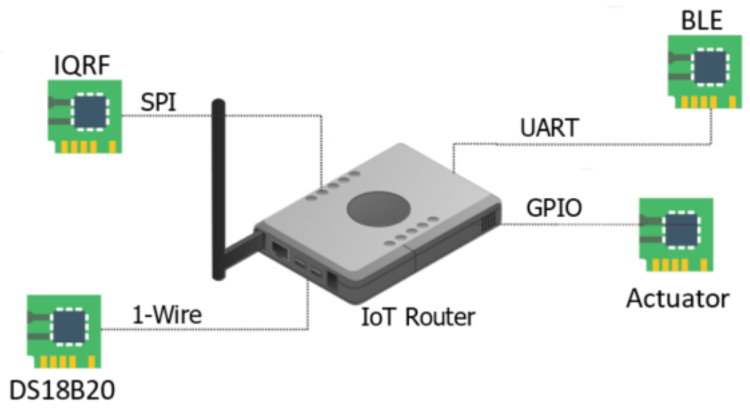


Figure 4.62. Scenario with data bridged between technologies.

The first test consists of reading data (temperature value) from an IQRF node deployed in the field and send it to a BLE node nearby. To keep it simple and easier to visualize, a mobile phone is used instead of the remote BLE sensor. Using the mobile app `Serial Bluetooth Terminal`, it is possible to see all the incoming messages in real time. Fig. 4.63 shows the architecture of this scenario.

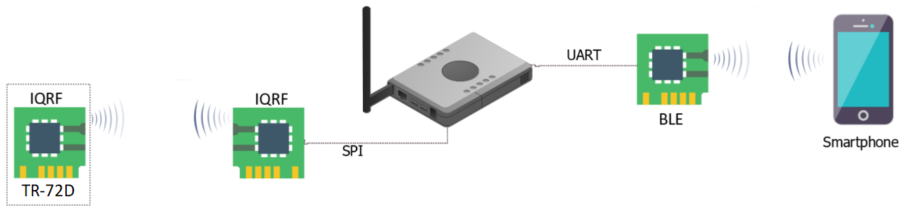


Figure 4.63. Data bridged between IQRF and BLE technologies.

Since the idea is to show how the traffic is bridged between different technologies, just few messages will be exchanged in both directions instead of the highest traffic possible. The second test can be seen as a variant of the first one. A typical scheme of IoT applications includes collecting data from remote sensors and use it to pilot actuators (sometimes remote as well). This test aims to represent this type of situations but without the wireless technologies, since the IoT Router offers this possibility as well. The Fig. 4.64 shows the architecture adopted.

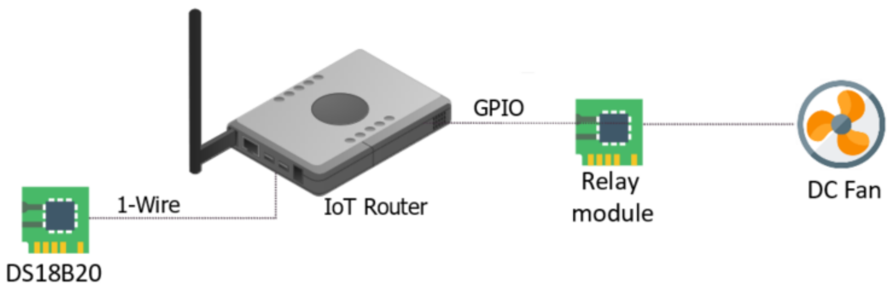


Figure 4.64. Data from local sensor pilots actuator.

The source of data is a temperature sensor (DS18B20) connected to the GPIO of the board. The GUI however is also able to simulate the behavior of this sensor generating random values in a certain interval, which can be useful for reproducibility of the results. On the other side a relay module is used as actuator to switch on a DC fan when the temperature value crosses a threshold. To trigger the relay, the temperature sensor is put close to a source of heat.

4.12.1 Bridge IQRF - BLE

The procedure to build a bridge scenario with the GUI remains the same as for the previous cases, using the GUI editor. It is important to note that the IQRF network `MUST` be implemented a priori (IQRF node bonded to coordinator) with its own IDE [41] in order for this to work. This is because the GUI is currently not able to compile code and upload it directly to the module. The nodes involved are:

- IQRF-DPA
- Extract
- BLE TX
- BLE RX

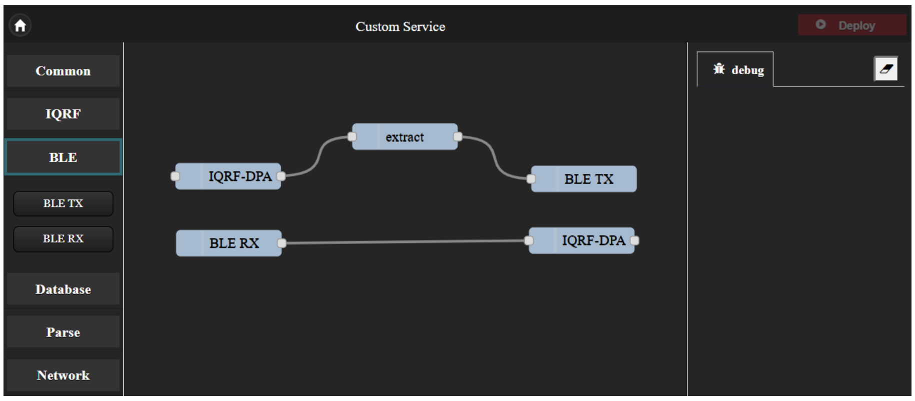


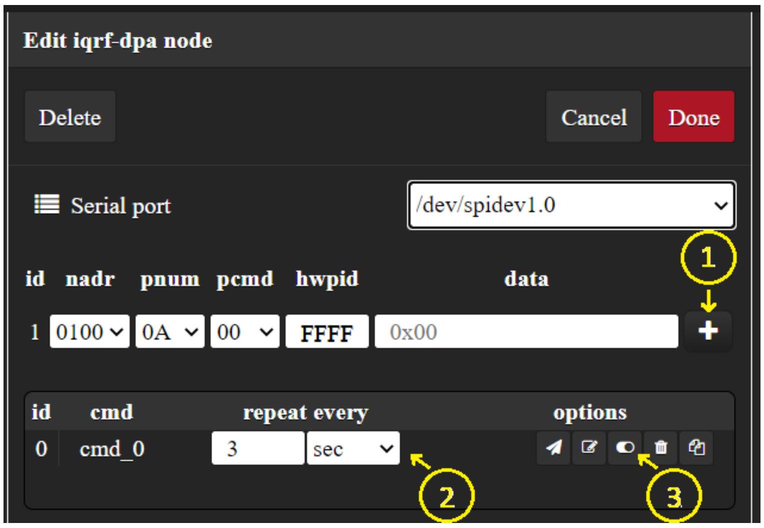
Figure 4.65. Design of a bridge scenario with the GUI.

Figure 4.65 shows how these nodes are connected. The idea is to query a IQRF TR-72D transceiver located nearby to get its temperature value and send it to the mobile phone over BLE. Since the IQRF-DPA node returns an array of bytes, it is necessary to extract the temperature value from there, for this purpose the `extract` node has been created. As regards the other direction, the data sent from the mobile phone is used as a DPA command for the IQRF network. The IQRF-DPA node is configured with the parameters of the table shown below.

Table 4.18. Parameters of IQRF DPA node in bridge simulation.

Field	Value	Meaning
Serial port	/dev/spidev1.0	SPI bus/device
nadr	0100	Node address [00 01], bytes swapped
pnum	0A	Thermometer peripheral
pcmd	00	Get temperature
hwpid	FFFF	FFFF = Any HW profile ID
data	leave empty	Not used

Once the form is filled, click on the plus button, Fig. 4.66 (pt 1), select an arbitrary interval of time (e.g. 3 seconds) and activate the command, Fig. 4.66 (pt 3).



Edit iqrf-dpa node

Delete | Cancel | Done

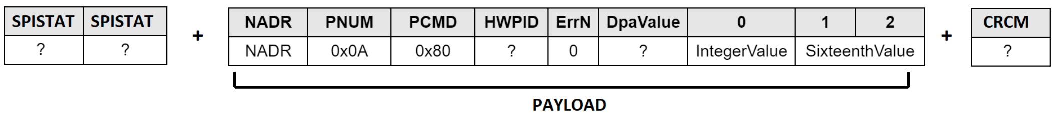
Serial port: /dev/spidev1.0

id	nadr	pnum	pcmd	hwpid	data
1	0100	0A	00	FFFF	0x00

id	cmd	repeat every		options
0	cmd_0	3	sec	

Figure 4.66. Set up DPA command for IQRF-DPA node.

When the temperature peripheral is used, the payload of the response message looks like this:



SPISTAT	SPISTAT
?	?

+

NADR	PNUM	PCMD	HWPID	ErrN	DpaValue	0	1	2
NADR	0x0A	0x80	?	0	?	IntegerValue	SixteenthValue	

+

CRCM
?

Figure 4.67. Response to the `getTemperature` DPA request[42].

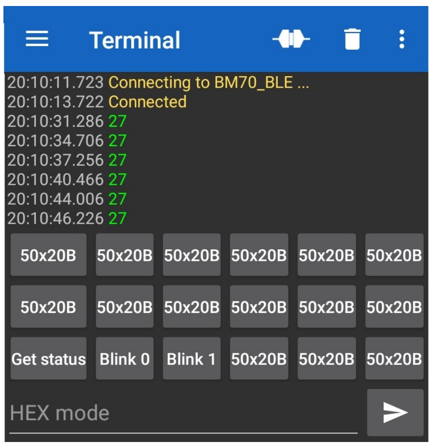
The byte extracted in this case is the `IntegerValue` of the payload. Since the fields `NADR` and `HWPID` contain two bytes each, the position of the temperature value (considering the raw SPI packet) is 10, therefore this is the parameter to use for the `extract` node, Table 4.19.

Table 4.19. Parameter of the `extract` node in the bridge simulation.

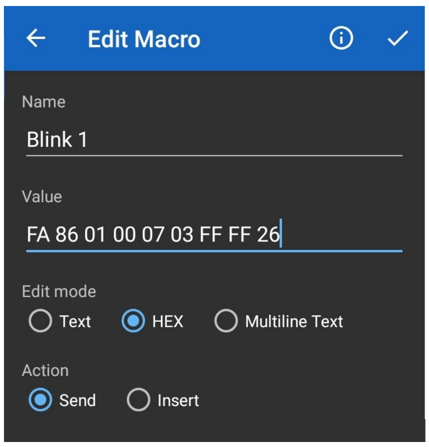
Field	Value
Byte position	10

As regards the BLE TX node, it doesn't need special configuration but the BLE module needs to be set into Application mode in order to use the default characteristic (Transparent UART). This can be done from the BLE RX node form, with a single click on the `App` button. At this point the module becomes visible and the mobile phone can connect to it through the app `Serial Bluetooth Terminal`. To send data from the phone to the IQRF network instead, it is sufficient to connect the BLE RX node to another IQRF-DPA node as shown in Fig. 4.65. For these two nodes no configuration is required as the DPA request for the IQRF module is prepared and sent from the mobile phone as shown below. This simplified scenario has the only purpose to test the BLE module and the bridge functionality of the IoT Router, real implementations might require higher logic. Now the scenario can be deployed and run as for the previous services (e.g. Fig. 4.57). In this case every 3 seconds the temperature read from the IQRF transceiver is displayed on the terminal of the mobile phone app, Fig. 4.68 (left).

Vice versa it is possible to send DPA commands from the terminal as DPA requests. An example is the packet shown below under the field `value`, Fig. 4.68 (right), which represents a DPA request for the bonded IQRF node to blink the green LED once. The picture has been taken directly from the mobile phone app.



```
20:10:11.723 Connecting to BM70_BLE ...
20:10:13.722 Connected
20:10:31.286 27
20:10:34.706 27
20:10:37.256 27
20:10:40.466 27
20:10:44.006 27
20:10:46.226 27
```



Name: Blink 1

Value: FA 86 01 00 07 03 FF FF 26

Edit mode: Text [✓ HEX] Multiline Text

Action: [✓ Send] Insert

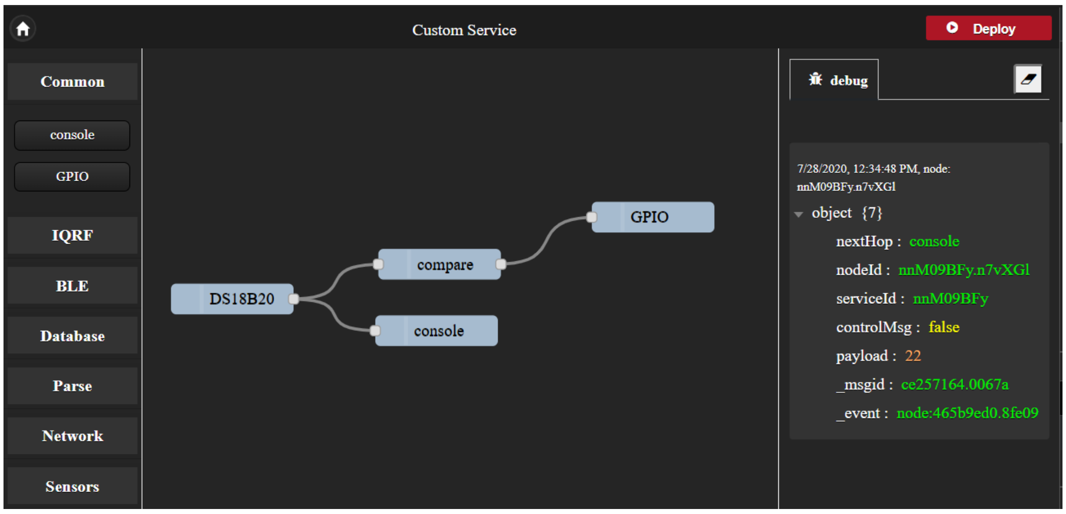
Figure 4.68. Temperature value received (left) & DPA request to blink LED (right).

Table 4.20. Byte composition of DPA request.

Byte	Role
FA	SPI write/read with DPA
86	SPI Data length (6 bytes + 0x80)
01	Node address (LSB)
00	Node address (MSB)
07	HW peripheral LED GREEN
03	Blink command
FF	HW profile ID (LSB)
FF	HW profile ID (MSB)
26	CRCM

4.12.2 Sensor & actuator

This section of the bridge simulation shows how the IoT Router can control actuators based on input data from sensors. In this case a temperature value is collected every 3 seconds from a DS18B20 sensor and it is compared with a threshold value of 27 °C. When the temperature is higher, a relay module connected to the GPIO 41 of the IoT Router is activated causing a small DC fan to turn on. Figure 4.69 shows which nodes are used for this test.



```
7/28/2020, 12:34:48 PM, node:
nnM09BFy.n7vXGl
▾ object {7}
    nextHop : console
    nodeId : nnM09BFy.n7vXGl
    serviceId : nnM09BFy
    controlMsg : false
    payload : 22
    _msgid : ce257164.0067a
    _event : node:465b9ed0.8fe09
```

Figure 4.69. Implementation of actuators with the GUI editor.

The DS18B20 is the same node used for section 4.11.1, it can be configured similarly as shown in Fig. 4.56, but with a shorter sampling interval (e.g. 3 seconds). The `compare` node returns true or false whether the input payload satisfies a conditions specified by the user (e.g. payload $\geqslant 27$) while the GPIO node can control the state of a GPIO pin (user defined) based on the boolean input. The `console` node has the only purpose to display the messages generated by the DS18B20 node. An example is shown on the debug console, Fig.4.69, where the payload of the message shows a recorded temperature of 22 °C. In this way it is easier to follow the temperature changes when the threshold is crossed. As regards the `compare` and `GPIO` nodes, the parameters described in tables 4.21 and 4.22 are used (double click on a node to configure it).

Table 4.21. Parameters of the `compare` node.

Field	Value
Rule	$\geqslant$
Value	27

Table 4.22. Parameters of the `GPIO` node.

Field	Value
GPIO pin	GPIO 41

Once the service is deployed it can be run with the `play` button, Fig. 4.70, from the dashboard page.

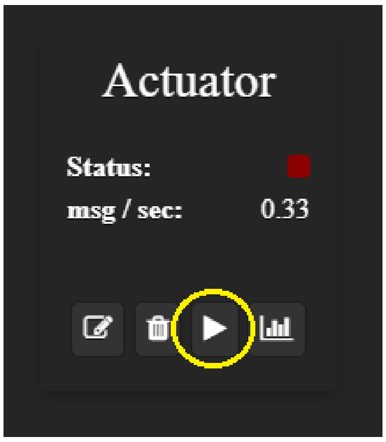


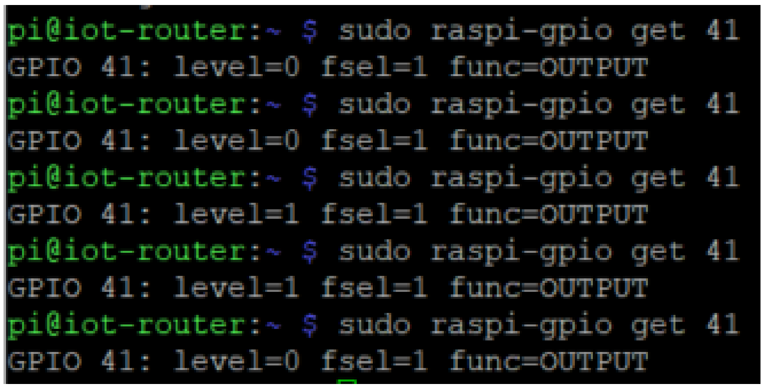
Figure 4.70. Service tag of the sensor/actuator scenario.

The temperature is periodically measured once the service is running. In this case, when it crosses 27 °C the relay module connected to the GPIO is activated and the DC fan will start moving. For reproducibility purposes this behavior can be observed even without the relay and the fan, it is sufficient to periodically run the following command from a SSH terminal of the IoT Router:

```
sudo raspi-gpio get 41
```

This command will log the state of the GPIO 41. Figure 4.71 shows how the output looks like, the field `level` represents the state of the pin:

- level=0 → LOW (relay open)
- level=1 → HIGH (relay closed)



```
pi@iot-router:~ $ sudo raspi-gpio get 41
GPIO 41: level=0 fsel=1 func=OUTPUT
pi@iot-router:~ $ sudo raspi-gpio get 41
GPIO 41: level=0 fsel=1 func=OUTPUT
pi@iot-router:~ $ sudo raspi-gpio get 41
GPIO 41: level=1 fsel=1 func=OUTPUT
pi@iot-router:~ $ sudo raspi-gpio get 41
GPIO 41: level=1 fsel=1 func=OUTPUT
pi@iot-router:~ $ sudo raspi-gpio get 41
GPIO 41: level=0 fsel=1 func=OUTPUT
```

Figure 4.71. SSH terminal of the IoT Router.

Chapter 5
Conclusion

During this thesis work, a lot of new knowledge has enlarged my view of the wireless technologies. In particular regarding some of the most widely used technologies within the Internet of Things, how they work, their protocols and limits. Developing software has also strengthened my interest towards the low level communication among hardware components as well as higher logic such as web development. The second chapter of this work presented an introduction to the main software used during the tests. The third chapter described the actual hardware involved and its main features. An overview of the available IoT technologies was also provided together with the possible database solutions. The fourth chapter started with the practical part of the thesis. Two main scenarios and the measurements to be carried out were listed, followed by the description of the designed Graphical User Interface and its dependency from Node-RED. A brief analysis of the impact of the GUI on the IoT Router was also provided as a reference for the next measurements. Always in the fourth chapter were described the bandwidth measurement of the Ethernet and Wi-Fi interfaces with the focus on TCP and UDP traffic in both directions. The results of these tests showed how the IoT Router is able to reach around 95% of its declared maximum speed via Ethernet (100 Mbit/s) and 30% via Wi-Fi dongle (150 Mbit/s) under the specified conditions. Although both solutions could guarantee sufficient bandwidth for many IoT use cases, it is recommended to use the Ethernet connection as main interface towards the public network. These measurements were followed by a series of simulations involving the hardware modules described in chapter 3. First of all the IQRF module was tested simulating the highest traffic possible via SPI using the DPA protocol. The impact was noticeable but it didn't represent a bottleneck for the IoT Router. This specific implementation in real cases is probably not very common though, since for mesh networks the data throughput will be lower with an increasing number of routing hops. Similarly, the BLE simulation was carried out. Its impact on the board was slightly higher than IQRF, proportionally to its higher traffic. Considering that applications involving BLE usually have different services other than just the transparent UART, it is possible that under other conditions a higher data rate can be achieved. The implementation of LoRa technology as a gateway was the next test performed. LoRa provides a ready to use software for ARM based processors, it was sufficient to install it and modify its configuration to have it up and running. As mentioned before, the software runs as a service in the background and its impact on the CPU and RAM was quite contained. The next step was to test the connection with local and remote databases. The IoT Router proved to be fairly stable while executing between 200 and 250 queries per second with the local solution, in fact all expected values were stored in the proper order. However, this process used roughly 40% of the CPU and caused its temperature to increase rapidly to 65°C, from the charts it was visible how the temperature would have increased more for a longer simulation. Nevertheless, for real cases with lower query execution time, the IoT Router works greatly as a local data-logger. Still as part of the first scenario (TCP/IP), the NB-IoT module was tested. Its implementation was carried out with the AT commands over serial port and its correct functionality was proven sending and receiving data through the T-mobile network. The transmission of roughly 6 Kbyte/min of data didn't affect the performance of the IoT Router. The latency was measured with an average value of 209.9 ms which appears to be suitable for IoT applications. The NB-IoT test was followed by the combination of two technologies: IQRF and BLE. This test was essentially the overlap of the two individual ones and, as expected, the Ethernet throughput increased as well as the CPU temperature. From the computational point of view, this simulation kept the CPU usage below 15% leaving plenty of room for other operations. The last task involving the TCP/IP protocol was carried out testing how the IoT Router would handle a network of sensors directly connected to it, in this case via 1-Wire interface. With the GUI was simulated the acquisition of data from 30 sensors whose values were sent to a UDP server. The traffic generated was fairly low (1.5 Kbyte/s) as for the impact on the CPU, showing how the IoT Router could handle tens of sensors being processed in the background without particular problems. At this point the tasks for the second scenario, i.e. not involving TCP/IP, were carried out. First it was presented a bridge between IQRF and BLE technologies where data was generated and sent in both directions, then the control of actuators driven by sensors values was tested. Both solutions were successfully implemented with the GUI and Node-RED. The bridge functionality was tested with two technologies but it could be extended to all others giving much more higher level control to the user. Given the number of available GPIO pins and other hardware peripherals, it also seems feasible for the IoT Router to control tens of actuators (eventually using expansion boards) without a noticeable impact on the performance. With all the considerations above, it seems reasonable to think that the IoT Router is able to handle most of the aforementioned technologies running simultaneously, with the exception of the NB-IoT and LoRa modules. These modules in fact share the same physical slot on the back of the board, therefore, just one at a time can be used. However, the fact that even other technologies can be brought on board (different custom modules via miniPCI connectors) makes the IoT Router a potential solution for a huge pool of situations within the Internet of Things.

References

[1] *Node-RED [online]. OpenJS Foundation, 2020 [cit. 2020-08-05]. Available at: .*
`https://nodered.org/`.

[2] *Iperf [online]. TCP/IP bandwidth measurement tool. France, 2020 [cit. 2020-08-04]. Available at: .*
`https://iperf.fr/`.

[3] *Wikipedia contributors. Microsoft Azure SQL Database. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 16 December 2019 07:14 UTC [cit. 2020- 08-04]. Available at:.*
`https://en.wikipedia.org/w/index.php?title=Microsoft_Azure_SQL_Database`.

[4] *Wikipedia contributors. InfluxDB. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 8 July 2020 22:14 UTC [cit. 2020-08-04]. Available at:.*
`https://en.wikipedia.org/w/index.php?title=InfluxDB&oldid=966739467`.

[5] *Go (programming language). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-11]. Available at:.*
`https://en.wikipedia.org/wiki/Go_(programming_language)`.

[6] *Wikipedia contributors. Graphite (software). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 28 May 2020 13:26 UTC [cit. 2020-08-04]. Available at:.*
`https://en.wikipedia.org/wiki/Graphite_(software)`.

[7] *Wikipedia contributors. VnStat. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 1 January 2020 03:48 UTC [cit. 2020-08-04]. Available at: .*
`https://en.wikipedia.org/wiki/VnStat`.

[8] *Wikipedia contributors. Wireshark. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 5 July 2020 13:38 UTC [cit. 2020-08-04]. Available at:.*
`https://en.wikipedia.org/w/index.php?title=Wireshark&oldid=966166988`.

[9] *Wireshark - About [online]. Kansas City: Wireshark Foundation, 2020 [cit. 2020-08-04]. Available at:.*
`https://www.wireshark.org/`.

[10] *Talk:Dumb terminal. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 9 June 2018 [cit. 2020-08-13]. Available at:.*
`https://en.wikipedia.org/wiki/Talk%3ADumb_terminal`.

[11] *Picocom - minimal dumb-terminal emulation program [online]. U.S. die.net, 2020 [cit. 2020-08-13]. Available at: .*
`https://linux.die.net/man/8/picocom`.

[12] *Point-to-Point Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2020-08-13]. Available at:.*
`https://en.wikipedia.org/wiki/Point-to-Point_Protocol`.

[13] *HNYK, Pavel, Lukas VOJTECH a Marek NERUDA. LPWAN IoT router prototype [online]. Prague, 2019 [cit. 2020-08-05]. Technical documentation. Czech Technical University in Prague.*

[14] *PCI Express. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-11]. Available at:.*
`https://en.wikipedia.org/w/index.php?title=PCI_Express&oldid=971973804`.

[15] *HNYK, Pavel. IOT_Router_Schematic [online]. Prague, 2019 [cit. 2020-08-05]. Technical documentation. Czech Technical University in Prague.*

[16] *Quectel. Quectel EC20 Mini PCIe, Multi-mode LTE Module [online]. Shanghai, China, 2015 [cit. 2020-08-05]. Available at:.*
`https://www.quectel.com/product/ec20r21minipcIe.htm`.

[17] *Quectel EC20EA-MINIPCIE LTE module [online]. Italy: RS Components Srl, 2020 [cit. 2020-08-14]. Available at:.*
`https://it.rs-online.com/web/p/moduli-gsm-e-gprs/9084195/`.

[18] Quectel BG96,LTE Cat M1 & Cat NB1 & EGPRS Module. Quectel [online]. 2019, 1(V1.7), 1-2 [cit. 2020-08-05]. Available at:.
`https://www.quectel.com/product/bg96minipcIe.htm`.

[19] *LoRaWANTM Concentrator Card Mini PCIe [online]. Stuttgart, Germany, 2019 [cit. 2020-08-05]. Available at:.*
`https://www.n-fuse.co/assets/products/lrwccx/lrwccx-mpcie-datasheet.pdf`.

[20] *LoRa®. PicoGW_hal. Github.com [online]. Camarillo (CA, USA): GitHub, 2019 [cit. 2020-08-05]. Available at:.*
`https://github.com/Lora-net/picoGW_hal`.

[21] LoRa®. *LoRa®. PicoGW_packet_forwarder [online]. Camarillo (CA, USA): GitHub, 2017 [cit. 2020-08-05]. Available at:.*
`https://github.com/Lora-net/picoGW_packet_forwarder`.

[22] *Concentrator Card LRWCCx-MPCIE for LoRaWAN® technology [online]. Stuttgart, Germany: n-fuse, 2020 [cit. 2020-08-14]. Available at:.*
`https://www.n-fuse.co/devices/LoRaWAN-Concentrator-Card-mini-PCIe.html`.

[23] *MAXIMINTEGRATED, ds18b20_current. Programmable Resolution 1-Wire Digital Thermometer [online]. Rev 6. San Jose, California: Maxim Integrated Products, 2019 [cit. 2020-08-04]. Available at:.*
`https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`.

[24] *Waterproof thermometer - sonda 2m DS18B20 [online]. Havlíčkův Brod: Arduino-shop.cz, 2020 [cit. 2020-08-05]. Available at:.*
`https://arduino-shop.cz/arduino/2029-teplomer-vodotesny-sonda-2m-ds18b20.html?gclid=Cj0KCQjws_r0BRCwARIsAMxfDRjWksf0uIm2nRazbhC6_4qTu013fJf8p4XX0-XsS3s9w3GTvAUD-YwaAgPaEALw_wcB`.

[25] *Relay module 5V - 1 channel optically separated [online]. Havlickuv Brod: Arduino-shop.cz, 2020 [cit. 2020-08-05]. Product code: 1517442331. Available at: .*
`https://arduino-shop.cz/`.

[26] *What is the Azure SQL Database service? [online]. Albuquerque, New Mexico: Microsoft, 2019 [cit. 2020-08-05]. Available at:.*

https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview.

[27] *InfluxDB Cloud [online]. San Francisco, CA: InfluxData, 2020 [cit. 2020-08-05]. Available at:.*
https://www.influxdata.com/products/influxdb-cloud/.

[28] *STRAKA, Tomáš. Testbed IDS/IPS bezpečnostní sondy SonIoT [online]. Prague, 2020 [cit. 2020-0- -05]. Available at:.*
https://dspace.cvut.cz/bitstream/handle/10467/86092/F3-DP-2020-Straka-Tomas-Testbed_IDS_IPS_bezpecnostni_sondy_SonIoT.pdf?sequence=-1&isAllowed=y.

[29] *IQRF DPA Framework: p 12-16 [online]. Prague: IQRF Tech, 2020 [cit. 2020-08-05]. Available at:.*
https://static.iqrf.org/Tech_Guide_DPA-Framework-414_200403.pdf.

[30] *Microchip Developer Help. Microchip Transparent UART Service for BM70-RN4870 [online]. Chandler, Arizona: Microchip Technology, 2020 [cit. 2020-08-05]. Available at:.*
https://microchipdeveloper.com/wireless:ble-mchp-transparent-uart-service.

[31] *MORICH, Kai. Serial Bluetooth Terminal [online]. Hockenheim, Germany: Google Commerce, 2020 [cit. 2020-08-05]. Available at: .*
https://play.google.com/store/apps/developer?id=Kai+Morich&hl=eng.

[32] *The Things Network [online]. –: The Things Industries, 2020 [cit. 2020-08-10]. Available at: .*
https://www.thethingsnetwork.org/.

[33] WIEDEMANN, Felix. Build a LoRa Gateway with n-fuse mPCIe card. Hackster.io [online]. 2018, 1(1), 1-5 [cit. 2020-08-05]. Available at:.
https://www.hackster.io/fewi/build-a-lora-gateway-with-n-fuse-mpcie-card-71f0e1.

[34] *SAVANT, Rajasa. Node-red-contrib-azure-sql. Flows.nodered.org/ [online]. OpenJS Foundation, 2020 [cit. 2020-08-04]. Available at: .*
https://flows.nodered.org/node/node-red-contrib-azure-sql.

[35] *T-mobile [online]. Czech Republic: T-Mobile Czech Republic, 2020 [cit. 2020-08-11]. Available at:.*
https://www.t-mobile.cz/osobni.

[36] *Narrowband IoT. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-08-11]. Available at:.*
https://en.wikipedia.org/w/index.php?title=Narrowband_IoT&oldid=968300943.

[37] Quectel. BG96 TCP/IP AT Commands Manual. AT Commands Manual [online]. 2017, 1(1.0), p. 17 [cit. 2020-08-12]. Available at:.
https://docplayer.net/82358690-Bg96-tcp-ip-at-commands-manual.html.

[38] *BG96 AT Commands Manual [online]. China: Quectel, 2018 [cit. 2020-08-11]. Available at: .*
https://docplayer.net/82358690-Bg96-tcp-ip-at-commands-manual.html.

[39] *Picocom [online]. San Francisco (CA): GitHub, 2018 [cit. 2020-08-12]. Available at:.*
https://github.com/npat-efault/picocom.

[40] *Iperf [online]. Public TCP/UDP server. France, 2020 [cit. 2020-08-04]. Available at:.*
`https://iperf.fr/iperf-servers.php`.

[41] *IQRF Startup package, OS v4.03D for TR-7xD. Iqrf.org [online]. Jicin, Czech Republic: IQRF Tech [cit. 2020-08-05]. Available at:.*
`https://www.iqrf.org/product-detail/iqrf-ide`.

[42] *IQRF DPA Framework: p 49-50 [online]. Prague: IQRF Tech, 2020 [cit. 2020-08-05]. Available at:.*
`https://static.iqrf.org/Tech_Guide_DPA-Framework-414_200403.pdf`.

Appendix A
List of abbreviations

4G	Fourth Generation
ACK	Acknowledgment
ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
APN	Access Point Name
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange
AT commands	Hayes command set
ATM	Asynchronous Transfer Mode
BLE	Bluetooth Low Energy
BSD	Berkeley Software Distribution
CDMA	Code Division Multiple Access
cm	Centimeters
CPU	Central Processing Unit
CRCM	Cyclic Redundancy Check Master
CSV	Comma-Separated Values
DB	Database
DC	Direct Current
DC-HSPA	Dual Carrier High Speed Packet Access
DPA	Direct Peripheral Access

EDGE	Enhanced Data rates for GSM Evolution
EEPROM	Electrically Erasable Programmable Read-Only Memory
EGPRS	Enhanced General Packet Radio Service
FCC	Federal Communications Commission
FDDI	Fiber Distributed Data Interface
FM	Firmware
GATT	Generic Attribute Profile
GB	Gigabytes
Gbit/s	Gigabit per second
GFSK	Gaussian Frequency-Shift Keying
GHz	Gigahertz
GLONASS	GLObal NAvigation Satellite System
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HDLC	High-Level Data Link Control
HEX	Hexadecimal
HSDPA	High-Speed Downlink Packet Access
HSPA	High Speed Packet Access
HSPA+	Evolved High Speed Packet Access
HSUPA	High-Speed Uplink Packet Access
HTTP	Hypertext Transfer Protocol
HW	Hardware
HWPID	Hardware Profile ID
I2C	Inter-Integrated Circuit

IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISED	Innovation, Science and Economic Development
ISM	Industrial, Scientific, Medical
JSON	JavaScript Object Notation
KB	Kilobytes
Kbit/s	Kilobit per second
KCC	Korea Certification
LAN	Local Access Network
LED	Light Emitting Diode
LoRa	Long Range
LPN	Low Power Network
LPWAN	Low Power Wide Area Network
LSB	Least Significant Byte
LTE	Long TErm Evolution
LTS	Long Term Support
MB	Megabytes
Mbit/s	Megabit per second
MCU	Micro Controller Unit
MHz	Megahertz
MIC	Manufacturer Installed Certificate
miniPCIe	Mini Peripheral Component Interconnect Express

MS	Microsoft
MS	Microsoft
MSB	Most Significant Byte
NADR	Node Address
NB-IoT	Narrow Band Internet of Things
NCC	National Certified Counselor
OS	Operative System
PPP	Point to Point Protocol
PTS	Precise Temperature Sensor
PWM	Pulse Width Modulation
RAM	Random Access Memory
REST	Representational State Transfer
RF	Radio Frequency
RPI CM3	Raspberry Pi Compute Module 3
RS232	Recommended Standard 232
RS485	Recommended Standard 485
RTT	Round-Trip Time
RX	Receive
SCTP	Stream Control Transmission Protocol
SD card	Secure Digital card
SIM card	Subscriber Identity Module Card
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SSH	Secure SHell
SSL	Secure Sockets Layer
SW	Software

TCP	Transmission Control Protocol
TD-SCDMA	Time Division Synchronous Code Division Multiple Access
TSDB	Time Series Database
TX	Transmit
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
UUID	Universally unique identifier
VS code	Visual Studio code
WCDMA	Wideband Code Division Multiple Access
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
WSN	Wireless Sensor Network
XML	Extensible Markup Language