

Introduction to the IoT



iot-open.eu



Erasmus+

Introduction to the IoT



iot-open.eu



Erasmus+

Table of Contents

| | |
|---|----|
| 1. Versions | 8 |
| 2. Preface | 9 |
| 2.1. Project information | 9 |
| 3. Introduction | 10 |
| 3.1. Definition of IoT | 10 |
| 3.1.1. What is IoT? | 10 |
| 3.1.2. "Thing" | 11 |
| 3.2. Enabling Technologies | 12 |
| 3.2.1. Small-Scale Computer Systems..... | 12 |
| 3.2.2. Medium-Scale Computer Systems | 13 |
| 3.2.3. Access to the Internet..... | 13 |
| 3.2.4. IP Addressing Evolution..... | 13 |
| 3.2.5. Data Storage and Processing | 13 |
| 3.2.6. Mobile Devices..... | 13 |
| 3.3. Mobility – New Paradigm for IoT Systems | 14 |
| 3.3.1. Cloud Computing | 14 |
| 3.3.2. Fog Computing | 14 |
| 3.3.3. Cognitive IoT Systems | 14 |
| 3.4. Hints for Further Readings on Development Boards, Kits and Sites | 15 |
| 3.5. Data Management Aspects in IoT | 16 |
| 3.6. Application Domains and Their Specifics..... | 17 |
| 4. IoT Hardware overview | 20 |
| 4.1. Most noticeable platforms | 20 |
| 4.2. Embedded Systems Communication Protocols | 20 |
| 4.2.1. Analog | 21 |
| 4.2.2. Digital | 21 |
| 4.2.3. SPI..... | 21 |
| 4.2.4. TWI (I2C) | 22 |
| 4.2.5. 1-Wire | 24 |
| 4.3. Arduino Overview | 28 |
| 4.3.1. Overview of the hardware device used | 29 |
| 4.3.2. Digital input/output pins..... | 30 |
| 4.3.3. Pulse Width Modulation | 30 |
| 4.3.4. Analog pins | 30 |
| 4.3.5. Power and other pins | 31 |

| | |
|---|-----|
| 4.3.6. Memory | 31 |
| 4.3.7. Interface..... | 31 |
| 4.3.8. Size of the board..... | 32 |
| 4.3.9. Arduino shields | 32 |
| 4.3.10. Sensors and sensing | 34 |
| 4.3.11. Drivers and driving | 57 |
| 4.3.12. IoT application example projects | 71 |
| 4.3.13. Setting up development/experimenting environment..... | 72 |
| 4.3.14. Arduino programming fundamentals | 77 |
| 4.4. Espressif SoC Overview | 108 |
| 4.4.1. Espressif SoC..... | 109 |
| 4.4.2. Espressif SoC networking | 121 |
| 4.4.3. ESP programming fundamentals..... | 122 |
| 4.4.4. ESP8266 Wifi Scanner..... | 149 |
| 4.4.5. ESP32 Wifi Scanner | 150 |
| 4.5. Raspberry Pi Overview | 163 |
| 4.5.1. Raspberry Pi general information..... | 164 |
| 4.5.2. Raspberry Pi Sensors | 174 |
| 4.5.3. Raspberry Pi Drivers and driving | 190 |
| 4.5.4. Raspberry Pi OS Guide..... | 201 |
| 4.5.5. Programming fundamentals Raspbian OS | 203 |
| 4.5.6. Programming fundamentals Windows 10 IOT Core..... | 219 |
| 5. Introduction to the IoT Communication and Networking | 255 |
| 5.1. Networking overview..... | 256 |
| 5.2. Communication models | 258 |
| 5.2.1. Device to device and Industry 4.0 revolution..... | 259 |
| 5.2.2. Device to gateway..... | 261 |
| 5.2.3. Device to cloud | 262 |
| 5.3. Media layers - Wired networking | 263 |
| 5.4. Media layers - Wireless protocols | 265 |
| 5.4.1. PHY+MAC+LLC layers | 266 |
| 5.4.2. NET (NWY) Layer | 271 |
| 5.4.3. Host layer protocols..... | 275 |
| 6. Data and Information Management in the Internet of Things | 282 |
| 6.1. IoT data lifecycle | 283 |
| 6.2. IoT data management versus traditional database management systems ... | 286 |
| 6.3. IoT data sources..... | 287 |

| | |
|--|-----|
| 6.4. Main IoT domains generating data | 287 |
| 6.5. Infrastructure and architectures for IoT Data processing: Cloud, Fog, and Edge computing..... | 289 |
| 6.6. IoT data storage models and frameworks..... | 291 |
| 6.7. IoT data processing models and frameworks | 291 |
| 6.8. IoT data semantics | 293 |
| 6.9. IoT data visualisation | 295 |
| 7. IoT security and privacy | 297 |
| 7.1. Types of vulnerabilities of IoT | 299 |
| 7.2. Monitoring of vulnerabilities | 300 |
| 7.3. Malware detection in IoT..... | 301 |
| 7.4. IoT security protocols..... | 306 |
| 7.5. IoT privacy | 310 |
| 7.6. Privacy preservation..... | 313 |
| 7.7. IoT privacy preservation threats | 314 |
| 7.8. Support of confidentiality and methods of authentication..... | 320 |
| 8. Introduction to the IoT Energy consumption | 326 |
| 8.1. Power efficiency in IoT | 326 |
| 8.2. Minimum Energy Performance Standards (MEPS) | 327 |
| 8.2.1. Vertical MEPS | 327 |
| 8.2.2. Horizontal MEPS..... | 328 |
| 8.2.3. Clustered MEPS..... | 328 |
| 8.2.4. Electronic components and their power requirements: motors, sensors, microcontrollers | 328 |
| 8.2.5. IoT software platform | 329 |
| 8.2.6. IoT Battery management systems | 330 |
| 9. Emerging technologies in IoT | 331 |
| 9.1. ROS - a new framework in IoT..... | 331 |
| 9.1.1. What is ROS? | 331 |
| 9.1.2. ROS features:..... | 331 |
| 9.1.3. Operating systems | 333 |
| 9.1.4. ROS architecture..... | 333 |
| 9.1.5. Introduction to ROS programming | 336 |
| 9.1.6. IoT bridge | 342 |
| 9.2. Autonomous transport systems | 346 |
| 9.3. Blockchain | 346 |
| 9.3.1. In search of consensus | 347 |

| | |
|--|-----|
| 9.3.2. Mechanisms of reaching consensus..... | 349 |
| 9.3.3. Ethereum: the new generation of Internet..... | 351 |
| 9.3.4. Conclusions | 352 |

Authors

IOT-OPEN.EU consortium partners collective monography. The full list of contributors is juxtaposed below.

ITMO University

- Aleksandr Kapitonov, Ph. D., Assoc. Prof.
- Dmitrii Dobriborsci, M. sc., Eng.
- Igor Pantiukhin, M. sc., Eng.
- Valerii Chernov, Eng.

ITT Group

- Raivo Sell, Ph. D., ING-PAED IGIP
- Rim Puks, Eng.
- Mallor Kingsepp, Eng.

Riga Technical University

- Agris Nikitenko, Ph. D., Eng.
- Karlis Berkolds, M. sc., Eng.
- Anete Vagale, M. sc., Eng.
- Rudolfs Rumba, M. sc., Eng.

Silesian University of Technology

- Piotr Czekalski, Ph. D., Eng.
- Krzysztof Tokarz, Ph. D., Eng.
- Oleg Antemijczuk, M. sc., Eng.
- Jarosław Paduch, M. sc., Eng.

Tallinn University of Technology

- Raivo Sell, Ph. D., ING-PAED IGIP

University of Messina

- Salvatore Distefano
- Rustem Dautov
- Riccardo Di Pietro
- Antonino Longo Minnolo

Graphic Design and Images

Blanka Czekalska, M. sc., Eng., Arch. Małgorzata Wiktorczyk, M. sc., Eng., Arch.

Editors

Ritankar Sahu

Reviewers

1. Versions

1. Versions

This page keeps track of the content reviews and versions done as a continuous maintenance process

Table 1: Versions and content updates

| # | Version | Change Date | Content updates summary | Other comments |
|---|---------|-------------|--|----------------|
| 1 | v 0.1 | 01.01.2019 | Fixed pre-publish version | |
| 2 | v 0.2 | 07.03.2019 | Updated content on BLE | |
| 3 | v 0.3 | 12.03.2019 | Corrected Python syntax in RPi samples | |

2. Preface

This book and its offshoots were prepared to provide comprehensive information about the Internet of Things. Its goal is to introduce IoT to the bachelor students, master students, technology enthusiasts and engineers that are willing to extend their current knowledge. This book is also designated for teachers and educators willing to extend their knowledge and prepare a course on IoT technology (full or partial).

We (authors) assume that persons willing to study this content do possess some general knowledge about IT technology, i.e. understand what embedded system is, know the general idea of programming and are aware of wired and wireless networking as it exists nowadays.

We believe this book is constituting a comprehensive manual to the IoT technology; however, it is not a full encyclopedy nor exhausts market review. The reason for it is pretty simple – IoT is so rapidly changing technology, that new devices, ideas and implementations appear every single day. Even so, once you read this book, you will be able to quickly move over IoT environment and market, chasing ideas with ease and implementing your own IoT infrastructure.

We also believe this book will help adults that took their technical education some time ago to update their knowledge.

We hope this book will let you find new brilliant ideas both in your professional life as well as see a new hobby or even startup innovative business.

Note: the sky is no longer the limit, so keep exploring with IoT!

2.1. Project information

This Intellectual Output was implemented under the Erasmus+ KA2: Strategic Partnerships in the Field of Education, Training, and Youth – Higher Education.
Project IOT-OPEN.EU – Innovative Open Education on IoT: Improving Higher Education for European Digital Global Competitiveness.
Project number: 2016-1-PL01-KA203-026471.

Erasmus+ Disclaimer

This project has been funded with support from the European Commission.
This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Copyright Notice

This content was created by the IOT-OPEN.EU consortium: 2016–2019.
The content is Copyrighted and distributed under CC BY-NC Creative Commons Licence, free for Non-Commercial use.



In case of commercial use, please contact IOT-OPEN.EU consortium representative.

3. Introduction

3. Introduction

Here comes the Internet of Things. The name that recently makes red-hot people in business, researchers, developers, geeks and ... students. The name that non-technology related people consider as kind of magic and even danger for their privacy. The name that the EU set as one of the emerging technologies and analysts estimate the worldwide market is expected to hit well over 500 billion US dollars in 2022.

What is IoT (Internet of Things) then? Surprisingly, the answer is not straightforward.

- Definition of IoT
- Enabling Technologies
- Mobility – New Paradigm for IoT Systems
- Hints for Further Readings on Development Boards, Kits and Sites
- Embedded Systems Communication Protocols
- Data Management Aspects in IoT
- Application Domains and Their Specifics

3.1. Definition of IoT

Let us roll back to 1970s first. In 1973 the first RFID device was patented. This device, even if does not look nor reminds modern IoT devices, was the key enabling technology. The low power (actually here passive) solution with remote antenna large enough to collect energy from the electromagnetic field and power the device brought an idea of uniquely identifiable items. That somehow mimics well known EAN barcodes and their evolution we use nowadays like QR codes, but here every single thing has a different identity, while EAN barcodes present class of products, not an individual one. The possibility to identify a unique identity remotely became a fundamental of the IoT as we know today. Please note RFID is not the only technology standing behind IoT. In 1990s rapid expansion of wireless networks, including broadband solutions like cellular-based transfers with its consequent generations brought the possibility to connect devices located in various, even distant geographical locations. Paralelly we experienced an exponential increase in the number of devices connected to the global, Internet network, including the Smartphone revolution that started around mid of the first decade of the XXI century. On the hardware level, microchips and processors became physically smaller and more energy efficient yet offering growing computing capabilities and memory size increase, along with significant price drops. All those facts drove the appearance of small, network-oriented, cheap and energy efficient electronic devices.

3.1.1. What is IoT?

Phrase "Internet of Things" has been used for the first time in 1999 by Kevin Ashton – an expert on digital innovation. Formally IoT was introduced by the International Telecommunication Union (ITU) in the ITU Internet report in 2005¹⁾. The understanding and definitions of IoT changed during the years, but now all agree that this cannot be seen as the technology issue only. According to IEEE "Special report: Internet of Things"²⁾ released in 2014, IoT is:

IEEE Definition of IoT

A network of items – each embedded with sensors – which are connected to the Internet.

It relates to the physical aspects of IoT only. Internet of Things also address other aspects that cover many areas ³⁾:

- enabling technologies,
- software,
- applications and services,
- business models,
- social impact.

We also cannot forget the management of elements of the IoT system and security and privacy aspects. IEEE, as one of the most prominent standardisation organisations, also work on standards related to the IoT. The primary document is IEEE P2413™ ⁴⁾. It covers the technological architecture of IoT as three-layered: sensing at the bottom, networking and data communication in the middle, and applications on the top. It is essential to understand that the IoT systems are not only the small, local range systems. ITU-T has defined IoT as:

ITU-T Definition of IoT

A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

In the book⁵⁾ by European Commission we can read similar description of what IoT is: "The IoT is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment." IoT has an impact on many areas of human activity: manufacturing, transportation, logistics, healthcare, home automation, media, energy saving, environment protection and many more. In this course, we will consider the technical aspects mainly.

3.1.2. "Thing"

In IoT world, the "thing" is always equipped with some electronic element that can be as simple as the RFID tag, active sensor sending data to the global network, or autonomous device that can react on environmental changes. In CERP-IoT book "Visions and Challenges"⁶⁾ in the context of "Internet of Things" a "thing" could be defined as:

CERP-IoT Definition of "Thing"

A real/physical or digital/virtual entity that exists and moves in space and time and is capable of being identified. Things are commonly identified either by assigned identification numbers, names and location addresses.

We can also find other terms used in the literature like "smart object", "device" or

3. Introduction

"nodes" ⁷⁾.

Passive Thing

We can imagine that almost everything in our surroundings is tagged with an RFID element. They do not need a power supply; they respond with the short message, usually containing the identification number. Modern RFID's can achieve the 6 to 7 meters of the range. Using the active RFID reader, we can easily locate lost keys, know if we still have the butter in the fridge, in which wardrobe there is our favourite t-shirt.

Active Thing

If the "thing" includes the sensor, it can send interesting data about current conditions. We can sense the environmental parameters like temperature, humidity, air pollution, pressure, localisation data, water level, light, noise, movement. This data, using different methods and protocols, can be sent to the central collector that connects to the Internet and further to the database or cloud. There the data can be processed, and using Artificial Intelligence algorithms can be used to decide actions that could be taken in different situations. Active things can also receive control signals from the main controller to control the environment: turn on/off the heating or light, water flowers, turn on the washing machine when there is sunlight enough to generate the required amount of electricity.

Autonomous Thing

This thing does not even require the controller to realise the proper decision. An autonomous vacuum cleaner can clean our house while it detects that we aren't present at home, and the floor needs to be cleaned. The fridge can order our favourite beverage once it detects that the last bottle is almost empty.

3.2. Enabling Technologies

In this chapter, we describe modern technologies that appeared in the last few years, enabling the idea of IoT to be widely implementable. In the ⁸⁾ we can read that "The confluence of efficient wireless protocols, improved sensors, cheaper processors and a wave of startups and established companies made the concept of the IoT mainstream". Similar analyze been done in ⁹⁾ where authors write that "the IoT is enabled by the latest developments in RFID, smart sensors, communication technologies and Internet protocols". To operate RFID and smart sensors need the microprocessor system to do the reading, convert the data into digital format, and send it to the Internet using the communication protocol. This process can be done by small- and medium-scale computer (embedded) systems. These are essential elements of technologies used in IoT systems.

3.2.1. Small-Scale Computer Systems

Last years we can observe rapid growth in the field of microprocessors. It includes not only the powerful desktop processors but also microcontrollers – elements that are used in small-scale embedded systems. We can also notice the popularity of microprocessor systems that can be easily integrated with other elements like sensors, actuators, connected to the network. Essential is also the availability of programming tools and environments supported by different companies and communities. An excellent example

of such systems is *Arduino*.

3.2.2. Medium-Scale Computer Systems

The same growth can be observed in medium-scale computers. They have more powerful processors, more memory and networking connectivity built in than small-scale computer systems. They can work under control of multitasking operating systems like *Linux*, *Windows*, embedded or real-time operating systems like *FreeRTOS*. Having many libraries, they can successfully work as hubs for local storage, local controllers and gateways to the Internet. The example of such systems we consider in our course is *Raspberry PI*.

3.2.3. Access to the Internet

Nowadays the Internet is (almost) everywhere. There are lots of wireless networks available in private and public places. The price of cellular access (3G/4G/5G) is low, offering a good performance of data transfer. Connecting the "thing" to the Internet has never been so easy.

3.2.4. IP Addressing Evolution

The main paradigm of IoT is that every unit can be individually addressed. With the addressing scheme used in IPv4, it wouldn't be possible. IPv4 address space delivers "only" 4 294 967 296 of unique addresses (2^{32}). If you think it's a big number, imagine that every person in the world has one IP-connected device – IPv4 covers about half of the human population. The answer is IPv6 with a 128-bit addressing scheme that gives 3.4×10^{38} addresses. It will be enough even if every person will have a billion devices connected to the Internet.

3.2.5. Data Storage and Processing

IoT devices generate the data to be stored and processed somewhere. If there is a couple of sensors, the amount of data is not very big, but if there are thousands of sensors generating data hundreds of times every second. It can be handled by the cloud – the huge place for the data with tools and applications ready to help with data processing. There are some big, global cloud available for rent offering not only the storage but also Business Intelligence tools, Artificial Intelligence analytic algorithms. There are also smaller private clouds created to cover the needs of one company only. Many universities have their own High-Performance Computing Centre.

3.2.6. Mobile Devices

Many people want to be connected to the global network everywhere, anytime having their "digital twin" with them. It is possible now with small, powerful mobile devices like smartphones. Smartphones are also elements of IoT world being together sensors, user interfaces, data collectors, wireless gateways to the Internet, and everything with mobility feature.

The technologies we mentioned here are the most recognisable, but there are many others, smaller, described only in the technical language in some standard description document, hidden under the colourful displays, between large data centres, making our

3. Introduction

IoT world operable. In this book, we will describe some of them.

3.3. Mobility – New Paradigm for IoT Systems

As defined IoT previously in its essence is a network of physical things or devices that might include not only sensors or simple data processing units but also complex actuators and significant hybrid computing power as well. Today IoT systems have transited from being perceived as sensor networks to smart-networked systems capable of solving complex tasks in mass production, public safety, logistics, medicine and other domains, which require a broader understanding and acceptance of current technological advancements.

Since the very beginning of sensor networks on of the main challenges have been data transport and data processing, where significant efforts have been put by the ICT community towards service based system architectures. The current trend, however, already provides considerable computing power available even in small mobile devices, and therefore, the concepts of future IoT already are shifted towards smarter and more accessible IoT devices.

3.3.1. Cloud Computing

Cloud-based computing is rather well known and adequately employed paradigm, where IoT devices can interact with remotely shared resources such as data storages, data processing, data-mining and other services that are not available to the system due it cost-effectiveness or other limitations. Although the cloud computing paradigm can handle vast amounts of data from IoT clusters, the transfer of extensive data to and from cloud computers presents a challenge due to limited bandwidth.¹⁰⁾ Consequently, there is a need to process data near data source, employing the increasing number of smart devices with enormous processing power and a rising number of service providers available for IoT systems as well.

3.3.2. Fog Computing

Fog-computing addressed the bottlenecks of cloud computing regarding data transport while providing the needed services to IoT systems. It is a new trend in computing that aims to process the data near the data source. Fog computing pushes applications, services, data, computing power, and decision making away from the centralised nodes to the logical extremes of a network. Fog computing significantly decreases the data volume that must be moved between end devices and cloud. Fog computing enables data analytics and knowledge generation to occur at the data source. Furthermore, the dense geographic distribution of fog helps to attain better localised accuracy for many applications as compared to the cloud¹¹⁾.

3.3.3. Cognitive IoT Systems

According to¹²⁾ Cognitive IoT besides a proper combination of hardware, sensors and data transport, comprises cognitive computing, which consists of the following main components:

- understanding – in case of IoT it means systems capability to process a significant amount of structured and unstructured data, extracting the meaning of the data –

3.4. Hints for Further Readings on Development Boards, Kits and Sites

- produce a model that binds data to reality;
- reasoning – involves decision making according to the understood model and acquired data;
- learning – a creation of new knowledge from the existing, sensed data and elaborated models.

Usually, cognitive IoT systems or C-IoT are expected to add more resilience to the overall solution. The resilience is a complex term and is differently explained under different contexts; however, there are common features for all resilient systems. C-IoT, as a part of their resilience, should be capable of self-failure detection and self-healing that minimises or gradually degrades the system's overall performance. In this respect, the non-resilient system fails or degrades in a step-wise manner. In case of security issues that system should be able to change its security keys, encryption algorithms and take other measures to cope with the detected threats. Abilities of self-optimisation often are considered as a part of C-IoT feature list to provide more robust solutions.

All three approaches from cloud to cognitive systems are focusing on adding value to IoT devices, system users and related systems on-demand. Since market and technology acceptance of mobile devices is still growing, and amount of produced data of those devices is growing exponentially, mobility as a phenomenon is one of the main driving forces of the technological advancements of the near future.

3.4. Hints for Further Readings on Development Boards, Kits and Sites

Some additional information about the *Arduino* boards, programming environment, the programming language can be obtained on the *Arduino* website¹³⁾. It also includes an Internet store, where the different type of Arduino boards or components can be bought and even forum and blog for the community where to look for the solution to various problems. Many *Arduino* projects, developed by *Arduino*, enthusiasts can be found in the *Arduino Project Hub*¹⁴⁾.

More information about the *Raspberry Pi* controllers can be found on the official website of it¹⁵⁾. It includes the blog, community, forums, education section, etc. There is also possible to download the Raspbian operating system.

There are many online platforms that provide online courses by different universities about relevant topics like Internet of Things, embedded systems, programming languages, connectivity and security, robotics, big data, computer vision and many more. Some of the most popular platforms are *Coursera*¹⁶⁾, *edX*¹⁷⁾, *Udacity*¹⁸⁾, *Udemy*¹⁹⁾, *Skillshare*²⁰⁾. Most of these courses are free of charge and at the end of these courses, a certificate about skills can be acquired (often for additional price).

The *Electronics Tutorials* website²¹⁾ offers multiple basic electronics tutorials topics including AC and DC circuit theory, amplifiers, semiconductors, filters, Boolean algebra, capacitors, power electronics, transistors, operational amplifiers, sequential logic, and many more. It contains an extensive description of theory with graphics and explanations.

*Instructables*²²⁾ is a project platform that includes plenty of Internet of Things projects

3. Introduction

for different knowledge levels. It is also possible to enrol to different classes with many lessons that teach about specific related topics that are not limited only to electronics but also cover issues such as sewing, food, craft, 3D printing, etc. One section of the Instructables website offers multiple contests and challenges about the related topic with valuable prizes.

Tinkercad is a simple, online 3D design and 3D platform that also allows to model and test circuits²³⁾. With *Tinkercad*, it is possible to program and simulate virtual Arduino board online, to use different libraries and serial monitor. There are also plenty of already existing starter examples.

3.5. Data Management Aspects in IoT

Data management is a critical task in IoT. Due to the high number of devices, things, already available (tens of billions), and considering the data traffic generated by each of them through sensor networks, infotainment (soft news) or surveillance systems, mobile social network clients, and so on, we are now in the ZettaByte ($ZB 2^{70}, 10^{21}$ bytes) era. This opened up several new challenges on (IoT) data management, giving rise to data sciences and BigData technologies. Such challenges have not to be considered as main issues to solve, but also as big opportunities fuelling digital economy with new directions such as Cloudonomics²⁴⁾ and IoTconomics, where data can be considered as a utility, a commodity to properly manage, curate, store, and trade. Therefore, to properly manage data in IoT contexts is not only critical but also of strategic importance for business players as well as for users, evolving into **prosumers** (producers-consumers).

From a technological perspective, the main aspects of dealing with IoT data management are:

- **data source:** data generation and production is a relevant part of IoT, involving sensors probing the physical system. In a cyber-physical-social system view, such sensors could also be virtual (e.g. software), or even human (e.g. citizens, crowdsensing). Main issues to deal with in data production are related to the type and format of data, heterogeneity in measurements and similar issues. Semantics is the key to solve these issue, also through specific standards such as Sensor Web Enablement and Semantic Sensor Network²⁵⁾;
- **data collection/gathering:** once data are generated, these should be gathered and made available for processing. The collection process needs to ensure that the data gathered are both defined and accurate so that subsequent decisions based on the findings are valid. Some types of data collection include census (data collection about everything in a group or statistical population), sample survey (collection method that includes only part of the total population), and administrative by-product (data collection is a byproduct of an organisation's day-to-day operations). Usually, wireless communication technologies such as Zigbee, BlueTooth, LoRa, Wi-Fi and 3-4G networks are used by IoT smart objects and things to deliver data to collection points;
- **filtering:** is a specific preprocessing activity, usually performed at data source or data collector (IoT) nodes (e.g. motes, base stations, hotspots, gateways), aiming at cleaning noisy data, filtering noise and not useful information;
- **aggregation/fusion:** in order to reduce bandwidth before sending data to processing nodes, these are further elaborated, compressed, aggregated and fused

3.6. Application Domains and Their Specifics

(sensor/data fusion) to reduce the overall volume of raw data to be transmitted and stored;

- **processing:** once data are properly collected, filtered, aggregated, and fused, they can be processed. Processing can be both local and remote, and usually, also include preprocessing activities aiming at preparing data for real processing. Local processing, when possible, is mainly tasked at a fast, lightweight computation on edges (Edge computing), quickly providing results and local analytics. More complex computation are usually demanded to remote (physical or virtual) servers, either provided by local nodes (e.g. communication servers, cloudlets) in a Fog computing fashion, or by Cloud providers as virtual machines hosted in data centres. This kind of computation can also involve historical data, providing global analytics, but hardly meets time-constrained applications and real-time requirements;
- **storage/archive:** remote servers are also used for permanently store and archive data, making these available for further processing, even to third parties. The database is often used for that, mainly based on distributed, NoSQL key-store technologies to improve reliability and performance;
- **delivering/presentation/visualization:** the results of processing activities have to be then delivered to requestors and users. These have to be therefore properly organised and formatted, ready for end-users. IoT data visualisation is becoming an integral part of the IoT. Data visualisation provides a way to display this avalanche of collected data in meaningful ways that clearly present insights are hidden within this mass amount of information;
- **security and privacy:** data privacy and security are among the most critical issues to address in IoT data management. Good results and reliable techniques for secure data transmission, such as TLS and similar, are available. This way, IoT data security issues mainly concern²⁶⁾ **securing IoT devices**, since they are usually resource constrained and therefore do not allow to adopt traditional cryptography scheme to data encryption/decryption. **Data privacy and integrity** should also be enforced in remote storage servers, anonymising data as well as allowing owners to properly manage (monitoring, removing) them while ensuring availability. Indeed, security and privacy issues vertically span into the whole IoT stack. A promising technique to address IoT security issues, attracting growing interests from both academic and business communities, is blockchain²⁷⁾.

3.6. Application Domains and Their Specifics

Application domains of the Internet of Things solutions are wide. Most prominent applications include (among others)²⁸⁾:

- building and home automation,
- smart water,
- smart metering,
- smart city (including logistics, retail, transportation),
- smart animal farming,
- industrial IoT,
- precision agriculture and smart farming,
- security and emergencies,

3. Introduction

- healthcare and wellness (including wearables),
- smart environment,
- energy management,
- robotics,
- smart grids.

Smart homes are one of the first examples that come to mind when talking about the domain applications of the Internet of Things. Smart home benefits include reduced energy wastage, the quality and reliability of devices, system security, reduced cost of basic needs, etc. Some home automation examples are environmental control (that monitors and controls heating, ventilation, air conditioning and sunscreens), electrical charging of vehicles, solar panels for electrical power and hot water, ambient lighting control, smart lighting for aquaria, home cooking, garage doors, smart plant watering systems indoors and outdoors, baby monitors, timed pet food dispensers, monitoring perishable goods (for example, in the refrigerator), remote monitoring (for instance, of washer cycle status), tracking and proactive maintenance scheduling, event-triggered task execution. Home security also plays a significant role in smart homes. Example applications are automatic door locks, sensors for opening doors and windows, pressure, motion and infrared sensors, security cameras, notifications about the security (to the owner or the police) and fitness related applications.

In **smart city**, multiple IoT-based services are applied to different areas of urban settings. The aim of the smart city is the best use of public resources, improvement of the quality of resources provided to people and reduction of operating costs of public administration²⁹⁾. Smart city can include many solutions like smart buildings, smart grids for improving energy management, smart tourism, monitoring of state of the roads and occupation of parking lots, public safety, environment monitoring, automatic street lighting, signalling with smart power devices, control of water levels for hydropower or flood warnings, electricity generating devices like solar panels and wind turbines, weather monitoring stations. **Transportation** in smart cities may include aviation, monitoring and forecasting of traffic slowdowns, timetables and current status, navigation and route planning, as well as vehicle diagnostics and maintenance reports, remote maintenance services, traffic accident information collection, fleet management using digital tachographs, smart parking, car/bicycle sharing services³⁰⁾. IoT in transportation makes cars connected.

Smart grid is a digital power distribution system. In this system, information is gathered using smart meters, sensors and other devices. After these data are processed, power distribution can be adapted accordingly. Smart grids are used to deliver sustainable, economical and secure electricity supplies efficiently.

In **precision agriculture** and **smart farming** IoT solutions can be used to monitor the moisture of the soil, conditions of the plants, control microclimate conditions and monitor the weather conditions in order to improve farming³¹⁾. The goal of using IoT in agriculture is the maximization of the harvest, reducing operational costs, being more efficient in general and reducing environmental pollution using low-cost automated solutions. An interaction between the farmer and the systems can be done using a human-machine interface. In the future smart precision farming can be a solution for such challenges like increasing worldwide demand for food, a changing climate, and a limited supply of water and fossil fuels³²⁾.

3.6. Application Domains and Their Specifics

Similar to precision agriculture that is part of IoT in industry, **smart factories** also tend to improve the manufacturing process by monitoring of pollutant gases, locating employees and with many other solutions.

Industrial IoT and **smart factories** are part of the Industry 4.0 revolution. In this model, modern factories are able to automate complex manufacturing tasks, thanks to the Machine-To-Machine communication model and thanks to it, provide more flexibility in the manufacturing process to enable personalised, short volume products manufacturing with ease.

In the **healthcare and wellness**, IoT applications can be used for monitoring and diagnosing of patients, managing of people and medical resources. It allows to remotely and continuously monitor the vital signs of patients to improve medical care and wellness of patients³³⁾. Important part of the smart welfare are wearables that include wristbands and smartwatches that monitor the level of activity, heart rate and other parameters. Smart healthcare includes remote monitoring, care of patients, self-monitoring, smart pills, smart home care, *Real-Time Health Systems* (RTHS) and many more. Medical robotics can also be part of the healthcare IoT system that includes medical robots in precision surgery or distance surgery; some robots are used in rehabilitation and hospitals (for example, *Panasonic HOSPI*³⁴⁾) for delivering medication, drinks, etc. to patients.

Wearables used in IoT applications should be highly energy efficient, ultra-low power and small sized. Wearables are installed with sensors and software for data and information collected about the user. Devices used in daily life like *Fitbit*³⁵⁾ are used to track people health and exercise progress in previously impossible ways, and smartwatches allow to access smartphones using this device on the wrist. But wearables are not limited only by wearing them on the wrist. They can also be glasses that equipped with the camera, a sports bundle attached to the shoes or camera attached to the helmet or as a necklace³⁶⁾.

4. IoT Hardware overview

IoT hardware infrastructure is mostly inheriting from the embedded systems of the SoC type. As IoT devices are by its nature network-enabled, many of the existing embedded platforms evolved towards network-enabled solutions, sometimes indirectly through delivering network processor (wired or wireless) as a peripheral device yet integrated on the development board (i.e. Arduino Uno with Ethernet Networking shield, GSM shield, etc.), sometimes a new system, integrating networking capabilities in one SoC (i.e. Espriff SoCs). More advanced devices that require OS to operate preliminarily benefited from externally connected peripheral network interfaces via common wired ports like USB (i.e. early versions of the Raspberry Pi, where WiFi card was delivered as USB stick), currently, usually integrate most of the network interfaces in a single board (i.e. RPi 3B, including Ethernet, WiFi and Bluetooth).

4.1. Most noticeable platforms

IoT market is an emerging one. New hardware solutions appear almost daily, while others disappear quick. At the moment of writing this publication (2016-2019), there are some core hardware solutions that seem to be prominent for at least a couple of years, however. We've provided a short review of those platforms in the following sections:

- AVR: Arduino - a development board that uses Atmel SoC, that is no doubt the most popular development platform for enthusiasts and professionals. Arduino itself barely offers networking capabilities yet; there is a vast number of extension boards including network interfaces (both wired and wireless).
- ESP: Espriff (Espressif Systems) - the great SoC solution of the single SoC including wireless network interfaces.
- ARM: Raspberry Pi (and its clones) - advanced boards, including Linux operating system with GUI interface, even able to replace desktop computers.

[Arduino Overview](#)

[Espressif SoC Overview](#)

[Raspberry Pi Overview](#)

4.2. Embedded Systems Communication Protocols

Understanding the principals of the communication are essential for further reading on hardware and programming. Most microcontrollers (including SoCs) can communicate in the protocols juxtaposed below right "out of the box". Interfaces can be implemented in hardware or (recently) in software. Some microcontrollers may require an external, dedicated protocol converter (a chip or a module).

IoT systems are typically structured into three basic layers ³⁷⁾. The lowest layer is the Perception (physical, acquisition) Layer, the intermediate is the Network Layer, and the higher is the Application Layer. The function of the perception layer is to keep contact with the physical environment. Devices working in this layer are designed as embedded systems. They include the microprocessor or microcontroller, memory, communication unit, and interfaces - sensors or actuators. Sensors are elements that convert a value of some physical parameter into an electrical signal, while actuators are elements

4.2. Embedded Systems Communication Protocols

that control environment parameters. Sensors and actuators are interfaced with the microcontroller using different connection types. This chapter describes some internal protocols used to communicate between microcontrollers and other electronic elements that can be named "embedded protocols". Description of the protocols used for wire and wireless transmission between the perception layer and higher layers is present in Introduction to the IoT Communication and Networking. The embedded protocol that can be used in specific implementation depends mainly on the type of the peripheral element. Some of them use an analogue signal that the microcontroller must convert to digital internally, some directly implement digital communication protocol.

4.2.1. Analog

Simple sensors do not implement the conversion and communication logic, and the output is just the analogue signal – voltage level depending on the value of the parameter that is measured. It needs to be further converted into a digital representation; this process can be made by analogue to digital converters (ADC) implemented as the internal part of a microcontroller or separate integrated circuit. Examples of the sensors with analogue output are a photoresistor, thermistor, potentiometer, resistive touchscreen.

4.2.2. Digital

Dummy, true/false information can be processed via digital I/O. Most devices use positive logic, where, i.e. +5V (TTL) or +3.3V (those are the most popular, yet there do exist other voltage standards) presents a logical one, while 0V presents logical zero. In real systems this bounding is fuzzy and brings some tolerance, simplifying, i.e. communication from 3.3V output to 5V input, without a need of the conversion (note, reverse conversion is usually not so straightforward, as 3.3V inputs driven by 5V output may burn easily). A sample of the sensor providing binary data is a button (On/Off).

4.2.3. SPI

One of the most popular interfaces to connect the sensor is SPI (Serial Peripheral Interface). It is a synchronous serial interface and protocol that can transmit data with speed up to 20Mbps. SPI is used to communicate microcontrollers with one or more peripheral devices over short distances – usually internally in the device. In SPI connection there is always one master device, in most cases the microcontroller (uC) that controls the transmission, and one or more slave devices - peripherals. To communicate SPI uses three lines common to all of the connected devices, and one enabling line for every slave element.

Table 2: SPI lines

| Line | Description | Direction |
|------|---------------------|-----------------|
| MISO | Master In Slave Out | peripheral → uC |
| MOSI | Master Out Slave In | uC → peripheral |
| SCK | Serial Clock | uC → peripheral |
| SS | Slave Select | uC → peripheral |

4. IoT Hardware overview

MISO is intended to send bits from slave to master, MOSI transmits data from master to slave. SCK line is used for sending clock pulses which synchronize data transmission. The clock signal is always generated by the master device. Every SPI compatible device has the SS (Slave Select) input that enables communication in this specific device. Master is responsible to generate this enable signal – separately for every slave in the system.

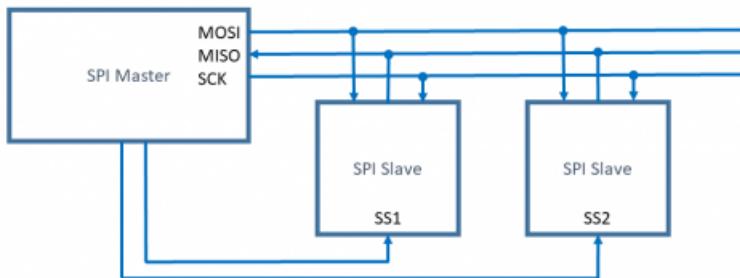


Figure 2: Sample SPI connection

SPI is used in many electronic elements like analogue to digital converters (ADC), real-time clocks (RTC), EEPROMs, LCD displays, communication interfaces (e.g. Ethernet, WiFi) and many others. Due to different hardware implementations, there are four modes of operation of the SPI protocol. The mode used in master must fit the mode that is implemented in the slave device.

Table 3: SPI modes

| Mode | Clock polarity | Clock phase | Idle state | Active state | Output edge | Data capture |
|--------|----------------|-------------|------------|--------------|-------------|--------------|
| mode 0 | 0 | 0 | 0 | 1 | falling | rising |
| mode 1 | 0 | 1 | 0 | 1 | rising | falling |
| mode 2 | 1 | 0 | 1 | 0 | rising | falling |
| mode 3 | 1 | 1 | 1 | 0 | falling | rising |

It results in different timings of the clock signal concerning the data sent. Clock polarity = 0 means that the idle state of the SCK is 0 so every data bit is synchronised with the pulse of logic 1. Clock polarity = 1 reverses these states. Output edge (rising/falling) says at which edge of active SCK signal sender puts a bit on the data line. Data capture edge says at what edge of SCK signal data should be captured by the receiver.

4.2.4. TWI (I2C)

TWI (Two Wire Interface) is one of the most popular communication protocol used in embedded systems. It has been designed by Philips as I2C (Inter-Integrated Circuit) for using in the audio-video appliances controlled by the microprocessor. There are many chips that can be connected to the processor with this interface including:

- EEPROM memory chips

4.2. Embedded Systems Communication Protocols

- RAM memory chips
- AD/DA converters
- Real-time clocks
- Sensors (temperature, pressure, gas, air pollution)
- Port extenders
- Displays
- Specialized AV circuits

TWI, as the name says, uses two wires for communication. One is the data line (SDA) and the second is the clock line (SCL). Both lines are common to all circuits connected to the one TWI bus. The method of the communication of TWI is the master-slave synchronous serial transmission. It means that data is sent bit after bit synchronised with the clock signal. SCL line is always controlled by the master unit (usually the processor), the signal on the SDA line is generated by the master or one of the slaves – depending on the direction of communication. The frequency rate of the communication is up to 100kHz for most of the chips, for some can be higher – up to 400kHz. New implementation allows even higher frequency rate is reaching 5MHz. At the output side of units, the lines have the open-collector or open-drain circuit. It means that there are external pull-up resistors needed to ensure proper operation of the TWI bus. Value of these resistors depends on the number of connected elements, speed of transmission and the power supply voltage and can be calculated with the formulas presented in Texas Instrument Application Report³⁸⁾. Usually, it is assumed between 1 kOhm and 4,7 kOhm.

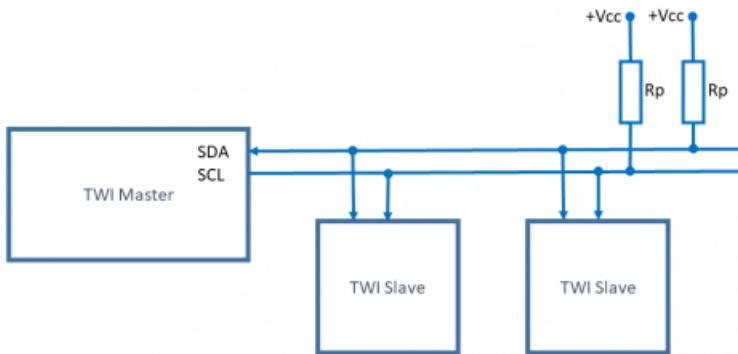


Figure 3: Sample TWI connection

The data is sent using frames of bytes. Every frame begins with the sequence of signals that is called the start condition. This sequence is detected by slaves and causes them to collect the next eight bits that form the address byte – unique for every circuit on the bus. If one of the slaves recognises its address remains active until the end of the communication frame, others become inactive. To inform master that some unit has been appropriately addressed slave responses with the acknowledge bit – it generates one bit of low level on the SDA line (the master generates clock pulse). After sending proper address data bytes are sent. The direction of the data bytes is controlled by the last bit of the address, for 0 data is transmitted by the master (Write), for 1 data is sent

4. IoT Hardware overview

by the slave (Read). The receiving unit must acknowledge every full byte (eight bits). There is no limitation on the number of data bytes in the frame, for example, samples from the AD converter can be read byte continuously after byte. At the end of the frame another special sequence is sent by the master – stop condition. It is also possible to generate another start condition without the stop condition. It is called repeated start condition.

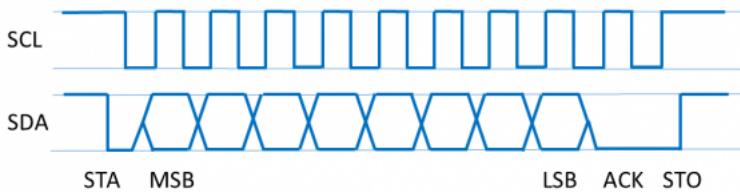


Figure 4: TWI frame

Address byte activates one chip on the bus only, so every unit must have a unique physical address. This byte usually consists of three elements: 4-bit field fixed by the producer, 3-bit field that can be set by connecting three pins of the chip to 0 (ground) or 1 (positive supply line), 1-bit field for setting the direction of communication (R/#W). Some elements (e.g. EEPROM memory chips) uses the 3-bit field for internal addressing so there can be only one such circuit connected to one bus. There are no special rules for the data bytes. First data byte sent by the master can be used for configuration of the slave chip. In memory units, it is used for setting the internal address of the memory for writing or reading. In multi-channel AD converters to choose the analogue input. The detailed information of the meaning of every bit of the transmission is present in the documentation of the specific integrated circuit. The I²C standard also defines the multi-master mode, but in most of the small projects, there is one master device only.

4.2.5. 1-Wire

1-Wire is a master-slave communication bus system designed formerly by Dallas Semiconductor Corp³⁹⁾ ensuring low data transmission speed, signalling and can be powered directly by data line signals. The 1-Wire concept is similar to I²C transmission standard, but can transmit data in longer distances than I²C but with lower speed. The implementation area is very wide and typically 1-Wire protocol is used to share data between small, inexpensive devices such as a digital thermometer, humidity or pressure sensors or actuator systems. A network chain of 1-Wire devices consists of one master device and many slave devices. Such a chain is called a MicroLAN. 1-Wire devices may be a part of the circuit board within a product, could be a single component device such as temperature probe, or may be attached to a remote device for monitoring purposes. Typical data acquisition and laboratory networks use CAT-5 cables to connect 1-Wire devices together, can be mounted in a socket of small PCB boards, attached to the device which must be monitored. In such implementations, the RJ11 connectors (telephones 6P2C/6P4C modular plugs) are very popular. Each 1-Wire device must contain logic unit to operate on the bus. The 1-Wire products include temperature, voltage, current sensors, loggers, timers, battery monitors, memory and many more. To connect them

4.2. Embedded Systems Communication Protocols

to a PC the special bus converter is needed. The most popular PC/1-Wire converters use USB, RS-232 serial, and parallel port interfaces allowing connect the MicroLAN to the host PC. 1-Wire devices can also be connected directly to the microcontroller boards.

1-Wire protocol description

Within the MicroLAN, there is always one master device, which may be a PC or a microcontroller unit. The master always initiates activity on the bus to avoid collisions on the network chain. If a collision occurs, the master device retries the communication. In the 1-Wire network, many devices can share the same bus line. To identify devices in the MicroLAN, each connected device has a unique 64-bit ID number. The least significant byte of the ID number defines the type of the device (temperature, voltage etc. sensors). The most significant byte represents a standard 8-bit CRC. The 1-Wire protocol description contains several broadcast commands and commands used to address the selected device. The master sends a selection command, then the address of a slave selected device. This way the next command is executed only by the addressed device. The 1-wire bus implements enumeration procedure which allows the master to get information about ID numbers of all connected slave devices to the MicrolAN network. Device address includes the device type, and a CRC allows to identify what type of slaves are currently connected to the network chain for inventory purposes. The 64-bit address space is searched as a binary tree. It allows to find up to 75 devices/second.

The physical implementation of the 1-Wire network is based on an open drain master device connected to one or more open drain slaves. One single pull-up resistor for all devices pull the bus up to 3/5 volts and can be used to power the slave devices. 1-Wire communication starts when a master or slave sets the bus to low voltage (connects the pull-up resistor to ground through its output MOSFET). Typical data speed of the 1-Wire interface is about 16.3 kbit/s.

1-Wire protocol allows for bursting the communication speed up by 10 factor. In this case, the master starts a transmission with a reset pulse pulling down the data line to 0 volts for at least 480 μ s. It resets all slave devices in the network chain bus. Then, any slave device shows that it exists generating the "presence" pulse. It holds the data line low for at least 60 μ s after the master releases the bus. To send a "1", the bus master sends a 1–15 μ s low pulse. To send a "0", the master sends a 60 μ s low pulse. The negative edge of the pulse is used to start a slave's monostable multivibrator. The slave's multivibrator clocks to read the data bus about 30 μ s after the falling edge. The slave's multivibrator has analogue tolerances that affect its timing accuracy, for the "0" pulses are 60 μ s long, and "1" pulses are limited to max. 15 μ s. When the designed solution doesn't contain a dedicated 1-Wire interface peripheral, a UART can be used as a 1-wire master. Dallas also offers the Serial or USB "bridge" chips, very useful when the distance between devices is long (greater than 100 m). For longer, up to 300 meter buses, the simple twisted pair telephone cable can be used. It will require adjustment of pull-up resistances from 5 to 1 k Ω . The basic sequence is a reset pulse followed by an 8-bit command, and after it, data can be sent/received in groups of 8-bits. In case of transmission errors, the weak data protection 8-bit CRC checking procedure can be used.

To find the devices, the enumeration broadcast command must be sent by a master. The slave device response with all ID bits to the master and at the end it returns a 0.

4. IoT Hardware overview

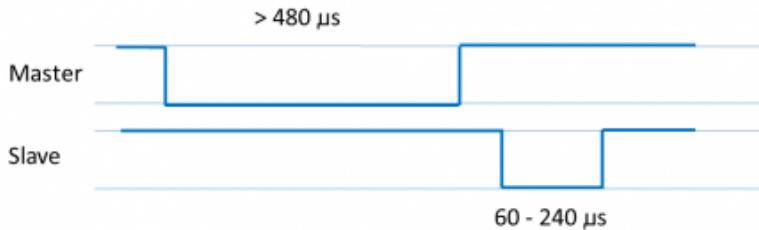


Figure 5: 1-Wire reset timings

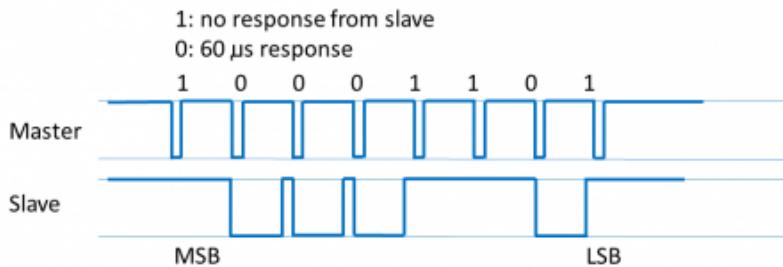


Figure 6: 1-Wire read timings

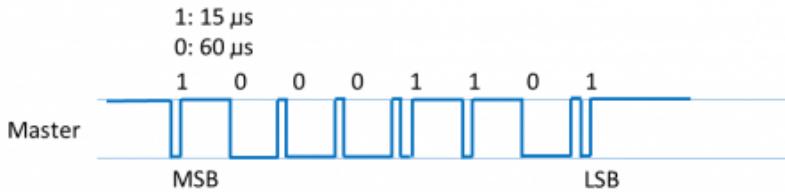


Figure 7: 1-Wire write timings

USB to 1-Wire Master

The DS9490B is a USB bridge and holder for a single F5-size iButton. The DS9490R is a USB bridge with 1-Wire RJ11 interface to accommodate 1-Wire receptacles and networks.

4.2. Embedded Systems Communication Protocols

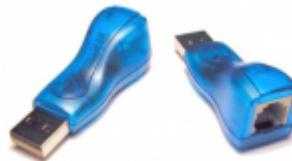


Figure 8: DS9490R USB Bridge

The bridge is based on the DS2490 chip developed by Dallas company, which allows to interconnect USB interface with 1-Wire bus. This required programming and electrical conversion between two different protocols in bidirectional way. The electrical wiring are present on Figure 3.

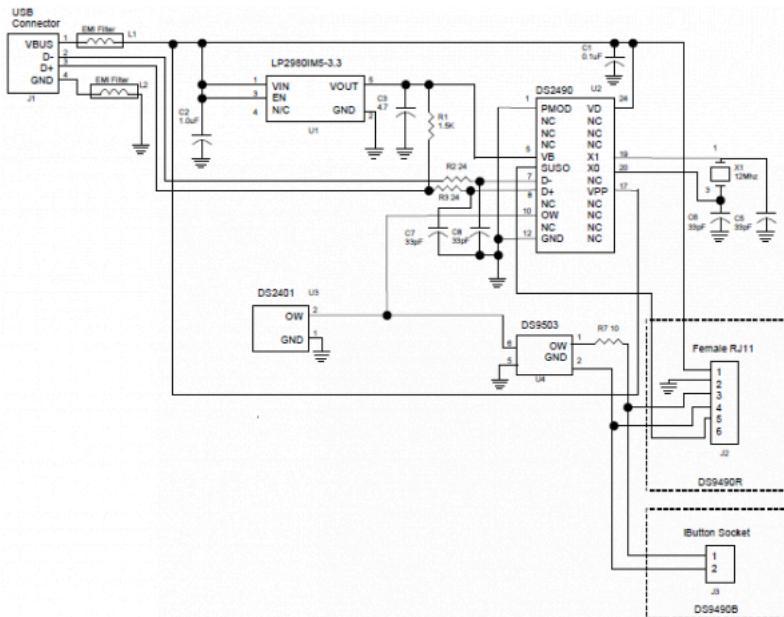


Figure 9: DS9490R USB schematic

The appropriate 1-Wire cable pinout uses RJ11 telephone connectors.

| PIN | SIGNAL NAME | DESCRIPTION |
|-----|-----------------|--------------------|
| 1 | V _{DD} | 5VDC Output |
| 2 | GND | Power Ground |
| 3 | OW | 1-Wire Data |
| 4 | GND_OW | 1-Wire Return |
| 5 | SUSO | USB Suspend Output |
| 6 | N.C. | No Connection |

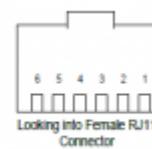


Figure 10: DS9490 1-Wire RJ11 SOCKET pinout

4. IoT Hardware overview

1-Wire Products

The list of Dallas/Maxim integrated 1-Wire devices contains a wide range of industrial implementations. The 1-Wire sensors and switches devices are very popular in the developer's community due to ease implementation. 1-Wire protocol can be fast implemented into the current IoT boards, most of the manufacturers share the software libraries allowing developers to include them in their projects in C, C++, assembly languages. The 1-Wire sensors (temperature, humidity, pressure etc.) are factory calibrated and reading the physical measurements follows the International System of Units (SI). 1-Wire products can be grouped as follows:

- Secure Authenticators,
- Memory EPROM, EEPROM ROM,
- Temperature Sensors and Temperature Switches,
- Data Loggers,
- 1-Wire Interface Products,
- Battery Monitors, Protectors, and Selectors,
- Battery ID and Authentication,
- Timekeeping and Real-Time Clocks.

4.3. Arduino Overview

In no doubt, Arduino became the most widespread SoC, particularly among enthusiasts, educators, amateurs, hobbyists, driving de-facto the embedded systems market for years.

Using cheap Atmel AVR microcontrollers, delivered along with development board and peripherals of almost any kind including sensors and actuators, where you do not need to develop your PCB nor solder to obtain the fully functional device, all that triggered new era where almost anyone can afford to have a development set and start playing the way only professionals used to do. Moreover, Arduino was not only the hardware but also the programming idea, delivering a simple development environment easy to use for beginners. Perhaps the most important impact of the Arduino to the daily use was to spread the idea of taking automation control from the industry and bring it on a massive scale to the regular life, homes, cars, toys; to automate daily life.

Beginnings of the Arduino are dated to the year 2003 in Italy yet. Their most popular development board was delivered to the market in fall 2010. While AVRs microcontrollers are considered to be embedded systems more than IoT, and most of the early Arduino boards didn't offer any network interface, even then it is essential to understand the idea of how to work with SoCs, so we start our guide here. However, there are many of the extension boards present, suitable for the standard development boards (so-called shields) that offer wired and wireless networking for Arduino. Some of the Arduino development boards nowadays do integrate networking SoC into the one board, i.e. Arduino Yun. Also, their clones, mostly made by Chinese manufacturers, evolved into more sophisticated products, integrating, i.e. Arduino Mega 2560 along with ESP8266 SoC into one development board.

Following chapters present the Arduino hardware overview, peripherals and programming as universal basics for IoT systems development using advanced

processors like i.e. ESP:

Setting up the programming environment

The syntax and the structure of the program

Data types and variables

Program control structures

Looping

Interrupts and sub-programs

Reading GPIOs, outputting data and tracing

4.3.1. Overview of the hardware device used

What is Arduino and why to use it?



Arduino is an open-source platform based on easy-to-use hardware and software.⁴⁰⁾ The Arduino project was started at the *Ivrea Interaction Design Institute* in Italy. Initially, the board aimed at students without a background in electronics and programming, but now boards are suitable for different IoT applications, wearable, embedded environments and other.

The Arduino board works by reacting on *inputs* that are received from various sensors and, after executing a *set of instructions*, an *output* is generated to respond to the environment. Input can be received by pressing a button, hearing the noise, perceiving an image of the situation using a camera and many other. The output actions on the environment are done using output sensors like actuator, blinking LED, audio device and other. The set of instructions are created using the *Arduino programming language* that is based on an open-source programming framework called *Wiring* and the *Arduino Software (IDE)* that is based on *Processing*.

Arduino microcontrollers can be used both in research and everyday applications. It is easy to use for people with different backgrounds, starting with students till experts. The *Arduino Forum*⁴¹⁾ is the place where users of Arduino can share their knowledge and get help and new ideas for developing their project.

The most common Arduino boards

Arduino boards can be divided into 6 sections depending on their specifications – entry level, enhanced features, Internet of things, education, wearable, and 3D printing boards.

The most common boards of Arduino are *Uno*, *Leonardo*, *Micro*, *Nano* (entry level), *Mega*, *Pro Mini* (enhanced features). Each of the board has different specifications and therefore can have different applications.

4. IoT Hardware overview



Figure 12: The most common Arduino boards.

4.3.2. Digital input/output pins

Digital input/output (I/O) pins are contacts on the Arduino board that can receive or transmit a digital signal. The status of the pin can be set either to 0 that represents LOW signal or to 1 - HIGH signal. The maximum current of the pin output is 40mA.

Table 4: The comparison of Arduino boards by the digital I/O pin number.

| | Uno | Leonardo | Micro | Mega | Nano | Pro Mini |
|-------------|-----|----------|-------|------|------|----------|
| Digital I/O | 14 | 20 | 20 | 54 | 22 | 14 |

4.3.3. Pulse Width Modulation

Pulse Width Modulation (PWM) is a function of a pin to generate a square wave signal, with a variable length of the HIGH level of the output signal. The PWM is used for digital pins to simulate the analogue output.

Table 5: The comparison of Arduino boards by the digital PWM pin number.

| | Uno | Leonardo | Micro | Mega | Nano | Pro Mini |
|-----|-----|----------|-------|------|------|----------|
| PWM | 6 | 7 | 7 | 12 | 6 | 6 |

4.3.4. Analog pins

Analog pins convert the analogue input value to a 10-bit number, using Analog Digital Converter (ADC). This function maps the input voltage between 0 and the reference voltage to numbers between 0 and 1023.

By default, the reference voltage is set to a microcontroller operating voltage. Usually, it is 5V or 3.3V. Also, other internal or external reference sources can be used, for example, AREF pin.

Table 6: The comparison of Arduino boards by the analog pin number.

| | Uno | Leonardo | Micro | Mega | Nano | Pro Mini |
|-------------|-----|----------|-------|------|------|----------|
| Analog pins | 6 | 12 | 12 | 16 | 8 | 6 |

4.3.5. Power and other pins

Power pins on the Arduino board connect the power source to the microcontroller and/or voltage regulators. They can also be used as a power source to the external components and devices.

The *VIN* pin is used to connect the external power source to the internal regulator, to provide the regulated 5V output. The input voltage of the board must be within the specific range, mostly between 7V and 12V.

The *5V* pin is used to supply a microcontroller with the regulated 5V from the external source or is used as a power source for the external components in the case when the board is already powered using the USB interface or the *VIN* pin.

The *3V3* pin provides the regulated 3.3V output for the board components and external devices. The *GND* (ground pin) is where the negative terminal of the power supply is applied.

The *Reset* pin and the *Reset* button are used to reset the Arduino board and the program. Resetting using the reset pin is done by connecting it to the *GND* pin.

4.3.6. Memory

There are three different types of memory on the Arduino board: Flash memory, SRAM and EEPROM.

The Flash memory stores the Arduino code, and it is a non-volatile type of memory. That means the information in the memory is not deleted when the power is turned off.

The SRAM (static random access memory) is used for storing values of variables when the program of Arduino is running. This is the volatile memory that keeps information only until the power is turned off, or the board is reset.

The EEPROM (electrically erasable programmable read-only memory) is a non-volatile type of memory that can be used as the long-term memory storage.

Table 7: The comparison of Arduino boards by memory size.

| | Uno | Leonardo | Micro | Mega | Nano | Pro Mini |
|-------------|------------|-----------------|--------------|-------------|-------------|-----------------|
| Flash (kB) | 32 | 32 | 32 | 256 | 32 | 32 |
| SRAM (kB) | 2 | 2 | 2.5 | 8 | 2 | 2 |
| EEPROM (kB) | 1 | 1 | 1 | 4 | 1 | 1 |

4.3.7. Interface

Communication interfaces for Arduino are used to send and receive information to and from other external devices. Standard interfaces for Arduino are USB, UART, I2C (two wire interface), SPI, Ethernet and WiFi.

Table 8: The comparison of Arduino boards by interface available.

4. IoT Hardware overview

| | Uno | Leonardo | Micro | Due | Nano | Pro Mini |
|-----------|------------|-----------------|--------------|------------|-------------|-----------------|
| USB | 1 USB B | 1 Micro | 1 Micro | 1 USB B | 1 Mini | - |
| UART | 1 | 1 | 1 | 4 | 1 | 1 |
| Wire(I2c) | 1 | 1 | 1 | 1 | 1 | 1 |
| SPI | 1 | 1 | 1 | 1 | 1 | 1 |

4.3.8. Size of the board

Arduino microcontrollers have different dimensions of the board depending on the components that are located on the board.

Table 9: The comparison of Arduino boards by the size of the board.

| | Uno | Leonardo | Micro | Mega | Nano | Pro Mini |
|-----------|-------------|-----------------|--------------|---------------|-------------|-----------------|
| Size (mm) | 68.6 x 53.4 | 68.6 x 53.3 | 48 x 18 | 101.52 x 53.3 | 18 x 45 | 18 x 33 |

4.3.9. Arduino shields

Arduino shields are the extension boards that can be plugged on top of the Arduino board extending its capabilities. The shields can give additional functionality to the Arduino board. There are multiple categories of the Arduino shields [<https://learn.sparkfun.com/tutorials/arduino-shields>] - *prototyping, improving connectivity, displays and cameras, sound and motor driver shields*.

Prototyping shields - are shields that do not give Arduino the additional functionality, but help with the wiring. Some example prototyping shields are *ProtoShield*, *ProtoScrew Shield*, *Go-Between Shield*, *LiPower Shield*, *Danger Shield*, *Joystick Shield* and *microSD Shield*.

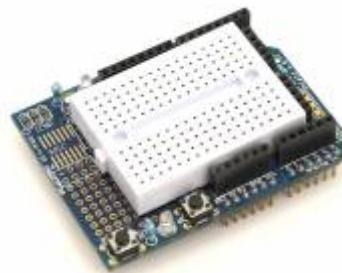


Figure 13: Prototype shield

Connectivity shields - are shields that can add new functionalities to the Arduino board like Ethernet, WiFi, Wireless, GPS, etc. Example shields are *Arduino Ethernet Shield*, *WiFly Shield*, *Arduino Wi-Fi Shield*, *Electric Imp Shield*, *XBee Shield*, *Cellular Shield w/ SM5100B* and *GPS Shield*.



Figure 14: Arduino wifi shield MKR WIFI 1010

Displays and camera shields - can provide Arduino with an LCD screen or add a camera. Example shields are *Color LCD Shield*, *EL Escudo* and *CMUcam*.



Figure 15: SparkFun Color LCD Shield

Sound shields - give the functionality to Arduino to play MP3 files, add speakers, listen to audio and sort it into different frequencies, etc. Example shields are *MP3 Player Shield*, *Music Instrument Shield*, *Spectrum Shield* and *VoiceBox Shield*.

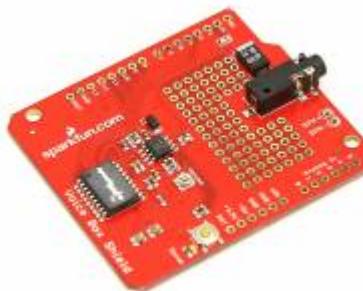


Figure 16: SparkFun MP3 Player Shield

Motor driver shields - allow Arduino to control DC motors, Servo motors, Stepper motors. Examples are *Ardumoto Motor Driver Shield*, *Monster Moto Shield* and *PWM Shield*.

4. IoT Hardware overview

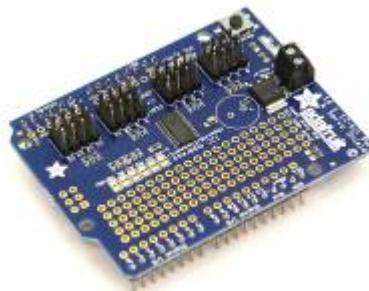


Figure 17: Adafruit Servo shield

4.3.10. Sensors and sensing

A sensor is an element which can turn a physical outer stimulus into an output signal which then can be used for further analysis, management or decision making. People also use sensors like eyes, ears and skin for gaining information about the outer world and act accordingly to their aims and needs. Sensors can be divided into multiple categories by the parameter that is perceived from the environment.

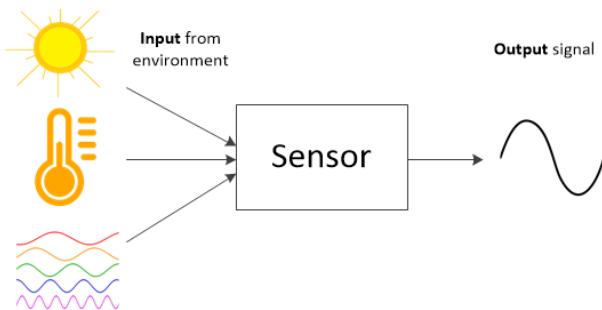


Figure 18: caption

Usually every natural phenomenon – temperature, weight, speed, etc. – needs specially customised sensors which can change every phenomenon into electronic signals that could be used by microprocessors or other devices. Sensors can be divided into many groups according to the physical nature of their operations - *touch, light, an electrical characteristic, proximity and distance, angle, environment and other sensors*.

Touch sensors

Button

A *pushbutton* is an electromechanical sensor that connects or disconnects two points in a circuit when the force is applied. Button output discrete value is either HIGH or LOW.



Figure 19: Pushbutton

A *microswitch*, also called a miniature snap-action switch, is an electromechanical sensor that requires a very little physical force and uses tipping-point mechanism. Microswitch has 3 pins, two of which are connected by default. When the force is applied, the first connection breaks and one of the pins is connected to the third pin.



Figure 20: Microswitch

The most common use of a pushbutton is as an input device. Both force solutions can be used as simple object detectors, or as end switches in the industrial devices.

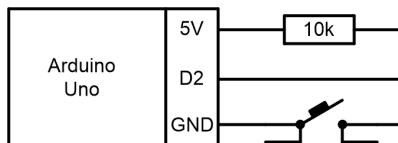


Figure 21: Schematics of Arduino Uno and a push button

An example code:

```
int buttonPin = 2; //Initialization of a push button pin number
int buttonState = 0; //A variable for reading the push button status

void setup() {
  Serial.begin(9600); //Begin serial communication
  pinMode(buttonPin, INPUT); // initialize the push button pin as an input
}

void loop() {
  // read the state of the push button value
  buttonState = !digitalRead(buttonPin);
  // check if the push button is pressed. If it is, the buttonState is HIGH
  if (buttonState == HIGH) {
    //print out text in the console
    Serial.println("The button state is HIGH - it is pressed.");
  } else {
    Serial.println("The button state is LOW - it is not pressed.");
  }
  delay(10); // delay in between reads for stability
}
```

4. IoT Hardware overview

```
}
```

Force sensor

A *force sensor* predictably changes resistance, depending on the applied force to its surface. Force-sensing resistors are manufactured in different shapes and sizes, and they can measure not only direct force but also the tension, compression, torsion and other types of mechanical forces. The voltage is measured by applying and measuring constant voltage to the sensor.

Force sensors are used as control buttons or to determine weight.



Figure 22: Force sensitive resistor (FSR)

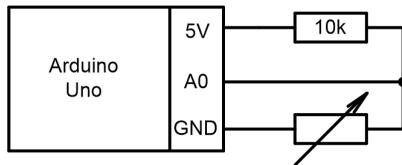


Figure 23: The voltage is measured by applying and measuring constant voltage to the sensor

An example code:

```
// Force Sensitive Resistor (FSR) is connected to the analog 0 pin
int fsrPin = A0;
// the analog reading from the FSR resistor divider
int fsrReading;

void setup(void) {
    // Begin serial communication
    Serial.begin(9600);
    // initialize the FSR analog pin as an input
    pinMode(fsrPin, INPUT);
}

void loop(void) {
    // read the resistance value of the FSR
    fsrReading = analogRead(fsrPin);
    // print
    Serial.print("Analog reading = ");
    Serial.println(fsrReading);
    delay(10);
}
```

```
}
```

Capacitive sensor

Capacitive sensors are a range of sensors that use capacitance to measure changes in the surrounding environment. A capacitive sensor consists of a capacitor that is charged with a certain amount of current till the threshold voltage. A human finger, liquids or other conductive or dielectric materials that touch the sensor, can influence a charge time and a voltage level in the sensor. Measuring a charge time and a voltage level gives information about changes in the environment.

Capacitive sensors are used as input devices and can measure proximity, humidity, fluid level and other physical parameters or serve as an input for electronic device control.



Figure 24: Touch button module

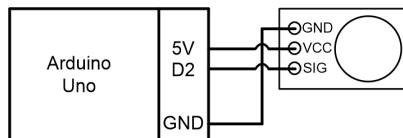


Figure 25: Arduino and capacitive sensor schematics

```
// capacitive sensor is connected to the digital 2 pin
int touchPin = 2;

// the digital reading value from the sensor
boolean touchReading = LOW;
//the variable that stores the previous state value
boolean lastState = LOW;

void setup() {
    // begin serial communication
    Serial.begin(9600);
    // initialize the capacitive sensor analog pin as an input
    pinMode(touchPin, INPUT);
}

void loop() {
    // read the digital value of the capacitive sensor
    touchReading = digitalRead(touchPin);
    //if the new touch has appeared
    if (currentState == HIGH && lastState == LOW) {
        Serial.println("Sensor is pressed");
        delay(10); //short delay
    }
    //save previous state to see relative changes
    lastState = currentState;
```

4. IoT Hardware overview

```
}
```

Light sensors

Photoresistor

A *photoresistor* is a sensor that perceives light waves from the environment. The resistance of the photoresistor is changing depending on the intensity of light. The higher is the intensity of the light, the lower is the resistance of the sensor. A light level is determined by applying a constant voltage sensor and measuring it. Photodiodes, compared to photoresistors, are slower and more influenced by temperature, thus they are more imprecise.

Photoresistors are often used in the energy effective street lightning.

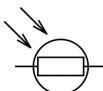


Figure 26: A photoresistor symbol



Figure 27: A photoresistor

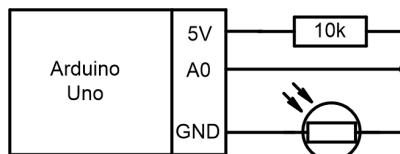


Figure 28: Arduino and photoresistor sensor schematics

An example code:

```
//define an analog A0 pin for photoresistor
int photoresistorPin = A0;
//the analog reading from the photoresistor
int photoresistorReading;

void setup()
{
    //Begin serial communication
    Serial.begin(9600);
    // initialize the analog pin of a photoresistor as an input
    pinMode(photoresistorPin, INPUT);
}
```

```

void loop()
{
    // read the value of the photoresistor
    photoresistorReading = analogRead(photoresistorPin);
    // print out value of the photoresistor reading to the serial monitor
    Serial.println(photoresistorReading);
    delay(10); //short delay
}

```

Photodiode

A photodiode is a sensor that converts the light energy into electrical current. A current in the sensor is generated by exposing a p-n junction of a semiconductor to the light. Information about the light intensity can be determined by measuring a voltage level. Photodiodes are reacting to the changes in the light intensity very quickly. Solar cells are just large photodiodes.

Photodiodes are used as precise light level sensors, receivers for remote control, electrical isolators and proximity detectors.



Figure 29: A photodiode symbol



Figure 30: A photodiode

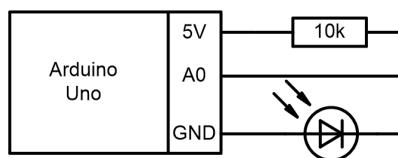


Figure 31: Arduino and photodiode sensor schematics

An example code:

```

//define an analog A0 pin for photodiode
int photodiodePin = A0;
//the analog reading from the photodiode
int photodiodeReading;

void setup()
{
    //Begin serial communication
    Serial.begin(9600);
}

```

4. IoT Hardware overview

```
// initialize the analog pin of a photodiode as an input  
pinMode(photodiodePin, INPUT);  
  
}  
  
void loop()  
{  
    //read the value of the photodiode  
    photodiodeReading = analogRead(photodiodePin);  
    //print out the value of the photodiode reading to the serial monitor  
    Serial.println(photodiodeReading);  
    delay(10); //short delay  
}
```

Phototransistor

A *phototransistor* is a light controlled electrical switch. In the exposed Base pin received light level, changes the amount of current, that can pass between two phototransistor pins - a collector and an emitter. A phototransistor is slower than the photodiode, but it can conduct more current.

Phototransistors are used as the optical switches, proximity sensors and electrical isolators.

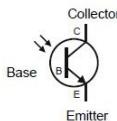


Figure 32: A phototransistor symbol



Figure 33: An phototransistor

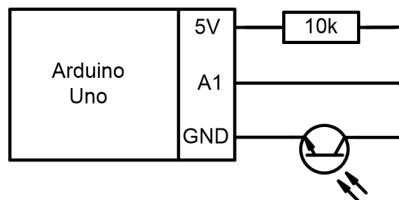


Figure 34: Arduino and phototransistor schematics

An example code:

```
//define an analog A1 pin for phototransistor  
int phototransistorPin = A1;  
//the analog reading from the phototransistor
```

```

int phototransistorReading;

void setup()
{
    //Begin serial communication
    Serial.begin(9600);
    // initialize the analog pin of a phototransistor as an input
    pinMode(phototransistorPin, INPUT);
}

void loop()
{
    //read the value of the phototransistor
    phototransistorReading = analogRead(phototransistorPin);
    //print out the value of the phototransistor reading to the serial monitor
    Serial.println(phototransistorReading);
    delay(10); //short delay
}

```

Electrical characteristic sensors

Electrical characteristic sensors are used to determine whether the circuit of the device is working properly. When the voltage and current sensors are used concurrently, the consumed power of the device can be determined.

Voltage sensor

A *voltage sensor* is a device or circuit for voltage measurement. A simple DC (direct current) voltage sensor consists of a voltage divider circuit with the optional amplifier for very small voltage occasions. For measuring the AC (alternating current), a transformer is added to a lower voltage; then it is connected to the rectifier to rectify AC to DC, and finally, an optoelectrical isolator is added for measuring circuit safety.

A voltage sensor can measure electrical load and detect a power failure. Examples of IoT applications are monitoring of appliance, line power, power coupling, power supply and sump pump.



Figure 35: Voltage sensor module 0-25V

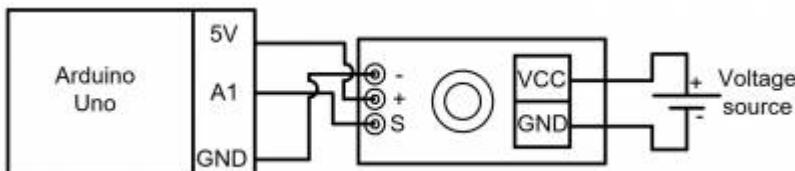


Figure 36: Arduino and voltage sensor schematics

4. IoT Hardware overview

The example code:

```
//define an analog A1 pin for voltage sensor
int voltagePin = A1;
//the analog reading from the voltage sensor
int voltageReading;

float vout = 0.0;
float vin = 0.0;
float R1 = 30000.0; // 30k resistor
float R2 = 7500.0; // 7.5k resistor

void setup()
{
    //Begin serial communication
    Serial.begin(9600);
    // initialize the analog pin of a voltage sensor as an input
    pinMode(voltagePin, INPUT);
}

void loop()
{
    //read the value of the voltage sensor
    voltageReading = analogRead(voltagePin);
    vout = (voltageReading * 5.0) / 1024.0;
    vin = vout / (R2/(R1+R2));

    Serial.print("Voltage is: ");
    //print out the value of the voltage to the serial monitor
    Serial.println(vin);
    delay(10); //short delay
}
```

Current sensor

A *current sensor* is a device or a circuit for current measurement. A simple DC sensor consists of a high power resistor with low resistance. The current is obtained by measuring the voltage on the resistor and applying formula proportional to the voltage. Other non-invasive measurement methods involve hall effect sensors for DC and AC and inductive coils for AC. Current sensors are used to determine the power consumption, to detect whether the device is turned on, short circuits.



Figure 37: Analog current meter module 0~50A

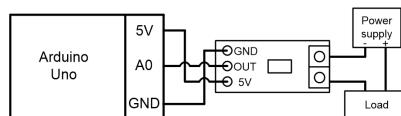


Figure 38: Arduino and current sensor module schematics

The example code:

```
//define an analog A0 pin for current sensor
const int currentPin = A0;
// scale factor of the sensor use 100 for 20A Module and 66 for 30A Module
int mVperAmp = 185;
int currentReading;
int ACSoffset = 2500;
double Voltage;
double Current;

void setup(){
  Serial.begin(9600);
}

void loop(){

currentReading = analogRead(currentPin);
Voltage = (currentReading / 1024.0) * 5000; // Gets you mV
Current = ((Voltage - ACSoffset) / mVperAmp); //calculating current value

Serial.print("Raw Value = " ); // shows pre-scaled value
Serial.print(currentReading);
Serial.print("\t Current = "); // shows the voltage measured
// the '3' after current allows to display 3 digits after decimal point
Serial.println(Current,3);
delay(1000); //short delay
}
```

Proximity and distance sensors

Optocoupler

An *optocoupler* is a device that combines light emitting and receiving devices. Mostly it is a combination of the infrared light-emitting diode (LED) and a phototransistor. Other optical semiconductors can be a photodiode and a photoresistor. There are two main types of optocouplers:

- an *optocoupler of a closed pair configuration* is enclosed in the dark resin and is used to transfer signals using light, ensuring electrical isolation between two circuits;
- a *slotted optocoupler* has a space between the light source and the sensor, light can be obstructed and thus can influence the sensor signal. It can be used to detect objects, rotation speed, vibrations or serve as a bounce-free switch;
- a *reflective pair configuration* the light signal is perceived as a reflection from the object surface. This configuration is used for proximity detection, surface colour detection and tachometer.

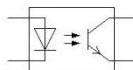


Figure 39: An optocoupler symbol

4. IoT Hardware overview



Figure 40: ELITR9909 reflective optocoupler sensor

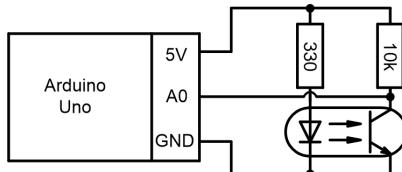


Figure 41: Arduino Uno and optocoupler schematics

An example code:

```
int optoPin = A0; // initialize an analog A0 pin for optocoupler
int optoReading; //the analog value reading from the optocoupler

int objecttreshold = 1000; //Object threshold definition
int whitetreshold = 150; //White colour threshold definition

void setup ()
{
    //Begin serial communication
    Serial.begin(9600);
    // initialize the analog pin of the optocoupler as an input
    pinMode(optoPin, INPUT);
}

void loop ()
{
    optoReading = analogRead(optoPin); //read the value of the optocoupler
    Serial.print ("The reading of the optocoupler sensor is: ");
    Serial.println(optoReading);

    //when the reading value is lower than the object threshold
    if (optoReading < objecttreshold) {
        Serial.println ("There is an object in front of the sensor!");
        //when the reading value is lower than the white colour threshold
        if (optoReading < white treshold) {
            Serial.println ("Object is in white colour!");
        } else { //when the reading value is higher than the white colour threshold
            Serial.println ("Object is in dark colour!");
        }
    }
    else { //when the reading value is higher than the object treshold
        Serial.println ("There is no object in front of the sensor!");
    }
    delay(500); //short delay
}
```

Infrared sensor

Infrared (IR) proximity sensor is used to detect objects and to measure the distance to them, without any physical contact. IR sensor consists of an infrared emitter, a receiving sensor or array of sensors and a signal processing logic. The output of a sensor differs depending on the type - simple proximity detection sensor outputs HIGH or LOW level when an object is in its sensing range, but sensors which can measure distance outputs an analogue signal or use some communication protocol, like I2C to send sensor measuring results. IR sensors are used in robotics to detect obstacles starting from few millimeters to several meters and in mobile phones to help detect accidental button touching.



Figure 42: Distance Sensor GP2Y0A21YK0F

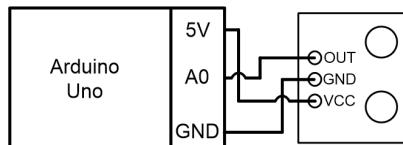


Figure 43: Arduino and IR proximity sensor circuit

An example code:

```

int irPin = A0; //define an analog A0 pin for IR sensor
int irReading; //the analog reading from the IR sensor

void setup()
{
    //Begin serial communication
    Serial.begin(9600);
    // initialize the analog pin of a IR sensor as an input
    pinMode(irPin, INPUT);
}

void loop()
{
    //read the value of the IR sensor
    irReading = analogRead(irPin);
    //print out the value of the IR sensor reading to the serial monitor
    Serial.println(irReading);
    delay(10); //short delay
}

```

Ultrasound sensor

Ultrasound (ultrasonic) sensor measures the distance to objects by emitting ultrasound and measuring its returning time. The sensor consists of an ultrasonic emitter and

4. IoT Hardware overview

receiver; sometimes they are combined in a single device for emitting and receiving. Ultrasonic sensors can measure greater distances and cost less than infrared sensors, but are more imprecise and interfere with each other measurement if more than one is used. Simple sensors have trigger pin and echo pin, when trigger pin is set high for the small amount of time ultrasound is emitted and on echo pin, response time is measured. Ultrasonic sensors are used in car parking sensors and robots for proximity detection.



Figure 44: Ultrasonic proximity sensor HC-SR04

Examples of IoT applications are robotic obstacle detection and room layout scanning.

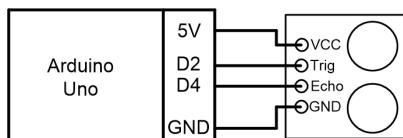


Figure 45: Arduino and ultrasound proximity sensor circuit

An example code:

```
int trigPin = 2; //define a trigger pin D2
int echoPin = 4; //define an echo pin D4

void setup()
{
    Serial.begin(9600); // Begin serial communication
    pinMode(trigPin, OUTPUT); // Set the trigPin as an Output
    pinMode(echoPin, INPUT); // Set the echoPin as an Input
}

void loop()
{
    digitalWrite(trigPin, LOW); // Clear the trigPin
    delayMicroseconds(2);

    // Set the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Read the echoPin, return the sound wave travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    distance= duration*0.034/2;

    // Printing the distance on the Serial Monitor
    Serial.print("Distance: ");
    Serial.println(distance);
}
```

Motion detector

Motion detector is a sensor that detects moving objects, most people. Motion detectors use different technologies, like passive infrared sensors, microwaves and Doppler effect, video cameras and previously mentioned ultrasonic and IR sensors. Passive IR sensors are the simplest motion detectors that sense people through detecting IR radiation that is emitted through the skin. When the motion is detected, the output of a motion sensor is a digital HIGH/LOW signal.

Motion sensors are used for security purposes, automated light and door systems. As an example in IoT, the PIR motion sensor can be used to detect motion in security systems a house or any building.



Figure 46: PIR motion sensor

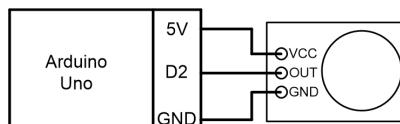


Figure 47: Arduino and PIR motion sensor circuit

An example code:

```
// Passive Infrared (PIR) sensor output is connected to the digital 2 pin
int pirPin = 2;
// the digital reading from the PIR output
int pirReading;

void setup(void) {
    // Begin serial communication
    Serial.begin(9600);
    // initialize the PIR digital pin as an input
    pinMode(pirPin, INPUT);
}

void loop(void) {
    // read the digital value of the PIR motion sensor
    pirReading = digitalRead(pirPin);
    //print out
    Serial.print("Digital reading = ");
    Serial.println(pirReading);

    if(pirReading == HIGH) { //motion was detected
        Serial.println("Motion Detected");
    }

    delay(10);
}
```

4. IoT Hardware overview

Angle sensors

Potentiometer

A **potentiometer** is a type of resistor, the resistance of which can be adjusted using a mechanical lever. The device consists of three terminals. The resistor between the first and the third terminal has fixed value, but the second terminal is connected to the lever. Whenever the lever is turned, a slider of the resistor is moved, it changes the resistance between the second terminal and side terminals. Variable resistance causes the change of the voltage variable, and it can be measured to determine the position of the lever. Thus, potentiometer output is an analogue value.

Potentiometers are commonly used as a control level, for example, a volume level for the sound and joystick position. They can also be used for angle measurement in feedback loops with motors, for example, in servo motors.

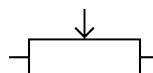


Figure 48: A symbol of potentiometer



Figure 49: A potentiometer

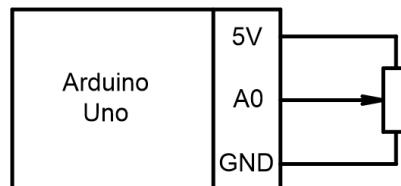


Figure 50: Arduino and potentiometer circuit

An example code:

```
// Potentiometer sensor output is connected to the analog A0 pin
int potentiocPin = A0;
// the analog reading from the potentiometer output
int potentioreading;

void setup(void) {
    // Begin serial communication
    Serial.begin(9600);
    // initialize the potentiometer analog pin as an input
    pinMode(potentiocPin, INPUT);
}
```

```
void loop(void) {
    // read the analog value of the potentiometer sensor
    potentioreading = analogRead(potentiopin);
    Serial.print("Potentiometer reading = ");
    Serial.println(potentioreading);
    delay(10);
}
```

The inertial measurement unit (IMU)

An IMU is an electronic device, that consist of accelerometer, gyroscope and sometimes also a magnetometer. Combination of these sensors returns orientation of the object in 3D space.

A *gyroscope* is a sensor that measures the angular velocity. The sensor is made of the microelectromechanical system (MEMS) technology and is integrated into the chip. The output of the sensor can be either analogue or digital value of information, using I2C or SPI interface. Gyroscope microchips can vary in the number of axes they can measure. The available number of the axis is 1, 2 or 3 axes in the gyroscope. For gyroscopes with 1 or 2 axes, it is essential to determine which axis the gyroscope measures and to choose a device according to the project needs. A gyroscope is commonly used together with an accelerometer, to determine the orientation, position and velocity of the device precisely. Gyroscope sensors are used in aviation, navigation and motion control.

A *magnetometer* is the sensor, that can measure the orientation of the device to the magnetic field of the Earth. A magnetometer is used in outdoor navigation for mobile devices, robots, quadcopters.

An *accelerometer* measures the acceleration of the object. The sensor uses a microelectromechanical system (MEMS) technology, where capacitive plates are attached to springs. When acceleration force is applied to the plates, the capacitance is changed, thus it can be measured. Accelerometers can have 1 to 3 axis. On 3-axis the accelerometer can detect orientation, shake, tap, double tap, fall, tilt, motion, positioning, shock or vibration of the device. Outputs of the sensor are usually digital interfaces like I2C or SPI. For precise measurement of the object movement and orientation in space, the accelerometer is often used together with a gyroscope. Accelerometers are used for measuring vibrations of cars, industrial devices, buildings and to detect volcanic activity. In IoT applications, it can be used as well for accurate motion detection for medical and home appliances, portable navigation devices, augmented reality, smartphones and tablets.



Figure 51: IMU BNO055 module

4. IoT Hardware overview

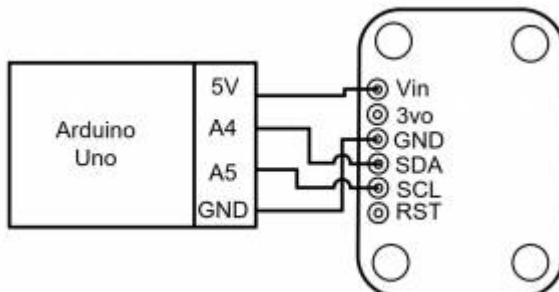


Figure 52: Arduino Uno and IMU BNO055 module schematics

The example code:

```
//Library for I2C communication
#include <Wire.h>
//downloaded from https://github.com/adafruit/Adafruit_Sensor
#include <Adafruit_Sensor.h>
//downloaded from https://github.com/adafruit/Adafruit_BNO055
#include <Adafruit_BNO055.h>
#include <utility/imumath.h>
Adafruit_BNO055 bno = Adafruit_BNO055(55);
void setup(void)
{
bno.setExtCrystalUse(true);
}
void loop(void)
{
//Read sensor data
sensors_event_t event;
bno.getEvent(&event);
// Print X, Y And Z orientation
Serial.print("X: ");
Serial.print(event.orientation.x, 4);
Serial.print("\tY: ");
Serial.print(event.orientation.y, 4);
Serial.print("\tZ: ");
Serial.print(event.orientation.z, 4);
Serial.println("");
delay(100);
}
```

Environment sensors

Temperature sensor

A *temperature sensor* is a device that is used to determine the temperature of the surrounding environment. Most temperature sensors work on the principle that the resistance of the material is changed depending on its temperature. The most common temperature sensors are:

- *thermocouple* - consists of two junctions of dissimilar metals.

4.3. Arduino Overview

- *thermistor* – includes the temperature-dependent ceramic resistor,
- *resistive temperature detector* – is made of a pure metal coil.

The main difference between sensors is the measured temperature range, precision and response time. Temperature sensor usually outputs the analogue value, but some existing sensors have a digital interface. ⁴²⁾

The temperature sensors most commonly are used in environmental monitoring devices and thermoelectric switches. In IoT applications, the sensor can be used for greenhouse temperature monitoring, warehouse temperature monitoring to avoid frozen fire suppression systems and tracking temperature of the soil, water and plants.



Figure 53: Thermistor

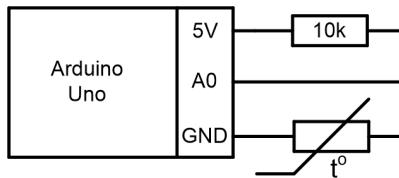


Figure 54: Arduino and thermistor circuit

An example code:

```
// Thermistor sensor output is connected to the analog A0 pin
int thermoPin = 0;
// the analog reading from the thermistor output
int thermoReading;

void setup(void) {
  // Begin serial communication
  Serial.begin(9600);
  // initialize the thermistor analog pin as an input
  pinMode(thermoPin, INPUT);
}

void loop(void) {
  // read the analog value of the thermistor sensor
  thermoReading = analogRead(thermoPin);
  Serial.print("Thermistor reading = ");
  //print out
  Serial.println(thermoReading);
  delay(10);
}
```

Humidity sensor

A *humidity sensor* (hygrometer) is a sensor that detects the amount of water or water vapour in the environment. The most common principle of the air humidity sensors is the change of capacitance or resistance of materials that absorb the moisture from the

4. IoT Hardware overview

environment. Soil humidity sensors measure the resistance between the two electrodes. The resistance between electrodes is influenced by soluble salts and water amount in the soil. The output of a humidity sensor is usually an analogue signal value.⁴³⁾

Example IoT applications are monitoring of humidors, greenhouse temperature and humidity, agriculture, art gallery and museum environment.



Figure 55: Temperature and humidity sensor module

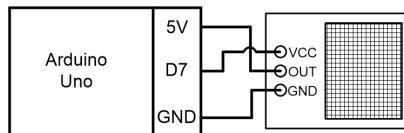


Figure 56: Arduino Uno and humidity sensor schematics

An example code:

```
#include <dht.h>

dht DHT;

#define DHT_PIN 7

void setup() {
    Serial.begin(9600);
}

void loop()
{
    int chk = DHT.read11(DHT_PIN);
    Serial.print("Humidity = ");
    Serial.println(DHT.humidity);
    delay(1000);
}
```

44)

Sound sensor

A *sound sensor* is a sensor that detects vibrations in a gas, liquid or solid environments. At first, the sound wave pressure makes mechanical vibrations, who transfers to changes in capacitance, electromagnetic induction, light modulation or piezoelectric generation to create an electric signal. The electrical signal is then amplified to the required output levels. Sound sensors, can be used to record sound, detect noise and its level.

Sound sensors are used in drone detection, gunshot alert, seismic detection and vault

safety alarm.

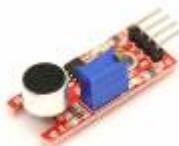


Figure 57: Digital sound detector sensor module

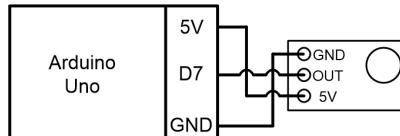


Figure 58: Arduino Uno and sound sensor schematics

An example code:

```
// Sound sensor output is connected to the digital 7 pin
int soundPin = 7;
// stores sound sensor detection readings
int soundReading = HIGH;

void setup(void) {
    // Begin serial communication
    Serial.begin(9600);
    // initialize the sound detector module pin as an input
    pinMode(soundPin, INPUT);
}

void loop(void) {
    // read the digital value whether the sound has been detected
    soundReading = digitalRead(soundPin);
    if (soundPin==LOW) { //when sound detector detected the sound
        Serial.println("Sound detected!"); //print out
    } else { //when the sound is not detected
        Serial.println("Sound not detected!"); //print out
    }
    delay(10);
}
```

Chemical/smoke and gas sensor

Gas sensors are a sensor group, that can detect and measure a concentration of certain gasses in the air. The working principle of electrochemical sensors is to absorb the gas and to create current from an electrochemical reaction. For process acceleration, a heating element can be used. For each type of gas different kind of sensor needs to be used. Multiple different types of gas sensors can be combined in a single device as well. The single gas sensor output is an analogue signal, but devices with multiple sensors used to have a digital interface.

Gas sensors are used for safety devices, to control air quality and for manufacturing

4. IoT Hardware overview

equipment. Examples of IoT applications are air quality control management in smart buildings and smart cities or toxic gas detection in sewers and underground mines.



Figure 59: MQ-7 gas sensor

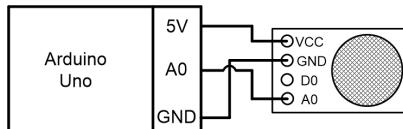


Figure 60: Arduino Uno and MQ2 gas sensor schematics

An example code:

```
int gasPin = A0; // Gas sensor output is connected to the analog A0 pin
int gasReading; // stores gas sensor detection reading

void setup(void) {
    Serial.begin(9600); // Begin serial communication
    pinMode(gasPin, INPUT); // initialize the gas detector pin as an input
}

void loop(void) {
    gasReading = analogRead(gasPin); // read the analog value of the gas sensor
    Serial.print("Gas detector value: ");
    Serial.println(gasReading);
    delay(10); //short delay
}
```

Level sensor

A *level sensor* detects the level of fluid or fluidised solid. Level sensors can be divided into two groups:

- *continuous level sensors* that can detect the exact position of the fluid. For the level detection usually, the proximity sensors, like ultrasonic or infrared, are used. Capacitive sensors can also be used by recording the changing capacitance value depending on the fluid level. The output can be either analogue or digital value.
- *point-level sensors* can detect whether a fluid is above or below the sensor. For the level detection, float or mechanical switch, diaphragm with air pressure or changes in conductivity or capacitance, can be used. The output is usually a digital value that indicates HIGH or LOW value.

Level sensors can be used as smart waste management, for measuring tank levels, diesel fuel gauging, liquid assets inventory, chemical manufacturing high or low-level alarms and irrigation control.



Figure 61: Liquid level sensor

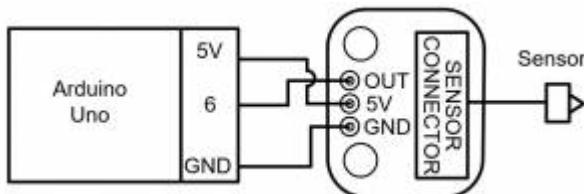


Figure 62: Arduino Uno and liquid level sensor schematics

An example code:

```

int levelPin = 6; //liquid level sensor output is connected to the digital 6 pin
int levelReading; //stores level sensor detection reading

void setup(void) {
    Serial.begin(9600); //Begin serial communication
    pinMode(levelPin, INPUT); //initialize the level sensor pin as an input
}

void loop(void) {
    levelReading = digitalRead(levelPin); //read the digital value of the level sensor
    Serial.print("Level sensor value: ");
    Serial.println(levelReading);
    delay(10); //short delay
}

```

Other sensors

Hall sensor

A *Hall effect sensor* detects strong magnetic fields, their polarities and the relative strength of the field. In the Hall effect sensors, a magnetic force influences current flow through the semiconductor material and creates a measurable voltage on the sides of the semiconductor. Sensors with analogue output can measure the strength of the magnetic field, while digital sensors give HIGH or LOW output value, depending on the presence of the magnetic field.

Hall effect sensors are used in magnetic encoders for speed measurements and magnetic proximity switches because it does not require contact, and it ensures high reliability. Example application can be sensing the position of rotary valves.

4. IoT Hardware overview



Figure 63: Hall-effect sensor module

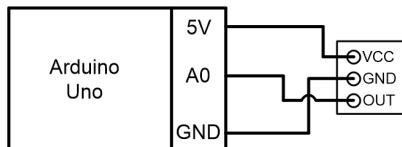


Figure 64: Arduino Uno and Hall sensor schematics

Thw example code:

```
int hallPin = A0; // Hall sensor output is connected to the analog A0 pin
int hallReading; // stores hallsensor detection reading

void setup(void) {
    Serial.begin(9600); // Begin serial communication
    pinMode(hallPin, INPUT); // initialize the hallsensor pin as an input
}

void loop(void) {
    hallReading = analogRead(hallPin); //read the analog value of the hall sensor
    Serial.print("Hall sensor value: "); //print out
    Serial.println(hallReading);
    delay(10); //short delay
}
```

Global positioning system

A *GPS receiver* is a device, that can receive information from global navigation satellite system and calculate its position on the Earth. GPS receiver uses a constellation of satellites and ground stations to compute position and time almost anywhere on the Earth. GPS receivers are used for navigation only in the outdoor area because it needs to receive signals from the satellites. The precision of the GPS location can vary.

A GPS receiver is used for device location tracking. Real world applications might be, i.e.: pet, kid or personal belonging location tracking.



Figure 65: Grove GPS receiver module

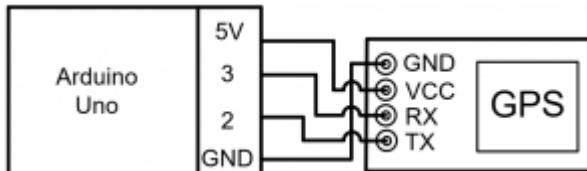


Figure 66: Arduino Uno and Grove GPS receiver schematics

The example code⁴⁵⁾:

```
#include <SoftwareSerial.h>
SoftwareSerial SoftSerial(2, 3);
unsigned char buffer[64]; // buffer array for data receive over serial port
int count=0; // counter for buffer array
void setup()
{
    SoftSerial.begin(9600); // the SoftSerial baud rate
    Serial.begin(9600); // the Serial port of Arduino baud rate.
}

void loop()
{
    if (SoftSerial.available()) // if date is coming from software serial
    {
        while(SoftSerial.available()) // reading data into char array
        {
            buffer[count++]=SoftSerial.read(); // writing data into array
            if(count == 64)break;
        }
        Serial.write(buffer,count); // if no data transmission ends, write it
        clearBufferArray(); // call clearBufferArray function to clear buffer
        count = 0; // set counter of while loop to zero
    }
    if (Serial.available()) // if data is available on hardware serial port
        SoftSerial.write(Serial.read()); // write it to the SoftSerial shield
}

void clearBufferArray() // function to clear buffer array
{
    for (int i=0; i<count;i++)
    {
        buffer[i]=NULL; // clear all index of array with command NULL
    }
}
```

4.3.11. Drivers and driving

Optical device drivers and their devices

Light-emitting diode

Light-emitting diode also called LED is a special type of diodes which emits light, unlike the other diodes. LED has a completely different body which is made of transparent

4. IoT Hardware overview

plastic that protects the diode and lets it emit light. Like the other diodes LED conducts the current in only one way, so it is essential to connect it to the scheme correctly. There are two safe ways how to determine the direction of the diode:

- in the cathodes side of the diode its side is chipped,
- anodes leg usually is longer than the cathodes leg.



Figure 67: 5mm Red LED

LED is one of the best light sources. Unlike incandescent light bulb LED transforms most of the power into light not warmth; it is more durable, works for a more extended period and can be manufactured in a smaller size.

LED colour is determined by the semiconductors material. Diodes are usually made from silicon then LEDs are made from elements like gallium phosphate, silicon carbide and others. Because the semiconductors used are different the voltage needed for the LED to shine is also different. In the table, you can see with which semiconductor you can get a specific colour and the voltage required to turn on the LED.

When LED is connected to the voltage and turned on a huge current starts to flow through it, and it can damage the diode. That is why all **LEDs have to be connected to current limiting resistor**.

Current limiting resistors resistance is determined by three parameters:

- **I_D** - current that can flow through the LED,
- **U_D** - Voltage that is needed to turn on the LED,
- **U** - combined voltage for LED and resistor.

To calculate the resistance needed for a diode, this is what you have to do:

1. Find out the voltage needed for the diode to work U_D; you can find it in the diodes parameters table.
2. Find out the amperage needed for the LED to shine I_D; it can be found in the LEDs datasheet, but if you can't find it then 20 mA current is usually correct and safe choice.
3. Find out the combined voltage for the LED and resistor; usually, it is the feeding voltage for the scheme.
4. Insert all the values into this equation: **R=(U-U_D)/I_D**
5. You get the resistance for the resistor for the safe use of the LED.
6. Find resistors nominal that is the same or bigger than the calculated resistance.

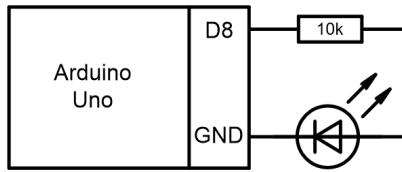


Figure 68: caption

The example of blinking LED code:

```
int ledPin = 8; //defining the pin of the LED

void setup()
{
    pinMode(ledPin,OUTPUT); // LED pin is set to output
}

void loop()
{
    //set pin output signal to HIGH - LED is working
    digitalWrite(ledPin,HIGH);
    //delay of 1000 milliseconds
    delay(1000);

    //set pin output signal to LOW - LED is not working
    digitalWrite(ledPin,LOW);
    //delay of 1000 milliseconds
    delay(1000);
}
```

Displays

Using *display* is a quick way to get a feedback information from the device. There are many display technologies compatible with Arduino. For IoT solutions low power, easy to use and monochrome displays are used:

- *Liquid-crystal display (LCD)*,
- *Organic light-emitting diode display (OLED)*,
- *Electronic ink display (E ink)*.

Liquid-crystal display (LCD)

LCD uses modulating properties of liquid crystal light to block the incoming light. Thus when a voltage is applied to a pixel, it has a dark colour. A display consists of layers of electrodes, polarised filters, liquid crystals and reflector or back-light. Liquid crystals do not emit the light directly; they do it through reflection or backlight. Because of this reason, they are more energy efficient. Small, monochrome LCDs are widely used in devices to show a little numerical or textual information like temperature, time, device status etc. LCD modules commonly come with an onboard control circuit and are controlled through parallel or serial interface.

4. IoT Hardware overview



Figure 69: Blue 16×2 LCD display

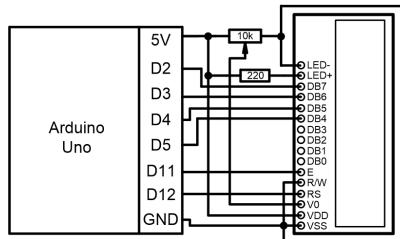


Figure 70: Arduino and LCD screen schematics

The example code:

```
#include <LiquidCrystal.h> //include LCD library

//define LCD pins
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
//create and LCD object with predefined pins
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  lcd.begin(16, 2); // set up the LCD's number of columns and rows:
  lcd.print("hello, world!"); // Print a message to the LCD.
}

void loop() {
  // set the cursor to column 0, line 1 - line 1 is the second row,
  // since counting begins with 0
  lcd.setCursor(0, 1);
  // print the number of seconds since reset
  lcd.print(millis() / 1000);
}
```

Organic light-emitting diode display (OLED)

OLED display uses electroluminescent materials that emit light when the current passes through these materials. The display consists of two electrodes and a layer of organic compound. OLED displays are thinner than LCDs, they have higher contrast, and they can be more energy efficient depending on usage. OLED displays are commonly used in mobile devices like smartwatches, cell phones and they are replacing LCDs in other devices. Small OLED display modules usually have an onboard control circuit that uses digital interfaces like I2C or SPI.



Figure 71: OLED I2C display

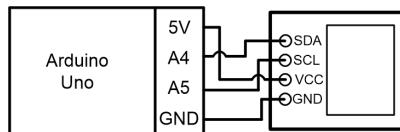


Figure 72: Arduino and OLED I2C schematics

```
//add libraries to ensure the functioning of OLED
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

void setup() {
    //Setting up initial OLED parameters
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
    display.setTextSize(1); //Size of the text
    display.setTextColor(WHITE); //Colour of the text - white

void loop() {

    // Uz ekrāna izvada sesnoru vērtības
    display.setCursor(0, 0);
    display.clearDisplay();
    display.print("Test of the OLED"); //print out the text on the OLED
    display.display();
    delay(100);
    display.clearDisplay();
}
```

Electronic ink display (E-ink)

E-ink display uses charged particles to create a paper-like effect. The display consists of transparent microcapsules filled with oppositely charged white and black particles between electrodes. Charged particles change their location, depending on the orientation of the electric field, thus individual pixels can be either black or white. The image does not need the power to persist on the screen; power is used only when the image is changed. Thus e-ink display is very energy efficient. It has high contrast and viewing angle, but it has a low refresh rate. E-ink displays are commonly used in e-riders, smartwatches, outdoor signs, electronic shelf labels.

4. IoT Hardware overview



Figure 73: E ink display module

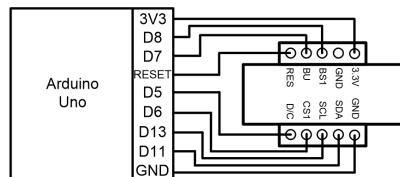


Figure 74: Arduino Uno and E ink display module schematics

```
#include <SmartEink.h>
#include <SPI.h>

E_ink Eink;

void setup()
{
    //BS LOW for 4 line SPI
    pinMode(8,OUTPUT);
    digitalWrite(8, LOW);

    Eink.InitEink();

    Eink.ClearScreen(); // clear the screen

    Eink.EinkP8x16Str(14,8,"NOA-Labs.com");
    Eink.EinkP8x16Str(10,8,"smart-prototyping.com");
    Eink.EinkP8x16Str(6,8,"0123456789");
    Eink.EinkP8x16Str(2,8,"ABCDEFG abcdefg");

    Eink.RefreshScreen();
}

void loop()
{}
```

Mechanical drivers

Relay

Relays are electromechanical devices that use electromagnets to connect or disconnect plates of a switch. Relays are used to control high power circuits with low power circuits. Circuits are mechanically isolated and thus protect logic control. Relays are used in household appliance automation, lighting and climate control.



Figure 75: 1 channel relay module

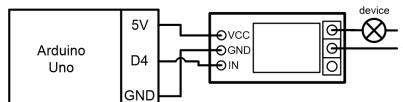


Figure 76: Arduino Uno and 1 channel relay module schematics

The example code:

```
#define relayPin 4 //define the relay pin

void setup()
{
    Serial.begin(9600);
    pinMode(relayPin, OUTPUT); //set relayPin to output
}

void loop()
{
    digitalWrite(relayPin,0); //turn relay on
    Serial.println("Relay ON"); //output text
    delay(2000); // Wait 2 seconds

    digitalWrite(relayPin,1); //turns relay off
    Serial.println("Relay OFF");
    delay(2000);
}
```

Solenoid

Solenoids are devices that use electromagnets to pull or push iron or steel core. They are used as linear actuators for locking mechanisms indoors, pneumatic and hydraulic valves and in-car starter systems.

Solenoids and relays both use electromagnets and connecting them to Arduino is very similar. Coils need a lot of power, and they are usually attached to the power source of the circuit. Turning the power of the coil off makes the electromagnetic field to collapse and creates very high voltage. For the semiconductor devices protection, a shunt diode is used to channel the overvoltage. For the extra safety, optoisolator can be used.

4. IoT Hardware overview



Figure 77: Solenoid

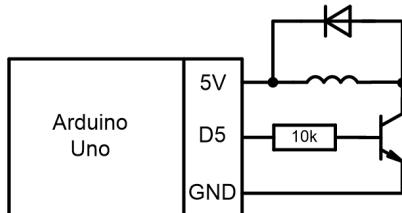


Figure 78: Arduino Uno and solenoid schematics

The example code:

```
#define solenoidPin 4 //define the solenoid pin

void setup()
{
    Serial.begin(9600);
    pinMode(solenoidPin, OUTPUT); //set solenoidPin to output
}

void loop()
{
    digitalWrite(solenoidPin,0); //turn solenoid on
    Serial.println("Solenoid ON"); //output text
    delay(2000); // Wait 2 seconds

    digitalWrite(solenoidPin,1); //turns solenoid off
    Serial.println("Solenoid OFF");
    delay(2000);
}
```

Speaker

Speakers are electroacoustic devices that convert the electrical signal into sound waves. A speaker uses a permanent magnet and a coil attached to the membrane. Sound signal, flowing through the coil, creates the electromagnetic field with variable strength, coil attracts to magnet according to the strength of the field thus making a membrane to vibrate and creating a sound wave. Other widely used speaker technology, called Piezo speaker, uses piezoelectric materials instead of magnets. Speakers are used to creating audible sound for human perception and ultrasonic sound for sensors and measurement equipment.



Figure 79: Speaker 8Ohm 0.5W

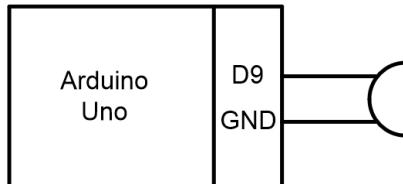


Figure 80: Arduino Uno and piezzo buzzer schematics

```

const int speakerPin = 9; // define the buzzer pin

void setup()
{
    pinMode(speakerPin, OUTPUT); // Set buzzer as an output
}

void loop()
{
    tone(speakerrPin, 1000); // Send 1KHz sound signal
    delay(1000);           // for 1 sec
    noTone(speakerPin);    // Stop sound
    delay(1000);           // for 1sec
}

```

DC motor (one direction)

Electric motor is an electro-technical device which can turn electrical energy into mechanical energy; motor turns because of the electricity that flows in its winding. Electric motors have seen many technical solutions over the year from which the simplest is the permanent-magnet DC motor.

DC motor is a device which converts direct current into a mechanical rotation. DC motor consists of permanent magnets in stator and coils in the rotor. By applying the current to coils, the electromagnetic field is created, and the rotor tries to align itself to the magnetic field. Each coil is connected to a commutator, which in turns supplies coils with current, thus ensuring continuous rotation. DC motors are widely used in power tools, toys, electric cars, robots, etc.



Figure 81: A DC motorwith gearbox 50:1

4. IoT Hardware overview

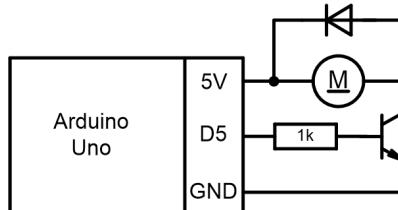


Figure 82: Arduino Uno and DC motor schematics

```
void setup ()
{
    pinMode(5,OUTPUT); //Digital pin 5 is set to output
    //the function for turning on the motor is defined
#define motON digitalWrite(5,HIGH)
    //the function for turning off the motor is defined
#define motOFF digitalWrite(5,LOW)
}
void loop ()
{
    motON; //turn on the motor
}
```

DC Motor with H-bridge

The *H bridge* has earned its name because of its resemblance to the capital 'H' wherein all the corners there are switches and in the middle – the electric motor. This bridge is usually used for operating permanent-magnet DC motor, electromagnets and other similar elements, because it allows operating with significantly bigger current devices, using a small current. By switching the switches, it is possible to change the motor direction. It is important to keep in mind that the switches need to be turned on and off in pairs.

When all of the switches are turned off, the engine is in the free movement. To slow down faster, the H bridge is turned on in the opposite direction.

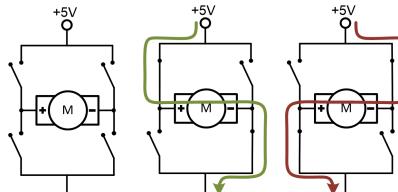


Figure 83: The flow of currents in the H bridge

If both positive or both negative switches are turned on at the top or at the bottom, then the engine stops, not allowing to have a free rotation - it is slowed down. The management can be reflected in the following table:

Table 10: The management of the H bridge switches

| Upper left | Upper right | Lower left | Lower right | Motor work mode |
|------------|-------------|------------|-------------|----------------------------|
| On | Off | Off | On | Turns in one direction |
| Off | On | On | Off | Turns in another direction |
| On | On | Off | Off | Braking |
| Off | Off | On | On | Braking |

When all of the switches are turned off, the engine is in the free movement. Not always it is enough for robotics, so sometimes the H bridge is turned on in the opposite direction to slow the motor down faster - the opposite direction is turned on rapidly.

Remember! Neither of these braking mechanisms is good for the H bridge or the power source. That is why this action is unacceptable without a particular reason because it can damage the switches or the power source.

The complicated part is the realisation of switches - if the switches don't work usually relays or appropriate power transistors are used. The biggest drawback for relays is that they can only turn the engine on or off. If the rotation speed needs to be regulated using the impulse width modulation, then transistors have to be used. MOSFET type transistors should be used for ensuring a large amount of power. Nowadays the stable operation of the bridge is ensured by adding extra elements. The manufactured bridges have one body, for example, the one that is included in the constructor - L293D.

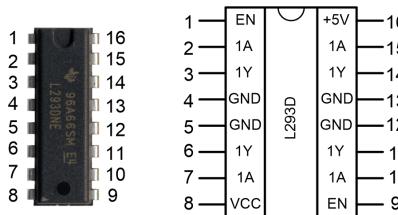


Figure 84: The L293D microchip and its representation in the circuit

The L293D microchip consists of two H bridges and is made for managing two motors. Each pin of the microchip has its function that is why it is very important not to mix them up; otherwise, the microchip can be damaged. All pins of the microchip have assigned a number. The enumeration begins with the special mark on the body: a split, a dot, a cracked edge, etc., and continues counter-clockwise. When creating a scheme, it is important to take into account pin numbers and the ones shown in the scheme. If some additional information about the microchip is necessary, it can be found in the datasheet of the microchip. Remember that the datasheet can be found by writing the number of the device (written on the body) and adding the word "datasheet" in the browser.

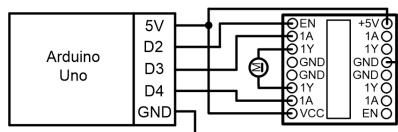


Figure 85: Arduino Uno and L293D H-bridge schematics

The example code:

4. IoT Hardware overview

```
int dirPin1 = 7;      //1st direction pin
int dirPin2 = 8;      //2nd direction pin
int speedPin = 5;    //Pin responsible for the motor speed

void setup ()
{
    pinMode (dirPin1,OUTPUT); //1st direction pin is set to output
    pinMode (dirPin2,OUTPUT); //2nd direction pin is set to output
    pinMode (speedPin,OUTPUT); //Speed pin is set to output
}

void loop ()
{
    analogWrite(speedPin, 100); //setting motor speed
    //speed value can be from 0 to 255

    int motDirection = 1; //motor direction can be either 0 or 1

    if (motDirection) //setting motor direction
    { //if 1
        digitalWrite(dirPin1,HIGH);
        digitalWrite(dirPin2,LOW);
    }
    else
    { //if 0
        digitalWrite(dirPin1,LOW);
        digitalWrite(dirPin2,HIGH);
    }
}
```

Stepper motor

Stepper motors are motors, that can be moved by a certain angle or step. Full rotation of the motor is divided into small, equal steps. Stepper motor has many individually controlled electromagnets, by turning them on or off, a motor shaft rotates by one step. Changing switching speed or direction can precisely control turn angle, direction or full rotation speed. Because of exact control ability, they are used in CNC machines, 3D printers, scanners, hard drives etc. Example of use: <https://learn.adafruit.com/adafruit-arduino-lesson-16-stepper-motors/breadboard-layout>



Figure 86: A stepper motor

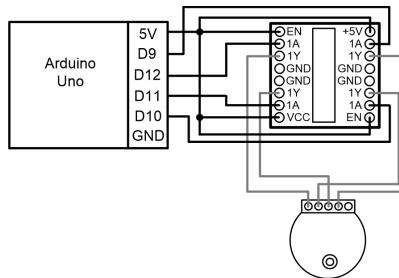


Figure 87: Arduino Uno and stepper motor schematics

The example code:

```
#include <Stepper.h> //include library for stepper motor

int in1Pin = 12; //defining stepper motor pins
int in2Pin = 11;
int in3Pin = 10;
int in4Pin = 9;

//define a stepper motor object
Stepper motor(512, in1Pin, in2Pin, in3Pin, in4Pin);

void setup()
{
    pinMode(in1Pin, OUTPUT); //set stepper motor control pins to output
    pinMode(in2Pin, OUTPUT);
    pinMode(in3Pin, OUTPUT);
    pinMode(in4Pin, OUTPUT);

    Serial.begin(9600);
    motor.setSpeed(20); //set the speed of stepper motor object
}

void loop()
{
    motor.step(5); //rotate 5 steps
}
```

Servomotor

Unlike the simple DC motor, *servomotor* is a particular management chain which allows effortless control over the speed or position of the motor. The management of the engine is realised using three connections – positive (usually red) and negative connection (brown or black) of the current as well as the connection for management (orange or yellow).

For the management, the pulse width technique is used.

4. IoT Hardware overview

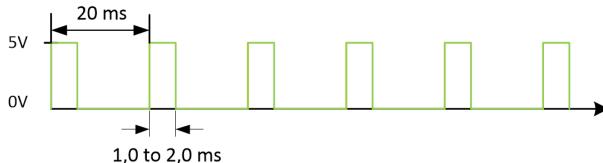


Figure 88: The pulse width modulated signal for the management of servomotor

From the image, it can be seen that the length of the servomotor impulse cycle is 20 milliseconds but the impulse length itself is 1 or 2 milliseconds (ms). These signal characteristics are true for the most enthusiast level servomotors, but it should be verified for each module in the manufacturer specification. Servomotor management chain meets the impulse every 20 ms, but the width of the pulse shows the position that the servomotor has to reach. For example, 1 ms corresponds to the 0-degree position but 2 ms - to 180-degree position against the starting point. When entering the defined position, the servomotor will keep it and resist any outer forces that are trying to change the current position. The graphical representation is in the following image.

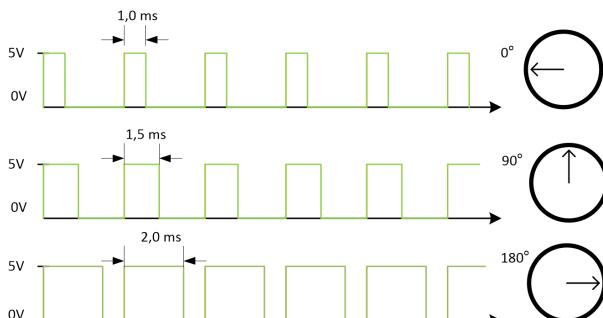


Figure 89: The pulse width modulated signal for different positions of servomotor

Just like other motors, servomotors have different parameters, where the most important one is the time of performance - the time that is necessary to change the position to the defined position. The best enthusiast level servomotors do a 60-degree turn in 0.09 seconds.

There are three types of servomotors:

- *positional rotation servomotor* – most widely used type of servomotor. With the help of a management signal, it can determine the position of the rotation axis from its starting position.
- *continuous rotation servomotor* – this type of motor allows setting the speed and direction of the rotation using the management signal. If the position is less than 90 degrees, it turns in one direction, but if more than 90 degrees it turns in the opposite direction. The speed is determined by the difference in value from 90 degrees; 0 or 180 degrees will turn the motor at its maximum speed while 91 or 89 degrees at its minimum rate.
- *linear servomotor* – with the help of additional transfers it allows moving forward or backward; it doesn't rotate.

Unfortunately using Arduino, the servomotor is not as easily manageable as DC motor. For this purpose, a special servomotor management library *Servo.h* has been created.



Figure 90: A servomotor

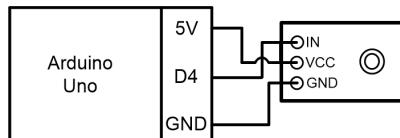


Figure 91: Arduino Uno and servomotor schematics

The example code:

```
#include <Servo.h> //include Servo library
Servo servo; //define a Servo object

void setup ()
{
    servo.attach(6); //connect servo object to pin D6
    servoLeft.write(90); //set position of servo to 90 degrees
    Serial.begin(9600);
}

void loop ()
{
    servoLeft.write(110); //set position of servo to 110 degrees
    delay(200); //wait for 200 ms
    servoLeft.write(70); //set position of servo to 70 degrees
    delay(200); //wait for 200 ms
}
```

4.3.12. IoT application example projects

Many IoT projects developed using an Arduino board can be found in the official *Arduino Project Hub*⁴⁶⁾. Here are stored multiple projects that are developed by Arduino enthusiasts. In many of the following examples, the Arduino Yun board is used, because it is easy to use a controller that contains the WiFi connection that is necessary for IoT solutions. Additionally, the Amazon services are used for storing and handling the sensor data.

One of the IoT projects available at the Arduino Project Hub is the ***Arduino Home Controller Activated by Alexa***⁴⁷⁾ (*Alexa* is the *Amazon Echo dot*⁴⁸⁾) that uses an Arduino Yun controller. In this solution, it is possible to control internet-connected devices around the home using voice, using Alexa and Amazon Web Service (AWS). Natural language voice commands are interpreted using Amazon Skill and a Lambda Function in the AWS. The developed system gives an opportunity to control four lights

4. IoT Hardware overview

installed in the room, garage, kitchen and living room, temperature and humidity, buzzer alarm and WebCam that takes a security photo and sends it by e-mail.

The **Home Security Model**⁴⁹⁾ is another example to monitor the security status in the real-time using sensors and internet connection. The system data can be accessed and controlled remotely using a smart device or a PC. In this project, the Arduino Yun and Arduino Mega controllers are used. The AWS IoT services are used for connecting the devices around the home. The system includes temperature and humidity, gas, water, vibration, current and ultrasonic sensors and it controls the light in multiple rooms and a buzzer.

The project **Plant Monitoring System uses AWS IoT**⁵⁰⁾ and gives the opportunity to track the temperature and soil moisture of the plant using AWS IoT. Using the Arduino Yun controller, the system monitors such parameters as the temperature, soil moisture and light value. These log data are stored in the AWS IoT service. Additionally, the *Amazon Simple Notification Service* (SNS) is used for sending notification informative e-mails about the status of the sensor measurements.

Here is another IoT project about the home automation system, called **Arduino based Amazon Echo using 1Sheeld** ⁵¹⁾. *1Sheeld* is a hardware shield that allows using sensors of the smartphone as the inputs for Arduino. In this project, a smartphone is used to play music (*Music Player Shield*), interact with the person by responding to questions (*Text to Speech Shield*), return the real-time (*Clock Shield*), return the information about the weather (*Internet Shield*) and it is controlled by voice commands (*Voice Recognition Shield*).

Few of the Arduino IoT related projects are provided also on the *hackster.io* website⁵²⁾.

The first project that is viewed is the **Harry Potter Weasleys' Clock using Bolt IoT**⁵³⁾. The idea of this project is taken from the Harry Potter movie where the wall clock indicated the location of family members. In this modern IoT project, using Arduino Uno microcontroller, servomotor and Bolt WiFi Module, the clock that using arrow represents the location of the person that has a smartphone has been developed. The clock and the tracked smartphone are connected to the Bolt cloud server.

54)

4.3.13. Setting up development/experimenting environment

Using breadboard

The breadboard is a mechanical base for electronics prototyping. Originally for this purpose was used the wooden board for cutting bread, that is why the name used now is the "breadboard", also known as a solderless breadboard. It is because an electrical connection on this board can be made without soldering. Thus components can be reused, and changes to the circuit can be made easily.

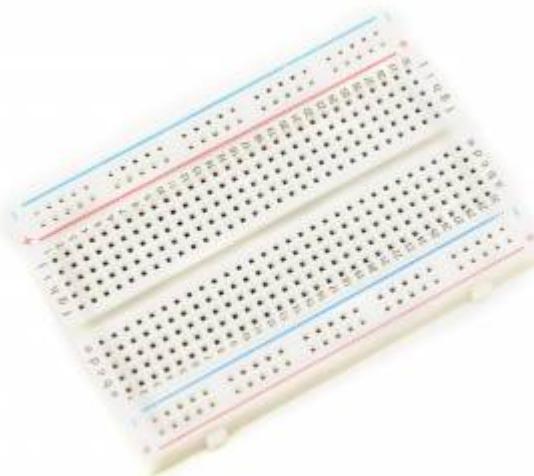


Figure 92: 400 point Breadboard

On the front surface of the breadboard, many small holes are connected on the back of the board in a specific pattern using metal clips. These clips hold component leg or wire in place when they are put into the hole.

As the example, the circuit that contains the LED and resistance is taken.

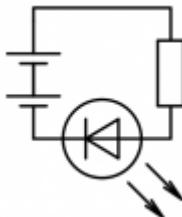
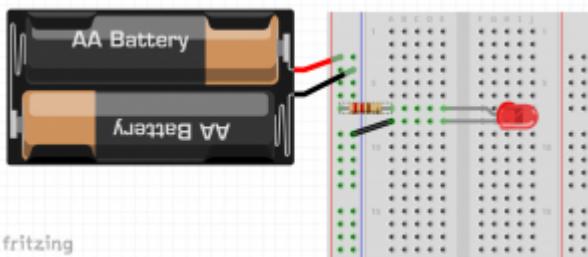


Figure 93: The example schematics

Following the schematics, all the necessary components are connected in the right way using the breadboard.



4. IoT Hardware overview

Figure 94: Example circuit connected using breadboard

Two side columns of the breadboard are made easily accessible from the rest of the breadboard and are commonly used for connecting the power to the circuit. Almost any DC (direct current) power source can be used to power the circuit, like batteries, Arduino board power pins, AC/DC regulators etc.

Two columns of 5 hole rows are used for connecting components. Extra connections can be made using wires. The gap in the middle allows using DIP (dual in-line package) circuits.

More information on breadboards can be found in the SparkFun webpage ⁵⁵⁾.

Soldering

Soldering is one of the essential skills in the world of electronics. The basic of electronics can also be learned without the knowledge of how to solder, however, the soldering allows to work on more exciting projects and to join a wide range of electronics enthusiasts. This skill is essential because nowadays the electronic and electrical equipment is being used more often. The essential element of the knowledge about electronics is not only the ability to understand the working principles of the electronics but also the skill to build, repair and supplement the electronic devices.

Materials

The main soldering material is a *solder*. As the name of the material indicates, it is a compound of various soft metals and consumable materials, which is usually similar to a wire, wrapped in a reel or other easy-to-use packaging.



Figure 95: Different packaging of solder

Different types of solder vary in *diameter* of the wire and the *chemical composition*. The type of solder that is used depends on the task.

According to the *chemical composition*, the solder is divided into two types: solder containing lead and lead-free solder. Historically, lead (Pb) is used in combination with tin (Sn) to ensure lower melting temperatures and better flow, which is essential for good contact between the parts.

Since 2006, many countries have forbidden using lead-containing solder, caring for the protection of nature and human health. When lead accumulates in the human body in significant amounts, it can cause poisoning. For this reason, it is important to remember – after using lead-containing solder, you should **carefully wash your hands**.

To avoid the risk of getting lead in the human body, the alternative is to use lead-free solder. Nevertheless few things need to be taken into account - the melting temperature

of lead-free solder is higher and grip with other materials is lower. For improving the grip, flux can be used. Some solders already have flux in the core of the wire, so it is not necessary to buy them separately.

The *diameter* of the solder wire depends on the size of details that need to be soldered. The bigger the detail, the bigger the diameter of the solder wire.

Tools

Tools used for soldering are a soldering iron, stand for soldering iron, as well as different tools for removing solder and keeping parts together.

A **soldering iron** is an electrical heater that is used for heating the solder to its melting temperature. As with all of the electrical devices, the instructions provided by the manufacturer should be followed when using the soldering iron. For specific tasks, there are also hot air and gas soldering irons. In this section, only the electric ones will be described.

There are different types of soldering irons so that they can be effectively used for diverse tasks. Soldering irons can vary in a shape of the tip, electrical capacity, heating temperature and control options. However, the primary indicator is the convenience of use. If the chosen soldering iron is too big, it will be difficult to solder small details. If it is too small, details probably will not get enough of the heat, and the solder will not stick properly to them. For beginners, it is advisable to use a soldering iron with a conical tip.



Figure 96: A soldering iron with the conical tip

In addition to the soldering iron, there are several useful tools for soldering:

- **Solder wick** - used for desoldering, removes the extra solder out of the board or details, it absorbs the melted solder. The solder wick consists of multiple fine interlaced lead threads.



Figure 97: Solder wick

- **Solder vacuum** - used for desoldering, removes the extra solder utilizing a vacuum.



Figure 98: Solder vacuum

- **Third-hand** - a holder of details or other parts. It is especially useful for beginners.



Figure 99: Third hand with the magnifying glass

Process of soldering

It is not easy to describe the process of soldering in words, because it is connected with the kinesthetic skills of people and the paid attention. However, it is essential to follow several pieces of advice for the good soldering:

1. Be careful with the hot soldering iron!
2. The working place should always be clean; it is advisable not to take a meal at the working place (because of the poisonous nature of the lead in the solder).

3. It is advisable to use the third hand when it is possible.
4. If possible, the temperature of the soldering iron should always be around 350°C.
5. If there is smoke coming from the soldering iron, the temperature should be decreased, or it should be turned off completely.
6. Before soldering, a special soldering iron cleaner (wet sponge or special paste for cleaning the tip) should be used.
7. The side of the soldering iron tip should be used when soldering rather than the very tip of it.
8. The good contact can be ensured when heating simultaneously both components that are soldered.
9. When the detail has been soldered, first the solder wire should be taken off and only then - the soldering iron.
10. Good soldering of the detail with the board looks like a tip of volcano rather than a ball or messy pile of solder.

In the following image, the correct and incorrect soldering techniques are indicated.

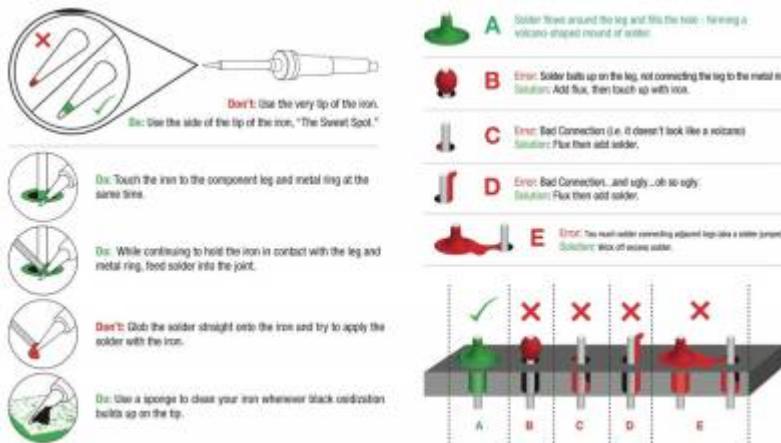


Figure 100: A visualization of the soldering technique ⁵⁶⁾

4.3.14. Arduino programming fundamentals

The following sub-chapters cover programming fundamentals in Arduino(C) C/C++, which complies with the most C/C++ notations and have some specific Arduino notations. Those who feel comfortable in programming will find these chapter somewhat introductory, while for those having no or small experience, it highly recommended covering this introduction. This chapter and its sub-chapters also target ESP and Raspberry Pi devices on the general level partially, however, in any case, programming environment configuration is different for every platform even, if Arduino IDE constitutes the joint part. Refer to the chapters on ESP and Raspberry Pi hardware platforms.

This manual refers to the particular version of the software (here Arduino IDE and related toolkits) available at the moment of writing of this book, thus accessing specific features

4. IoT Hardware overview

may change over time along with the evolution of the platform. Please refer attached documentation (if any) and browse Internet resources to find latest guidances on how to configure development platform, when in doubt.

Setting up the programming environment

The syntax and the structure of the program

Data types and variables

Program control structures

Looping

Interrupts and sub-programs

Timing

Reading GPIOs, outputting data and tracing

Setting up the programming environment

Before starting programming the microcontroller, it is necessary to connect it to the computer.

Connection

Arduino Uno microcontroller is taken as a board for programming example tasks. It can be connected to a computer, using *Universal Serial Bus* (USB) port, using the appropriate USB cable. A microcontroller can be used together with a prototyping board or a robot. In the simplest programming tasks, it can be used as an independent device.

Power

The microcontroller has to be powered via an external power supply or USB port. The microcontroller determines the power source automatically. If external power supplies other than USB are used, GND and VIN ports should be used to connect the power supply. The manufacturer recommends the use of a voltage of 7-12 V to ensure a normal operation of the microcontroller. If the voltage is exceeded, before reaching 20 V, then the power supply circuits of the microcontroller may get overheated. If the supply voltage is lower than 7 V, then the microcontroller may function unstable, and the result will be unpredictable.

In addition to the above mentioned, the microcontroller can provide a small power supply for external circuits by connecting them according to the microcontroller pins.

Table 11: The power pins of Arduino UNO.

| Pin | Description |
|-----|--|
| VIN | The input of a power supply when USB port is not used, i.e., an external power supply is used. |
| 5V | A regulated 5V power supply, which can be provided via both USB and VIN. |

| Pin | Description |
|------|---|
| 3.3V | A 3.3V supply voltage for external circuits. The maximum current that this output can provide is 50 mA. If it is exceeded, the power supply circuits of the microcontroller may be permanently damaged. |
| GND | Ground or port 0. |

Inputs/Outputs

Each of the 14 digital inputs/outputs (I/O) of the microcontroller can be used to send or receive signals using the `pinMode()`, `digitalWrite()` and `digitalRead()` commands, which will be more detailed discussed in the chapter about the basics of programming. All I/O operate in the range of 0V to 5V. Each of the I/O is capable of receiving or sending no more than 40 mA of current. They all have internal load resistors in the range of 20-50kΩ.

Descriptions of other microcontroller pin and their specific use are explained below. In addition to these I/O, the microcontroller also provides other specific functions that will be described below.

Table 12: Specific I/O pins of Arduino UNO.

| Pin | Description |
|--|--|
| O(RX) and 1(TX) | <i>Serial I/O</i> for serial communication. RX is used for receiving data, and TX for sending data to external devices. For data transmitting and receiving, the voltage must not exceed 5 V. |
| 2 and 3 | <i>External interrupt</i> pins that can be used to receive an external interrupt in cases when the value is low, the value is changed, etc. For this functionality the function <code>attachInterrupt()</code> is used. |
| PWM: 3, 5, 6, 9, 10, 11 | <i>Pulse Width Modulation (PWM)</i> pins are used to provide 8-bit PWM signal that often can be used for motor control or other specific use cases. For this functionality the <code>analogWrite()</code> function is used. |
| SPI: 10(SS), 11(MOSI), 12(MISO), 13(SCK) | Pins that support <i>Serial Peripheral Interface (SPI)</i> communications. For this feature, the SPI library is used. |
| LED: 13 | This pin is used to manage the built-in LED. LED can be turned on by setting the value of pin HIGH and turned off by setting pin value LOW. |
| A4(SDA) and A5(SCL) | <i>Two Wire Interface (TWI)</i> pins, <i>Serial Data Line (SDA)</i> and <i>Serial Clock Line (SCL)</i> , are the alternative of the data exchange using serial communication. For supporting TWI, the Wire library should be used. |
| AREF | It is the reference voltage for the analogue inputs. For this functionality <code>analogReference()</code> is used. |
| Reset | Gives the opportunity to reset the microcontroller by setting this pin to LOW. |

4. IoT Hardware overview

Installing the programming environment

To start the development of software for a microcontroller, it is necessary to install and properly configure the development environment that consists of the program editor and the Arduino UNO driver. Below are described all the steps that are needed to prepare the programming environment for Windows 10 OS.

Step 1. Preparing Arduino UNO and the USB cable

Before installing the programming environment, it is necessary to prepare the Arduino UNO board and the USB cable for connecting the board to the computer.



Figure 101: The Arduino UNO board



Figure 102: USB B cable for Arduino UNO

Step 2. Downloading the Arduino software development environment

The open-source *Arduino Software* (Integrated development environment (IDE)) can be found in the official Arduino website ⁵⁷⁾. The appropriate installation file depends on the OS of the computer and the access rights of the user.

Download the Arduino IDE

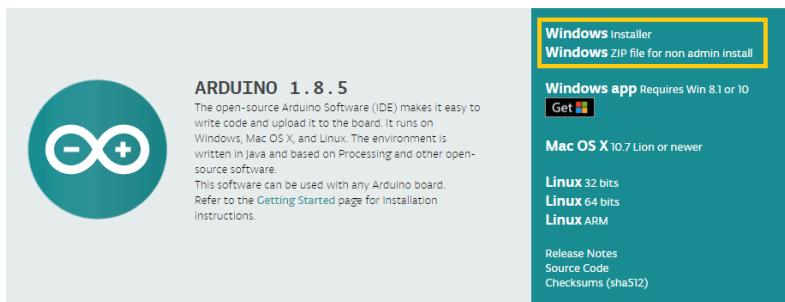


Figure 103: Downloading the installation file for Windows OS from Arduino original website.

For Windows OS, the "*Windows Installer*" should be clicked and then the file should be saved on the computer. When the installation file has downloaded, the file should be run. If the ZIP file was downloaded, it is necessary to unarchive it and to run the installer. Follow the instructions of the installer. If the operating system asks for permission to install the driver of the board - allow it.

It is also possible to use Arduino Web Editor (can be found on the same website) to work online with the Arduino board, but this option will not be considered in this manual.



Figure 104: Arduino Web Editor

Step 3. Connecting to Arduino

Using USB cable, Arduino needs to be connected to a free USB port of a computer. The blue LED on the Arduino board starts to shine continuously. Aforementioned is the indicator that the Arduino board is working.

The green LED will blink, and that will indicate to the performance of the manufacturer test software. In case if the green LED is not flashing, it is not an error.

Step 4. Starting up the programming environment

The Arduino programming environment can be started with the double-click on the desktop shortcut of the Arduino software. The language of the environment will respond to the one that is set up in the OS of the computer, that means if it is English, then the

4. IoT Hardware overview

menu of the programming environment will also be in the English language. To change the language preferences, it is necessary to follow the instructions in the following webpage ⁵⁸⁾.

Step 5. Open the example program

In the Arduino IDE open *File*→*Examples*→*01.Basics*→*Blink* as shown in the image below.

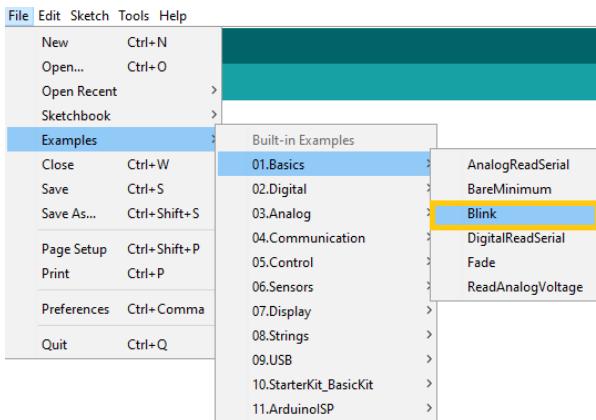
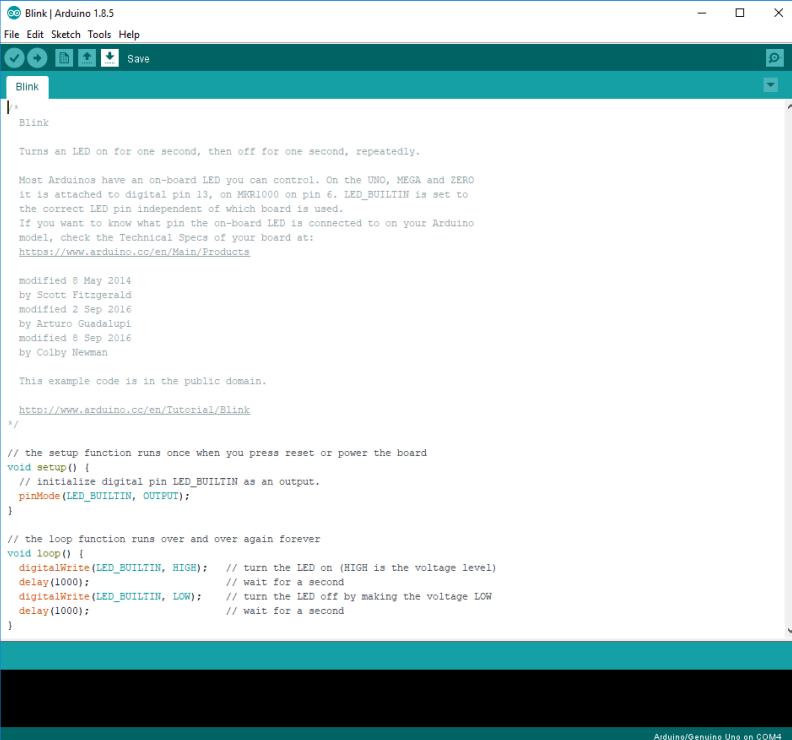


Figure 105: The path to open "Blink" example in the Arduino IDE

This will open in the new window an example program for turning on and off green LED that is situated on the Arduino UNO board with the 1 second delay.



The screenshot shows the Arduino IDE interface with the 'Blink' sketch open. The code is as follows:

```
File Edit Sketch Tools Help
Blink Save
Blink
/*
  Blink
  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
 */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}

```

Arduino/Genuino Uno on COM4

Figure 106: The example Blink program.

Step 6. Choosing the microcontroller

In this step it is necessary to choose the type of board that is used. In this example the Arduino UNO board is used that is why in the menu of Arduino IDE choose *Tools→Board→Arduino/Genuino Uno* as shown in the image below.

4. IoT Hardware overview

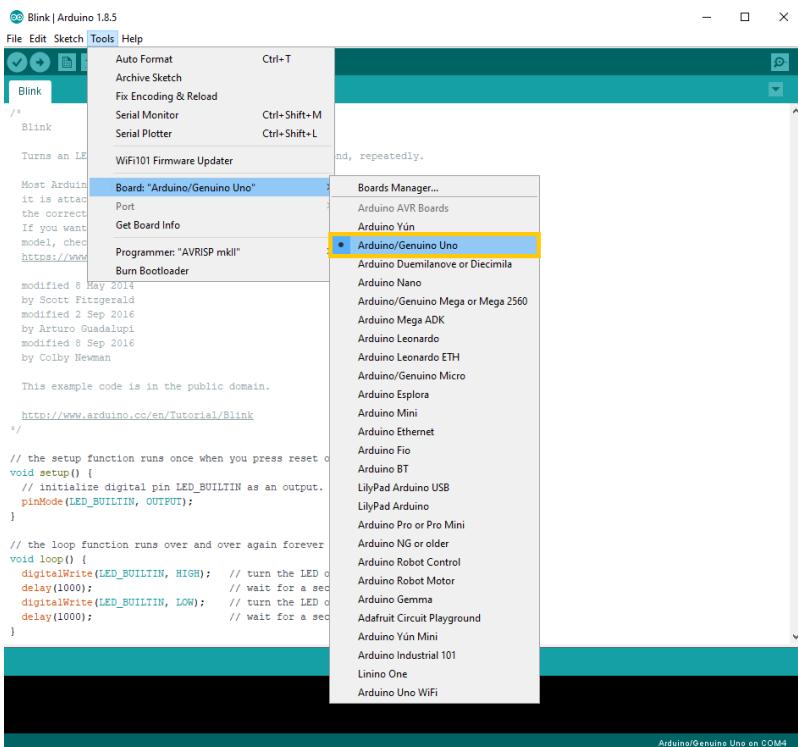


Figure 107: The path to choose the type of board.

Step 7. Setting up COM port

To ensure transmitting and receiving data to/from the microcontroller, it is necessary to set the serial communication port - COM port. All ports are numbered in order, and for Arduino microcontroller, it is usually higher than COM3, i.e. COM4, COM5, etc. In the image below it is COM4.

4.3. Arduino Overview

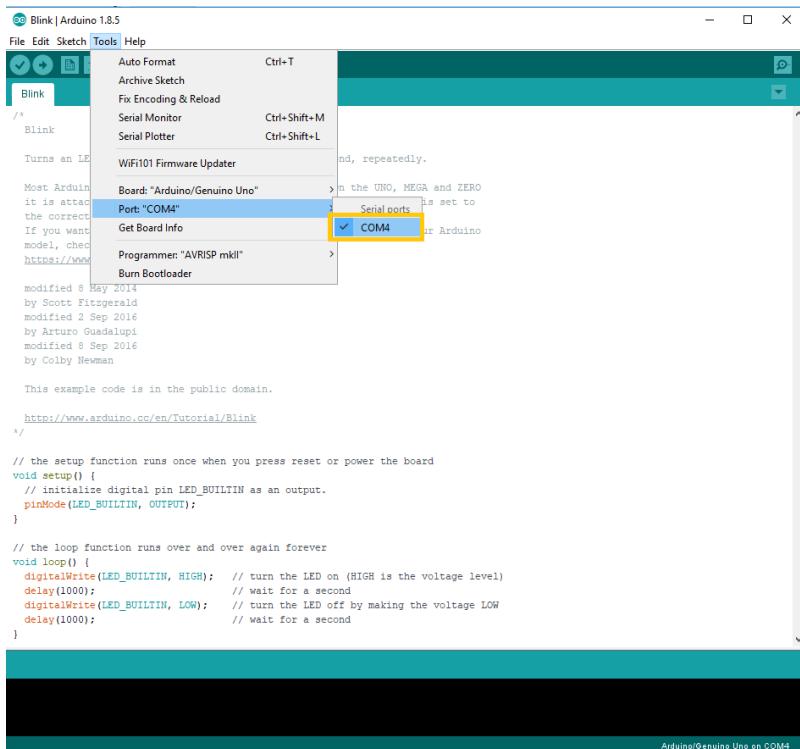


Figure 108: The path to choose the port for Arduino connection.

Step 8. Uploading the example program to the board

Now the program can be uploaded to the Arduino board using the *Upload* button in the top left corner of the software, then wait for a few seconds, during which you can see the data sending indicators - LEDs are blinking fast (indicates sending or receiving data) - and wait for the message to be "*Upload is complete*".

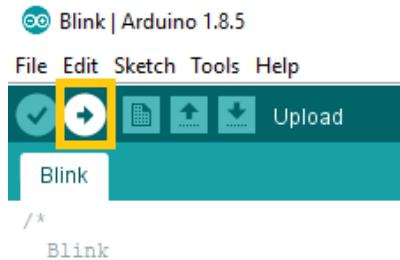


Figure 109: Uploading program to the board.

After a few seconds, the green LED will blink with a one-second interval like it is written

4. IoT Hardware overview

in the source code. If this can be observed successfully then everything is done to start learning the basics of programming.

In case if the blinking green LED cannot be observed, instructions for troubleshooting can be read in the following link ⁵⁹⁾.

If you want to get acquainted yourself with microcontroller capabilities or programming basics independently, look at one of these sources of information:

- Examples for the accomplishing tasks of different level of difficulty ⁶⁰⁾.
- Reference for the programming language used ⁶¹⁾.

Check yourself

1. What power supply Arduino UNO microcontroller requires?
2. How to operate with inputs/outputs of the microcontroller?
3. Try different examples in the menu of Arduino IDE.

The syntax and the structure of the program

Syntax

Arduino IDE is a software that allows writing Arduino code. Each file with Arduino code is called *a sketch*. The Arduino programming language is similar to the C++ language. For the Arduino IDE to compile the written code without errors, it is important to follow the pre-defined syntax.

Define

`"#define"` is a component that allows giving a name to a constant value at the very beginning of the program.

```
#define constant 4
```

In this example, the value 4 is assigned to the *constant*.

Note that at the end of this expression semicolon (;) is not necessary and between the name and the value, the sign of equality (=) should not be added!

Include

`"#include"` is a component that allows to include libraries from the outside of the program. Just like the `#define`, `#include` is not terminated with the semicolon at the end of the line!

```
#include <Servo.h>
```

In this example, the library called *“Servo.h”* that manages servomotors has been added to the sketch.

Comments

There are two ways to write comments in the sketch so that the written text is not compiled as a part of the running code.

```
//Single line comment is written here
```

The double backslash is used when the only single line should be commented.

```
/*Multi-line comments are written here
Second line of the comment
...
*/
```

In this way, the backslash followed by asterisk defines the beginning of the block comment, and the asterisk followed by backslash defines the end of the block comment.

Semicolon

The semicolon (;) is used at the end of each statement of the code.

```
int pin = 5;
```

In this example, a single statement is *int pin = 5* and the semicolon at the end tells the compiler that this is the end of the statement and it can continue with the next one.

Curly braces

Curly braces are used to enclose a further block of instructions. Curly braces usually follow different functions that will be viewed in the further sections.

Each opening curly brace should always be by a closing curly brace. Otherwise, the compiler will show an error.

```
void function(datatype argument) {
    statements(s)
}
```

Use of curly braces in the own defined function.

Structure of the program

Below is given the example, how a new empty sketch looks like.

4. IoT Hardware overview



The screenshot shows the Arduino IDE interface. At the top, there are several icons: a checkmark, a circular arrow, a file folder, an upload arrow, a download arrow, and a settings gear. Below the icons is a toolbar with a magnifying glass icon. The main area is titled "test\$". It contains the following code:

```
void setup() {  
    // put your setup code here, to run once:  
  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
}
```

Figure 110: The empty sketch in the Arduino IDE

Each Arduino sketch contains multiple parts:

1. Global definition section - the section where variables and constants are defined that the working range is in the whole sketch that means both in the initialization and the loop sections. This section is at the very beginning of the sketch, before the setup function.
2. Initialization section - is executed only once before running the basic program. In this part usually, all variables, I/O of the board pins, constants, are defined, etc. The most essential is to define all inputs and outputs that will be used in the program that defines how each pin will be used.

The construction of the setup function:

```
void setup()      //The result data type and the name of the function  
{               //Beginning of the initialization function  
    //The body of the function - contains all executable statements  
}               //The end of the initialization function
```

As it was already mentioned, this function will execute only once. The function is described by the result data type (number, symbol array or something else). In this example, the keyword *setup* means that the setup function does not have the result that means it is executed only once and that is all. The name necessarily should be *setup*, so that the built-in subsystem of the board program execution could differ the initialisation section from the rest of the code.

1. Loop section - the part that is executed continuously, it reads inputs, processes data and gives output signals. After running the last statement of the function, the program continues again with the first statement of the same loop function. This is the main part of the program execution that encloses all the steps of program execution, including the logic.

The construction of the loop function is the following:

```
void loop()      //The result data type and the name of the function
{
    //Logic      //The body of the function - contains all executable statements
}                //The end of the loop function
```

The result data type of this function is the same as previous - *void* - that shows that the function does not have the result, it will be executed in the loop continuously while the program is working.

Blink LED

The code of the *Blink LED* program code will be viewed now. The example can be opened by following the path in Arduino IDE: *File→Examples→01.Basics>Blink*.

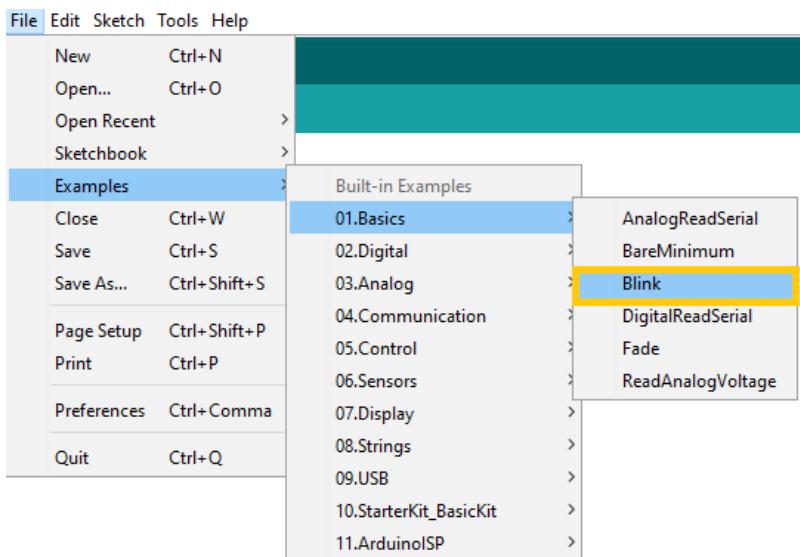
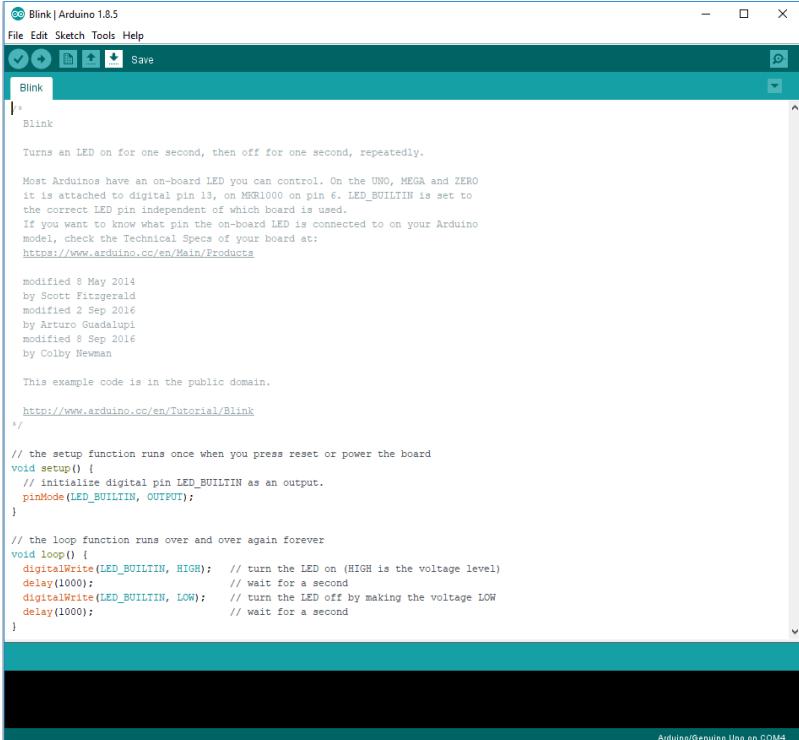


Figure 111: The path to open the *Blink LED* example program

When the *Blink LED* example program is opened, the following sketch should open in the programming environment:

4. IoT Hardware overview



The screenshot shows the Arduino IDE interface with the 'Blink' sketch open. The code is displayed in the main window, and the status bar at the bottom right indicates 'Arduino/Genuino Uno on COM4'. The code itself is the classic 'Blink' example, which turns an LED on for one second and off for one second, repeatedly.

```
/*
  Blink | Arduino 1.8.5
  File Edit Sketch Tools Help
  ✓ Save
  Blink
  */
  Blink
  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
  */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

Figure 112: The *Blink* LED example program

The code of the example program is the following:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  // LED_BUILTIN stands for the built-in LED on the board.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

In the source code of the program following things can be seen:

1. It defined that the `LED_BUILTIN` is set to be the output of the program. In this example sketch, the output periodically sends the specific signal that is in the level of the logical 1 (+5V) and 0 (0V). Sending the output signal to the built-in LED: the

LED is periodically turned on and off.

2. Continuous executable function *loop()* is created that allocates 1 second (1000 ms) of time to each level of signal. It is done by pausing the execution of the program. While the program is not changing the states of the inputs/outputs, they remain unchanged. In this way, when the +5 V signal is sent to the LED output, and the program execution is paused, the LED will continue to shine until the level of the output will be set to 0 V.
3. The last row indicates that the program will be paused for a 1 second also when the output level is set to be 0 V. In this way the period of LED on and off are equal. After executing this program, the program returns to the first line of the *loop()* function and the execution starts from the beginning.

Hello World

"Hello World" program is the simplest program because it simply outputs the text to the screen. Here is the *Hello World* program for Arduino that outputs the text on the Serial Monitor each second:

```
void setup() {  
    Serial.begin(9600); //establishes the connection with the serial port  
}  
  
void loop() {  
    Serial.println("Hello World"); //prints out the line with the text  
    delay(1000); //pause for 1 second  
}
```

Serial Monitor can be found following the path: *Tools*→*Serial Monitor*.

In the code can be seen that the *setup()* function contains the following command:

```
Serial.begin(9600);
```

This statement opens the serial port at the initialisation of the program so that the Serial Monitor can be used for outputting text or values on the screen.

For printing out text the following command is used:

```
Serial.println("Hello World");
```

Check yourself

1. How to attach any library to a sketch?
2. What command are expressions not usually separated by the semicolon?
3. How to establish serial communication between devices?
4. How does *delay()* command works?

*Stops LED blinking specified number of milliseconds

*Stops program execution for a specified number of seconds

4. IoT Hardware overview

*Stops program execution for a specified number of milliseconds

Data types and variables

Data types

Each variable has its data type that determines its place in the memory of the microcontroller and also the way how it can be used. There are plenty of different data types. Further will be viewed the most used ones:

- **byte** - numeric type of 8 bits that stores numbers from 0 to 255.

```
byte exampleVariable = 123;
```

- **int** - the whole number of 16 bits that can contain values from -32 767 to 32 768.

```
int exampleVariable = 12300;
```

- **float** - a data type for real numbers that uses 32 bits and store numbers approximately from -3.4E38 to 3.4E38.

```
float exampleVariable = 12300.546;
```

- **array** - a set of the same data type that can be accessed using serial number or index. The index of the first element is always 0. The values of an array can be initialised at the definition of to do it during the execution of the program. In the following example the array with the name “*first array*” and data type *int* has been created. The value of the array with the index 0 will be 12 and the value with the index 3 is 15.

```
int firstArray[] = {12,-3,8,15};
```

Square brackets of the array can be used to access some value in the array by index. In the following example the element with the index 1 (that is -3) is assigned to the *secondVariable* variable.

```
int secondVariable = firstArray[1];
```

Array can be easily processed in the cycle. The next example shows how to store automatically the necessary values from the analogue signal input to the previously defined array.

```
//the cycle that repeats 4 times
for(int i = 0; i < 4; i = i + 1){
    //reads value from the pin 2 and stores it in the //exampleArray//.
    exampleArray[i] = analogRead(2);
}
```

This cycle in the example starts with the index 0 ($i=0$), and it increases by 1 while it is smaller than 4 (not including). That means in the last cycle the index value will be 3, because when the i equals 4, the inequality $i < 4$ is not true, and the cycle stops working.

- **bool** - the variables of this data type can take values *true* or *false*. Arduino environment allows following values to these variables: **TRUE**, **FALSE**, **HIGH** (logical 1 (+5V)) and **LOW** (logical 0 (0V)).

Data type conversion

Data type conversion can be done using multiple techniques - casting or data type conversion using specific functions.

- **Casting** - cast operator translates one data type into another straightforward. The desired type of variable should be written in the brackets before the variable data type of which needs to be changed. In the following example where the variable type is changed from float to int, the value is not rounded but truncated. Casting can be done to any variable type.

```
int i;
float f=4.7;

i = (int) f; // now i is 4
```

- **Converting** - *byte()*, *char()*, *int()*, *long()*, *word()*, *float()* functions are used to convert any type of variable to the specified data type.

```
int i = int(123.45); //the result will be 123
```

- **Converting String to float** - function *toFloat()* converts *String* type of variable to the *float*. In the following example is shown the use of this function. In case if the value cannot be converted, because the *String* doesn't start with a digit, the returned value will be 0.

```
String string = "123fkm";
float f = string.toFloat(); // the result will be 123.00
```

- **Converting String to Int** - function *toInt()* converts *String* type of variable to the *Int*. In the following example is shown the use of this function.

```
String string = "123fkm";
int i = string.toInt(); //the result will be 123
```

Arithmetic operators

Arithmetic operations on Arduino are used to do mathematical calculations with the numbers or numerical variables. The arithmetic operators are the following:

- **Addition (+)** - one of the four primary arithmetic operations that are used to add numbers. Addition operator can be used to add only numbers, only numeric variables or the mix of both. The following example shows the use of the addition operator.

```
int result = 1 + 2; //the result of the addition operation will be 3
```

- **Subtraction (-)** - the operation that subtracts one number from another where result is the difference between these numbers.

```
int result = 3 - 2; //the result of the subtraction operation will be 1
```

4. IoT Hardware overview

- **Multiplication (*)** - the operation that multiplies numbers and gives the result.

```
int result = 2 * 3; //the result of the multiplication operation will be 6
```

- **Division (/)** - the operation that divides one number by another. If the result variable has the *integer* type, the result will always be the whole part of the division result without the fraction behind it. If the precise division is necessary, it is important to use *float* type of variable for this purpose.

```
//the result of the division operation will be 3  
//(only the whole part of the division result)  
int result = 7 / 2;  
//the result of the division operation will be 3.5  
float result2 = 7.0 / 2.0;
```

- **Modulo (%)** - the operation that finds the remainder of the division of two numbers.

```
//the result of the modulo operation will be 1,  
//because if 7 is divided by 3, the remaining is 1  
int result = 7 % 3;
```

- **Assignment operator (=)** - the operator that assigns the value on the right to the variable that is on the left of the assignment operator. The work of an assignment operator can be seen in any of the previously viewed operation examples.

Compound operators

Compound operators in Arduino are a short way of writing down the arithmetic operations with variables. All of these operations are done on integer variables. These operands are often used in the loops, when it is necessary to manipulate with the same variable in each iteration of the cycle. The compound operators are following:

- **Increment (++)** - increases the value of integer variable by one.

```
int a = 5;  
a++; //the operation a=a+1; the result will be 6
```

- **Decrement (- -)** - decreases the value of integer variable by one.

```
int a = 5;  
a--; //the operation a=a-1; the result will be 4
```

- **Compound addition (+=)** - adds the right operand to the left operand and assigns the result to the left operand.

```
int a = 5;  
a+=2; //the operation a=a+2; the result will be 7
```

- **Compound subtraction (-=)** - subtracts the right operand from the left operand and assigns the result to the left operand.

```
int a = 5;  
a+=3; //the operation a=a-3; the result will be 2
```

- **Compound multiplication (`*=`)** - multiplies the left operand by the right operand and assigns the result to the left operand.

```
int a = 5;  
a*=3; //the operation a=a*3; the result will be 15
```

- **Compound division (`/=`)** - divides the left operand with the right operand and assigns the result to the left operand.

```
int a = 6;  
a/=3; //the operation a=a/3; the result will be 2
```

- **Compound modulo (`%=`)** - takes modulus using two operands and assigns the result to the left operand.

```
int a = 5;  
//the result will be the remaining  
//part of the operation a/2; it results in 1  
a%=2;
```

- **Compound bitwise OR (`|=`)** - bitwise OR operator that assigns the value to the operand on the left.

```
int a = 5;  
a|=2; //the operation a=a|2; the result will be 7
```

- **Compound bitwise AND (`&=`)** - bitwise AND operator that assigns the value to the operand on the left.

```
int a = 6;  
a&=; //the operation a=a&2; the result will be 2
```

Check yourself

1. What is the data type used for a variable range 0...255?
2. What should data type be used for more precise measurements?
3. Make data type conversion char to string.
4. What is the main advantage of using compound operators?

Program control structures

Control structure

If is a statement that checks the condition and executes the following statements if the condition is *true*. There are multiple ways how to write down the *if* statement:

```
//1st example  
if (condition) statement;  
  
//2nd example  
if (condition)
```

4. IoT Hardware overview

```
statement;

//3rd example
if (condition) { statement; }

//4th example
if (condition)
{
    statement;
}
```

When both *true* and *false* cases of the condition should be viewed, the **else** part should be added to the *if* statement in the following ways:

```
if (condition) {
    statement1;      //executes when the condition is true
}
else {
    statement2;      //executes when the condition is false
}
```

If more conditions should be viewed, the **else if** part is added to the *if* statement:

```
if (condition1) {
    statement1;      //executes when the condition1 is true
}
else if (condition2) {
    statement2;      //executes when the condition2 is true
}
else {
    statement3;      //executes in all of the rest cases
}
```

The example when the *x* variable is compared and in the cases when it is higher than *10*, the *digitalWrite()* method executes.

```
if (x>10)
{
    //statement is executed if the x>10 expression is true
    digitalWrite(LEDpin, HIGH)
}
```

Logical operators

Logical operators are widely used together with the condition operator *if* that is described below.

Comparison operators

There are multiple comparison operators used for comparing variables and values. All of these operators compare the value of the variable on the left to the value of the variable on the right. Comparison operators are the following:

- == (equal to) - if they are equal, the result is *TRUE*, otherwise *FALSE*.

- != (not equal to) - if they are not equal, the result is *TRUE*, otherwise *FALSE*.
- < (less than) - if the value of the variable on the left is less than the value of the variable on the right, the result is *TRUE*, otherwise *FALSE*.
- <= (less than or equal to) - if the value of the variable on the left is less than or equal to the value of the variable on the right, the result is *TRUE*, otherwise *FALSE*.
- > (greater than) - if the value of the variable on the left is greater than the value of the variable on the right, the result is *TRUE*, otherwise *FALSE*.
- >= (greater than or equal to) - if the value of the variable on the left is greater than or equal to the value of the variable on the right, the result is *TRUE*, otherwise *FALSE*.

Examples:

```
if (x==y){ //equal
  //statement
}

if (x!=y){ //not equal
  //statement
}

if (x<y){ //less than
  //statement
}

if (x<=y){ //less than or equal
  //statement
}

if (x>y){ //greater than
  //statement
}

if (x>=y){ //greater than or equal
  //statement
}
```

Boolean operators

Three Boolean logical operators in the Arduino environment are the following:

- ! (logical NOT) - reverses the logical state of the operand. If a condition is *TRUE* the logical NOT operator will turn it to *FALSE* and the other way around.
- && (logical AND) - the result is *TRUE* when both operands on the left and on the right of the operator are *TRUE*. If even one of them is *FALSE* the result is *FALSE*.
- || (logical OR) - the result is *TRUE* when at least one of the operands on the left and on the right of the operator is *TRUE*. If both of them are *FALSE* the result is *FALSE*.

Examples:

```
//logical NOT
if (!a) { //the statement inside "if" will execute when the "a" is FALSE
  b = !a; //the reverse logical value of "a" is assigned to the variable "b"
}
```

4. IoT Hardware overview

```
//logical AND
//the statement inside "if" will execute when the
//values both of the "a" and "b" are TRUE
if (a && b){
    //statement
}

//logical OR
//the statement inside "if" will execute when at least one of the
//"/"a" and "b" values is TRUE
if (a || b){
    //statement
}
```

Switch case statement

Switch statement similar like *if* statement controls the flow of program. The code inside *switch* is executed in various conditions. A *switch* statement compares the values of a variable to the specified values in the *case* statements. Allowed data types of the variable are *int* and *char*. The *break* keyword exits the *switch* statement.

Examples:

```
switch (x) {
    case 0: //executes when the value of x is 0
        // statements
        break; //goes out of the switch statement

    case 1: //executes when the value of x is 1
        // statements
        break; //goes out of the switch statement

    default: //executes when none of the cases above is true
        // statements
        break; //goes out of the switch statement
}
```

Check yourself 1. Which code part is the correct one?

- if (value == 1) digitalWrite(13, HIGH)
- if (value == 1); digitalWrite(13, HIGH)
- if (value == 1) DigitalRead(13,1)

2. What is the output of the next code part?

```
int x = 0;

switch(x)
{
    case 1: cout << "One";
    case 0: cout << "Two";
```

```
case 2: cout << "Hello, world!";  
}
```

3. In which cases 'switch structure' should be used?

Looping

For

For is a cycle operator that allows specifying the number of times when the same statements will be executed. In this way, similar to the *loop* function it allows to control the program execution. Each time when all statements in the body of the cycle are executed is called the *iteration*. In this way, the cycle is one of the basic programming techniques that is used for all programs and automation in general.

The construction of a *for* cycle is the following:

```
for (initialization ; condition ; operation with the cycle variable) {  
    //the body of the cycle  
}
```

Three parts of the *for* construction is the following:

- *initialization* section usually initialises the value of the variable that will be used to iterate the cycle; this value can be 0 or any other value;
- *condition* allows managing the number of cycle iterations; the statements in the body of the cycle are executed when the condition is *TRUE*;
- *operations with the cycle variable* allows defining the number of cycle iterations.

The example of the *for* cycle:

```
for (int i = 0; i < 4; i = i + 1)  
{  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(1000);  
}
```

On the initialization of the *for* cycle the variable *i=0* is defined. The condition states that the *for* cycle will be executed while the value of variable *i* will be less than 4 (*i<4*). In the operation with the cycle variable it is increased by 1 each time when the cycle is repeated.

In this example above, the LED that is connected to the pin 13 of the Arduino board will turn on/off four times.

While

While cycle operator is similar to the *for* cycle operator that is described above, but it does not contain the cycle variable. Because of this, the *while* cycle allows to execute

4. IoT Hardware overview

previously unknown number of iterations. The management of the cycle is realized using only *condition* that needs to be *TRUE* for the next operation to execute.

The construction of the *while* cycle is the following:

```
while (condition that is TRUE)
{
    //the body of the cycle
}
```

That way the *while* cycle can be used as a good instrument for the execution of a previously unpredictable program. For example, if it is necessary to wait until the signal from the pin 2 reaches the defined voltage level - 100, the following code can be used:

```
int inputVariable = analogRead(2);
while (inputVariable < 100)
{
    digitalWrite(13, HIGH);
    delay(10);
    digitalWrite(13, LOW);
    delay(10);
    inputVariable = analogRead(2);
}
```

In the cycle, the LED that is connected to the pin 13 of the Arduino board will be turned on/off while the signal will reach the specified level.

Do while

Do while cycle works the same way like the *while* loop. The difference is that in the *while* cycle the condition is checked before entering the loop, but in the *do while* cycle the condition is checked after execution of the statements in the loop and then if the condition is *TRUE* the loop repeats. As a result, the statements inside the cycle will execute at least once, even if the test condition is *FALSE*.

The construction of a *do while* cycle is the following:

```
do {
    //the body of the cycle
} while (condition that is TRUE);
```

If the same code is taken from the *while* loop example and used in the *do while* cycle, the difference is that the code will execute at least once, even if the *inputVariable* value is more than or equal to 100. The example code:

```
int inputVariable = analogRead(2);
do {
    digitalWrite(13, HIGH);
    delay(10);
    digitalWrite(13, LOW);
    delay(10);
    inputVariable = analogRead(2);
} while (inputVariable < 100);
```

Check yourself

1. What is a kind of the loop, where the condition is checked after the loop body is executed?
2. How long will the operators in the body of the loop operate (while ($x < 100$))?
3. What value will be for variable *a* after code executing

```
int a; for(a = 0; a < 10; a++) {}
```

4. Which of the following operators are not loop(s) in Arduino IDE?

- do while
- while
- repeat until
- for

Interrupts and sub-programs

Functions

Functions are the set of statements that are executed always when the function is called. Two functions that were mentioned before are already known - *setup()* and *loop()*. The programmer is usually trying to make several functions that contain all the statements and then to call them in the *setup()* or *loop()* functions.

The structure of the function is following:

```
type functionName(arguments) //a return type, name and arguments of the function
{
    //the body of a function - statements to execute
}
```

For the example, a function that periodically turns on and off the LED is created:

```
void exampleFunction()
{
    digitalWrite(13, HIGH); //the LED is ON
    delay(1000);
    digitalWrite(13, LOW); //the LED is OFF
    delay(1000);
}
```

In the example code can be seen that the return type of *exampleFunction* function is *void* that means the function does not have the return type. This function also does not have any arguments, because the brackets are empty.

This function should be called inside the *loop()* function in the following way:

```
void loop()
{
```

4. IoT Hardware overview

```
exampleFunction(); //the call of the defined function inside loop()  
}
```

The whole code in the Arduino environment looks like this:

```
void loop()  
{  
    exampleFunction(); //the call of the defined function inside loop()  
}  
  
void exampleFunction()  
{  
    digitalWrite(13, HIGH); //the LED is ON  
    delay(1000);  
    digitalWrite(13, LOW); //the LED is OFF  
    delay(1000);  
}
```

It can be seen that the function is defined outside the *loop()* or *setup()* functions.

When some specific result must be returned as a result of a function, then the function return type should be indicated, for example:

```
//the return type is "int"; arguments are still optional  
int sumOfTwoNumbers(int x, int y)  
{  
    //the value next to the "return" should have the "int" type;  
    //this is what will be returned as a result.  
    return (x+y);  
}
```

In the *loop()* this function would be called in a following way:

```
void loop()  
{  
    //the call of the defined function inside loop()  
    int result = sumOfTwoNumbers(2, 3);  
}
```

Interrupts

Interrupt is a signal that stops the normal execution of a program in the processor, to be able to handle tasks with higher priority. These tasks usually are called *Interrupt service routine (ISR)* or interrupt handler. Interrupt signals can be generated from the external source, like a change of value on the pin and from the internal source, like a timer. When the interrupt signal is received, the processor stops executing the code and starts the IRS. After completing the IRS, the processor returns to the normal program execution state.

IRS should be as short as possible, and the return type of it is void. Some of normal Arduino functions do not work or behave differently in the IRS, for example, *delay()* function does not work in the IRS. Variables, used in the IRS must be volatile variables.

Interrupts are used to detect important real-time events, which occur during the normal code execution of the code, without continuously checking them, like pushing a button.

Different Arduino types have different external interrupt pin availability. In most Arduino boards pins, number 2 and 3 are used for interrupts.

To attach interrupt, the function `attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)` is called. This function has 3 parameters:

1. *pin* – the number of a pin number where the interrupt signal generating device will be attached,
2. *ISR* – the name of a function of interrupt service routine,
3. *mode* - defines when interrupt signal is triggered. There are four basic *mode* values:
 - LOW - interrupt is triggered when the pin value is LOW,
 - HIGH - interrupt is triggered when the pin value is HIGH,
 - CHANGE - interrupt is triggered when the pin value is changed,
 - RISING - interrupt is triggered when the pin value is changed from LOW to HIGH.

The example program that uses interrupt:

```
volatile bool button =0; //a variable to save button state

void setup() {
    //define LED pin
    pinMode(13,OUTPUT);
    // define button pin
    pinMode(2,INPUT_PULLUP);
    // attach interrupt to button pin
    attachInterrupt(digitalPinToInterrupt(2),ButtonIRS,FALLING);
}

void ButtonIRS() { // IRS function
    button =!button;
}

void loop() {
    digitalWrite (13,button);
}
```

Check yourself

1. What are the built-in functions used for?

- To reduce the size of the program
- To delete unnecessary functions
- To simplify the source file
- To increase the speed of the program

2. Which of the following statements are true?

- built-in functions must return a value.
- built-in functions cannot return values.

4. IoT Hardware overview

- The compiler can ignore the declaration of the built-in function.
- built-in functions cannot contain more than 10 lines of code.

3. Is it possible to guarantee that the declared built-in function is really built-in?

- guarantee is not possible, in each case it is different
- can be confidently ensured that the function you have declared as built-in is really built-in

Timing

Delay

The simplest solution to make functions work for a certain time is to use `delay()`⁶²⁾ function. `delay()` function stops program execution for instructed time.

Blinking led is a simple demonstration of delay functionality:

```
digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
delay(1000);                      // stop program for a second
digitalWrite(LED_BUILTIN, LOW);     // turn the LED off
delay(1000);                      // stop program for a second
```

Limitations that come with using `delay()`: most tasks stop, there can be no sensor reading, calculation, pin manipulation. Some tasks continue to work, like receiving serial transmissions and outputting set PWM values. Alternative of using `delay` is to use `millis()`.

Millis

`millis()`⁶³⁾ returns number in milliseconds since Arduino began running current program. The number will reset after approximately 50 days. `millis()` can be used to replace `delay()`.

Here is an example code of blinking led using `millis()`. Millis is used as a timer. Every new cycle time is calculated since the last led state change if passed time is equal to or greater than the threshold value led is switched.:.

```
// Unsigned long should be used to store time values
//store value of current millis reading
unsigned long currentTime = 0;
//store value of time when last time LED state was switched
unsigned long previousTime = 0;

bool ledState = LOW; //variable for setting led state

const int stateChangeTime = 1000; //time at which switch led states

void setup() {
  pinMode (LED_BUILTIN, OUTPUT); // led setup
}

void loop() {
  currentTime = millis(); //read and store current time
```

```

//calculate passed time since last stateChange
//if more time has passed than stateChangeTime, start state Change
if (currentTime - previousTime >= stateChangeTime) {

    previousTime = currentTime; //store led state change time
    ledState = !ledState; //change led state to oposite
    digitalWrite(LED_BUILTIN, ledState); //write led state to led
}
}

```

Protothread

Protothread is a mechanism for concurrent programming in embedded systems with limited resources. In Arduino, it can be used to achieve a periodical function call, like sensor reading, output state change, calculations etc. Most Arduino used microcontrollers have only one core, it can't execute multiple functions simultaneously. By using *millis()* and loop, we can schedule a function call if appropriate conditions are met, thus avoiding unnecessary taking processors time.

In example code second LED is added, to demonstrate how to blink two led's concurrent with different frequencies. In the same manner, servo control, button reading and other functionality can be added.

```

// Unsigned long should be used to store time values
//store value of current millis reading
unsigned long currentTime = 0;
//store value of time when last time LED1 state was switched
unsigned long previousTime1 = 0;
//store value of time when last time LED2 state was switched
unsigned long previousTime2 = 0;

bool led1State = LOW; //variable for setting led1 state
bool led2State = LOW; //variable for setting led2 state

const int stateChangeTimeLed1 = 1000; //time at which switch led1 states
const int stateChangeTimeLed2 = 200; //time at which switch led2 states

void setup() {
    pinMode (LED_BUILTIN, OUTPUT); // led1 setup
    pinMode (2,OUTPUT); // led1 setup
}

void loop() {
    currentTime = millis(); //read and store current time

    //calculate passed time since the last stateChange
    //if more time has passed than stateChangeTime, start state Change

    if (currentTime - previousTime1 >= stateChangeTimeLed1) {
        previousTime1 = currentTime; //store led1 state change time
        led1State = !led1State; //change led1 state to oposite
        digitalWrite(LED_BUILTIN, led1State); //write led1State to led1
    }
    if (currentTime - previousTime2 >= stateChangeTimeLed2) {
        previousTime2 = currentTime; //store led2 state change time
        led2State = !led2State; //change led 2state to oposite
    }
}

```

4. IoT Hardware overview

```
    digitalWrite(2, led2State); //write led2state to led2
}
}
```

There are dedicated libraries made by the Arduino community to implement protothreads.⁶⁴⁾

Check yourself

1. What are the drawbacks of using *delay()*?
 - Program execution stops
 - Can't read sensors during *delay()*
 - Simple code
2. What is the difference between *delay()* and *millis()*?
3. Can Arduino run truly parallel programs?
4. How is concurrency achieved on Arduino?

Reading GPIOs, outputting data and tracing

Digital I/O

The digital inputs and output of Arduino allow connecting different type of sensors and actuators to the board. Digital signals can take two values - HIGH (1) or LOW (0). These types of inputs and outputs are used in applications when the signal can have only two states.

pinMode()

The function *pinMode()* is essential in order to indicate whether the specified pin will behave like an input or an output. This function does not return any value. Usually, the mode of a pin is set in the *setup()* function of a program - only once on initialization.

The syntax of a function is the following:

```
pinMode(pin, mode)
```

The parameter *pin* is the number of the pin.

The parameter *mode* can have three different values - *INPUT*, *OUTPUT*, *INPUT_PULLUP* depending on whether the pin will be used as an input or an output. The *INPUT_PULLUP* mode means that the pin will work as an input whose state will be inverted (set to the opposite). More about Pullup Resistors can be found in the Arduino homepage⁶⁵⁾.

digitalWrite()

The function *digitalWrite()* writes a HIGH or LOW value to the pin. This function is used for digital pins, for example, to turn on/off LED. This function as well does not return any value.

The syntax of a function is the following:

```
digitalWrite(pin, value)
```

The parameter *pin* is the number of the pin. The parameter *value* can take values HIGH or LOW. If the mode of the pin is set to the OUTPUT, the *HIGH* sets voltage to +5V and *LOW* - to 0V.

It is also possible to use this function for pins that are set to have the INPUT mode. In this case, *HIGH* or *LOW* values enable or disable the internal pull-up resistor.

digitalRead()

The function *digitalRead()* works in the opposite direction than the function *digitalWrite()*. It reads the value of the pin that can be either *HIGH* or *LOW* and returns it.

The syntax of a function is the following:

```
digitalRead(pin)
```

The parameter *pin* is the number of the pin.

On the opposite as the functions viewed before, this one has the return type, and it can take a value of *HIGH* or *LOW*.

Analog I/O

The analogue inputs and outputs are used when the signal can take a range of values, unlike the digital signal that takes only two values (HIGH or LOW). For measuring the analogue signal, Arduino has built-in *analogue-to-digital converter* (ADC) that returns the digital value of the voltage level.

analogWrite()

The function *analogWrite()* is used to write an analog value (*Pulse Width Modulation* (PWM)) of the integer type as an output of the pin. The example of use is turning on/off the LED in various brightness levels or setting different speeds of the motors. The value that is written to the pin stays unchanged until the next value is written to the pin.

The syntax of a function is the following:

```
analogWrite(pin, value)
```

The parameter *pin* is the number of the pin.

The parameter *value* is the

This function does not have the return type.

analogRead()

The function *analogRead()* is used for analogue pins (A0, A1, A2, etc.) and it reads the value that is on the analogue pin.

4. IoT Hardware overview

The syntax of a function is the following:

```
analogRead(pin)
```

The parameter *pin* is the number of the pin whose value is read.

The return type of the function is the integer value between 0 and 1023. The reading of each analogue input takes around 100 microseconds that are 0.0001 seconds.

Check yourself

1. To assign the Arduino pin operation mode, which function is using?

- function digitalWrite()
- function pinMode()
- directive #define

2. What command for analogue input read is used?

3. The digital output on Arduino works as a “power source” with voltage

- 5V
- 12V
- 3.3V

4.4. Espressif SoC Overview

Arduino along with a vast amount of peripheral boards lacks integration of the networking capabilities in one SoC. Espressif ESP series was the natural answer for this disadvantage as their ESP 8266 with integrated WiFi, introduced in 2014, is widely recognised as a turning point for the IoT market, delivering de-facto fully functional IoT chip, providing high performance and low power to the end users and developers. ESP 32 launched in 2016 brought even more disrupting effect to the IoT ecosystems, introducing additional Bluetooth interface to the above. By the January 2018 company announced they delivered to the market 100 000 000⁶⁶⁾, thus constituting a de-facto standard for the IoT market devices.

Following chapters provide an overview of the networking programming with the use of ESP SoCs.

Please note, interfacing with sensors and actuators has been already covered in the chapter related to Arduino:

Arduino Overview

The major difference is that ESP SoCs (both 8266 and 32) use 3.3V logic, while most (but not all!) Arduinos use 5V logic. This can be easily handled using one or bi-directional voltage converters/adapters. Additionally, many ESP boards and development kits offer double power source, including 5V, even if the device itself still operates on 3.3V. Yet programming principals used to be the same while using digital protocols and GPIOs

Espressif SoC

Espressif SoC networking

ESP programming fundamentals

4.4.1. Espressif SoC

Espressif System-on-chip (SoC) devices are low-cost microcontrollers with full TCP/IP stack capability produced by Shanghai-based Chinese manufacturer, Espressif Systems⁶⁷⁾. The two most popular series of these microcontrollers are:

- ESP8266
- ESP32

ESP 8266 General information

The ESP8266 is a low-cost system on chip (SoC) microcontroller with Wi-Fi and full TCP/IP stack capability⁶⁸⁾. The chip first came to the market with the ESP-01 module, made by a third-party manufacturer, Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style AT commands. The low price and the fact that there were very few external components on the module which suggested that it could eventually be very inexpensive in volume, attracted many users to explore it. The ESP8285 is an ESP8266 with 1 MiB of built-in flash, allowing for single-chip devices capable of connecting to Wi-Fi. The successor to these microcontroller chips is the ESP32. For now the ESP8366 family includes the following chip:

- ESP8266EX (figure 113),



Figure 113: ESP8266EX chip

Esp8266 Architecture overview

Main standard features of the ESP8266EX are:

Processor:

4. IoT Hardware overview

- **Main processor:** L106 32-bit RISC microprocessor core based on the Tensilica Xtensa Diamond Standard 106Micro running at 80 MHz. Both the CPU and flash clock speeds can be doubled by overclocking on some devices. CPU can be run at 160 MHz, and flash can be sped up from 40 MHz to 80 MHz. Success varies chip to chip.

Memory:

- 32 KiB instruction RAM
- 32 KiB instruction cache RAM
- 80 KiB user data RAM
- 16 KiB ETS system data RAM
- External QSPI flash: up to 16 MiB is supported (512 KiB to 4 MiB typically included)

Interfaces:

- IEEE 802.11 b/g/n Wi-Fi
- Integrated TR switch, balun, LNA, power amplifier and matching network WEP or WPA/WPA2 authentication, or open networks
- 16 GPIO pins
- SPI
- I²C (software implementation)
- I²S interfaces with DMA (sharing pins with GPIO)
- UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
- 10-bit ADC (successive approximation ADC)

(Figure 114) shows function block of ESP8266 chip diagram⁶⁹⁾.

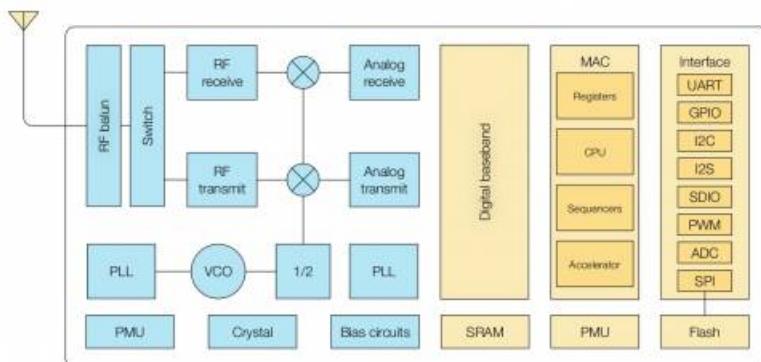


Figure 114: ESP8266 chip main functions

ESP8266 modules

There are many ESP8266 based modules on the market⁷⁰⁾. These modules combine ESP8266EX microcontroller and additional components mounted on PCB.

The most popular are these produced by AI-Thinker and remain the most widely

available.⁷¹⁾:

- ESP-01 (512 KiB Flash)
- ESP-01S (1 MiB Flash)
- ESP-12 (FCC and CE approved)
- ESP-12E
- ESP-12F (4 MiB Flash, FCC and CE approved)

Popular modules from other manufacturers:

- Sparkfun ESP8266 Thing
- Wemos D1 mini, D1 mini Pro⁷²⁾

The Espressif company also produces ready-made modules using the aforementioned chip. This is the series of ESP8266-based modules made by Espressif (Table 13):

| Name | Active pins | LEDs | Antenna | Shielded | Dimensions (mm) | Notes |
|------------------------------|-------------|------|-------------|----------|-----------------|--|
| ESP-WROOM-02 ⁷³⁾ | 18 | No | PCB trace | Yes | 18 × 20 | FCC ID 2AC7Z-ESPWROOM02 |
| ESP-WROOM-02D ⁷⁴⁾ | 18 | No | PCB trace | Yes | 18 × 20 | FCC ID 2AC7Z-ESPWROOM02D. Revision of ESP-WROOM-02 compatible with both 150-mil and 208-mil flash memory chips |
| ESP-WROOM-02U ⁷⁵⁾ | 18 | No | U.FL socket | Yes | 18 × 20 | Differs from ESP-WROOM-02D in that includes an U.FL compatible antenna socket connector |
| ESP-WROOM-S2 ⁷⁶⁾ | 20 | No | PCB trace | Yes | 16 × 23 | FCC ID 2AC7Z-ESPWROOMS2 |

Table 13: Espressif ESP8266 modules

The most widely used and chippest ESP-01 is presented on (Figure 115) and its pinout on (Figure 116)

4. IoT Hardware overview



Figure 115: ESP-01

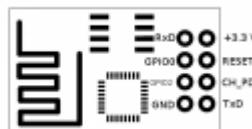


Figure 116: ESP-01 pinout

Module ESP12F with pinout is presented on (Figure 117) and its pinout on (Figure 118)



Figure 117: ESP-12F

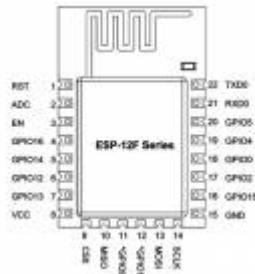


Figure 118: ESP-12F pinout

Among the other modules, it is worth to be interested in WEMOS modules ⁷⁷⁾ (Figure 119) (Figure 120). The WEMOS company offers dedicated sensor modules and inputs/outputs compatible with the processor modules. They are called WEMOS shields (Figure

121).

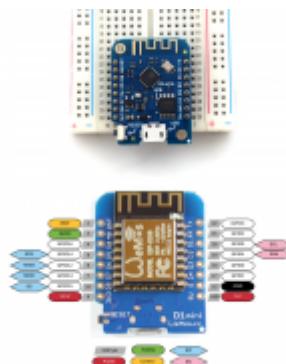


Figure 119: Wemos D1 mini with pinout



Figure 120: Wemos D1 Pro

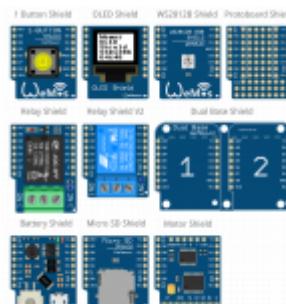


Figure 121: Wemos I/O shields

ESP32 NodeMCU pins (figure 122):

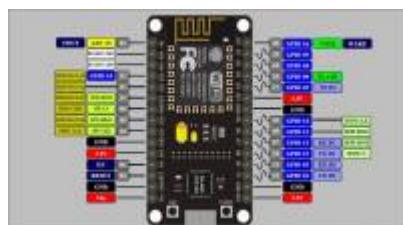


Figure 122: NodeMCU v.2 pins

4. IoT Hardware overview

Esp32 General information

ESP32 is a low-cost, low-power system on a chip (SoC) series microcontrollers with Wi-Fi & dual-mode Bluetooth capabilities⁷⁸⁾. ESP32 SoC is highly integrated with built-in antenna switches, power amplifier, low-noise receive amplifier, filters, and power management modules. Inside all family there is a single-core or dual-core Tensilica Xtensa LX6 microprocessor with a clock rate of up to 240 MHz. ESP32 is designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It features all the state-of-the-art characteristics of low-power chips, including fine-grained clock gating, multiple power modes, and dynamic power scaling. For now the ESP32 family includes the following chips:

- ESP32-D0WDQ6 (figure 123),
- ESP32-D0WD (figure 124),
- ESP32-D2WD (figure 125),
- ESP32-S0WD (figure 126),
- ESP32-PICO-D4 - SiP (system in package) (figure 127) - additionally contains crystal oscillator, 4MiB flash memory, filter capacitors and RF matching links.



Figure 123: ESP32-D0WDQ6



Figure 124: ESP32-D0WD



Figure 125: ESP32-D2WD



Figure 126: ESP32-S0WD



Figure 127: ESP32-PICO-D4

Esp32 Architecture overview

(Figure 16) shows function block diagramm of ESP32 chip. Main common features of the ESP32 are: 79) 80)

Processors:

- **Main processor:** Tensilica Xtensa 32-bit LX6 microprocessor
 - **Cores:** 2 or 1 (depending on variation) (All chips in the ESP32 series are dual-core except for ESP32-S0WD, which is single-core.)
 - Internal 8 MHz oscillator with calibration
 - External 2 MHz to 60 MHz crystal oscillator (40 MHz only for Wi-Fi/BT functionality)
 - External 32 kHz crystal oscillator for RTC with calibration
 - **Clock frequency:** up to 240 MHz
 - **Performance:** up to 600 DMIPS
- **Ultra low power co-processor:** allows you to do ADC conversions, I2C connecting, computation, and level thresholds while in a deep sleep.

Wireless connectivity:

- **Wi-Fi:** 802.11 b/g/n/e/i (802.11n @ 2.4 GHz up to 150 Mbit/s) with simultaneous Infrastructure BSS Station mode/SoftApp mode/Promiscuous mode,
- **Bluetooth:** v4.2 BR/EDR and Bluetooth Low Energy (BLE) with multi-connections Bt and BLE and simultaneous advertising and scanning capability,

Memory: Internal memory:

- **ROM:** 448 KiB (*For booting and core functions*)
- **SRAM:** 520 KiB (*For data and instruction*)

4. IoT Hardware overview

- **RTC fast SRAM:** 8 KiB (*For data storage and main CPU during RTC Boot from the deep-sleep mode.*)
- **RTC slow SRAM:** 8 KiB (*For co-processor accessing during deep-sleep mode*)
- **eFuse:** 1 Kibit (*Of which 256 bits are used for the system (MAC address and chip configuration) and the remaining 768 bits are reserved for customer applications, including Flash-Encryption and Chip-ID.*)
- **Embedded flash:** (*Flash connected internally via IO16, IO17, SD_CMD, SD_CLK, SD_DATA_0 and SD_DATA_1 on ESP32-D2WD and ESP32-PICO-D4.*)
 - 0 MiB (ESP32-D0WDQ6, ESP32-D0WD, and ESP32-S0WD chips)
 - 2 MiB (ESP32-D2WD chip)
 - 4 MiB (ESP32-PICO-D4 SiP module)
- **External flash & SRAM:** ESP32 supports up to four 16 MiB external QSPI flashes and SRAMs with hardware encryption based on AES to protect developers' programs and data. ESP32 can access the external QSPI flash and SRAM through high-speed caches.
- Up to 16 MiB of external flash are memory-mapped onto the CPU code space, supporting 8-bit, 16-bit and 32-bit access. Code execution is supported.
- Up to 8 MiB of external flash/SRAM memory is mapped onto the CPU data space, supporting 8-bit, 16-bit and 32-bit access. Data-read is supported on the flash and SRAM. Data-write is supported on the SRAM.

ESP32 chips with embedded flash do not support the address mapping between external flash and peripherals.

Peripheral input/output:

- Rich peripheral interface with DMA that includes capacitive touch (10x touch sensors),
- 12 -bit ADCs (analog-to-digital converter) up to 18 channels,
- 2 x 8 bit DACs (digital-to-analog converter),
- 2 x I²C (Inter-Integrated Circuit),
- 3x UART (universal asynchronous receiver/transmitter),
- CAN 2.0 (Controller Area Network),
- 4 x SPI (Serial Peripheral Interface),
- 2 x I²S (Integrated Inter-IC Sound),
- RMII (Reduced Media-Independent Interface),
- Motor PWM (pulse width modulation),
- LED PWM up to 16 channels,
- Hall sensor ,
- Internal temperature sensor.

Security:

- Secure boot
- Flash encryption

4.4. Espressif SoC Overview

- IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration:
 - AES,
 - SHA-2,
 - RSA,
 - elliptic curve cryptography (ECC),
 - random number generator (RNG)

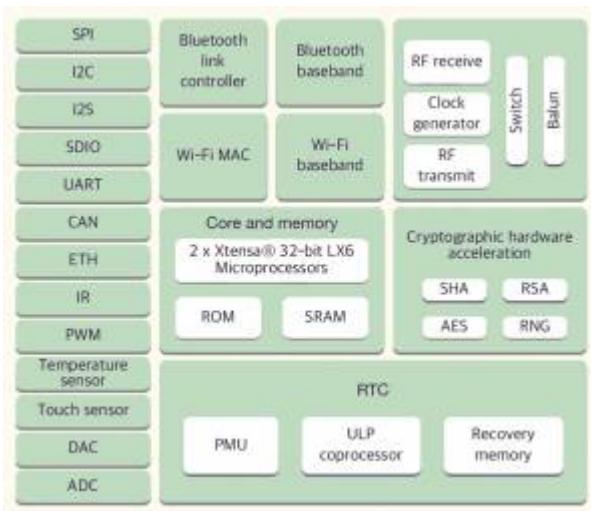


Figure 128: ESP 32 Function block diagram

ESP32 modules

The company also produces ready-made modules using the aforementioned processors. These modules combines ESP32 microcontroller and additional components mounted on PCB with EM shield:

- ESP32-WROOM-32 with 4MiB flash memory, and antenna on PCB. (figure 129)
- ESP32-WROOM-U with 4MiB flash memory and u.fl antenna connector, (figure 130),
- ESP32-WROVER - with 4MiB flash memory, 4MiB pseudo static RAM and antenna on PCB (figure 131)
- ESP32-WROVER-I - as ESP32-WROVER with additional u.fl antenna connector. (figure 132)

4. IoT Hardware overview



Figure 129: ESP32-WROOM-32



Figure 130: ESP32-WROOM-U



Figure 131: ESP32-WROVER



Figure 132: ESP32-WROVER-I

ESP32 development kits

To accelerate the design of circuits, developers can use specially prepared sets with ESP32 which are ready to use. The original Espressif best known small development boards are:

- ESP32-DevkitC (figure 133)
- ESP32-PICO-KIT-V4 (figure 134)



Figure 133: ESP-32-DevkitC



Figure 134: ESP-32-PICO-KIT-V4

General purpose input-output (GPIO) connector

Each ESP32 is equipped with standard 38/40-pins male connector containing universal GPIO ports, VCC 3.3/5V, GND, CLK, I2C/SPI buses pins which developers can use to connect their external sensors, switches and other controlled devices to the ESP32 board and then program their behaviour within the code loaded to the board.

4. IoT Hardware overview

- ESP32-DevkitC v2 pins (figure 135)

:

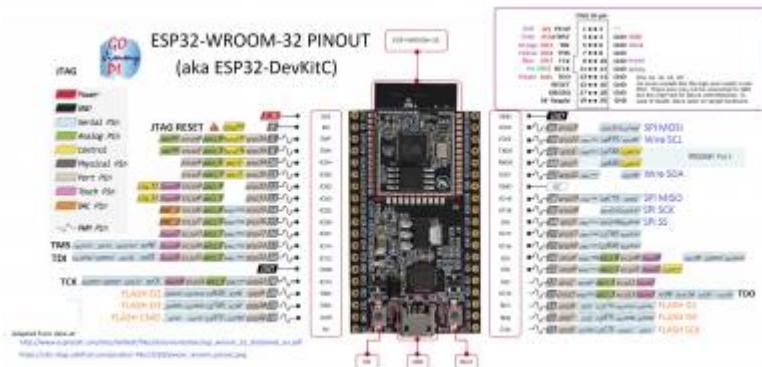




Figure 137: ESP32-Pico D4 pins

4.4.2. Espressif SoC networking

Using Espressif SoC devices in Arduino platform we can use all the previously described Arduino examples for sensors and actuators. But the most interesting are those that use wireless networking functions. Both Espressif chip ESP32 and ESP8266 families can use similar network modes. As a WiFi device Espressif SoC can work in different networking modes (applies to medial layers 1-3 ISO-OSI model):

- as WiFi client connected to WiFi router (figure 138),

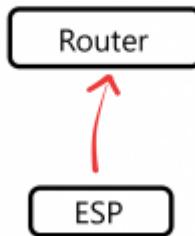


Figure 138: ESP client mode

- as independent WiFi access point (figure 139),

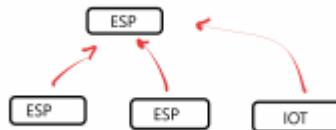


Figure 139: ESP AP mode

- as a repeater with devices connected to ESP and ESP connected to external router (figure 140),

4. IoT Hardware overview

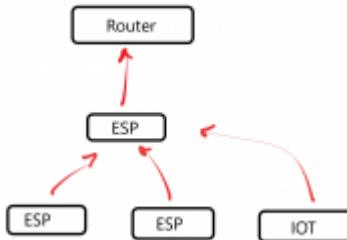


Figure 140: ESP dual mode

- as client and server in mesh network (figure 141),

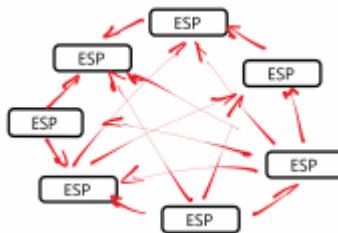


Figure 141: ESP mesh networking

ESP32 can also use Bluetooth networking in the same configuration as WiFi.

4.4.3. ESP programming fundamentals

The following sub-chapters cover programming fundamentals for ESP chips in C/C++, which complies with the most C/C++ notations and have some specific notations. Please note, this is a particular implementation of the programming, compatible with Arduino standards and Arduino(C) IDE. Following chapters present specific aspects of the network programming with ESP chips. Other structures (program flow control, GPIO, etc.) are shown in the Arduino section of this manual and apply straightforwardly to the ESP programming.

This manual refers to the particular version of the software (here Arduino IDE, ESP 8266 and ESP32 and related toolkits) available at the moment of writing of this book, thus accessing particular features may change over time along with the evolution of the platform. Please refer attached documentation (if any) and browse Internet resources to find latest guidances on how to configure development platform, when in doubt. In particular, following links present up-to-date help on "how to start" guides with ESP 8266 and ESP32 platforms and Arduino IDE. Please refer here, when guides present in following sub-chapters are outdated:

- ESP 8266 Github: <https://github.com/esp8266/Arduino>
- ESP 32 Github <https://github.com/espressif/arduino-esp32>

Further reading:

Setting up development environment for ESP SoC programming

ESP AT networking

ESP Network Layers

ESP Application Layer

ESP32 Parallel programming

Setting up development environment for ESP SoC programming

Following subchapters present guides for setting up the environment to enable you to use Arduino(C) IDE to develop solutions for ESP8266 and ESP32. Installation methodology varies much as ESP8266 is already integrated with so-called Board Manager in Ardiono(C) iDE, while ESP32 is somehow external to the Boards Manager and requires some steps to be done, to obtain fully integrated environment. Some of the steps vary in details among various operating systems you use (Windows/Mac/Linux), see details, but in any case, there is a common idea, what to do in following steps. Once it is done, programming of both ESP8266 and ESP32 does not differ much from programming, i.e. Arduino Uno, natively supported by Arduino(C) IDE.

There do exists other solutions i.e. PlatformIO⁸¹⁾ + Visual Studio Code IDE⁸²⁾ or Atom IDE⁸³⁾, MicroPython⁸⁴⁾, NodeMCU⁸⁵⁾(for ESP8266), etc. Some of them provide advanced features, like debugging (usually requires extra hardware, i.e. JTAG), multithreaded and parallel programming, etc. A detailed description of those solutions is out of the scope of this manual, however. Please refer to the links provided below to obtain detailed steps, how to configure other development environments.

Download and run Arduino IDE

We consider here a software that is natively installed as a binary package on your operating system, not a WEB version of the IDE. To obtain the latest version of the binary package, visit the following website: [https://www.arduino.cc/en/Main/Software^{86\)}](https://www.arduino.cc/en/Main/Software), as on figure 142 or install it via integrated software management application (Windows App Store, iTunes, Linux Application Store, etc.).

4. IoT Hardware overview

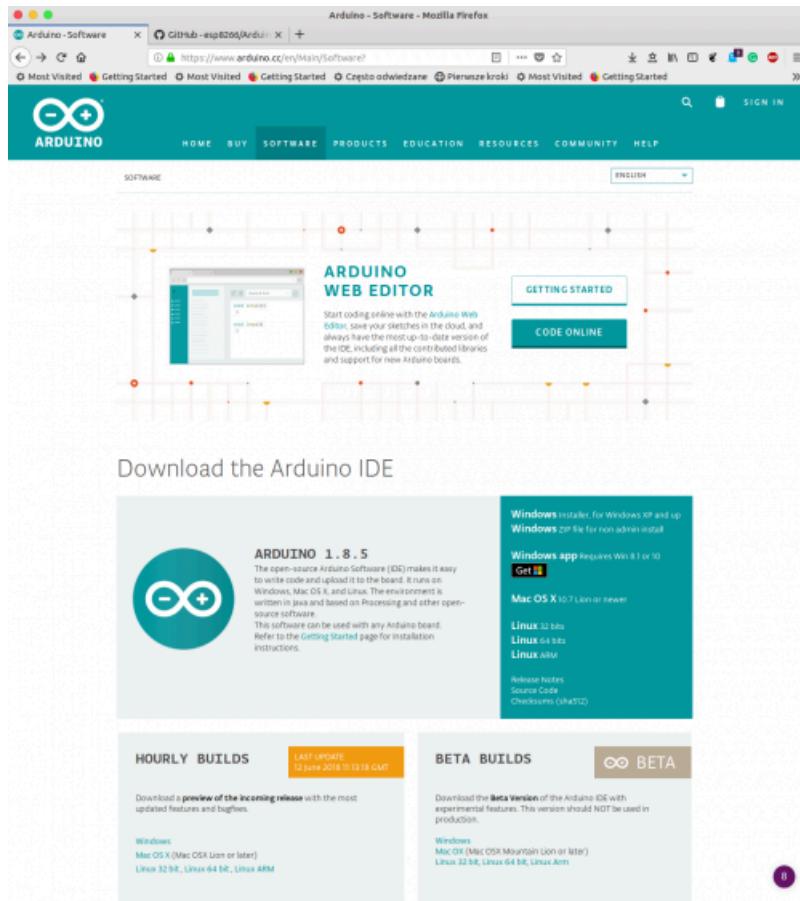


Figure 142: Arduino(C) IDE downloads web page

Linux distros usually offer an outdated version of the Arduino(C) IDE through its repositories; thus we suggest to install one through downloading package directly from the Arduino IDE website.

There are two versions of the Arduino(C) IDE for Windows: one you can download from their website and the other is available via Microsoft Windows Application Store. If you experience compilation problems with Windows Store version, please use the installer from their website and install it manually.

Configuring Arduino IDE for ESP 8266 development

Assuming you've purchased any of the ESP 8266 development boards, it is essential to write the first program. In any case, you'll need a way to write, compile and upload your code to the ESP 8266 SoC. Here Arduino(C) IDE comes handy, however before you start,

4.4. Espressif SoC Overview

you need to let the Arduino(C) IDE knows, how to compile and communicate with your ESP 8266 chip. Below there is a short manual, presenting how to install development extension to the Arduino(C) IDE through the Boards Manager. Note, other solutions (i.e. manual installation via Github pull) is also possible, but we do not consider this option here. ESP 8266 core for Arduino(C) IDE is maintained by the community and the latest version is available here (along with up-to-date installation guide): [https://github.com/esp8266/Arduino⁸⁷](https://github.com/esp8266/Arduino).

Install ESP 8266 boards via Board Manager

Start Arduino(C) IDE (figure 143):

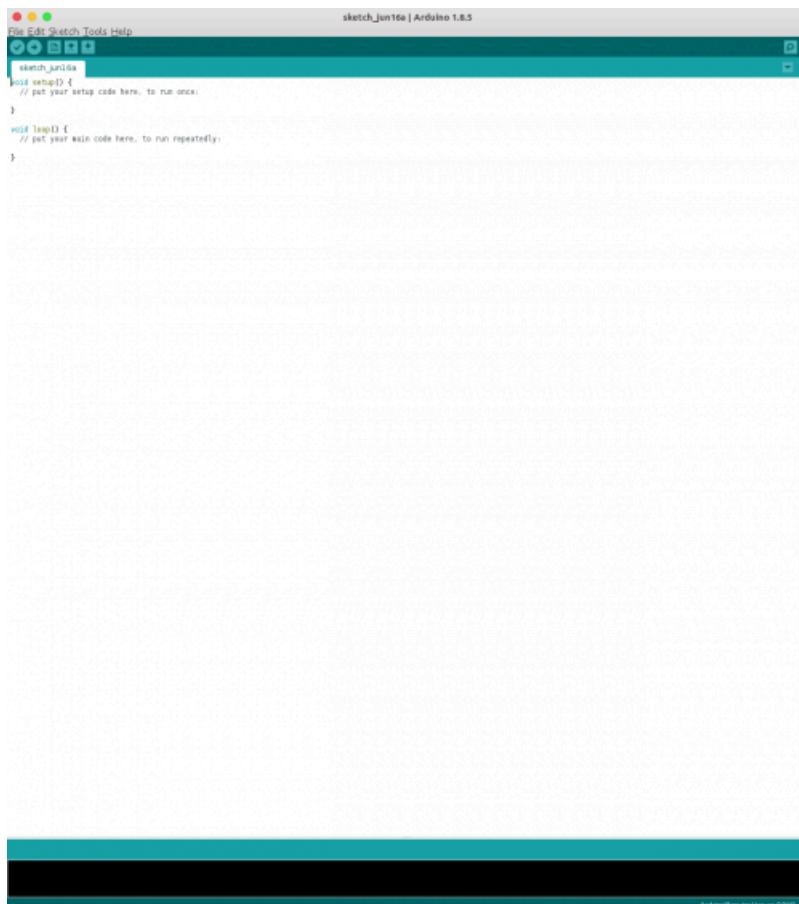


Figure 143: Arduino IDE

Enter application menu *File/Preferences* :

4. IoT Hardware overview



Figure 144: System menu and preferences

and then go to the **Additional Boards Manager URLs:**, enter following URL and accept changes **[OK]** (figure 145):

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

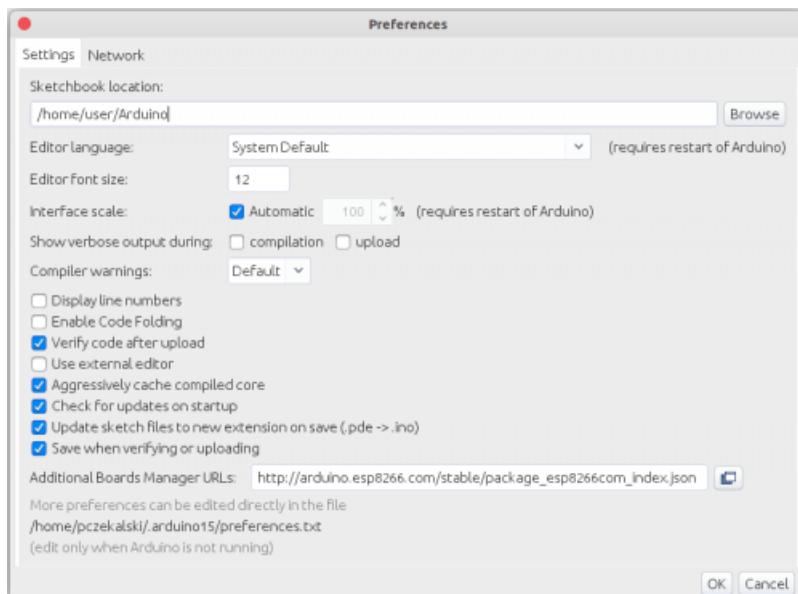


Figure 145: Preferences dialog

Once finished, you need to tell the Board Manager, which definitions and tools to download. Open *Tools/Board:/Boards Manager* (figure 146):

4.4. Espressif SoC Overview

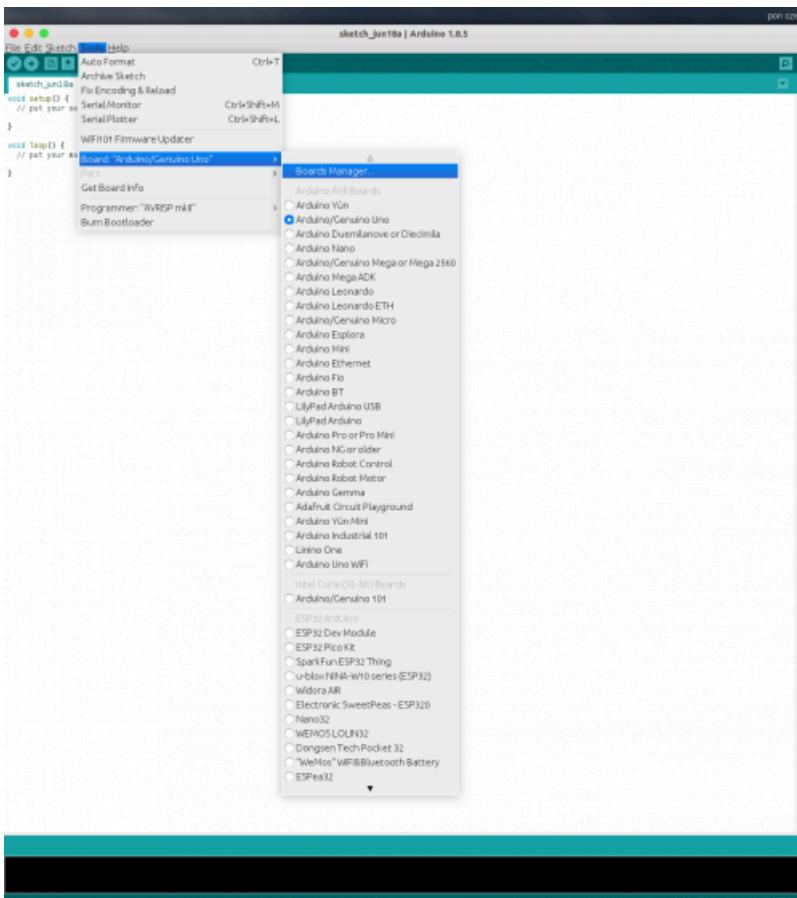


Figure 146: Board Manager menu

and filter all boards entering *ESP8266* in the *Search* area, as on figure 147, then click [**Install!**]:

4. IoT Hardware overview

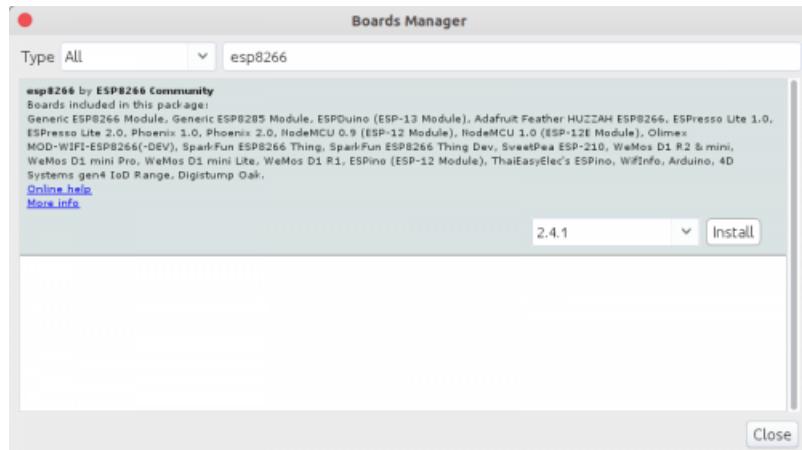


Figure 147: Board installation dialog

Note - installation downloads number of resources over Internet connection and may take some time.

Configure project to compile for ESP8266

Once you're done with Arduino(C) IDE configuration now it is time to start programming. The process above installs a number of board definitions. First, depending on the development kit you own, select the appropriate type of the development board in the Board Manager menu, i.e. *WeMos D1 R2 & mini* (image 148):

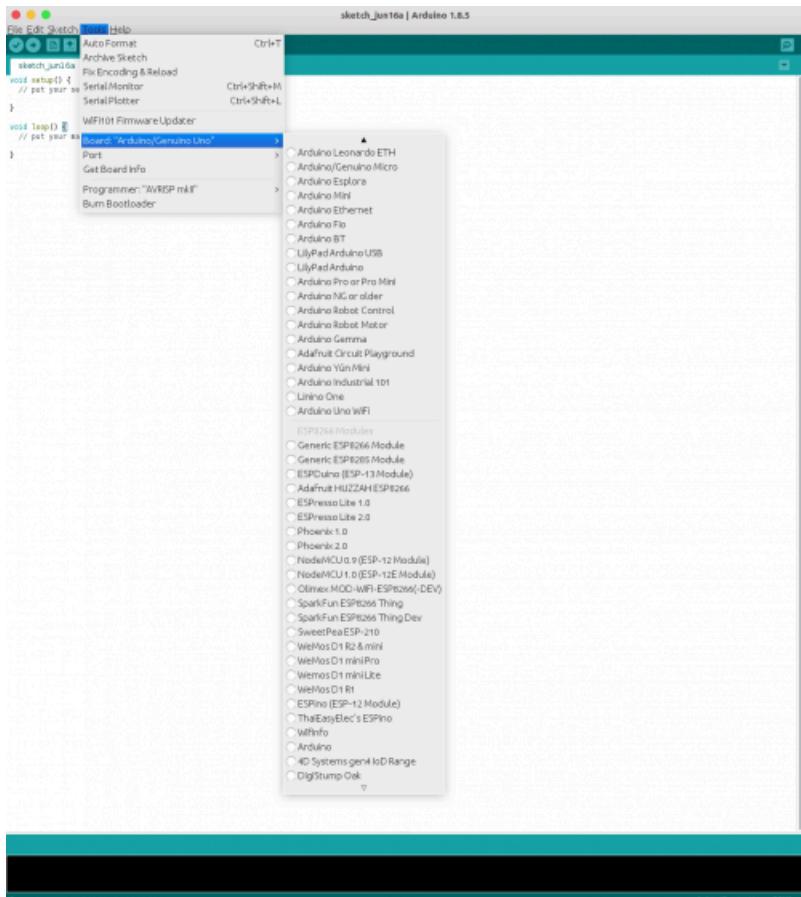


Figure 148: Selecting ESP8266 development board

Communication between development machine and ESP SoC

Most of the ESP development boards come with integrated programming interface via serial to USB converter (usually CH340, CP210x, FTDI, Prolific, etc.) that you connect to the development machine using the USB cable. Connector standards vary, but nowadays the most popular seems to be Micro USB connector. There is also possible to upload your compilation using wireless transmission (OTA - Over The Air), or dedicated programming device, but we do not consider it here as too complicated to implement (requires special firmware) and also insecure. You need to select correct device interface - it differs, depending on the operating system you use, see details below how to identify it in your computer.

Windows

4. IoT Hardware overview

Look into the Device Manager to identify COM port, your board is represented by, in the Windows OS (figure 149, here COM4):

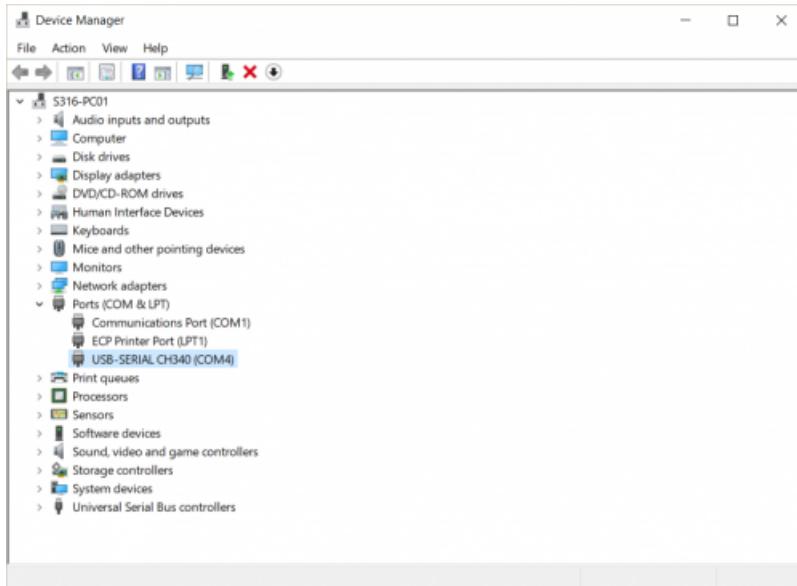


Figure 149: Windows COM port identification

Linux

In case of the Linux distributions, run in the terminal:

```
lsusb
```

and

```
ls /dev/ttyUSB*
```

or

```
ls /dev/ttyACM*
```

to identify your board (figure 150):

```
pi@RPI1:~ $ lsusb
Bus 001 Device 004: ID 1a86:7523 QinHeng Electronics HL-340 USB-Serial adapter
Bus 001 Device 005: ID 0603:1002 Novatek Microelectronics Corp.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast
Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@RPI1:~ $ ls /dev/ttyUSB*
/dev/ttyUSB0
pi@RPI1:~ $
```

Figure 150: Linux serial devices identification

Mac OS

The similar way to Linux distros, look for the devices using:

```
ls /dev/tty.usbmodem*
```

or

```
ls /dev/tty.usbserial*
```

to get a list of connected devices providing serial port feature (figure 151):

4. IoT Hardware overview

```
riccardo@MacBook-Pro-di-Riccardo:~ riccardo$ lsusb
Bus 002 Device 002: ID 05ac:025a Apple Inc. Apple Internal Keyboard / Trackpad
Bus 002 Device 003: ID 0a5c:4500 Broadcom Corp. BRCM20702 Hub
Bus 002 Device 004: ID 05ac:8289 Apple Inc. Bluetooth USB Host Controller
Bus 002 Device 017: ID 2341:0043 2341 Communication Device Serial: 754333137393
51212232
Bus 000 Device 001: ID 1d6b:ILPT Linux Foundation USB 3.0 Bus
MacBook-Pro-di-Riccardo:~ riccardo$ ls /dev/tty.usb*
/dev/tty.usbmodem1411
MacBook-Pro-di-Riccardo:~ riccardo$
```

Figure 151: Mac OS serial devices identification

Configure details for the SoC

Now select appropriate device in the Arduino IDE (figure 152) and select communication speed (figure 153). Eventually, change the other parameters. Details on the flash size, its organisation, communication speed and frequencies should be provided by your hardware vendor. If you experience errors during programming or programming hangs, try to reduce programming speed, as your computer may be not quick enough to deliver data stream over USB. Devices usually present their maximum programming (flashing) speed, and it is common they tolerate lower speeds.



Figure 152: Selecting serial port

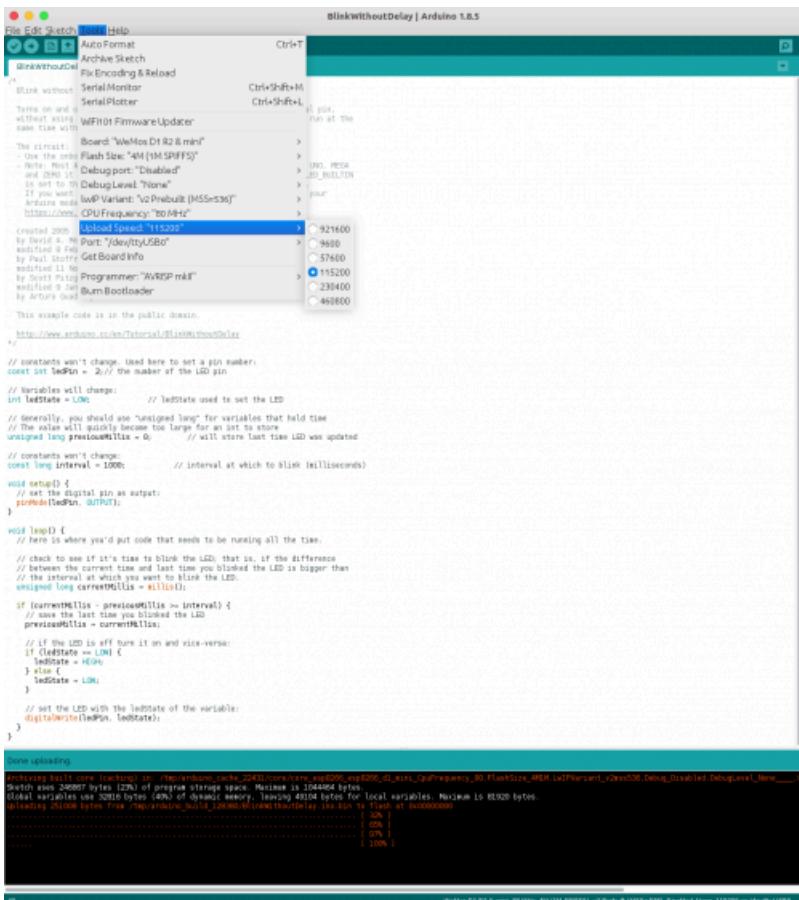


Figure 153: Selecting flashing speed

Troubleshooting Access Denied error

In case of the Linux and Mac, depending on the security context you run your Arduino(C) IDE, you may experience *Access Denied* error. There are several workarounds to this problem, starting from running Arduino IDE with `sudo` credentials (not recommended) through creating `udev` rules (advanced) to simplest - providing credentials on-demand and ad-hoc. The disadvantage of this method is you must usually run those command every time you reboot OS or reconnect your board, yet is simplest to handle (figure 154 and 155):

```
sudo usermod -a -G dialout $USER
```

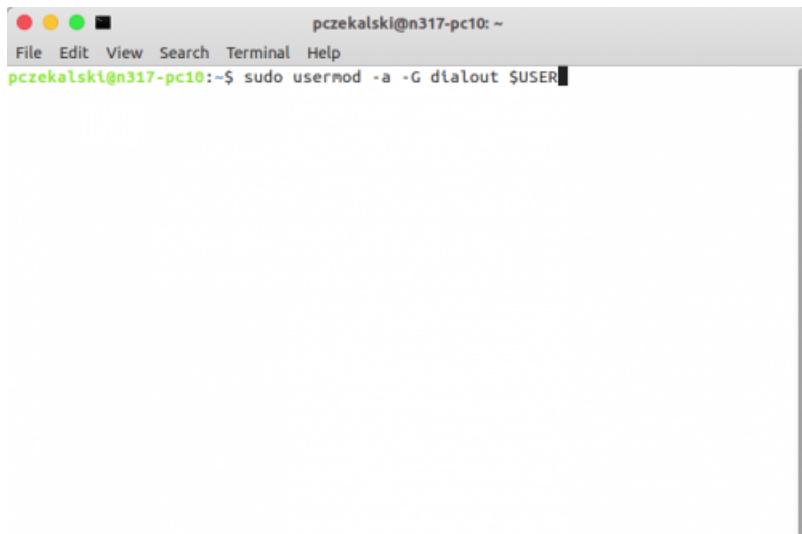
```
sudo chmod a+r /dev/ttyUSB0
```

4. IoT Hardware overview



A screenshot of a Linux terminal window. The title bar shows the user's name and the path: "pczekalski@n317-pc10: ~/Downloads/arduino-1.8.5". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command line shows the user running "sudo usermod -a -G dialout \$USER". The terminal window has a light gray background and a dark gray border.

Figure 154: Linux serial port access denied troubleshooting, part 1



A screenshot of a Linux terminal window. The title bar shows the user's name and the path: "pczekalski@n317-pc10: ~". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command line shows the user running "sudo usermod -a -G dialout \$USER". The terminal window has a light gray background and a dark gray border.

Figure 155: Linux serial port access denied troubleshooting, part 2

A need to run one or two of the commands above strongly depends on the operating system configuration.

In case of the Windows OS, marking application to “Run as Administrator” may help to remove permission related errors (figure 156):

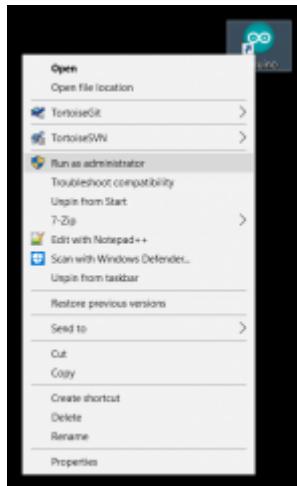


Figure 156: Running Arduino IDE as Administrator in Windows OS

Preparing ESP 32 development environment with Arduino IDE

ESP 32 SoC is a continuation and extension to the ESP 8266 SOCs. At the moment of writing this manual, installation of the ESP 32 development environment is not supported via integrated Board Manager of the Arduino(C) IDE, as presented above in the ESP8266 section. Uploading your binary to the ESP 32 chip via USB to serial converter requires Python as the ESP 32 flashing tool is written in the form of the Python script. In case you're aware of what Python programming language is and how to install it on your machine, please refer to the Python website: www.python.org⁸⁸⁾ and install Python before continuing.

Installing ESP 32 core for Arduino IDE

Depending on the operating system you use, there is necessary to perform some steps to obtain fully functional ESP 32 development environment for Arduino(C) IDE. ESP 32 core for Arduino is maintained by the community, and the latest version is available here (along with up-to-date installation guide): <https://github.com/espressif/arduino-esp32>⁸⁹⁾. Steps tend to have the same meaning, but tools to obtain the result are different for different operating systems.

Linux

A guide for the most popular distros (Ubuntu/Debian) is presented below.

Modify user credentials

This step is optional, see related ESP 8266 section. Run following command in the terminal to add a user to the `dialout` group:

```
sudo usermod -a -G dialout $USER
```

Download and install git client

Run following command in the terminal:

4. IoT Hardware overview

```
sudo apt-get install git
```

Download and install python tools and packages

Run following commands in the terminal:

```
wget https://bootstrap.pypa.io/get-pip.py && \
sudo python get-pip.py && \
sudo pip install pyserial
```

Create destination folders and clone repository

Assuming your Arduino IDE is installed in your home directory, (`~/Arduino`), run the following code in terminal:

```
mkdir -p ~/Arduino/hardware/espressif && \
cd ~/Arduino/hardware/espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32
```

Pull depending modules and tools

Run following commands in the terminal:

```
git submodule update --init --recursive && \
cd tools && \
python2 get.py
```

Then start Arduino IDE.

Windows

Installing ESP 32 core for Arduino requires Git client, both GUI and bash as well as command line operations.

Install Git and clone repository

You can download and install Git here: [https://git-scm.com/download/win^{90\)}](https://git-scm.com/download/win). Once installed, choose *Clone Existing Repository* (figure 157):

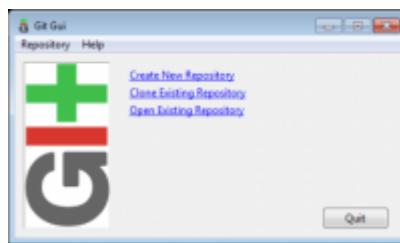


Figure 157: Git GUI for Windows

Use ESP 32 core for Arduino repository address as source:

```
https://github.com/espressif/arduino-esp32.git
```

The destination folder depends on where you've installed Arduino IDE and your Windows

4.4. Espressif SoC Overview

user name. The common location is:

```
C:/Users/[YOUR_USER_NAME]/Documents/Arduino/hardware/espressif/esp32
```

note, you do not need to create this folder manually. If you install fresh copy of the Arduino IDE, perhaps there will be no hardware subfolder, but Git GUI will create remaining path for you. Once entered *Source* and *Destination*, click [**Clone**] (figure 158). Cloning may take a while.

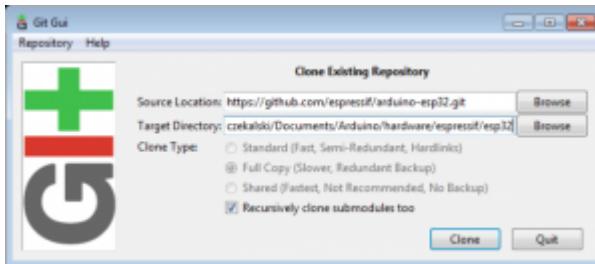


Figure 158: Configure Git GUI for cloning

Pull depending modules

Use Git Bash command line (not a Windows command line!) to install dependencies (figure 159). Change directory to the esp32 git local copy then run submodules install:

```
cd Documents/Arduino/hardware/espressif/esp32/
```

```
git submodule update --init --recursive
```

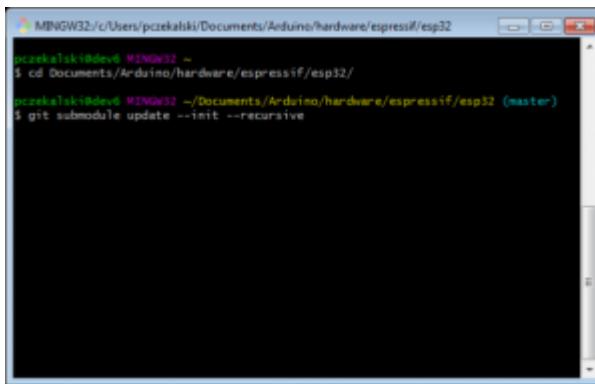


Figure 159: Updating sub modules using Git Bash

Download ESP32 tools

Open Windows command line (not a Git Bash command line!), navigate to the tools folder (figure 160):

4. IoT Hardware overview

```
cd C:/Users/[YOUR_USER_NAME]/Documents/Arduino/hardware/espressif/esp32/tools
```

then run:

```
get.exe
```

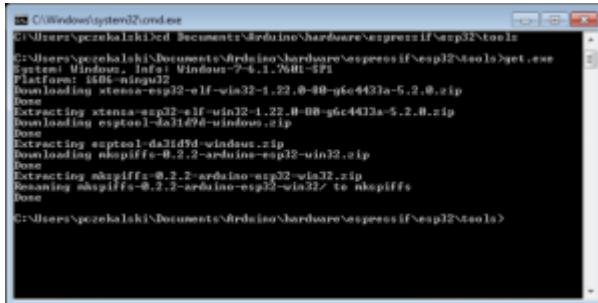


Figure 160: Get ESP 32 tools

Mac OS

Instruction for the Mac is similar to this for Linux. They require terminal to issue a set of commands to install ESP32 development kit and tools.

Create destination folders and clone source

```
mkdir -p ~/Documents/Arduino/hardware/espressif && \
cd ~/Documents/Arduino/hardware/espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32
```

Pull depending modules and tools

Run following commands in the terminal:

```
git submodule update --init --recursive && \
cd tools && \
python get.py
```

Then start Arduino IDE. *Troubleshooting*

If you get the following error during installation:

```
xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools), missing x
```

then install command line developer tools using:

```
xcode-select --install
```

Configure project to compile for ESP 32 Once ESP 32 platform is installed, start Arduino IDE and you should see new board definitions, similar to those presented on the figure 161:



Figure 161: ESP 32 boards in Arduino IDE

Refer to your board vendor for information about compatible configurations and setting up upload parameters. Detailed description and information on selecting communication port and upload speed is presented in the ESP 8266 section, above.

ESP AT networking

Flashing AT firmware

To use the ESP8266 chip as a modem (figure 162) we must first load the appropriate AT-command firmware.

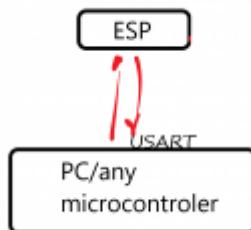


Figure 162: ESP8266 as a modem

4. IoT Hardware overview

Download software:

1. Download the latest ESP Flash Download Tool (v3.6.4 at the time of writing) from here.
2. Download the latest AT release from here or here.

Flashing procedure:

- Detect ESP8266 module parameters. Start ESP Flash Download Tool ("ESPFlashDownloadTool_v3.6.4"), set the COM port corresponding to your programmer, then click the START button in order to detect the specs of the board. After detection, you should see something like this (figure 163):

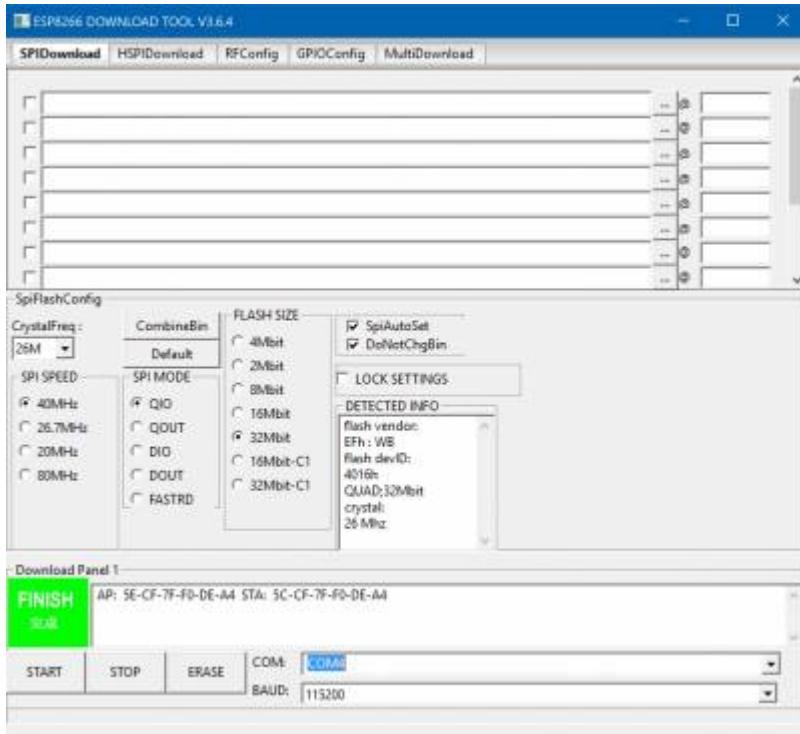


Figure 163: Programming ESP8266 - detected parameters

- Gather information. Make a note of the flash memory size. In this example, we have a 32Mbit flash.
- Load the correct size of combined AT binary firmware file ("*.bin") and set the offset as 0x0, you should see something like this (figure 164):

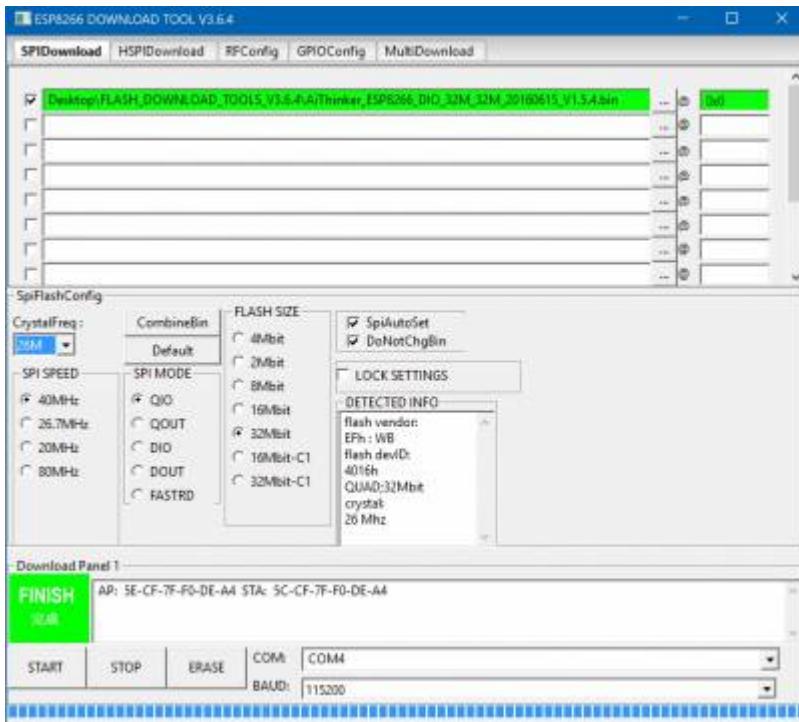


Figure 164: Programming ESP8266 - setting proper image file

- Then, click the START button and wait until the flashing process is over.

Reflashing procedure:

If necessary, to restore the original firmware:

- Detect ESP8266 module parameters. Start ESP Flash Download Tool ("ESPFlashDownloadTool_v3.6.4"), set the COM port corresponding to your programmer, then click the START button in order to detect the specs of the board. After detection, you should see something like this (figure 165):

4. IoT Hardware overview

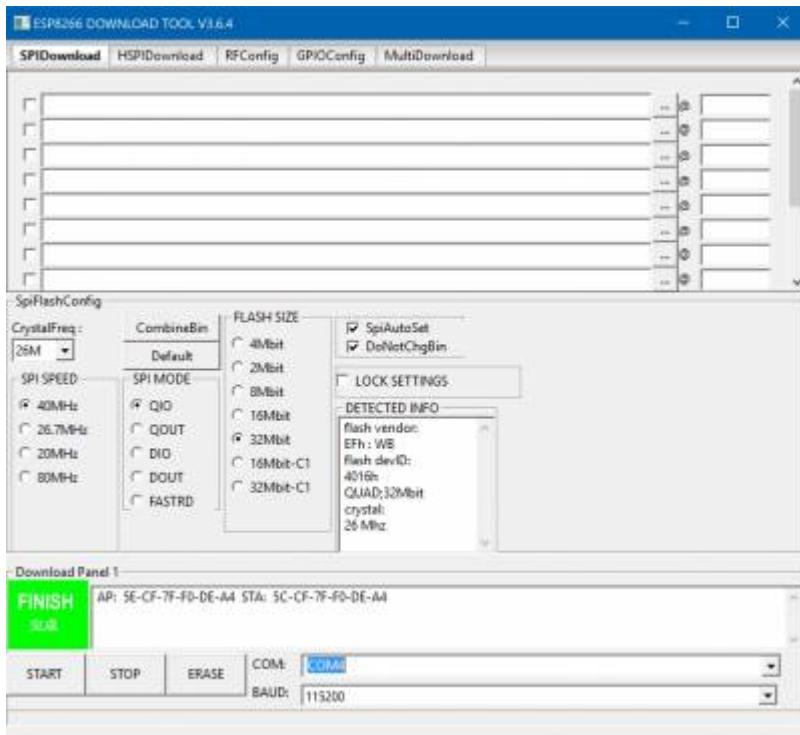


Figure 165: Programming ESP8266 - detected parameters

- From the downloaded AT firmware folder, open the "readme.txt" file containing the information for flashing the firmware. Inside the file, there should be a "BOOT MODE" section, as follows:

```
# BOOT MODE
## download
### Flash size 8Mbit: 512KB+512KB
boot_v1.2+.bin          0x00000
userl.1024.new.2.bin     0x01000
esp_init_data_default.bin 0xfc000 (optional)
blank.bin                0x7e000 & 0xfe000

### Flash size 16Mbit: 512KB+512KB
boot_v1.5.bin           0x00000
userl.1024.new.2.bin     0x01000
esp_init_data_default.bin 0x1fc000 (optional)
blank.bin                0x7e000 & 0x1fe000

### Flash size 16Mbit-C1: 1024KB+1024KB
boot_v1.2+.bin          0x00000
userl.2048.new.5.bin     0x01000
esp_init_data_default.bin 0x1fc000 (optional)
blank.bin                0xfe000 & 0x1fe000
```

```
## Flash size 32Mbit: 512KB+512KB
boot_v1.2+.bin          0x00000
user1.1024.new.2.bin     0x01000
esp_init_data_default.bin 0x3fc000 (optional)
blank.bin                0x7e000 & 0x3fe000

## Flash size 32Mbit-C1: 1024KB+1024KB
boot_v1.2+.bin          0x00000
user1.2048.new.5.bin     0x01000
esp_init_data_default.bin 0x3fc000 (optional)
blank.bin                0xfe000 & 0x3fe000
```

- Indicate - correct for your ESP8266 flash size - firmware files & addresses. The firmware is broken down into several files. They need to be provided to the ESP Flash Download Tool, together with the corresponding addresses found in the readme.txt file above. For our ESP8266 example it should look like this (figure 166)

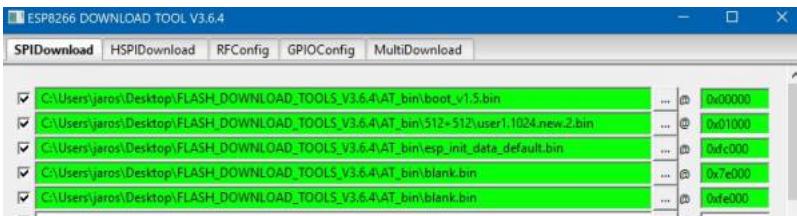


Figure 166: Programming ESP8266 - reflashing settings

- Then, click the START button and wait until the flashing process is over.

Basic ESP8266 networking

After uploading AT firmware and connecting module to PC, we can use ESP8266 as a modem with simple **AT** commands.

We can connect ESP8266 to PC with TTL-Serial-to-USB adapter, or we can use any microcontroller with a serial interface. The default baud rate settings are 115200,N,8,1. Next from any terminal type command:

```
AT
```

and press enter. If you get **OK**, the ESP8266 module is ready to use. Let's try out some other commands. For example, let's figure out exactly what firmware version we're dealing with. To do that, we'll use the following command:

```
AT+GMR
```

As a Wifi device ESP8266 can connect to the network in such modes:

- mode 1 - client mode - the ESP8266 connecting to an existing wireless network,
- mode 2 - access point mode (AP) - other wireless network devices can be connected to the ESP8266,

4. IoT Hardware overview

- mode 3 - dual mode (router) - the ESP8266 act as an access point and connect at the same time to existing wireless network.

By default, the ESP8266's stock firmware is set to AP mode. If you'd like to confirm that, send the following command:

```
AT+CWMODE?
```

You should get this response: +CWMODE:2, where 2 corresponds to AP mode. To switch ESP8266 to client device mode we use the following command:

```
AT+CWMODE=1
```

Now we can scan the airwaves for all WiFi access points in range. To do that, we send:

```
AT+CWLAP
```

Then the ESP8266 will return a list of all the access points in range. In with each line will be item consisting of the security level of the access point, the network name, the signal strength, MAC address, and wireless channel used. Possible security levels of the access point <0-4> mean:

- 0 - open
- 1 - WEP
- 2 - WPA_PSK
- 3 - WPA2_PSK
- 4 - WPA_WPA2_PSK

Now we can connect to the available access point using proper "ssid_name" and "correct_password" with the command:

```
AT+CWJAP="ssid_name","correct_password"
```

If everything is OK, the ESP8266 will answer:

```
WIFI CONNECTED  
WIFI GOT IP  
OK
```

It means that ESP8266 is connected to the chosen AP and got a proper IP address. To check what the assigned address is we send the command:

```
AT+CIFSR
```

To set up ESp8266 to behave both as a WiFi client as well as a WiFi Access point.

```
AT+CWMODE=3
```

ESP Network Layers

Programming networking services with ESP requires a connection on the networking layer between parties, mostly TCP.

ESP SoC can act as Access Point (AP): a device you connect to, like you connect a notebook to the Internet router, and as a client: ESP then behaves like any wifi enabled device, i.e. tablet or mobile phone, connecting to the Internet infrastructure. Interestingly, ESP 8366 SoC can act simultaneously in both modes at once, even, if it has only one WiFi interface!

Below there is sample code, how to implement both modes, using ESP libraries that came during installation of the development environment for Arduino IDE.

The third example shows how to send and receive a UDP packet while in client mode. It is the full solution to connect ESP to the NTP (Network Time Protocol) server to obtain current date and time from the Internet.

Last examples show, how to make a handy WiFi scanner showing available networks nearby.

ESP8266 AP (Access Point) mode

This sketch based on standard example demonstrates how to program ESP8266 in AP mode:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

/* Set these variables to your desired credentials. */
const char *ssid = "APmode";
const char *password = "password";

ESP8266WebServer server(80);

void hRoot() {
    server.send(200, "text/html", "<h1>You are connected</h1>");
}

/* Initialization */
void setup() {
    delay(1500);
    /* You can remove the password parameter if you want the AP to be open. */
    WiFi.softAP(ssid, password);

    IPAddress myIP = WiFi.softAPIP();

    server.on("/", hRoot);
    server.begin();
}

void loop() {
    server.handleClient();
}
```

ESP8266 client mode

This sketch (standard example) demonstrates how to program ESP8266 in client mode:

4. IoT Hardware overview

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

ESP8266WiFiMulti WiFiMulti;

void setup() {
    delay(1000);

    // We start by connecting to a WiFi network
    WiFi.mode(WIFI_STA);
    WiFiMulti.addAP("SSID", "password");

    while(WiFiMulti.run() != WL_CONNECTED) {
        delay(500);
    }
    delay(500);
}

void loop() {
    const uint16_t port = 80;
    const char * host = "192.168.1.1"; // ip or dns

    // Use WiFiClient class to create TCP connections
    WiFiClient client;

    if (!client.connect(host, port)) {
        delay(5000);
        return;
    }

    // This will send the request to the server
    client.println("Send this data to server");

    //read back one line from server
    String line = client.readStringUntil('\r');
    Serial.println(line);

    Serial.println("closing connection");
    client.stop();

    Serial.println("wait 5 sec...");
    delay(5000);
}
```

ESP8266 and UDP

This sketch (based on standard example) demonstrates how to program ESP8266 as NTP client using UDP packets (send and receive):

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

char ssid[] = "*****"; // your network SSID (name)
char pass[] = "*****"; // your network password
```

4.4. Espressif SoC Overview

```
unsigned int localPort = 2390;           // local port to listen for UDP packets

// NTP servers
IPAddress ntpServerIP; // 0.pl.pool.ntp.org NTP server address
const char* ntpServerName[] =
    {"0.pl.pool.ntp.org","1.pl.pool.ntp.org","2.pl.pool.ntp.org","3.pl.pool.ntp.org"};

const int timeZone = 1; //Central European Time
int servernbr=0;

// NTP time stamp is in the first 48 bytes of the message
const int NTP_PACKET_SIZE = 48;

//buffer to hold incoming and outgoing packets
byte packetBuffer[ NTP_PACKET_SIZE];

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp;

void setup()
{
    Serial.begin(115200);
    Serial.println();

    Serial.print("Connecting to ");
    Serial.println(ssid);

    // WiFi.persistent(false);
    WiFi.mode(WIFI_OFF);
    delay(2000);

    // We start by connecting to a WiFi network
    WiFi.mode(WIFI_STA);
    delay(3000);
    WiFi.begin(ssid, pass);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");

    Serial.println("WiFi connected");
    Serial.println("DHCP assigned IP address: ");
    Serial.println(WiFi.localIP());

    Serial.println("Starting UDP");
    udp.begin(localPort);
    Serial.print("Local port: ");
    Serial.println(udp.localPort());

    // first ntp server
    servernbr = 0;
}

void loop()
{
```

4. IoT Hardware overview

```
//get a random server from the pool

WiFi.hostByName(ntpServerName[servernbr], ntpServerIP);
Serial.print(ntpServerName[servernbr]);
Serial.print(":");
Serial.println(ntpServerIP);

sendNTPpacket(ntpServerIP); // send an NTP packet to a time server
// wait to see if a reply is available
delay(1000);

int cb = udp.parsePacket();
if (!cb) {
    Serial.println("no packet yet");
    if (servernbr = 5) {
        servernbr =0;
    }
    else {
        servernbr++;
    }
}
else {
    Serial.print("packet received, length=");
    Serial.println(cb);
    // We've received a packet, read the data from it
    udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, esxtract the two words:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
    Serial.print("Seconds since Jan 1 1900 = " );
    Serial.println(secsSince1900);

    // now convert NTP time into everyday time:
    Serial.print("Unix time = ");
    // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // subtract seventy years:
    unsigned long epoch = secsSince1900 - seventyYears;
    // print Unix time:
    Serial.println(epoch);

    // print the hour, minute and second:
    // UTC is the time at Greenwich Meridian (GMT)
    Serial.print("The UTC time is ");
    // print the hour (86400 equals secs per day)
    Serial.print((epoch % 86400L) / 3600);
    Serial.print(':');
    if ( ((epoch % 3600) / 60) < 10 ) {
        // In the first 10 minutes of each hour, we'll want a leading '0'
        Serial.print('0');
    }
}
```

```

// print the minute (3600 equals secs per minute)
Serial.print((epoch % 3600) / 60);
Serial.print(':');
if ( (epoch % 60) < 10 ) {
    // In the first 10 seconds of each minute, we'll want a leading '0'
    Serial.print('0');
}
Serial.println(epoch % 60); // print the second
}
// wait ten seconds before asking for the time again
delay(10000);
}

// send an NTP request to the time server at the given address
void sendNTPpacket(IPAddress& address)
{
    Serial.print("sending NTP packet to: ");
    Serial.println( address );
    // set all bytes in the buffer to 0
    memset(packetBuffer, 0, NTP_PACKET_SIZE);
    // Initialize values needed to form NTP request
    // (see URL above for details on the packets)
    packetBuffer[0] = 0b1100011; // LI, Version, Mode
    packetBuffer[1] = 0;        // Stratum, or type of clock
    packetBuffer[2] = 6;        // Polling Interval
    packetBuffer[3] = 0xEC;     // Peer Clock Precision
    // 8 bytes of zero for Root Delay & Root Dispersion
    packetBuffer[12] = 49;
    packetBuffer[13] = 0x4E;
    packetBuffer[14] = 49;
    packetBuffer[15] = 52;

    // all NTP fields have been given values, now
    // you can send a packet requesting a timestamp:
    udp.beginPacket(address, 123); //NTP requests are to port 123
    udp.write(packetBuffer, NTP_PACKET_SIZE);
    udp.endPacket();
}

```

4.4.4. ESP8266 Wifi Scanner

This sketch demonstrates how to scan WiFi networks. ESP8266 is programmed in access point mode. All found WiFi networks will be printed in TTY serial window.

```

#include "ESP8266WiFi.h"

void setup() {
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect
    // from an AP if it was previously connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

```

4. IoT Hardware overview

```
}

void loop() {
    Serial.println("scan start");

    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("scan done");
    if (n == 0)
        Serial.println("no networks found");
    else
    {
        Serial.print(n);
        Serial.println(" networks found");
        for (int i = 0; i < n; ++i)
        {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(": ");
            Serial.print(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.print(")");
            Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE) ? " ":"*");
            delay(10);
        }
    }
    Serial.println("");

    // Wait a bit before scanning again
    delay(5000);
}
```

4.4.5. ESP32 Wifi Scanner

There are many different development software and tools which can be used for ESP32 programming⁹¹:

- Arduino COre (C++)
- ESP-IDF (Espressif IoT Development Framework)
- Mongoose OS
- MicroPython
- Simba Embedded Programming Platform
- Lua
- JacvaScript
- mruby
- BASIC

Of course, for programming ESP32 We can use all the previously described Arduino examples for sensors and actuators. But in our example, we will focus on programming in ESP-IDF, as this is the native Development Platform for ESP32. A detailed description of the installation of the development environment can be found [here](#).

4.4. Espressif SoC Overview

This example shows how to use the All Channel Scan or Fast Scan to connect to a Wi-Fi network. In the Fast Scan mode, the scan will stop as soon as the first network matching the SSID is found. In this mode, an application can set the threshold for the authentication mode and the Signal strength. Networks that do not meet the threshold requirements will be ignored. In the All Channel Scan mode, the scan will end after all the channels are scanned, and the connection will start with the best network. The networks can be sorted based on Authentication Mode or Signal Strength. The priority for the Authentication mode is: WPA2 > WPA > WEP > Open.

```
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "esp_event_loop.h"
#include "nvs_flash.h"

/*Set the SSID and Password via "make menuconfig"*/
#define DEFAULT_SSID CONFIG_WIFI_SSID
#define DEFAULT_PWD CONFIG_WIFI_PASSWORD

#if CONFIG_WIFI_ALL_CHANNEL_SCAN
#define DEFAULT_SCAN_METHOD WIFI_ALL_CHANNEL_SCAN
#elif CONFIG_WIFI_FAST_SCAN
#define DEFAULT_SCAN_METHOD WIFI_FAST_SCAN
#else
#define DEFAULT_SCAN_METHOD WIFI_FAST_SCAN
#endif /*CONFIG_SCAN_METHOD*/

#if CONFIG_WIFI_CONNECT_AP_BY_SIGNAL
#define DEFAULT_SORT_METHOD WIFI_CONNECT_AP_BY_SIGNAL
#elif CONFIG_WIFI_CONNECT_AP_BY_SECURITY
#define DEFAULT_SORT_METHOD WIFI_CONNECT_AP_BY_SECURITY
#else
#define DEFAULT_SORT_METHOD WIFI_CONNECT_AP_BY_SIGNAL
#endif /*CONFIG_SORT_METHOD*/

#if CONFIG_FAST_SCAN_THRESHOLD
#define DEFAULT_RSSI CONFIG_FAST_SCAN_MINIMUM_SIGNAL
#if CONFIG_EXAMPLE_OPEN
#define DEFAULT_AUTHMODE WIFI_AUTH_OPEN
#elif CONFIG_EXAMPLE_WEP
#define DEFAULT_AUTHMODE WIFI_AUTH_WEP
#elif CONFIG_EXAMPLE_WPA
#define DEFAULT_AUTHMODE WIFI_AUTH_WPA_PSK
#elif CONFIG_EXAMPLE_WPA2
#define DEFAULT_AUTHMODE WIFI_AUTH_WPA2_PSK
#else
#define DEFAULT_AUTHMODE WIFI_AUTH_OPEN
#endif
#else
#define DEFAULT_RSSI -127
#define DEFAULT_AUTHMODE WIFI_AUTH_OPEN
#endif /*CONFIG_FAST_SCAN_THRESHOLD*/

static const char *TAG = "scan";

static esp_err_t event_handler(void *ctx, system_event_t *event)
```

4. IoT Hardware overview

```
{  
    switch (event->event_id) {  
        case SYSTEM_EVENT_STA_START:  
            ESP_LOGI(TAG, "SYSTEM_EVENT_STA_START");  
            ESP_ERROR_CHECK(esp_wifi_connect());  
            break;  
        case SYSTEM_EVENT_STA_GOT_IP:  
            ESP_LOGI(TAG, "SYSTEM_EVENT_STA_GOT_IP");  
            ESP_LOGI(TAG, "Got IP: %s\n",  
                    ip4addr_ntoa(&event->event_info.got_ip.ip_info.ip));  
            break;  
        case SYSTEM_EVENT_STA_DISCONNECTED:  
            ESP_LOGI(TAG, "SYSTEM_EVENT_STA_DISCONNECTED");  
            ESP_ERROR_CHECK(esp_wifi_connect());  
            break;  
        default:  
            break;  
    }  
    return ESP_OK;  
}  
  
/* Initialize Wi-Fi as sta and set scan method */  
static void wifi_scan(void)  
{  
    tcpip_adapter_init();  
    ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));  
  
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();  
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));  
    ESP_LOGI(TAG, DEFAULT_SSID);  
    ESP_LOGI(TAG, DEFAULT_PWD);  
    wifi_config_t wifi_config = {  
        .sta = {  
            .ssid = DEFAULT_SSID,  
            .password = DEFAULT_PWD,  
            .scan_method = DEFAULT_SCAN_METHOD,  
            .sort_method = DEFAULT_SORT_METHOD,  
            .threshold.rssi = DEFAULT_RSSI,  
            .threshold.authmode = DEFAULT_AUTHMODE,  
        },  
    };  
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));  
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));  
    ESP_ERROR_CHECK(esp_wifi_start());  
}  
  
void app_main()  
{  
    // Initialize NVS  
    esp_err_t ret = nvs_flash_init();  
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES) {  
        ESP_ERROR_CHECK(nvs_flash_erase());  
        ret = nvs_flash_init();  
    }  
    ESP_ERROR_CHECK( ret );  
  
    wifi_scan();  
}
```

4.4. Espressif SoC Overview

To properly set up Station mode, it is necessary to enter SSID and password. To enter these values, before compiling the program, run the command:

```
make menuconfig
```

and then

```
make all
```

or

```
make flash
```

ESP Application Layer

ESP8266 Samples

ESP8266 Web Server Sample

This example can be compiled in Arduino IDE. It allows through the website to change the output state of PIN 4 and PIN 5⁹²⁾. We can connect LED to these pins and change its state remotely using a web browser. Before compiling this example it is necessary to change this two lines, to enable the module to connect to the WIFI network:

```
const char* ssid = "... put here your own SSID name ...";  
const char* password = "... put here your SSID password.. ";
```

Now please check in the serial console the ESp8266 IP number and connect with any browser to address: http://esp8266_ipnumber

```
// Load Wi-Fi library  
#include <ESP8266WiFi.h>  
  
// Replace with your network credentials  
const char* ssid = "... put here your own SSID name ...";  
const char* password = "... put here your SSID password.. ";  
  
// Set web server port number to 80  
WiFiServer server(80);  
  
// Variable to store the HTTP request  
String header;  
  
// Auxiliar variables to store the current output state  
String gpio5State = "off";  
String gpio4State = "off";  
  
// Assign output variables to GPIO pins  
const int gpiopin5 = 5;  
const int gpiopin4 = 4;  
  
void setup() {
```

4. IoT Hardware overview

```
Serial.begin(115200);
// Initialize the output variables as outputs
pinMode(gpiopin5, OUTPUT);
pinMode(gpiopin4, OUTPUT);
// Set outputs to LOW
digitalWrite(gpiopin5, LOW);
digitalWrite(gpiopin4, LOW);

// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("ESP8266 IP address: ");
Serial.println(WiFi.localIP());
server.begin();
}

void loop(){
    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,
        Serial.println("New Client."); // print a message out in the serial port
        String currentLine = ""; // make a String to hold incoming data from the client
        while (client.connected()) { // loop while the client's connected
            if (client.available()) { // if there's bytes to read from the client,
                char c = client.read(); // read a byte, then
                Serial.write(c); // print it out the serial monitor
                header += c;
                if (c == '\n') { // if the byte is a newline character
                    // if the current line is blank, you got two newline characters in a row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0) {
                        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
                        // and a content-type so the client knows what's coming, then a blank line:
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                        client.println();

                        // turns the GPIOs on and off
                        if (header.indexOf("GET /5/on") >= 0) {
                            Serial.println("GPIO 5 on");
                            gpiopin5State = "on";
                            digitalWrite(gpiopin5, HIGH);
                        } else if (header.indexOf("GET /5/off") >= 0) {
                            Serial.println("GPIO 5 off");
                            gpiopin5State = "off";
                            digitalWrite(gpiopin5, LOW);
                        } else if (header.indexOf("GET /4/on") >= 0) {
                            Serial.println("GPIO 4 on");
                            gpiopin4State = "on";
                        }
                    }
                }
            }
        }
    }
}
```

```

        digitalWrite(gpiopin4, HIGH);
    } else if (header.indexOf("GET /4/off") >= 0) {
        Serial.println("GPIO 4 off");
        gpio4State = "off";
        digitalWrite(gpiopin4, LOW);
    }

    // Display the HTML web page
    client.println("<!DOCTYPE html><html>");
    client.println("<head><meta name=\"viewport\""
                  "content=\"width=device-width, initial-scale=1\">");
    client.println("<link rel=\"icon\" href=\"data:,\">");
    // CSS to style the on/off buttons
    // Feel free to change the background-color and
    // font-size attributes to fit your preferences
    client.println("<style>html { font-family: Helvetica; display:"
                  "inline-block; margin: 0px auto; text-align: center; }");
    client.println(".button { background-color: #195B6A;"
                  "border: none; color: white; padding: 16px 40px;}");
    client.println("text-decoration: none;
                  font-size: 30px; margin: 2px; cursor: pointer; }");
    client.println(".button2 {background-color: #77878A;}</style></head>");

    // Web Page Heading
    client.println("<body><h1>ESP8266 Web Server</h1>");

    // Display current state, and ON/OFF buttons for GPIO 5
    client.println("<p>GPIO 5 - State " + gpio5State + "</p>");
    // If the output5State is off, it displays the ON button
    if (gpio5State=="off") {
        client.println("<p><a href=\"/5/on\"><button"
                      "class=\"button\">ON</button></a></p>\"");
    } else {
        client.println("<p><a href=\"/5/off\"><button"
                      "class=\"button button2\">OFF</button></a></p>\"");
    }

    // Display current state, and ON/OFF buttons for GPIO 4
    client.println("<p>GPIO 4 - State " + gpio4State + "</p>");
    // If the output4State is off, it displays the ON button
    if (gpio4State=="off") {
        client.println("<p><a href=\"/4/on\"><button"
                      "class=\"button\">ON</button></a></p>\"");
    } else {
        client.println("<p><a href=\"/4/off\"><button"
                      "class=\"button button2\">OFF</button></a></p>\"");
    }
    client.println("</body></html>");

    // The HTTP response ends with another blank line
    client.println();
    // Break out of the while loop
    break;
} else { // if you got a newline, then clear currentLine
    currentLine = "";
}
} else if (c != '\r') { // if you got anything
    // else but a carriage return character,

```

4. IoT Hardware overview

```
        currentLine += c;      // add it to the end of the currentLine
    }
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}
```

After connecting with web browser to ESP8266 there will be such web page (figure 167), and we can change the input status of PIN 4 and 5 simply by pressing the appropriate button



Figure 167: ESP8266 web page

ESP32 Samples

ESP32 "Hello World"

This is simple program printing "Hello World" and it is written in Espressif IoT Development Framework

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"

void app_main()
{
    printf("Hello world!\n");

    /* Print chip information */
    esp_chip_info_t chip_info;
    esp_chip_info(&chip_info);
    printf("This is ESP32 chip with %d CPU cores, WiFi%s%s, ",
           chip_info.cores,
           (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
           (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
}
```

```

printf("silicon revision %d, ", chip_info.revision);

printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
       (chip_info.features & CHIP_FEATURE_EMB_FLASH) ?
           "embedded" : "external");

for (int i = 10; i >= 0; i--) {
    printf("Restarting in %d seconds...\n", i);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
printf("Restarting now.\n");
fflush(stdout);
esp_restart();
}
}

```

ESP32 Web Server

This example of ESP32 programming in Arduino and shows how to implement simple www server.

First we do a little initialisation

```

//##### LIBRARIES #####
#include <WiFi.h>
#include <ESP32WebServer.h>
#include <WiFiClient.h>
//##### VARIABLES #####
String webpage = ""; // General purpose variable to hold HTML code
const char* ssid      = "ssdi";      // WiFi SSID
const char* password  = "password"; // WiFi Password

int status = WL_IDLE_STATUS;
int curr_index;
String SensorStatusBME;

// Site's Main Title
String siteheading      = "ESP32 Webserver";
// Sub-heading for all pages
String subheading       = "Sensor Readings";
// Appears on the tab of a Web Browser
String sitetitle        = "ESP32 Webserver";
// A foot note e.g. "My Web Site"
String yourfootnote     = "ESP32 Webserver Demonstration";
// Version of your Website
String siteversion       = "v1.0";

```

Then we must implement the main www server activities. Mind, to access the server from outside of your network WiFi (LAN) e.g. on port 80 when in NAT mode, add a rule on your router that forwards a connection request to http://your_network_WAN_address:80 to http://your_network_LAN_address:80 and then you can access your ESP server from virtually anywhere on the Internet.

```
ESP32WebServer server(80);
```

4. IoT Hardware overview

```
void setup()
{
    Serial.begin(115200); // initialize serial communications
    curr_index = 1;

    time_to_measure = millis();

    StartWiFi(ssid, password);
    StartTime();
    //-----
    Serial.println("Use this URL to connect: http://" + WiFi.localIP().toString() + "/");
    // If the user types at their browser
    // http://192.168.0.100/ control is passed here and then
    // to user_input, you get values for your program...
    server.on("/", homepage);
    // If the user types at their browser
    // http://192.168.0.100/homepage or via menu control
    // is passed here and then to the homepage, etc
    server.on("/homepage", homepage);

    // If the user types something that is not supported, say so
    server.onNotFound(handleNotFound);
    // Start the webserver
    server.begin(); Serial.println(F("Webserver started..."));
}

void handleNotFound() {
    String message = "The request entered could not be found,
                     please try again with a different option\n";
    server.send(404, "text/plain", message);
}

void homepage() {
    append_HTML_header();
    webpage += "<P class='style2'>This is the server home page</p><br>";
    webpage += "<p class='style2'>";
    webpage += "This is sample webpage";
    webpage += "</p><br>";
    webpage += "<p>This page was displayed on : " + GetTime() + " Hr</p>";
    String Uptime = (String(millis() / 1000 / 60 / 60)) + ":";
    Uptime += ((millis() / 1000 / 60 % 60) < 10) ? "0" +
              String(millis() / 1000 / 60 % 60) :
              String(millis() / 1000 / 60 % 60) + ":";
    Uptime += ((millis() / 1000 % 60) < 10) ? "0" +
              String(millis() / 1000 % 60) :
              String(millis() / 1000 % 60);
    webpage += "<p>Uptime: " + Uptime + "</p>";
    append_HTML_footer();
    server.send(200, "text/html", webpage);
}

void page1() {

    append_HTML_header();
    webpage += "<H3>This is the server Page-1</H3>";
    webpage += "<P class='style2'>This is the server home page</p>";
    webpage += "<p class='style2'>";
```

```
webpage += "This is sample 1 page";
webpage += "</p>";
append_HTML_footer();
server.send(200, "text/html", webpage);
}
```

next we must start Wifi :

```
void StartWiFi(const char* ssid, const char* password) {
    int connAttempts = 0;
    Serial.print(F("\r\nConnecting to: ")); Serial.println(String(ssid));
    WiFi.begin(ssid, password);

    status = WiFi.status();

    while (status != WL_CONNECTED) {
        Serial.print(".");
        // wait 10 second for re-trying
        delay(10000);
        status = WiFi.status();
        Serial.println(status);
        if (connAttempts > 5) {
            Serial.println("Failed to connect to WiFi");
            // printWiFiStatus();
        }
        connAttempts++;
    }
    Serial.print(F("WiFi connected at: "));
    Serial.println(WiFi.localIP());
}
```

and last step is to implement main loop function:

```
void loop() {

    delay( 2000 );
    server.handleClient();

}
```

ESP32 Parallel programming

As it is known, some of the microcontrollers, in order to increase performance provide more than one core. ESP32 is one of them providing two physical cores. In practice, it means that the program developed can run simultaneously on both cores. Thereby it is possible to optimize some of the tasks in a way that they are not waiting for each other but running in parallel instead. This is the main advantage of parallel programming comparing to a sequential one. However, it requires both dedicated program control structures and hardware support.

At the time while this chapter is being written, the simplest way of developing a parallel code on ESP32 is via using FreeRTOS™ (<https://www.freertos.org/>), which is a widely used real-time library for different microcontrollers. The RTOS allows using most of the real-time and parallel programming features including semaphores, process assignments

4. IoT Hardware overview

to cores and more. The following code chunks explain how to apply the most useful parallel programming features.

Let's start with an example of blinking LED and Text output (based on material found here: https://exploreembedded.com/wiki>Hello_World_with_ESP32_Explained).

The first task is task1, that simply outputs a string "Hi there!" to default serial port with delay of 100ms i.e. 10 times per second:

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "driver/gpio.h"

#define BLINK_GPIO 13

void task1_SayHi(void * parameters)
{
    while(1)
    {
        printf("Hi there!\n");
        vTaskDelay(100 / portTICK_RATE_MS);
    }
}
```

- #include "freertos/FreeRTOS.h" and #include "freertos/task.h" - adds needed libraries of FreeRTOS™
- #define BLINK_GPIO 13 - defines output pin that will be used to switch on or off the LED
- portTICK_RATE_MS refers to constant portTICK_PERIOD_MS that is used to calculate real-time from the tick rate - with the resolution of one tick period;

The second task is to blink a LED with a period of 2 seconds (1 second on, 1 second off):

```
void task2_BlinkLED(void * parameters)
{
    gpio_pad_select_gpio(BLINK_GPIO);
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
    while(1) {
        /*Sets the LED low for one second*/
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(1000 / portTICK_RATE_MS);

        /*Sets the LED high for one second*/
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
}
```

Once both task functions are defined, they can be executed simultaneously:

```
void app_main()
{
    nvs_flash_init();
    xTaskCreate(&task1_SayHi, "task1_SayHi", 2000, NULL, 5, NULL);
    xTaskCreate(&task2_BlinkLED, "task2_BlinkLED", 2000, NULL, 5, NULL );
}
```

- `nvs_flash_init()` - initializes a non-volatile memory in flash memory, so it can be used by concurrent tasks
- `xTaskCreate` - creates a task, without specifying a core, on which it is executed, with rather low priority (5). More on parameters can be found here: http://esp32.info/docs/esp_idf/html/dd/d3c/group__xTaskCreate.html

In fact, the code does not run in parallel physically, it uses the full speed of the ESP32 that is far beyond human perception speed and shares the computation time between both tasks. Therefore for the human, it seems to be running in parallel. Each of the tasks uses Idle (defined by `vTaskDelay()`) time of the other task. Since both are simply the time slot is enough to complete.

To run the code physically in parallel it is necessary to assign task explicitly to the particular core, which requires a slight modification of the `main()` function:

```
void app_main()
{
    nvs_flash_init();
    xTaskCreatePinnedToCore(&task1_SayHi, "task1_SayHi", 2000, NULL, 5, NULL, 0);
    xTaskCreatePinnedToCore(&task2_BlinkLED, "task2_BlinkLED", 2000, NULL, 5, NULL, 1);
}
```

- `xTaskCreatePinnedToCore` - creates a task and assigns it to the particular core, on which it is executed. In this case, `task1_SayHi()` is assigned to core 0, while `task2_BlinkLED()` to core 1. For more information refer to http://esp32.info/docs/esp_idf/html/db/da4/task_8h.html#a25b035ac6b7809ff16c828be270e1431

While ESP32 provide two computing nodes, other devices like particular serial port or other peripherals are only single devices. In some cases, it might be needed to access those devices by multiple processes in a way that does not disturb the others. In a terminology of parallel programming, those "single" devices are called resources that need to be shared or simply shared resources. To share a resource it is necessary to have a signal that is available to all processes and that determines if the resource is available or not. Those signals are dedicated data structures and are called - semaphores. Depending on the particular platform they might represent a different data structure to address particular use case. RTOS support three main semaphore types - **Binary** (True/False), **Counting** (represents a queue) and **Mutex** (binary semaphore with priority). More details on each type and use examples might be found here <https://www.freertos.org/Inter-Task-Communication.html>. To explain the concept of resource sharing here a simple binary-semaphore example is provided. Example uses two SayHi tasks to share the same output device:

Since we need to define a semaphore at the beginning a setup function is also needed:

4. IoT Hardware overview

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "driver/gpio.h"

SemaphoreHandle_t xSemaphore = NULL;

void setup()
{
    vSemaphoreCreateBinary( xSemaphore );
}
```

Now it is possible to define the task functions and modify them in a way they use the same resource

```
void task1_SayHi(void * parameters)
{
    while(1)
    {
        /*check and waits for semaphore to be released for 100 ticks.
        If the semaphore is available it is taken / blocked */
        if( xSemaphoreTake( xSemaphore, ( TickType_t ) 100 ) == pdTRUE )
        {
            printf("TASK1: Hi there!\n");
            vTaskDelay(100 / portTICK_RATE_MS);
            xSemaphoreGive( xSemaphore );
        }
        else
        {
            //Does something else in case the semaphore is not available
        }
    }
}

void task2_SayHi(void * parameters)
{
    while(1)
    {
        /*check and waits for semaphore to be released for 100 ticks.
        If the semaphore is available it is taken / blocked */
        if( xSemaphoreTake( xSemaphore, ( TickType_t ) 100 ) == pdTRUE )
        {
            printf("TASK2: Hi there!\n");
            vTaskDelay(100 / portTICK_RATE_MS);
            xSemaphoreGive( xSemaphore );
        }
        else
        {
            //Does something else in case the semaphore is not available
        }
    }
}
```

Now both of the tasks are ready to be executed on the same or different cores as explained previously.

Please note that semaphore mechanism is a powerful tool to synchronize tasks, prioritize tasks or simply make sure that a single resource is used properly in a multi-task application.

4.5. Raspberry Pi Overview

Raspberry Pi (referred as RPi or RPI) and its clones i.e. Orange Pi, Banana Pi, Ordroid, Cubie, Olimex, are the class of devices located somewhere between low constraint IoT boards and regular PC/Mac machines.

While authors create this text, Raspberry Pi is a standard and reference solution for other manufacturers. However 3rd party manufacturers indeed offer more powerful solutions (regarding processor power, RAM size, connectivity capabilities, and integrated flash) usually by the cost of no support and not so well tailored operating system that lacks many features and present serious bugs.

Those devices technically are very close to smartphones and are far away from energy-efficient IoT solutions powered by a single battery that lasts for weeks or even years. They need DC, usually 5 or 12V and about 2-3W total, with external power adapter. It is still far less than even most efficient ultrabooks or PCs, requiring some 50-90W PSUs. They also use an operating system booted from storage like regular PCs - usually from flashed MicroSD card or embedded eMMC flash. The OS is mostly Linux based, but there do exist Microsoft Windows for certified Raspberry Pi devices. It is how this class of devices differ from, i.e. Arduino, where software is in the SoC model. The RPi and clones are holding a one-board solution that includes a processor, memory, storage slot, USB and networking. Many devices also offer hardware-based graphics acceleration, usually integrated with the processor core. Some devices like Orange Pi frequently provide an integrated flash for OS storage, so you do not necessarily need to boot and use an external flash like USB dongle or TransFlash card. The most common processor in this class of the devices is an ARM architecture family, in case of the RPI it is Broadcom (i.e. BCM2936), other manufacturers use, i.e. Exynos, All-Winner and Samsung manufactured processors. What is pretty similar to the low-power, constrained IoT boards, RPi and clones offer GPIO, and you can connect various sensors and expansion boards (called here "hats"), and you have a wide choice of operating systems and modules. You can also extend the hardware by connecting hats that offer to sense and to actuate but sometimes advanced computing like dedicated coprocessors or FPGA-based AI. Interestingly, their GPIOs usually provide (among others) popular protocols like I2C, SPI, One-Wire, so you can directly connect with many sensors known as designed for Arduino-compatible development boards. This way you can use those boards like conventional IoT devices with integrated networking capabilities, similar to, i.e. ESP chips.

What is much different from low constrained IoT devices is that they offer at least a command terminal you can connect to, and also most boards offer a capability to connect it to the external display via HDMI, analogue output or dedicated connector for LCD. They also provide the ability to interact with HID devices like regular keyboards, mouses, via USB but also wireless, i.e. using a Bluetooth connection. Of course, those features are dependent on operating systems. Manufacturers usually are trying to keep those development boards as small as possible, and it is a case that among high-end devices they also offer some constrained solution yet usually 50% smaller in size and power consumption (i.e. RPi zero). Many boards also offer dedicated camera connector.

Being so far from the low-power, constrained IoT devices does not exclude them from IoT

4. IoT Hardware overview

devices, however. They find their application everywhere, when there is a need for higher processing resources (i.e. voice recognition), high capacity and complex networking operations, i.e. gatewaying other devices to the Internet, convert networking protocols, implement software-based or hardware-assisted Artificial Intelligence, implementing rich user interface (GUI) where constrained devices are not powerful enough to fulfil the requirements yet there is still a limited power source, or there is not a need to set up a regular, PC-based solution, because of its cost. Most of the devices belonging to this class still can be switched to low power consumption modes, where low power means a dozen mA here.

On the other hand, most modern representatives of those devices are powered with multicore processors and large RAM and are powerful enough to replace the desktop computer in daily operations like web browsing, multimedia playback, software development and so on.

Raspberry Pi general information

Raspberry Pi Sensors

Raspberry Pi Drivers and driving

Raspberry Pi OS Guide

Programming fundamentals Raspbian OS

Programming fundamentals Windows 10 IOT Core

4.5.1. Raspberry Pi general information

The Raspberry Pi is a series of small single-board computers developed in the UK by the Raspberry Pi Foundation to promote modern computer science in schools and developing electronic communities. Adding the 40-pin GPIO connector to the computer board allows developers not only improving their programming skills but also open them new horizons in controlling processes and devices not available for desktop computers. According to the Raspberry Pi Foundation, the entire boards' sales in July 2017 has reached nearly 15 million units. The first generation of this new board type was developed and then released in February 2012 - *Raspberry Pi Model B*. Each Raspberry Pi board contains hardware modules which together makes it fully usable PC like a computer which size fits the typical *Credit Card* (85/56 mm) size and small power consumption <3.5 W. This makes this kind of single board computers one of the most popular in developers community. For today there exist thousands of hardware implementation projects available for users who want to learn the modern hardware and software controlling units within their projects. The general Raspberry Pi features are listed below.

Hardware

Hardware boards (depending on the manufactured model) contains interfaces: Ethernet, Bluetooth, Wi-Fi, USB, AUDIO, HDMI and GPIO ports⁹³⁾. The Raspberry Pi boards have evolved through several versions varying in memory capacity, System on Chips (SoC) and processor units. First generation models of Raspberry Pi used the Broadcom BCM2835 (ARMv6 architecture) based on 700 MHz ARM11176JZF-S processor and VideoCore IV graphics processing Unit (GPU). Models Pi 1 and B+ developed later uses

the five-point USB/Ethernet hub chip while the Pi 1 Model B only contains two. On the Pi Zero, the USB port is connected directly to the SoC and uses the (OTG) micro USB port.

Processor

The first Raspberry Pi 2 models use the 900 MHz Broadcom BCM2836 SoC 32-bit quad-core ARM Cortex-A7 processor, with shared 256 KB L2 cache. After this earlier models, the Raspberry Pi 2 V1.2 has been upgraded to a Broadcom BCM2837 SoC equipped with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor. Latest Raspberry Pi 3 series uses the same SoC. They use the Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, equipped with 512 KB shared L2 cache. The Raspberry Pi 3B+ uses the same processor (BCM2837B0) but running at 1.4 GHz. Next Raspberry Pi generations are going to be more and more powerful, but their power consumption is still rising to force developers to use CPU and GPU heatsinks.

RAM

Older B board models were designed with 128 MB RAM which was by default allocated between the GPU and CPU. The Model B (including Model A) release the RAM was extended to 256 MB split to three regions. The default split was 192 MB (RAM for CPU), which is sufficient for standalone 1080p video decoding, or for 3D modelling. Models B with 512 MB RAM initially, memory was split to files released (arm256_start.elf, arm384_start.elf, arm496_start.elf) for 256 MB, 384 MB and 496 MB CPU RAM (and 256 MB, 128 MB and 16 MB video RAM). The Raspberry Pi 2 and 3 are shipped with 1 GB of RAM. The Raspberry Pi Zero and Zero W contains 512 MB of RAM.

Networking

The Model A, A+ and Pi Zero have no dedicated Ethernet interface and can be connected to a network using an external USB Ethernet or Wi-Fi adapter. In Models B and B+, the Ethernet port is built-in to the USB Ethernet adapter using the SMSC LAN9514 chip. The Raspberry Pi 3 and Pi Zero W (wireless) models are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on Broadcom BCM43438 FullMAC chip. The Raspberry Pi 3 also has a 10/100 Ethernet port.

Peripherals

The Raspberry Pi may be controlled with any generic USB computer keyboard and mouse. It can also use USB storage, USB to MIDI converters, and virtually any other device/component which is USB compatible. Other peripherals can be attached through the various pins and connectors on the surface of the Raspberry Pi.

Video

The video controller supports standard modern TV resolutions, such as HD and Full HD, and higher. It can emit 640×350 EGA; 640×480 VGA; 800×600 SVGA; 1024×768 XGA; 1280×720 720p HDTV; 1280×768 WXGA variant; 1280×800 WXGA variant; 1280×1024 SXGA; 1366×768 WXGA variant; 1400×1050 SXGA+; 1600×1200 UXGA; 1680×1050 WXGA+; 1920×1080 1080p HDTV; 1920×1200 WUXGA. Higher resolutions, such as, up to 2048×1152, may work or even 3840×2160 at 15 Hz. Although the Raspberry Pi 3 does not include H.265 hardware decoders, the CPU is more powerful than its predecessors, potentially fast enough for software decode H.265-encoded videos. The Raspberry Pi 3

4. IoT Hardware overview

GPU runs at a higher clock frequency - 300 MHz or 400 MHz, compared to 250 MHz previous versions. The Raspberry Pis is capable of generating 576i and 480i composite video signals, as used on old-style (CRT) TV screens and less-expensive monitors through standard connectors – either RCA or 3.5 mm phono connector depending on models. The television signal standards supported are PAL-BGHID, PAL-M, PAL-N, NTSC and NTSC-J.

Real-time clock

None of the current Raspberry Pi models is equipped with a built-in real-time clock. Developers which needs the real clock time in their project can retrieve the time from a network time server (NTP) or use the external RTC module connected to the board via SPI or I²C interface. To save the file system consistency of time, the Raspberry Pi automatically saves the time on shutdown, and reload it time at boot. One of the best RTC solutions for keeping the proper boards time is to use the I²C DS1307 chip containing hardware clock with battery power supply.

Specification

Table 14: Raspberry Pi models comparative table

| Version | Model A | | | |
|--------------------|--|---|--------------------------------|-------------------------|
| | RPi 1 Model A | RPi 1 Model A+ | RPi 3 Model A+ | RPi 1 |
| Release date | 2/1/2013 | 11/1/2014 | 11/1/2018 | April–June 2019 |
| Target price (USD) | 25 | 20 | 25 | 35 |
| Instruction set | ARMv6Z (32-bit) | | ARMv8 (64-bit) | ARMv6Z (32-bit) |
| SoC | Broadcom BCM2835 | | BroadcomBCM2837B0 | Broadcom BCM2837B0 |
| FPU | VFPv2; NEON not supported | | VFPv4 + NEON | VFPv2; NEON |
| CPU | 1× ARM1176JZF-S 700 MHz | | 4× Cortex-A53 1.4 GHz | 1× ARM1176JZF-S 700 MHz |
| GPU | Broadcom VideoCore IV @ 250 MHz (BCM2837: 3D part of GPU @ 300 MHz, video @ 250 MHz) OpenGL ES 2.0 (BCM2835, BCM2836: 24 GFLOPS / BCM2837: 28.8 GFLOPS) MPEG-2 and VC-1 (with license), 1080p30 H.264/MPEG-4 AVC high-profile decoding | | | |
| Memory (SDRAM) | 256 MB (shared with GPU) | 512 MB (shared with GPU) as of 4 May 2016 | (shared with GPU) | |
| USB 2.0 ports | 1 (direct from BCM2835 chip) | | 1 (direct from BCM2837B0 chip) | 2 (via on-board hub) |
| Video input | 15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera | | | |

4.5. Raspberry Pi Overview

| Version | Model A | | |
|------------------------------|--|---|---|
| Video outputs | HDMI (rev 1.3) composite video (RCA jack), MIPI display interface (DSI) for raw LCD panels | HDMI (rev 1.3), composite video (3.5 mm TRRS jack), MIPI display interface (DSI) for raw LCD panels | HDMI (rev video (RCA display interface (DSI) for raw LCD panels |
| Audio inputs | As of revision 2 boards via I ² S | | |
| Audio outputs | Analog via 3.5 mm phone jack; digital via HDMI and, as of revision 2 boards, I ² S | | |
| On-board storage | SD, MMC, SDIO card slot (3.3 V with card power only) | MicroSDHC slot | SD, MMC, SDHC |
| On-board network | None | 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2/BLE | 10/100 Mbit |
| Low-level peripherals | 8× GPIOplus the following, which can also be used as GPIO: UART, I ² Cbus, SPI bus with two chip selects, I ² Saudio+3.3 V, +5 V, ground | 17× GPIO plus the same specific functions, and HAT ID bus | 8× GPIOplus which can also be used as GPIO: UART, I ² Cbus, SPI bus with two chip selects, I ² Saudio+3.3 V, +5 V, ground. An additional 17× GPIO plus the same specific functions, and HAT ID bus available on user is willing to make connections |
| Power ratings | 300 mA (1.5 W) | 200 mA (1 W) | 700 mA (3.5 W) |

4. IoT Hardware overview

| Version | Model A | | | | |
|---------------------|--|---|---|-----------------|------------------------|
| Power source | 5 V via MicroUSB or GPIO header | | | | |
| Size | 85.60 mm (3.370 in excluding connectors) | × 56.5 mm × 2.224 in), protruding | 65 mm × 56.5 mm × 10 mm (2.56 in × 2.22 in × 0.39 in), same as HAT board | 65 mm x 56.5 mm | 85.60 mm connectors |
| Weight | 31 g (1.1 oz) | 23 g (0.81 oz) | | 45 g (1.6 oz) | |
| Console | Adding a USB network interface via tethering or a serial cable with optional GPI | | | | |
| Generation | 1 | 1 + | 3+ | 1 | |
| Obsolescence | | | | | |
| Statement | n/a | n/a | in production until at least January 2023 | n/a | |
| Version | Model A | | | | |

Raspberry Pi boards

For today on the market exists few models of Raspberry Pi boards from tiny to more powerful. User can choose the right board to fit the price and functionality to his project development needs. Below figures are listed form the tiny/cheap to most sophisticated Raspberry Pi models.



Figure 168: Raspberry Pi Zero⁹⁴⁾

4.5. Raspberry Pi Overview

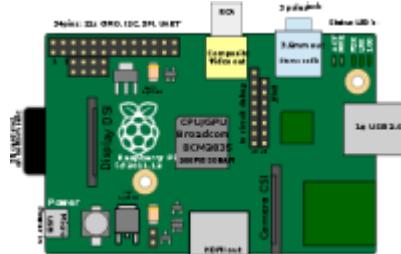


Figure 169: Raspberry Pi 1 Model A

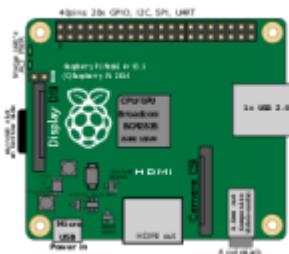


Figure 170: Raspberry Pi 1 Model A+ revision 1.1⁹⁵⁾

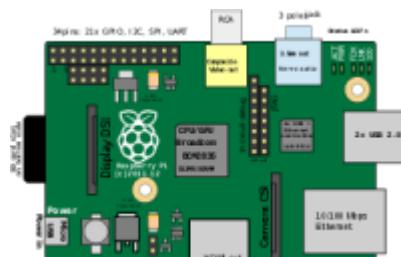


Figure 171: Raspberry Pi 1 Model B revision 1.2⁹⁶⁾

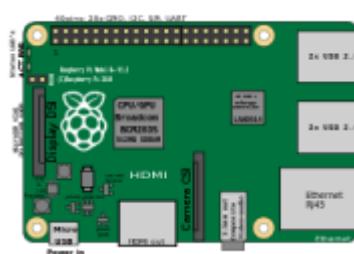


Figure 172: Raspberry Pi 2⁹⁷⁾

4. IoT Hardware overview

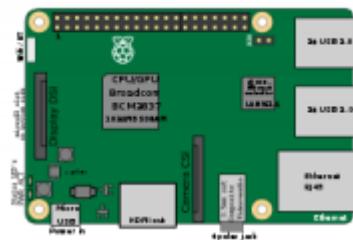


Figure 173: Raspberry Pi 3⁹⁸⁾

General-purpose input-output (GPIO) connector

Each Raspberry Pi model is equipped with standard 34/40-pins male connector containing universal GPIO ports, VCC 3.3/5V, GND, CLK, I2C/SPI buses pins which developers can use to connect their external sensors, switches and other controlled devices to the Raspberry Pi board and then program their behaviour within the code loaded to the board.

- Raspberry Pi 1 Models A+ and B+, Pi 2 Model B, Pi 3 Model B and Pi Zero (and Zero W) GPIO J8 have a 40-pin pinout. Raspberry Pi 1 Models A and B have only the first 26 pins.

| GPIO# | 2nd func. | Pin# | Pin# | 2nd func. | GPIO# |
|---------------------------------|------------------------|------|------|-------------|--------|
| | +3.3 V | 3 | 2 | +5 V | |
| 2 | SDA1 (I ^C) | 3 | 4 | +5 V | |
| 3 | SCL1 (I ^C) | 5 | 6 | GND | |
| 4 | GCLK | 6 | 8 | TXD0 (UART) | 14 |
| | GND | 9 | 10 | RXD0 (UART) | 15 |
| 17 | GEN0 | 11 | 12 | GEN1 | 18 |
| 27 | GEN2 | 13 | 14 | GND | |
| 22 | GEN3 | 15 | 16 | GEN4 | 23 |
| | +3.3 V | 17 | 18 | GEN5 | 24 |
| 10 | MOSI (SPI) | 19 | 20 | GND | |
| 9 | MISO (SPI) | 21 | 22 | GEN6 | 25 |
| 11 | SCLK (SPI) | 23 | 24 | CE0_N (SPI) | 8 |
| | GND | 25 | 26 | CE1_N (SPI) | 7 |
| (Pi 1 Models A and B stop here) | | | | | |
| EEPROM | ID_SD | 27 | 28 | ID_SC | EEPROM |
| 5 | N/A | 29 | 30 | GND | |
| 6 | N/A | 31 | 32 | | 12 |
| 13 | N/A | 33 | 34 | GND | |
| 19 | N/A | 35 | 36 | N/A | 16 |
| 26 | N/A | 37 | 38 | Digital IN | 20 |
| | GND | 39 | 40 | Digital OUT | 21 |

Figure 174: Raspberry Pi 1 pins

- Model B rev. 2 also has a pad (called P5 on the board and P6 on the schematics) of 8 pins offering access to an additional 4 GPIO connections.

4.5. Raspberry Pi Overview

| Function | 2nd func. | Pin# | Pin# | 2nd func. | Function |
|----------|-----------|------|------|------------|----------|
| N/A | +5 V | 1 | 2 | +3.3 V | N/A |
| GPIO28 | GPIO_GEN7 | 3 | | GPIO_GEN8 | GPIO29 |
| GPIO30 | GPIO_GEN9 | 5 | | GPIO_GEN10 | GPIO31 |
| N/A | GND | 7 | 8 | GND | N/A |

Figure 175: Raspberry Pi 2 & 3 pins

HDMI port

Each Raspberry Pi model is equipped with the standard mini HDMI port allows user connect the monitor or TV set with the board. The electronic schematic is shown on the picture.

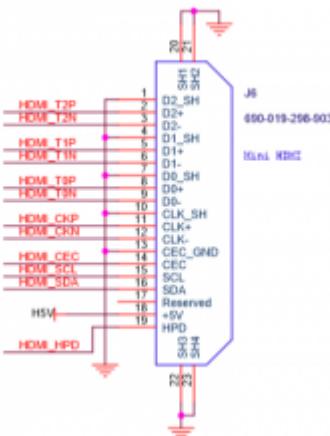


Figure 176: Raspberry HDMI port connection schematic

Camera port CSI

Raspberry Pi boards Zero, 1, A+, 2, 3 are equipped with Camera interface(CSI) port allowing user connect the CCD camera following the MIPI standard.

4. IoT Hardware overview

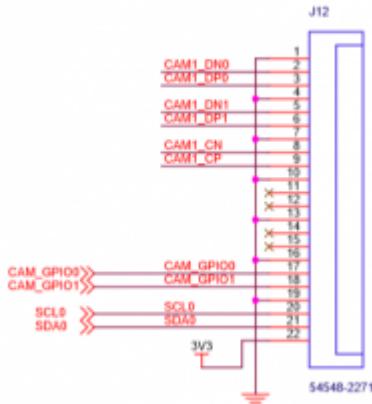


Figure 177: Raspberry CSI camera schematic⁹⁹⁾

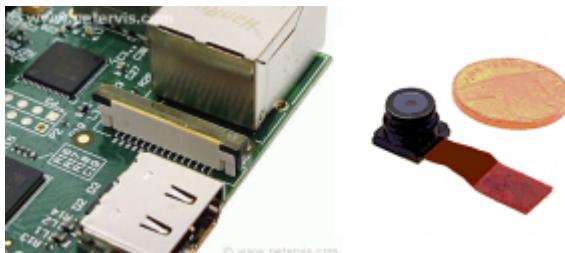


Figure 178: Raspberry CSI camera view¹⁰⁰⁾

Display port (DSI)

Raspberry Pi boards 2, 3 are equipped with LCD Display interface(DSI) port allowing the user to connect the LCD touch display to the board. The official Raspberry Pi LCD touch display shown in the figure below is 800×480 dpi 7" size can be connected to the Raspberry board using the DSI interface. Such an assembly can be used in the projects to display controlling application view and with the ability to handle fingers touchscreen controls the project behaviour. The LCD can be mounted in portrait/landscape orientation fitting the best user needs.

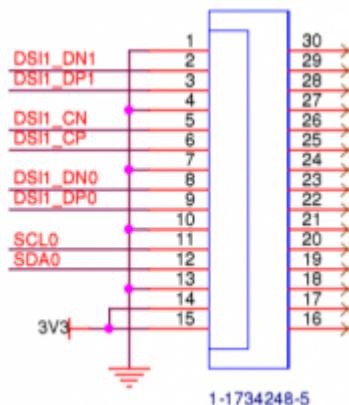


Figure 179: Raspberry DSI display port schematic¹⁰¹⁾



Figure 180: Raspberry DSI LCD display kit¹⁰²⁾

USB and LAN ports

Raspberry PI models boards Zero, 1, A+, 2, 3 contains USB ports (from 1 up to 4) and models boards 1, A+, 2, 3 the LAN port for TCP/IP network connections. This ports can be used for mouse/keyboard connection or if the software has appropriate driver installed to handle other USB devices.



Figure 181: Raspberry LAN/USB ports view¹⁰³⁾

4. IoT Hardware overview

4.5.2. Raspberry Pi Sensors

Raspberry Pi boards offer an easy way to connect different sensors and control devices. With specially designed I/O pins available to program them by developers the amount of possible implementations growth year by year. Any I/O General Purpose Input-Output Ports (GPIO) can be set as Digital Input or Output. The board contains two PWM pins which can be used as output analogue signals. Some of the interface libraries, such as pigpio or wiringPi, support this feature. It is also the way the Raspberry Pi outputs analogue audio.

Touch sensors

Button

A *push button* is an electromechanical sensor that connects or disconnects two points in a circuit when the force is applied. Button output discrete value is either HIGH or LOW.

A *microswitch*, also called a miniature snap-action switch, is an electromechanical sensor that requires a very little physical force and uses tipping-point mechanism. Microswitch has 3 pins, two of which are connected by default. When the pressure is applied, the first connection breaks and one of the pins is connected to the third pin.

The most common use of a push button is as an input device. Both force solutions can be used as simple object detectors, or as end switches in the industrial devices.



Figure 182: A push button¹⁰⁴⁾ and a microswitch¹⁰⁵⁾

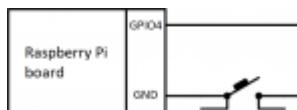


Figure 183: Schematics of Raspberry Pi and a push button

To proper work with button the GPIO4 must be configured as an digital input. Pressing the **push button** connects the GPIO4 pin to the boards GND. On Raspberry Pi GPIO input pins are normally pulled up to 3.3V. When button is pressed and GPIO4 is read using GPIO.input, it will return the **False** result. Each GPIO pin can be configured to use internal pull-up or pull-down resistors. Using a GPIO pin as an input, these resistors can be configured using the optional `pull_up_down` parameter in the `GPIO.setup`. If this parameter is omitted, resistors will not be activated. In this case the input may floating giving unpredicted results during read it. If the GPIO pin is set to `GPIO.UD_UP`, the pull-up resistor is enabled; if it is set to `GPIO.PUD_DOWN`, the pull-down resistor is enabled.

An example code:

```
# Python code for Raspberry Pi

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
s_pin = 7 # select the GPIO4 pin

# set the GPIO4 port to input mode
GPIO.setup(s_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    input_state = GPIO.input(s_pin)
    if input_state == False:
        print('Button Pressed')
        time.sleep(0.2)
```

Running the code as superuser shows:

```
pi@raspberrypi ~ $ sudo python switch.py
Button Pressed
Button Pressed
Button Pressed
Button Pressed
```

Force sensor

A *force sensor* predictably changes resistance, depending on the applied force to its surface. Force-sensing resistors are manufactured in different shapes and sizes, and they can measure not only direct force but also the tension, compression, torsion and other types of mechanical forces. The voltage is measured by applying and measuring constant voltage to the sensor.

Force sensors are used as control buttons or to determine weight.



Figure 184: .5 inch force sensing resistor (FSR)¹⁰⁶

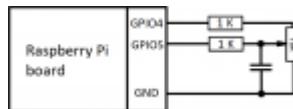


Figure 185: Raspberry Pi and Force Sensitive Resistor circuit schematics

An example code:

4. IoT Hardware overview

```
# Python code for Raspberry Pi

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

a_pin = 7    # select the GPIO4 pin
b_pin = 29   # select the GPIO5 pin

def discharge():
    GPIO.setup(a_pin, GPIO.IN)
    GPIO.setup(b_pin, GPIO.OUT)
    GPIO.output(b_pin, False)
    time.sleep(0.005)

def charge_time():
    GPIO.setup(b_pin, GPIO.IN)
    GPIO.setup(a_pin, GPIO.OUT)
    count = 0
    GPIO.output(a_pin, True)
    while not GPIO.input(b_pin):
        count = count + 1
    return count

def analog_read():
    discharge()
    return charge_time()

while True:
    print(analog_read())
    time.sleep(1)
```

Running the code as superuser shows:

```
$ sudo python pot_step.py
10
12
10
16
23
43
53
67
72
86
105
123
143
170
```

The idea of how to read the force sensor changing value is called **step response**. It works by checking how the circuit responds to the step change when an output is switched from low to high. Raspberry Pi isn't equipped with an ADC converter. So it is impossible to read voltage directly. However, it can be measured how long the

capacitor will fill with the charge to the extent that it gets voltage above 1.65V or so that constitutes a high digital input. The speed at which the capacitor fills with charge depends on the value of the variable resistor (R_t). The lower the resistance, the faster the capacitor fills with charge, and the voltage rises. To get the proper value, the circuit must empty the capacitor each time before the reading starts. In the schematic the GPIO4 is used to charge the capacitor and GPIO5 is used to discharge the capacitor through the 10K resistor. Both resistors are used to make sure that there is no way too much current can flow as the capacitor is charged and discharged. To discharge it, connection GPIO4 is set to be an input, effectively disconnecting R_c and R_t from the circuit. Connection GPIO5 is then set to be an output and low. It is held there for 5 milliseconds, to empty the capacitor.

Capacitive sensor

Capacitive sensors are a range of sensors that use capacitance to measure changes in the surrounding environment. A capacitive sensor consists of a capacitor that is charged with a certain amount of current till the threshold voltage. A human finger, liquids or other conductive or dielectric materials that touch the sensor, can influence a charge time and a voltage level in the sensor. Measuring a charge time and a voltage level gives information about changes in the environment.

Capacitive sensors are used as input devices and can measure proximity, humidity, fluid level and other physical parameters or serve as an input for electronic device control.



Figure 186: Digital capacitive touch sensor v2.0 switch module ¹⁰⁷⁾

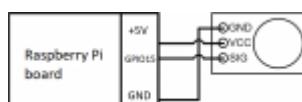


Figure 187: Raspberry Pi and capacitive sensor schematics

```
# Python code for Raspberry Pi

import time
import pigpio # http://abyz.co.uk/rpi/pigpio/python.html
RXD=15       # define the RxD serial input port

pi = pigpio.pi()
if not pi.connected:
    exit(0)

pigpio.exceptions = False # Ignore error if already set as bit bang read.
pi.bb_serial_read_open(RXD, 9600) # Set baud rate here.
pigpio.exceptions = True
```

4. IoT Hardware overview

```
pi.bb_serial_invert(RXD, 1) # Invert line logic.  
stop = time.time() + 60.0  
while time.time() < stop:  
    (count, data) = pi.bb_serial_read(RXD)  
    if count:  
        print(data)  
    time.sleep(0.2)  
  
pi.bb_serial_read_close(RXD)  
  
pi.stop()
```

Proximity and distance sensors

Ultrasound sensor

Ultrasound (ultrasonic) sensor measures the distance to objects by emitting ultrasound and measuring its returning time. The sensor consists of an ultrasonic emitter and receiver, and sometimes they are combined in a single device for emitting and receiving. Ultrasonic sensors can measure greater distances and cost less than infrared sensors, but are more imprecise and interfere with each other measurement if more than one is used. Simple sensors have trigger pin and echo pin, when trigger pin is set high for the small amount of time ultrasound is emitted and on echo pin, response time is measured. Ultrasonic sensors are used in car parking sensors and robots for proximity detection.



Figure 188: Ultrasonic proximity sensor HC-SR04¹⁰⁸⁾

Examples of IoT applications are robotic obstacle detection and room layout scanning.



Figure 189: Raspberry Pi and ultrasound proximity sensor circuit

An example code:

```
# Python code for Raspberry Pi  
  
import RPi.GPIO as GPIO  
import time  
  
TRIG = 7 # define a trigger pin GPIO4  
ECHO = 29 # define an echo pin GPIO5
```

```
print ("Distance Measurement In Progress")

GPIO.setup(TRIG, GPIO.OUT)      # set the GPIO4 as trigger output port
GPIO.setup(ECHO,GPIO.IN)        # set the GPIO5 pin as echo input

GPIO.output (TRIG,False)

print ("Waiting for Sensor to Settle")
time.sleep(2)

GPIO.output (TRIG, True)
time.sleep (0.00001)
GPIO.output (TRIG, False)

while GPIO.input(ECHO) == 0:
    pulse_start = time.time()

while GPIO.input(ECHO) == 1:
    pulse_end = time.time()

pulse_duration = pulse_end - pulse_start
distance = pulse_duration*17150
distance = round(distance,2)          # Calculating the distance
print ("Distance:", distance, "cm")
```

Running the code as superuser shows:

```
pi@raspberrypi > $ sudo python range_sensor.py
Distance Measurement To Settle
Distance: 23.54 cm
pi@raspberrypi > $
```

Motion detector

Motion detector is a sensor that detects moving objects, usually people. Motion detectors use different technologies, like passive infrared sensors, microwaves and Doppler effect, video cameras and previously mentioned ultrasonic and IR sensors. Passive IR sensors are the simplest motion detectors that sense people through detecting IR radiation that is emitted through the skin. When the motion is detected, the output of a motion sensor is a digital HIGH/LOW signal.

Motion sensors are used for security purposes, automated light and door systems. As an example in IoT, the PIR motion sensor can be used to detect motion in security systems a house or any building.



Figure 190: PIR motion sensor HC-SR501¹⁰⁹⁾

4. IoT Hardware overview

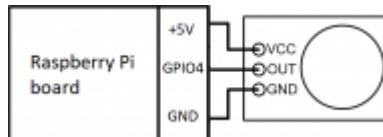


Figure 191: Raspberry Pi and PIR motion sensor circuit

An example code:

```
# Python code for Raspberry Pi

pirPin = 7; // Passive Infrared (PIR) sensor output is connected to the GPIO4 pin

GPIO.setup(pirPin ,GPIO.IN)          # set the GPIO5 pin as echo input

while 1:
    # read the digital value of the PIR motion sensor GPIO4
    pirReading = GPIO.input(pirPin)
    print (piReading)                # print out

    if pirReading == True:           # motion was detected
        print ('Motion Detected')
    time.sleep(10)
```

Gyroscope

A gyroscope is a sensor that measures the angular velocity. The sensor is made of the microelectromechanical system (MEMS) technology and is integrated into the chip. The output of the sensor can be either analogue or digital value of information, using I2C or SPI interface. Gyroscope microchips can vary in the number of axes they can measure. An available number of the axis is 1, 2 or 3 axes in the gyroscope. For gyroscopes with 1 or 2 axes, it is important to determine which axis the gyroscope measures and to choose a device according to the project needs. A gyroscope is commonly used together with an accelerometer, to determine the orientation, position and velocity of the device precisely.

Gyroscope sensors are used in aviation, navigation and motion control.



Figure 192: MPU 6050 GY-521 breakout board¹¹⁰⁾

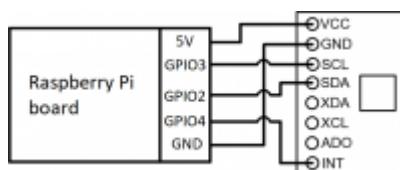


Figure 193: Raspberry Pi and MPU 6050 GY-521 gyro breakout schematics

The example code for the FXAS21002C sensor used in the breakout board:

```
# Python code for Raspberry Pi
#!/usr/bin/env python

from __future__ import division, print_function
from nxp_imu import IMU
import time

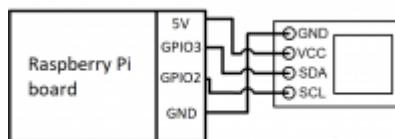
imu = IMU(gs=4, dps=2000, verbose=True)
header = 67
print('*'*header)
print("|{:17} | {:20} | {:20} |".format("Accels [g's]", "Magnet [uT]", "Gyros [dps]"))
print('*'*header)
for _ in range(10):
    a, m, g = imu.get()
    print('|{:>5.2f} {:>5.2f} {:>5.2f} |{:>6.1f} {:>6.1f} |{:>6.1f} |'.format(
        a[0], a[1], a[2],
        m[0], m[1], m[2],
        g[0], g[1], g[2]))
    )
    time.sleep(0.50)
print('*'*header)
print(' uT: micro Tesla')
print(' g: gravity')
print('dps: degrees per second')
print('')
```

Compass

A **compass** is the sensor, that can measure the orientation of the device to the magnetic field of the Earth. Solid state compass consists of the magnetometer and accelerometers in a single chip to precisely calculate the position of the device. Devices communicate through I2C or SPI interfaces and can return calculated heading, pitch and roll and raw accelerometer and magnetometer values. Compass is used in outdoor navigation for mobile devices, robots, quadcopters.



Figure 194: Compass module HMC5883L¹¹¹⁾



4. IoT Hardware overview

Figure 195: Raspberry Pi and Compass module HMC5883L schematics

The example code:

```
1. Install i2c:  
   sudo apt-get install i2c-tools  
2. edit file /etc/modprobe.d/raspi-blacklist.conf  
   and comment out the line blacklist i2c-bcm2708  
3. edit /etc/modules, and add the lines:  
   i2c-bcm2708  
   i2c-dev  
4. Allow i2c access from users other than root,  
   by creating the file /etc/udev/rules.d/99-i2c.rules with this line:  
   SUBSYSTEM=="i2c-dev", MODE="0666"  
5. Reboot the Pi. When it goes up again, type:  
   ls /dev/i2c*  
On Pi (Model B, Revision 2 version, early 2013) it generates:  
/dev/i2c-0 /dev/i2c-1  
Optional: For python, install the smbus python library with:  
1. apt-get install python-smbus  
2. Install Python 3, can't hurt, and i2clibraries needs it.  
   Just type sudo apt-get install python3  
3. Test if the compass is detected, by typing:  
   i2cdetect -y 1 (for Revision 1 Pis, replace 1 with 0).  
4. Replace with 0 for Revision 1 Raspberry Pis and with  
   1 for Revision 2 boards. This is the output:  
0 1 2 3 4 5 6 7 8 9 a b c d e f  
00: -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- 1e --  
20: -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- --  
  
Tip: if you don't see it, it's because you haven't welded  
the pins to the sensor. Just press with your finger. Or weld it.  
1. Add the quick2wire code. Pull from git:  
   git clone https://github.com/quick2wire/quick2wire-python-api.git.  
   On /etc/profile, add: export QUICK2WIRE_API_HOME=/home/pi/quick2wire-python-api  
2. export PYTHONPATH=$PYTHONPATH:$QUICK2WIRE_API_HOME  
   Add i2clibraries. Pull from git:  
   clone https://bitbucket.org/thinkbowl/i2clibraries.git
```

Environment sensors

Temperature sensor

A *temperature sensor* is a device that is used to determine the temperature of the surrounding environment. Most temperature sensors work on the principle that the resistance of the material is changed depending on its temperature. The most common temperature sensors are:

- *thermocouple* - consists of two junctions of dissimilar metals,
- *theristor* - includes the temperature-dependent ceramic resistor,

4.5. Raspberry Pi Overview

- *resistive temperature detector* – is made of a very pure metal coil.

The main difference between sensors is the measured temperature range, precision and response time. Temperature sensor usually outputs the analogue value, but some existing sensors have a digital interface. ¹¹²⁾

The temperature sensors most commonly are used in environmental monitoring devices and thermoelectric switches. In IoT applications, the sensor can be used for greenhouse temperature monitoring, warehouse temperature monitoring to avoid frozen fire suppression systems and tracking temperature of the soil, water and plants.



Figure 196: Thermistor sensor ¹¹³⁾

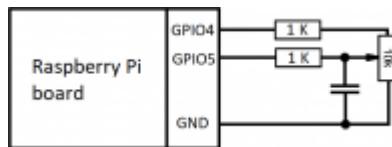


Figure 197: Raspberry Pi and thermistor circuit

An example code is similar to the Raspberry Pi force sensor sample. The thermistor changes its resistance depends on the environment temperature, and it can be read using a similar code:

```
# Python code for Raspberry Pi

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

a_pin = 7    # select the GPIO4 pin
b_pin = 29   # select the GPIO5 pin

def discharge():
    GPIO.setup(a_pin, GPIO.IN)
    GPIO.setup(b_pin, GPIO.OUT)
    GPIO.output(b_pin, False)
    time.sleep(0.005)

def charge_time():
    GPIO.setup(b_pin, GPIO.IN)
    GPIO.setup(a_pin, GPIO.OUT)
    count = 0
    GPIO.output(a_pin, True)
    while not GPIO.input(b_pin):
        count = count + 1
    return count
```

4. IoT Hardware overview

```
def analog_read():
    discharge()
    return charge_time()

while True:
    print(analog_read())
    time.sleep(1)
```

Humidity sensor

A *humidity sensor* (hygrometer) is a sensor which detects the amount of water or water vapour in the environment. The most common principle of the air humidity sensors is the change of capacitance or resistance of materials which absorb the moisture from the environment. Soil humidity sensors measure the resistance between the two electrodes. A resistance between electrodes is influenced by soluble salts and water amount in the soil. The output of a humidity sensor is usually an analogue signal value.¹¹⁴⁾

Example IoT applications are monitoring of humidors, greenhouse temperature and humidity, agricultural environment and art gallery and museum environment.

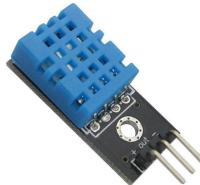


Figure 198: DHT11 temperature and humidity sensor breakout¹¹⁵⁾

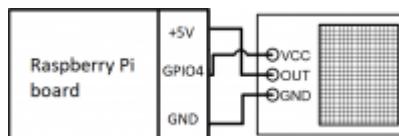


Figure 199: Raspberry Pi and humidity sensor schematics

An example code:

1. Enter this at the command prompt to download the library:
`git clone https://github.com/adafruit/Adafruit_Python_DHT.git`
2. Change directories with:
`cd Adafruit_Python_DHT`
3. Now enter this:
`sudo apt-get install build-essential python-dev`
4. Then install the library with:
`sudo python setup.py install`

```
# Python code for Raspberry Pi
#!/usr/bin/python

import sys
import Adafruit_DHT

while True:
    humidity, temperature = Adafruit_DHT.read_retry(11, 7) # read GPIO4 Pin 7
    print ('Temp: {0:0.1f} C  Humidity: {1:0.1f} %'.format(temperature, humidity))
```

116)

Sound sensor

A *sound sensor* is a sensor that detects vibrations in a gas, liquid or solid environments. At first, the sound wave pressure makes mechanical vibrations, who transfers to changes in capacitance, electromagnetic induction, light modulation or piezoelectric generation to create an electric signal. The electrical signal is then amplified to the required output levels. Sound sensors, can be used to record sound, detect noise and its level.

Sound sensors are used in drone detection, gunshot alert, seismic detection and vault safety alarm.



Figure 200: Digital sound detector sensor module¹¹⁷⁾

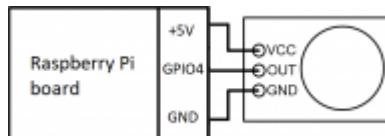


Figure 201: Raspberry Pi and sound sensor schematics

An example code:

```
# Python code for Raspberry Pi

import time
import RPi.GPIO as GPIO
from ghus import Bridge

GPIO.setmode(GPIO.BCM) # use board pin numbers
pin = 7 # define GPIO4 as Input
GPIO.setup(pin, GPIO.IN)

def callback (pin)
```

4. IoT Hardware overview

```
if GPIO.input (pin)
    print ("Sound detected!")
else:
    print ("Sound detected!")
# activate when pin changed its state
GPIO.add_event_detect(pin,GPIO_BOTH, bouncetime=300)
# assign function to GPIO PIN run it on changes
GPIO.add_event_callback(pin,callback)

# infinite loop
while True:
    time.sleep(1)
```

Chemical/smoke and gas sensor

Gas sensors are a sensor group, that can detect and measure a concentration of certain gasses in the air. The working principle of electrochemical sensors is to absorb the gas and to create current from the electrochemical reaction. For process acceleration, a heating element can be used. For each type of gas different kind of sensor needs to be used. Multiple different types of gas sensors can be combined in a single device as well. The single gas sensor output is an analogue signal, but devices with multiple sensors have a digital interface.

Gas sensors are used for safety devices, to control air quality and for manufacturing equipment. Examples of IoT applications are air quality control management in smart buildings and smart cities or toxic gas detection in sewers and underground mines.



Figure 202: MQ2 gas sensor¹¹⁸⁾

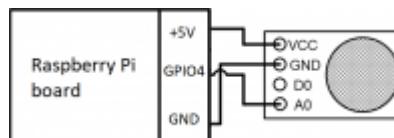


Figure 203: Raspberry Pi and MQ2 gas sensor schematics

An example code:

```
# Python code for Raspberry Pi

1. git clone https://github.com/tutRPi/Raspberry-Pi-Gas-Sensor-MQ
2. cd Raspberry-Pi-Gas-Sensor-MQ
3. sudo python example.py
```

```
# Python code for Raspberry Pi
#!/usr/bin/env python
```

```
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time
import math

DO    = 17
Buzz = 18
GPIO.setmode(GPIO.BCM)

def setup():
    ADC.setup(0x48)
    GPIO.setup (DO, GPIO.IN)
    GPIO.setup (Buzz, GPIO.OUT)
    GPIO.output (Buzz, 1)

def Print(x):
    if x == 1:
        print ('')
        print ('*****')
        print (' * Safe~ *')
        print ('*****')
        print ('')
    if x == 0:
        print ('')
        print ('*****')
        print (' * Danger Gas! *')
        print ('*****')
        print ('')

def loop():
    status = 1
    count = 0
    while True:
        print (ADC.read(0))

        tmp = GPIO.input(DO);
        if tmp != status:
            print(tmp)
            status = tmp
            if status == 0:
                count += 1
                if count % 2 == 0:
                    GPIO.output(Buzz, 1)
                else:
                    GPIO.output(Buzz, 0)
            else:
                GPIO.output(Buzz, 1)
            count = 0

    time.sleep(0.2)

def destroy():
    GPIO.output(Buzz, 1)
    GPIO.cleanup()

if __name__ == '__main__':
    try:
```

4. IoT Hardware overview

```
setup()  
loop()  
except KeyboardInterrupt:  
destroy()
```

Other sensors

Global positioning system

A *GPS receiver* is a device, that can receive information from global navigation satellite system and calculate its position on the Earth. GPS receiver uses a constellation of satellites and ground stations to compute position and time almost anywhere on the Earth. GPS receivers are used for navigation only in the outdoor area because it needs to receive signals from the satellites. The precision of the GPS location can vary.

A GPS receiver is used for device location tracking. Real world applications might be pet, kid or personal belonging location tracking.



Figure 204: LS20031 GPS receiver¹¹⁹⁾

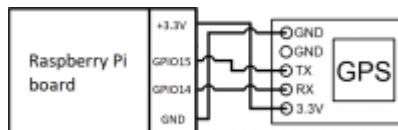


Figure 205: Raspberry Pi and LS20031 GPS receiver schematics

The example code:

```
# Python code for Raspberry Pi  
#!/usr/bin/python  
  
import os  
import pygame, sys  
from pygame.locals import *  
import serial  
  
#initialise serial port on /ttyUSB0  
ser = serial.Serial('/dev/ttyUSB0',4800,timeout = None)  
# set font size MAX 100  
fontsize = 50  
  
# calculate window size  
width = fontsize * 17  
height = fontsize + 10
```

```

# initialaise pygame
pygame.init()
windowSurfaceObj = pygame.display.set_mode((width,height),1,16)
fontObj = pygame.font.Font('freesansbold.ttf',fontSize)
pygame.display.set_caption('GPS Location')
redColor = pygame.Color(255,0,0)
greenColor = pygame.Color(0,255,0)
yellowColor = pygame.Color(255,255,0)
blackColor = pygame.Color(0,0,0)

fix = 1
color = redColor
x = 0
while x == 0:
    gps = ser.readline()
    # print (all NMEA strings)
    print (gps)
    # check gps fix status
    if gps[1:6] == "GPGSA":
        fix = int(gps[9:10])
        if fix == 2:
            color = yellowColor
        if fix == 3:
            color = greenColor
    # print (time, lat and long from #GPGGA string)
    if gps[1 : 6] == "GPGGA":
        # clear window
        pygame.draw.rect(windowSurfaceObj,blackColor,Rect(0,0,width,height))
        pygame.display.update(pygame.Rect(0,0,width,height))
        # get time
        time = gps[7:9] + ":" + gps[9:11] + ":" + gps[11:13]
        # if 2 or 3D fix get lat and long
        if fix > 1:
            lat = " " + gps[18:20] + "." + gps[20:22] + "." + gps[23:27] + gps[28:29]
            lon = " " + gps[30:33] + "." + gps[33:35] + "." + gps[36:40] + gps[41:42]
        # if no fix
        else:
            lat = " No Valid Data "
            lon = " "
        # print new values
        msgSurfaceObj = fontObj.render(str(time), False,color)
        msgRectobj = msgSurfaceObj.get_rect()
        msgRectobj.topleft =(2,0)
        windowSurfaceObj.blit(msgSurfaceObj, msgRectobj)

        msgSurfaceObj = fontObj.render(str(lat), False,color)
        msgRectobj = msgSurfaceObj.get_rect()
        msgRectobj.topleft =(210,0)
        windowSurfaceObj.blit(msgSurfaceObj, msgRectobj)

        msgSurfaceObj = fontObj.render(str(lon), False,color)
        msgRectobj = msgSurfaceObj.get_rect()
        msgRectobj.topleft =(495,0)
        windowSurfaceObj.blit(msgSurfaceObj, msgRectobj)
        pygame.display.update(pygame.Rect(0,0,width,height))
        fix = 1
        color = redColor

```

4. IoT Hardware overview

```
# check for ESC key pressed, or GPS Location window closed, to quit
for event in pygame.event.get():
    if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):
        pygame.quit()
        sys.exit()
}
```

4.5.3. Raspberry Pi Drivers and driving

Optical device drivers and their devices

Light-emitting diode

Light-emitting diode also called LED is a special type of diodes which emits light, unlike the other diodes. LED has a completely different body which is made of transparent plastic that protects the diode and lets it emit light. Like the other diodes LED conducts the current in only one way, so it is essential to connect it to the scheme correctly. There are two safe ways how to determine the direction of the diode:

- in the cathodes side of the diode its side is chipped,
- anodes leg usually is longer than the cathodes leg.



Figure 206: White LED¹²⁰

LED is one of the best light sources. Unlike incandescent light bulb LED transforms most of the power into light not warmth; it is more durable, works for a more extended period and can be manufactured in a smaller size.

LED colour is determined by the semiconductors material. If usually diodes are made from silicon then LEDs are made from elements like gallium phosphate, silicon carbide and others. Because the semiconductors used are different the voltage needed for the LED to shine is also different. In the table, you can see with which semiconductor you can get a specific colour and the voltage required to turn on the LED.

When LED is connected to the voltage and turned on a considerable current starts to flow through it, and it can damage the diode. That is why all **LEDs have to be connected to current limiting resistor**.

Current limiting resistors resistance is determined by three parameters:

- **I_D** - current that can flow through the LED,
- **U_D** - Voltage that is needed to turn on the LED,
- **U** - combined voltage for LED and resistor.

To calculate the resistance needed for a diode, this is what you have to do:

1. Find out the required voltage for the diode to work U_D ; you can find it in the diodes parameters table.
2. Find out the amperage is necessary for the LED to shine I_D ; it can be found in the LEDs datasheet, but if you can't find it, then 20 mA current is usually correct and safe choice.
3. Find out the combined voltage for the LED and resistor. Usually, it is the feeding voltage for the scheme.
4. Insert all the values into this equation: $R = (U - U_D) / I_D$
5. You get the resistance for the resistor for the safe use of the LED.
6. Find resistors nominal that is the same or some higher resistance than the calculated one.

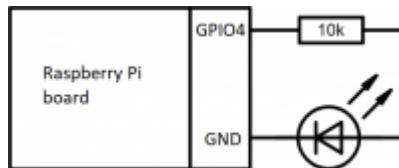


Figure 207: Raspberry Pi and LED control schematic

The example of blinking LED code:

```
# Raspberry Pi Python sample code

import RPi.GPIO as GPIO
import time

LED = 18      # GPIO04 port

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(LED,GPIO.OUT)
print ("LED on")
GPIO.output(LED,GPIO.HIGH)
time.sleep(1)
print ("LED off")
GPIO.output(LED,GPIO.LOW)
```

Displays

Using *display* is a quick way to get a feedback information from the device. There are many display technologies compatible with Arduino. For IoT solutions low power, easy to use and monochrome displays are used:

- *Liquid-crystal display (LCD)*,
- *Organic light-emitting diode display (OLED)*,
- *Electronic ink display (E ink)*.

4. IoT Hardware overview

Liquid-crystal display (LCD)

LCD uses modulating properties of liquid crystal light to block the incoming light. Thus when the voltage is applied to a pixel, it has a dark colour. A display consists of layers of electrodes, polarising filters, liquid crystals and reflector or back-light. Liquid crystals do not emit the light directly; they do it through reflection or backlight. Because of this reason, they are more energy efficient. Small, monochrome LCDs are widely used in devices to show a short numerical or textual information like temperature, time, device status etc. LCDs commonly come with an onboard control circuit and are controlled through parallel or serial interface.

Figure 208: 16×2 LCD display¹²¹⁾

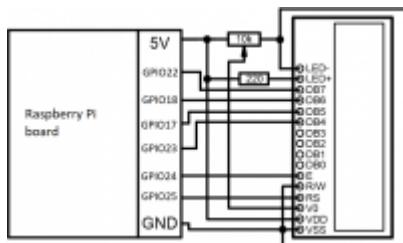


Figure 209: Raspberry Pi and LCD screen schematics

The example code:

```
# Raspberry Pi Python sample code
#!/usr/bin/python
# Example using a character LCD connected to a Raspberry Pi

import time
import Adafruit_CharLCD as LCD

# Raspberry Pi pin setup
lcd_rs = 25
lcd_en = 24
lcd_d4 = 23
lcd_d5 = 17
lcd_d6 = 18
lcd_d7 = 22
lcd_backlight = 2

# Define LCD column and row size for 16x2 LCD.
lcd_columns = 16
lcd_rows = 2

lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6,
                           lcd_d7, lcd_columns, lcd_rows, lcd_backlight)

lcd.message('Hello\nworld!')
# Wait 5 seconds

time.sleep(5.0)
lcd.clear()
text = raw_input("Type Something to be displayed: ")
```

```

lcd.message(text)

# Wait 5 seconds
time.sleep(5.0)
lcd.clear()
lcd.message('Goodbye\nWorld!')

time.sleep(5.0)
lcd.clear()

```

Organic light-emitting diode display (OLED)

OLED display uses electroluminescent materials that emit light when the current passes through these materials. The display consists of two electrodes and a layer of organic compound. OLED displays are thinner than LCDs, they have higher contrast, and they can be more energy efficient comparing to LCDs. OLED displays are commonly used in mobile devices like smartwatches, cell phones and they are replacing LCDs in other devices. Small OLED displays have an onboard control circuit that uses digital interfaces like I2C or SPI.



Figure 210: OLED I2C display ¹²²⁾

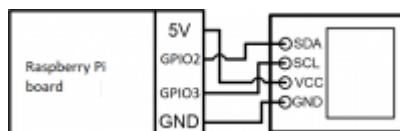


Figure 211: Raspberry Pi and OLED I2C schematics

```

1. git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git
2. cd Adafruit_Python_SSD1306
For Python 2:
3. sudo python setup.py install
For Python3:
4. sudo python3 setup.py install

```

```

cd examples
Choose one of existing examples:
- animate.py
- buttons.py
- image.py
- shapes.py
- stats.py

```

4. IoT Hardware overview

Electronic ink display (E-ink)

E-ink display uses charged particles to create a paper-like effect. The display consists of transparent microcapsules filled with oppositely charged white and black particles between electrodes. Charged particles change their location, depending on the orientation of the electric field, thus individual pixels can be either black or white. The image does not need the power to persist on the screen. Power is used only when the image is changed; thus e-ink display is very power efficient for presenting static information. It has high contrast and viewing angle, but it has a low refresh rate. E-ink displays are commonly used in e-readers, smartwatches, outdoor signs, electronic shelf labels.



Figure 212: E ink display module ¹²³⁾

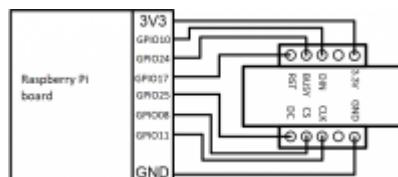


Figure 213: Raspberry Pi and E ink display module schematics

```
# Raspberry Pi Python sample code
# from https://www.instructables.com/id/Waveshare-EPaper-and-a-RaspberryPi

import time, datetime, sys, signal, urllib, requests
from EPD_driver import EPD_driver
def handler(signum, frame):
    print ('SIGTERM')
    sys.exit(0)
signal.signal(signal.SIGTERM, handler)
bus = 0
device = 0
disp = EPD_driver(spi = SPI.SpiDev(bus, device))
print ("disp size : %dx%d"%(disp.xDot, disp.yDot))
print ('-----init and Clear full screen-----')
disp.Dis_Clear_full()
disp.delay()
# display part
disp.EPD_init_Part()
disp.delay()
imagenames = []
search = "http://api.duckduckgo.com/?q=Cat&format=json&pretty=1"
if search:
    req = requests.get(search)
    if req.status_code == 200:
```

```

for topic in req.json()["RelatedTopics"]:
    if "Topics" in topic:
        for topic2 in topic["Topics"]:
            try:
                url = topic2["Icon"]["URL"]
                text = topic2["Text"]
                if url:
                    imagenames.append( (url,text) )
            except:
                # print topic
                pass
            try:
                url = topic["Icon"]["URL"]
                if url:
                    imagenames.append( url )
            except:
                # print topic
                pass
    else:
        print (req.status_code)
# font for drawing within PIL
myfont10 = ImageFont.truetype("amiga_forever/amiga4ever.ttf", 8)
myfont28 = ImageFont.truetype("amiga_forever/amiga4ever.ttf", 28)
# mainimg is used as screen buffer, all image composing/drawing is done in PIL,
# the mainimg is then copied to the display (drawing on the disp itself is no fun)
mainimg = Image.new("1", (296,128))
name = ("images/downloaded.png", "bla")
skip = 0
while 1:
    for name2 in imagenames:
        print ('-----')
        skip = (skip+1)%7
        try:
            starttime = time.time()
            if skip==0 and name2[0].startswith("http"):
                name = name2
                urllib.urlretrieve(name[0], "images/downloaded.png")
                name = ("images/downloaded.png", name2[1])
                im = Image.open(name[0])
                print (name, im.format, im.size, im.mode)
                im.thumbnail((296,128))
                im = im.convert("1") #, dither=Image.NONE)
                # print ('thumbnail', im.format, im.size, im.mode)
                loadtime = time.time()
                print ('@:load+resize:', (loadtime - starttime))
                draw = ImageDraw.Draw(mainimg)
                # clear
                draw.rectangle([0,0,296,128], fill=255)
                # copy to mainimg
                ypos = (disp.xDot - im.size[1])/2
                xpos = (disp.yDot - im.size[0])/2
                print ('ypos:', ypos, 'xpos:', xpos)
                mainimg.paste(im, (xpos,ypos))
                # draw info text
                ts = draw.textsize(name[1], font=myfont10)
                tsy = ts[1]+1
                oldy = -1
                divs = ts[0]/250

```

4. IoT Hardware overview

```
for y in range(0, divs):
    newtext = name[1][(oldy+1)*len(name[1])/divs:(y+1)*len(name[1])/divs]
    # print (divs, oldy, y, newtext)
    oldy = y
    draw.text((1, 1+y*tsy), newtext, fill=255, font=myfont10)
    draw.text((1, 3+y*tsy), newtext, fill=255, font=myfont10)
    draw.text((3, 3+y*tsy), newtext, fill=255, font=myfont10)
    draw.text((3, 1+y*tsy), newtext, fill=255, font=myfont10)
    draw.text((2, 2+y*tsy), newtext, fill=0, font=myfont10)
#draw time
now = datetime.datetime.now()
tstr = "%02d:%02d:%02d"%(now.hour,now.minute,now.second)
# draw a shadow, time
tpx = 36
tpy = 96
for i in range(tpy-4, tpy+32, 2):
    draw.line([0, i, 295, i], fill=255)
    draw.text((tpx-1, tpy ), tstr, fill=0, font=myfont28)
    draw.text((tpx-1, tpy-1), tstr, fill=0, font=myfont28)
    draw.text((tpx , tpy-1), tstr, fill=0, font=myfont28)
    draw.text((tpx+2, tpy ), tstr, fill=0, font=myfont28)
    draw.text((tpx+2, tpy+2), tstr, fill=0, font=myfont28)
    draw.text((tpx , tpy+2), tstr, fill=0, font=myfont28)
    draw.text((tpx , tpy ), tstr, fill=255, font=myfont28)
del draw
im = mainimg.transpose(Image.ROTATE_90)
drawtime = time.timetime.timetime.time
```

Mechanical drivers

Relay

Relays are electromechanical devices that use electromagnets to connect or disconnect plates of a switch. Relays are used to control high power circuits with low power circuits. Circuits are mechanically isolated and thus protect logic control. Relays are used in household appliance automation, lighting and climate control.



Figure 214: 1 channel relay module¹²⁴⁾

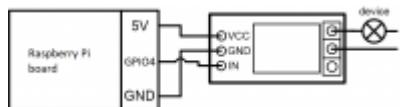


Figure 215: Raspberry Pi and 1 channel relay module schematics

The example code:

```
# Raspberry Pi Python sample code

import RPi.GPIO as GPIO
import time

REL = 18      # GPIO04 port

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(REL,GPIO.OUT)
print ("REL on")
GPIO.output(REL,GPIO.HIGH)
time.sleep(1)
print ("REL off")
GPIO.output(REL,GPIO.LOW)
```

Solenoid

Solenoids are devices that use electromagnets to pull or push iron or steel core. They are used as linear actuators for locking mechanisms indoors, pneumatic and hydraulic valves and in-car starters.

Solenoids and relays both use electromagnets and connecting them to Arduino is very similar. Coils need a lot of power, and they are usually connected to the power source of the circuit. Turning the power of the coil off makes the electromagnetic field to collapse and creates very high voltage. For the semiconductor devices protection, a shunt diode is used to channel the overvoltage. For the extra protection, optoisolator can be used.

4. IoT Hardware overview



Figure 216: Long-stroke latching solenoid¹²⁵⁾

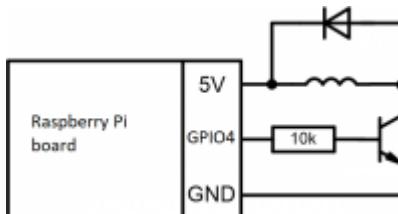


Figure 217: Raspberry Pi and solenoid schematics

The example code:

```
# Raspberry Pi Python sample code

import RPi.GPIO as GPIO
import time

SOL = 18      # GPIO04 port

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(SOL,GPIO.OUT)
print ("SOL on")
GPIO.output(SOL,GPIO.HIGH)
time.sleep(1)
print ("SOL off")
GPIO.output(SOL,GPIO.LOW)
```

DC motor (one direction)

Electric motor is an electro-technical device which can turn electrical energy into mechanical energy; motor turns because of the electricity that flows in its winding. Electric motors have seen many technical solutions over the year from which the simplest is the permanent-magnet DC motor.

DC motor is a device which converts direct current into the mechanical rotation. DC motor consists of permanent magnets in stator and coils in the rotor. By applying the current to coils, the electromagnetic field is created, and the rotor tries to align itself to the magnetic field. Each coil is connected to a commutator, which in turns supplies coils with current, thus ensuring continuous rotation. DC motors are widely used in power tools, toys, electric cars, robots, etc.

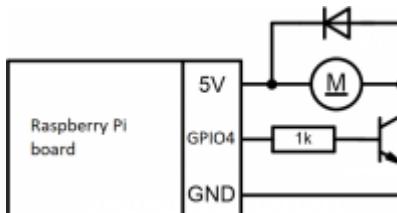
Figure 218: A DC motor¹²⁶⁾

Figure 219: Raspberry Pi and DC motor schematics

```
# Raspberry Pi Python sample code

import RPi.GPIO as GPIO
import time

DCM = 18      # GPIO04 port

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(DCM,GPIO.OUT)
print ("DCM on")
GPIO.output(DCM,GPIO.HIGH)
time.sleep(1)
print ("DCM off")
GPIO.output(DCM,GPIO.LOW)
```

Stepper motor

Stepper motors are motors, that can be moved by a certain angle or step. Full rotation of the motor is divided into small, equal steps. Stepper motor has many individually controlled electromagnets, by turning them on or off, the motor shaft rotates by one step. Changing switching speed or direction can precisely control turn angle, direction or full rotation speed. Because of very precise control ability they are used in CNC machines, 3D printers, scanners, hard drives etc. Example of use: <https://learn.adafruit.com/adafruit-arduino-lesson-16-stepper-motors/breadboard-layout>

Figure 220: A stepper motor¹²⁷⁾

4. IoT Hardware overview

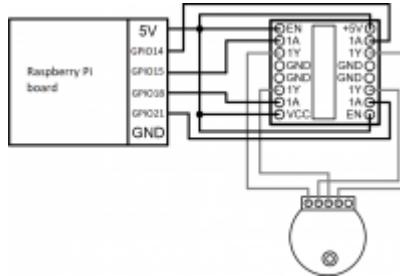


Figure 221: Raspberry Pi and stepper motor schematics

The example code:

```
# Raspberry Pi Python sample code
# from https://www.rototron.info/raspberry-pi-stepper-motor-tutorial/

from time import sleep
import RPi.GPIO as GPIO

DIR = 20      # Direction GPIO Pin
STEP = 21     # Step GPIO Pin
CW = 1         # Clockwise Rotation
CCW = 0        # Counterclockwise Rotation
SPR = 48       # Steps per Revolution (360 / 7.5)

GPIO.setmode(GPIO.BCM)
GPIO.setup(DIR, GPIO.OUT)
GPIO.setup(STEP, GPIO.OUT)
GPIO.output(DIR, CW)

step_count = SPR
delay = .0208

for x in range(step_count):
    GPIO.output(STEP, GPIO.HIGH)
    sleep(delay)
    GPIO.output(STEP, GPIO.LOW)
    sleep(delay)

sleep(.5)
GPIO.output(DIR, CCW)
for x in range(step_count):
    GPIO.output(STEP, GPIO.HIGH)
    sleep(delay)
    GPIO.output(STEP, GPIO.LOW)
    sleep(delay)

GPIO.cleanup()
```

This code may result in motor vibration and jerky motion especially at low speeds. One way to counter these result is with microstepping. Adding the code above avoid it:

```
MODE = (14, 15, 18)    # Microstep Resolution GPIO Pins
GPIO.setup(MODE, GPIO.OUT)
```

```
RESOLUTION = {'Full': (0, 0, 0),
              'Half': (1, 0, 0),
              '1/4': (0, 1, 0),
              '1/8': (1, 1, 0),
              '1/16': (0, 0, 1),
              '1/32': (1, 0, 1)}
GPIO.output(MODE, RESOLUTION['1/32'])

step_count = SPR * 32
delay = .0208 / 32
```

4.5.4. Raspberry Pi OS Guide

Supported Operating Systems (OS)

Raspberry Pi all models are based on ARM processors which are typically quad-core Cortex-A7 CPUs. This means that most of popular multitasking OS systems can be uploaded and used to create and develop user software operations. The list of supported OS systems contains Linux, Windows and thirty part OS systems. The following list of OS are specially designed for Raspberry Pi boards:



Figure 222: *Raspbian*¹²⁸⁾



Figure 223: *Ubuntu Mate*¹²⁹⁾



Figure 224: *Snappy Ubuntu Core*¹³⁰⁾



Figure 225: *Windows 10 IOT Core*¹³¹⁾

4. IoT Hardware overview



Figure 226: *OSMC*¹³²⁾



Figure 227: *LibreELEC*¹³³⁾



Figure 228: *Pinet*¹³⁴⁾



Figure 229: *Risc OS*¹³⁵⁾

Before start installing the OS system on the Raspberry Pi board developer must prepare his hardware for it¹³⁶⁾. It means that the minimum hardware equipment is needed:

1. Raspberry Pi board,
2. Monitor or TV with HDMI port,
3. HDMI cable,
4. USB keyboard,
5. USB mouse,
6. Power supply,
7. at least 8GB micro SD card (C10 class is welcome).

Connecting all establishes the minimum PC desktop kit which will allow to install and run the selected OS system on the SD card.

Downloading OS system

There is few ways to get the right OS system for Raspberry Pi board:

1. Buy pre-installed SD card from RS¹³⁷⁾ or PiHut¹³⁸⁾,

2. Install Raspbian with NOOBS¹³⁹⁾,
3. Download Raspbian image directly from Raspberry Pi software repository¹⁴⁰⁾,
4. Download the Windows 10 IOT OS from Microsoft Windows Insider Preview¹⁴¹⁾,

Other Raspberry OS systems

For other then Windows and NOOBS systems use the *Etcher SD*¹⁴²⁾ card image utility which is designed to format and upload to the SD card different operating systems images. Then follow its instructions.

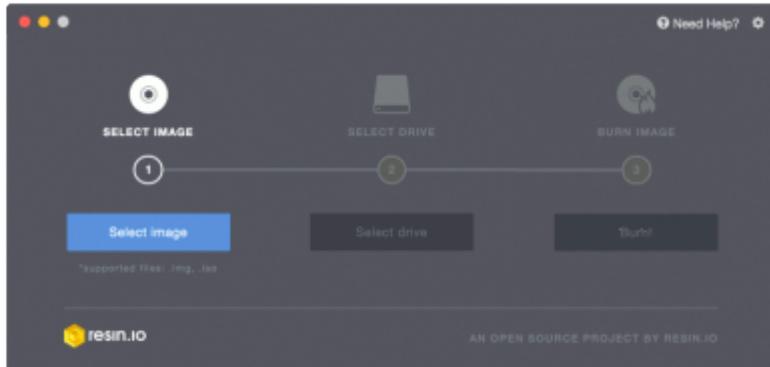


Figure 230: *Etcher SD card image utility view*

4.5.5. Programming fundamentals Raspbian OS

Installing the Raspbian OS

To install the OS system on SD card the best way is to use specially designed software which will provide SD card formatting tool.

Step 1

Download and install the **SD Formatter**¹⁴³⁾ tool. Run the SD Formatter tool:

4. IoT Hardware overview

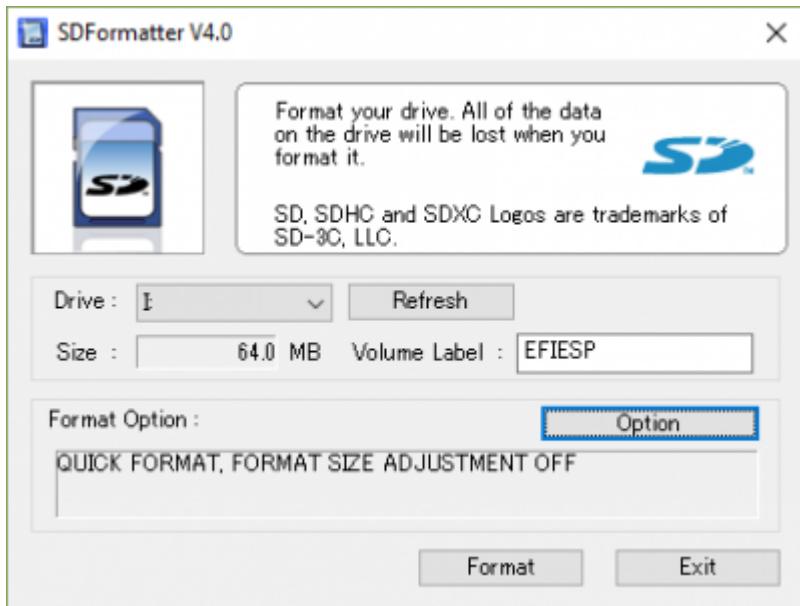


Figure 231: *SD Formatter* view

Step 2

Insert the SD card into the computer SD card reader,

Step 3

Run the **SD Formatter**, select the drive letter for the SD card and format it,

Step 4

Simply drag and drop the extracted NOOBS OS image files from unzipped NOOBS folder onto the SD card drive. The necessary files will be transferred to the SD card.

Step 5

Gently remove SD card from the reader and push it into the Raspberry Pi SD card slot.

Step 6

Power on the Raspberry Pi board and follow its instructions.

After the board reboot the Raspbian screen displays:

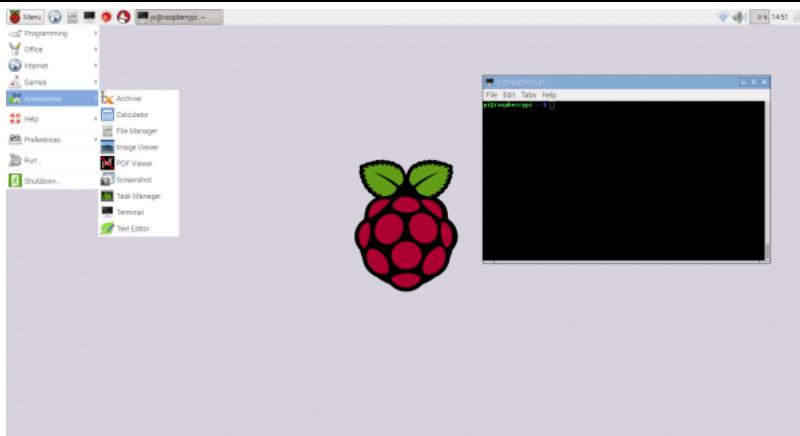


Figure 232: Raspbian desktop view

Raspberry Pi Python programming Guide

Python language

 Python belongs to the high-level programming languages class which was first time developed by Guido van Rossum in 1991. The Python is similar to C++, C# or Java programming languages. It is very useable with a clean syntax and easy to learn even for programming beginners.

Raspberry Raspbian OS is shipped with pre installed two versions of Python language: Python2 and Python3 which are available from the Raspbian Menu:



Figure 233: Python menu view

Choosing the Python version from the menu, the command window with cursor opens:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 20 2018, 15:12:02)
[MSC v.1900 64 bit (AMD64)] on Win32
```

4. IoT Hardware overview

```
Type "copyright", "credits" or "license()" for more information.  
>>>
```

To test the simply program "**Hello World!**" User just can entry:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 20 2018, 15:12:02)  
[MSC v.1900 64 bit (AMD64)] on Win32  
Type "copyright", "credits" or "license()" for more information.  
>>>printf("Hello World")  
Hello World!  
>>>
```

Writing the same code in C language will look following:

```
#include <stdio.h>  
int main()  
{  
    printf ("Hello World!");  
    return 0;  
}
```

Python program features

Python can automate tasks using the batch commands renaming and move large amounts of files like shell scripts. It can be used as a command line with IDLE, Python's REPL (read, eval, print, loop) functions. However, there are more useful tasks which can be done with Python. For example, Python can be used to program things like:

- Web applications
- Desktop applications and utilities
- Special GUIs
- Small databases
- 2D games

Because the Python stay very popular, it has a large collection of libraries, which speeds up the development process. There exist libraries for – game programming, rendering graphics, GUI interfaces, web frameworks, and scientific computing.

Many of the programmings stuff in C language can also be programmed in Python. Python is generally slower at computations regarding the C compiler, but its ease for use, which makes Python an very popular tool for prototyping programs and designing applications which are not computationally intensive. One of the best Python tutorials can be found in the book: *Learning Python, 5th Edition by Mark Lutz*.

Installing and updating Python

Python 2 and Python 3 come pre-installed on Raspbian OS systems, but if necessary to install Python on another Linux OS or to update it, the simple commands can be executed at the command prompt:

```
sudo apt-get install python3
```

Installs or updates Python 3.

```
sudo apt-get install python
```

Installs or updates Python 2.

Opening the PYTHON REPL

To access the Python REPL (where the user can type Python commands just like the command line) the user can enter `python` or `python3` commands depending on which version of Raspbian to use in the command prompt.

```
pi@raspberrypi ~
login as: pi
pi@10.0.0.53's password:
Linux raspberrypi 3.18.11-v7+ #781 SMP PREEMPT Tue Apr 21 18:07:59 BST 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Aug 11 20:33:47 2015 from mediastudio.local
pi@raspberrypi ~ $ python
Python 2.7.3 (default, Mar 18 2014, 05:13:23)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
pi@raspberrypi ~ $ python3
Python 3.2.3 (default, Mar  1 2013, 11:53:50)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Figure 234: Python REPL view

Pressing (CTRL-D) exits the REPL.

Running a Python Program

To run the program without making it executable, the user must navigate to the location where the file exists and enter the command:

```
python hello-world.py
```

Make Python file executable

Making a Python program executable allows to run the program without entering `python` before the file name. User can make a file executable executing the following commands in the command prompt:

```
chmod +x file-name.py
```

Now to run the program:

4. IoT Hardware overview

```
./file-name.py
```

Additional resources for Python programming

1. The Python syntax and semantics: Python Semantics
2. The Python Package Index (PyPi): PyPi
3. The Python Standard Library: PSL

Python Building the first program

The IDLE Editor is designed to write and execute programs using Python language. The IDLE Editor is a part of the Raspbian OS. To execute it User can simply press (CTRL+N) and enter the Python commands like:

```
# Raspberry Pi Python sample code

import RPi.GPIO as GPIO
import time

# blinking function
def blink(pin):
    GPIO.output(pin,GPIO.HIGH)
    time.sleep(1)
    GPIO.output(pin,GPIO.LOW)
    time.sleep(1)
return

# Use the Raspberry Pi GPIO pin numbers
GPIO.setmode(GPIO.BOARD)
# set up GPIO output channel
GPIO.setup(5, GPIO.OUT)
# blink GPIO5 5 times
for i in range(0,5):
    blink(5)
GPIO.cleanup()
```

To save the the file and run it User must simply press F5. This sample above blinks the LED diode connected to the GPIO17 pin on the Raspberry Pi board.

It is also possible to run Python programs without using Python interpreter. Programs like Py2exe or Pyinstaller are designed to package the Python code into stand-alone executable programs.

Python Data types and Variables

Python aims to be consistent and straightforward in the design of its syntax. The best advantage of this language is that it can dynamically set the variable types depending on values types which are set for variables.

Base types

Python has a wide range of data types, like many simple programming languages:

1. Number,

2. String,
3. List,
4. Tuple
5. Dictionary,

Numbers

Standard Python methods are used to create the Numbers:

```
var = 1234      # Creates Integer number assignment  
var = 'George' # Creates String type var
```

Python can automatically convert types of the number from one type to another. The Type can be also defined explicitly.

```
int a = 10  
long a = 123L  
float a = 12.34  
complex a = 3.23J  
<code>  
  
==String==  
To define Strings use enclosing characters in quotes.  
Python uses single quotes ', double " " and triple """ to denote strings.  
<code Python>  
Name = "George"  
lastName = "Smith"  
message = """this is the string message which is spanning across multiple lines."""
```

List

List contains a series of values. To declare list variables uses brackets []

```
A = [] # blank list variable  
B = [1, 2, 3] # list with 3 numbers  
C = [1, 'aa', 3] # list with different types
```

List are zero-based indexed. Data can be assigned to a specific element of the list using an index into the list.

```
mylist[0] = 'sasa'  
mylist[1] = 'wawa'  
  
print mylist[1]
```

List aren't limited to a single dimension.

```
myTable = [[],[]]
```

In two-dimensional array the first number is always the rows number, when the second is the columns number.

4. IoT Hardware overview

Tuple

Python Tuples are defined as a group of values like a list and can be processed in similar ways. When assigned Tuples got the fixed size. In Python, the fixed size is immutable. The lists are dynamic and mutable. To define Tuples parenthesis () must be used.

```
TestSet = ('Piotr', 'Jan', 'Adam')
```

Dictionary

To define the Dictionaries in the Python the lists of Key:Value pairs are used. This datatype is used to hold a related information that can be associated through Keys. The Dictionary is used to extract a value based on the key name. Lists uses the index numbers to access its members, when dictionaries uses a key. Dictionaries generally are used to sort, iterate and compare data.

To define the Dictionaries the braces ({}) are used with pairs separated by a comma (,) and the key values associated with a colon(:). Dictionaries Keys must be unique.

```
box_nbr = {'Alan': 111, 'John': 222}
box_nbr['Alan'] = 222      # set the associated 'Alan' key to value 222'
print (box_nbr['John'])    # print the 'John' key value.
box_nbr['Dave'] = 111      # Add a new key 'Dave' with value 111
print (box_nbr.keys())     # print the keys list in the dictionary
print ('John' in box_nbr)  # check if 'John' is in the dictionary.
                           # This returns true.
```

All variables in Python hold references to objects, and are passed to functions. Function can't change the value of variable references in its body. The object's value may be changed in the called function with the "alias".

```
>>> alist = ['a', 'b', 'c']
>>> def myfunc(al):
    al.append('x')
    print al

>>> myfunc(alist)
['a', 'b', 'c', 'x']
>>> alist
['a', 'b', 'c', 'x']
```

Python Program control structures

if Statements

If an expression returns **True** statements are carried out. Otherwise they aren't.

```
if expression:
    statements
```

Sample:

```
no = 11
if no >10:
    print ("Greater than 10")
    if no <=30
        printf ("Between 10 and 30")
```

Output:

```
>>>
Greater than 10
Between 10 and 30
>>>
```

else Statements

An **else** statement follows an **if** statement and contains code that is called when the if statement is **False**.

```
x = 2
if x == 6
    printf ("Yes")
else:
    printf ("No")
```

elif Statements

The **elif** (shortcut of else if) statement is used when changing **if** and **else** statements. A series of **if elif** statements can have a final **else** block, which is called if none of the **if** or **elif** expression is **True**

```
num = 12
if num == 5:
    printf ("Number = 5")
elif num == 4:
    printf ("Number = 4")
elif num == 3:
    printf ("Number = 3")
else:
    printf ("Number = 12")
```

Output:

```
>>>
Number = 12
>>>
```

Boolean Logic

Python uses logic operators like **and**, **or** and **not**.

The **and** operator uses two arguments, and evaluates to **True** if, and only if, both of the arguments are **True**. Otherwise, it evaluates to **False**

4. IoT Hardware overview

```
>>> 1 == 1 and 2 == 2
True
>>> 1 == 1 and 2 == 3
False
>>> 1 != 1 and 2 == 2
False
>>> 4 < 2 and 2 > 6
False
>>>
```

Boolean operator **or** uses two arguments. and evaluates as **True** if either (or both) of its arguments are **True**, and **False** if both arguments are **False**.

The result of **not True** is **False**, and **not False** goes to **True**

```
>>> not 2 == 2
False
>>> not 6 > 10
True
>>>

== Operator Precedence==
Operator Precedence uses mathematical idea of operation order i.e. multiplication begin per
<code Python>
>>> False == False or True
True
>>> False == (False or True)
False
>>> (False == False) or True
>>>True
>>>
```

Python Looping

while Loop

An **if** statement is run once if its condition evaluates to **True**, and never if it evaluates to **False**.

A **while** statement is similar, except that it can be run more than once. The statements inside it are repeatedly executed, as long as the condition holds. Once it evaluates to **False**, the next section of code is executed.

```
i = 1
while i<=4:
    print (i)
    i+=1
print ('End')
```

Output:

```
>>>
1
2
```

```
3  
4  
End  
>>>
```

The **infinite loop** is a particular kind of the while loop, it never stops running. Its condition always remains **True**

```
while 1 == 1:  
    print ('in the loop')
```

To end the **while** loop prematurely, the **break** statement can be used. When encountered inside a loop, the **break** statement causes the loop to finish immediately.

```
i = 0  
while 1==1:  
    print (i)  
    i += 1  
    if i >=3:  
        print('breaking')  
        break;  
    print ('finished')
```

Output:

```
>>>  
0  
1  
2  
3  
breaking  
finished  
>>>
```

Another statement that can be used within loops is **continue**.

Unlike **break**, **continue** jumps back to the top of the loop, rather than stopping it.

```
i = 0  
while True:  
    i+=1  
    if i == 2:  
        printf ('skipping 2')  
        continue  
    if i == 5:  
        print ('breaking')  
        break  
    print (i)  
print ('finished')
```

Output:

```
>>>
```

4. IoT Hardware overview

```
1
skipping 2
3
4
breaking
finished
>>>
```

for Loop

```
n = 9
for i in range (1,5):
    ml = n * i
    print ("{} * {} = {}".format (n, i, ml))
```

Output:

```
>>>
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
>>>
```

Python sub-programs and interrupts

Subprograms

One of the most important in mathematics concept is to use functions. Functions in computer languages implement mathematical functions. The executing function produces one or more results, which are dependent by the parameters passed to it.

In general, a function is a structuring element in the programming language which groups a set of statements so they can be called more than once in a program. Programming without functions will need to reuse code by copying it and changing its different context. Using functions enhances the comprehensibility and quality of the program. It also lowers the memory usage, development cost and maintenance of the software.

Different naming is used for Functions in programming languages, e.g. as subroutines, procedures or methods.

Python language defines function by a def statement. The function syntax looks:

```
def function-name(Parameter list):
    statements, i.e. the function body
```

Function bodie can contain one or more return statement. It can be situated anywhere in the function body. A return statement ends the function execution and returns the result, i.e. to the caller. If the return statement does not contain expression, the value "None" is returned.

```
def Fahrenheit(T_in_celsius):
    """ returns the temperature in degrees Fahrenheit """
    return (T_in_celsius * 9 / 5) + 32

for t in (22.6, 25.8, 27.3, 29.8):
    print(t, ": ", fahrenheit(t))
```

Output:

```
>>>
22.6 : 72.68
25.8 : 78.44
27.3 : 81.14
29.8 : 85.64
>>>
```

Optional Parameters

Functions can be called with optional parameters, also named default parameters. If function is called without parameters the default values are used. The following code greets a person. If no person name is defined, it greets everybody:

```
def Hello(name="everybody"):
    """ Say hello to the person """
    print("Hello " + name + "!")

Hello("George")
Hello()
```

Output:

```
>>>
Hello George!
Hello everybody!
>>>
```

Docstrings

The string is usually the first statement in the function body, which can be accessed with `function_name.__doc__`. This is Docstring statement.

```
def Hello(name="everybody"):
    """ Say hello """
    print("Hello " + name + "!")
print("The docstring of the function Hello: " + Hello.__doc__)
```

Output:

```
>>>
The function Hello docstring: Say hello
>>>
```

4. IoT Hardware overview

Keyword Parameters

The alternative way to make function calls is to use keyword parameters. The function definition stay unchanged.

```
def sumsub(a, b, c=0, d=0):
    return a - b + c - d
print(sumsub(12,4))
print(sumsub(42,15,d=10))
```

Only Keyword parameters are valid which are not used as positional arguments. If keyword parameters don't exist, the next call to the function will need all four arguments, even if the c needs just the default value:

```
print(sumsub(42,15,0,10))
```

Return Values

In above examples, the return statement exist in sumsub but not in Hello function. The return statement is not mandatory. if explicitly return statement doesn't exist in the sample code it will not show any result:

```
def no_return(x,y):
    c = x + y
res = no_return(4,5)
print(res)
```

Any result will not be displayed in:

```
>>>
```

Executing this script, the None will be printed. if a function doesn't contain expression the None will also be returned:

```
def empty_return(x,y):
    c = x + y
    return
res = empty_return(4,5)
print(res)
```

Otherwise the expression value following return will be returned. In this example 11 will be printed:

```
def return_sum(x,y):
    c = x + y
    return c
res = return_sum(6,5)
print(res)
```

Output:

```
>>>
9
>>>
```

Multiple Values Returning

Any function can return only one object. An object can be a numerical value - integer, float, list or a dictionary. To return i.e. three integer values, we can return a list or a tuple with these three integer values. It means that function can indirectly return multiple values. This following example calculates the Fibonacci boundary for a positive number, returns a 2-tuple. The Largest Fibonacci Number smaller than x is the first and the Smallest Fibonacci Number larger than x is next. The return value is stored via unpacking into the variables lub and sup:

```
def fib_intervall(x):
    """ returns the largest Fibonacci number, smaller than x and the lowest
    Fibonacci number, higher than x"""
    if x < 0:
        return -1
    (old, new, lub) = (0,1,0)
    while True:
        if new < x:
            lub = new
            (old,new) = (new,old+new)
        else:
            return (lub, new)

    while True:
        x = int(input("Your number: "))
        if x <= 0:
            break
        (lub, sup) = fib_intervall(x)
        print("Largest Fibonacci Number < than x: " + str(lub))
        print("Smallest Fibonacci Number > than x: " + str(sup))
```

Interrupts handling

Following Python rules for working with signals and their handlers are listed below:

1. A particular signal handler, once set, remains installed until it is explicitly reset (Python emulates the BSD style interface), except the SIGCHLD handler, which follows the underlying implementation.
2. The critical section signal can't be "blocked" temporarily (it is not supported by all Unix flavours).
3. Python signal handlers are called asynchronously as far as the Python user is concerned, they can only occur between the "atomic" instructions of the Python interpreter. Signals arriving during long calculations implemented in C (such as regular expression matches on large bodies of text) may be delayed for an arbitrary amount of time.
4. When a signal arrives during an I/O operation, it is possible that the I/O operation raises an exception after the signal handler returns. It depends on the underlying Unix system's semantics.

4. IoT Hardware overview

5. The C signal handler always returns, it makes little sense to catch synchronous errors like SIGFPE or SIGSEGV.
6. Python installs a small number of signal handlers by default: SIGPIPE is ignored and SIGINT is translated into a KeyboardInterrupt exception. All of them can be overridden.
7. If both signals and threads are used in the same program some care must be taken. When using signals and threads simultaneously always perform signal() operations in the main thread of execution.
8. Any thread can perform an alarm(), getsignal(), pause(), setitimer() or getitimer(); only the main thread can set a new signal handler, and the main thread will be the only one to receive signals. It means that signals can't be used as a means of inter-thread communication.
9. Use locks instead.

The example program is shown below. It uses the alarm() function to limit the time spent waiting to open a file; very useful if the file needs to be transmitted over a serial device that may not be turned on, which would typically cause the os.open() to hang indefinitely. The best solution is to make 5-second alarm before opening the file; if the operation takes too long, the alarm signal will be sent, and the handler raises an exception.

```
# Raspberry Pi Python sample code
import RPi.GPIO as GPIO
import time

state = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(19, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def interrupt_handler(channel):
    global state

    print("interrupt handler")

    if channel == 19:
        if state == 1:
            state = 0
            print("state reset by event on pin 19")
        elif channel == 26:
            if state == 0:
                state = 1
                print("state set by event on pin 26")

GPIO.add_event_detect(26, GPIO.RISING,
                     callback=interrupt_handler,
                     bouncetime=200)

GPIO.add_event_detect(19, GPIO.RISING,
                     callback=interrupt_handler,
                     bouncetime=200)
```

```
while (True):  
    time.sleep(0)
```

4.5.6. Programming fundamentals Windows 10 IOT Core

Installing the Windows 10 IOT Core

Microsoft Windows 10 IOT OS system is available for download from Windows Insider Preview Downloads page¹⁴⁴⁾.

Step 1

Register into the Microsoft Insider Program. To download images of the Microsoft IOT Core user must be logged into Microsoft Insider Program web page.

Step 2

On the download page User must choose which OS edition he wants to use in his project ieg. Windows 10 IoT Core Insider preview - build 17083 or 17035. The core numbers are changing depending on the latest developer editions available in the Microsoft repository. Microsoft IOT development policy is straightly tied with the latest Visual Studio environment. To fully use its power users are asked to use latest Visual Studio and Windows 10 IOT core builds in the development process synchronously. The Windows 10 IOT Core platform is still under development and improvement.

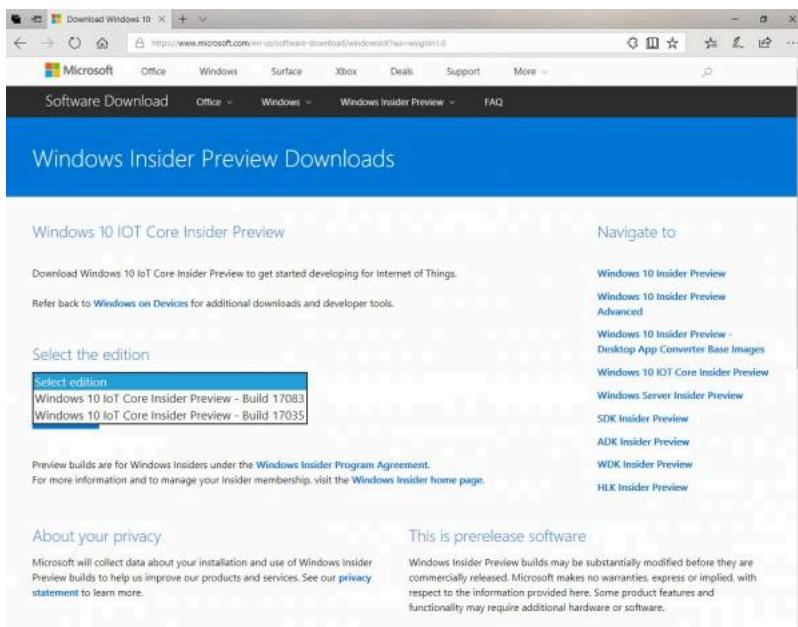


Figure 235: Windows Insider Program Win10 Core download page

4. IoT Hardware overview

Step 3

Install the right Windows10_InsiderPreview_IoTCore_RPi_ARM32_en-us_17035.iso image on your Windows 10 PC. This package installs the *Windows IOT Core Image Helper* application and stores the latest Raspberry Pi Windows 10 core image *flash.ffc* file into the C:\Program Files (x86)\Microsoft IoT\ directory,

Step 4

Insert the SD card to your computer SD cards reader slot,

Step 5

Run the *Windows IOT Core Image Helper* and follow its instructions - select the SD card drive letter, select the right FFU image in the C:\Program Files (x86)\Microsoft IoT\FFU\RaspberryPi2 folder,



Figure 236: *Windows IOT Core Helper view*

Step 6

Start formatting the SD card and install the FFU image on it,

Step 7

Gently remove SD card from the reader and push it into the Raspberry Pi SD card slot.

Step 8

Power on the Raspberry Pi board and follow the Windows 10 Core setup commands configuring the Windows 10 Core features.

After the board reboot the Main Windows 10 Core screen displays:



Figure 237: *Windows 10 Core system view*

Raspberry Pi Programming Guide

This chapter describes the typical programming technics used in Raspberry Pi boards developing projects.

Raspberry Pi under Windows 10 IOT Core

To create and develop control applications on the Raspberry Pi boards needs the following development environment:

1. PC with Windows 10 System installed,
2. Visual Studio 2015 or higher,
3. Raspberry PI 2 or 3 board with Windows 10 IOT Core installed,
4. Configured TCP/IP network for Raspberry Pi and Windows 10 Desktop computer

4. IoT Hardware overview

(Local LAN or Wi-Fi subnet),

Programming skills needed:

1. C# language knowledge,
2. XML/XAML language knowledge,
3. Windows API understanding.

For better development Raspberry applications, the Windows IoT Remote Client is welcome. This application is available for download from Microsoft Store. This application captures keyboard, mouse and screen from the Raspberry Pi board running the Windows 10 IOT Core system on the desktop PC. It allows developers to use standalone Raspberry board without a connected mouse, keyboard and monitor to it.

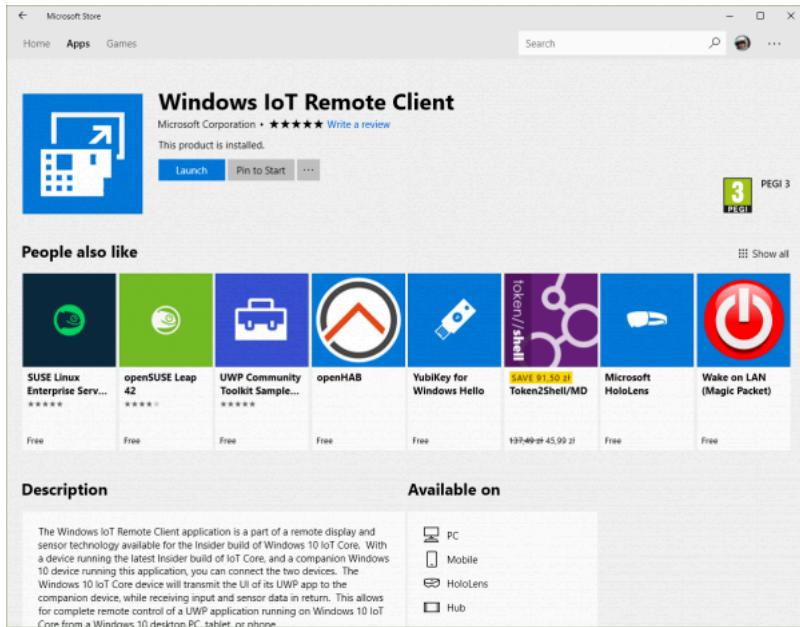


Figure 238: Microsoft Store - Windows IoT Remote Client

To write and develop applications under Windows 10 IOT Core developer must possess basic knowledge of how Windows operating system interacts with User applications. The major advantage of using Windows 10 IOT Core is that Microsoft concept based on use the same Kernel API available on different hardware platforms - desktop PCs, IOT boards suitable to run Windows Core, Tablets, phones etc. It reduces development costs due to the unifying system environment, and the only difference is in display view of same application code written in C#/C++. Windows 10 Core is specially designed to handle applications working as standalone on the IoT platforms in 24/7 time model.

Configuring the Windows 10 IOT Core platform

After installing the Windows 10 IOT Core, the developer must configure IoT platform using Windows Device Portal (WDP).

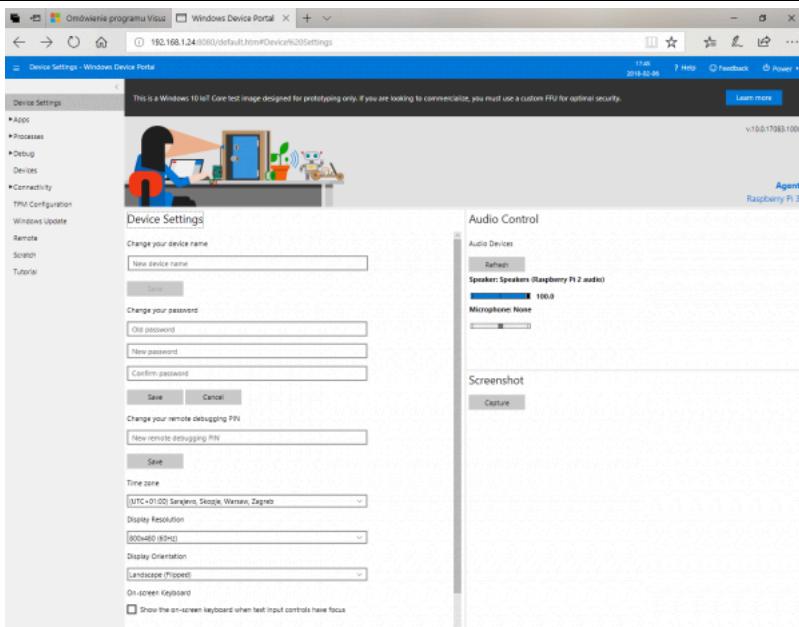


Figure 239: Windows Device Portal view

IoT board can be managed using any Internet browser - Chrome, Microsoft Edge, Firefox etc. To open the WDB portal on the IOT board user must enter the board IP address - IPaddress:8080/default.htm. The site is protected with username/password. Default account credentials are: **administrator/p@ssw0rd**. Following tabs in the WDP, it is possible to configure all necessary IoT platform settings, check the current board status, download development crash/debug information, configure network/Bluetooth settings, download drivers and configure security TPM modules. If all tasks are ready, the developer can start to write his own IoT application under Windows 10 IOT Core. Following steps are recommended before IoT board will be used for application:

Step 1

In the **Device Settings** user is recommended to **Change your device name**. The default name is *minwinpc*. The aim to change it is that if a user uses many of the IoT devices connected to the same network segment, it is difficult to recognise which role each device is set for. Enumerating IoT devices will show boards with the same name but with different IP addresses. Proper naming will make it easy to know what role each device plays.

Step 2

Because RPI boards don't have their own RTC clock modules Windows 10 IOT Core sets the time using the NTP services during its work. So very important in the industrial implementations and in a case when the time is important in developed applications is to set the proper time zone for the board. In the **Device settings** user is recommended to select proper **Time zone**

4. IoT Hardware overview

Step 3

Security reasons - the default password for the newly flashed device is *p@ssw0rd*. It is strongly recommended right after the first board boot to change it - to set it unique! It will prevent the IoT device from remote hacking. The password can be changed in **Device Settings** tab.

Step 4

The Windows 10 IOT Core comes with *Cortana* service ready. If the board is equipped with microphone and speakers, it is always possible to turn the *Cortana* service on for voice commands communication with the board.

Step 5

If the IoT board needs special hardware connected to it then in the **Devices/Device Manager** user can upload and install appropriate driver for it in case if it is not preinstalled in the IoT Core.

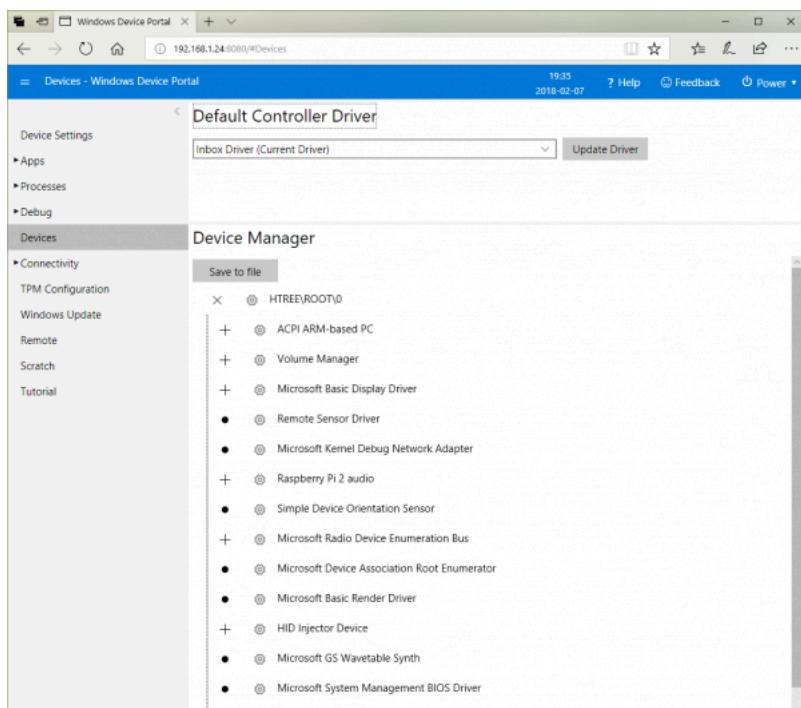


Figure 240: Device Manager view

Step 6

Raspberry Pi boards 1/2/3 are equipped with network connection modules. In case if the board under Windows 10 IOT Core is connected to LAN RJ45 connector the IP number can be set via DHCP server. In case if user wants to use Wi-Fi connection or to activate

4.5. Raspberry Pi Overview

Bluetooth then he can do it directly on the board main display or manage them via Windows Device Portal.

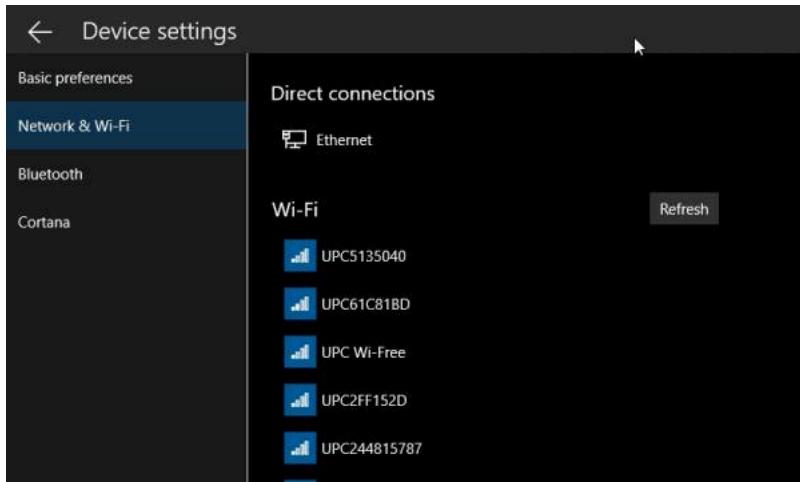


Figure 241: Network& Wi-Fi view

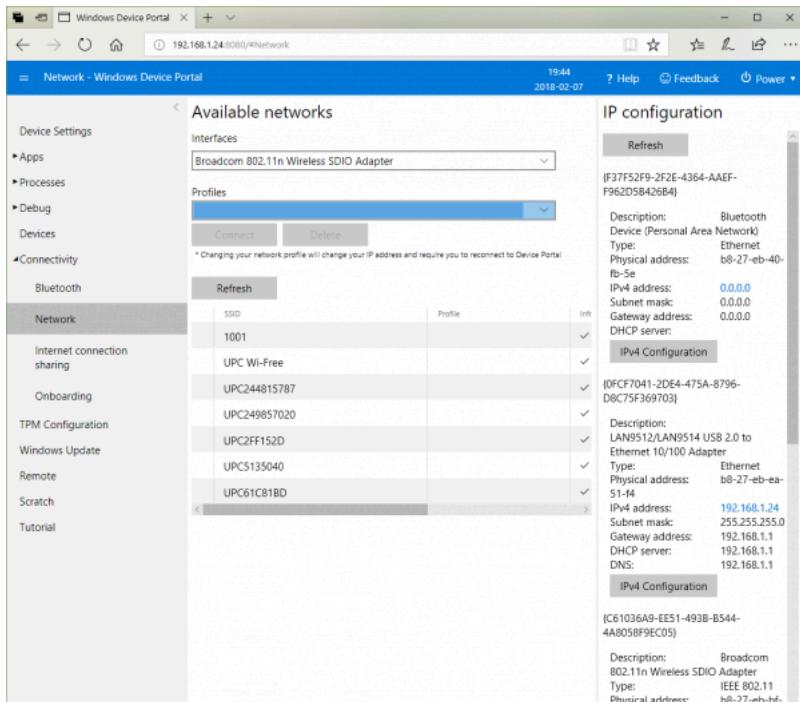


Figure 242: Network view

4. IoT Hardware overview

Step 7

Security. In a case when IoT device must be protected for remote hacking one of solutions is to use Trusted Platform Modules (TPM) module following ISO/IEC 11889 standards for a secure cryptoprocessor, a dedicated microcontroller designed to secure hardware through integrated cryptographic keys. The chip contain physical security mechanisms to protect it from tampering, and malicious software is unable hack the TPM security functions. Some of the TPM key advantages are:

- Generate and store the cryptographic keys.
- Use the TPM unique RSA key technology for platform device authentication, which is burned into the chip.
- Help ensure platform integrity.

The most common TPM functions are used for system integrity measurements, key creation and use. During the system boot process, the boot code is loaded (including firmware and OS components) and can be measured and recorded in the TPM module. The integrity measurements are used for evidence how OS started and to be sure when correct boot software was used with TPM-based key. Windows 10 IoT Core supports few TPM modules standards which can be connected to the 40-pin GPIO connector.



Figure 243: *TPM module view*

Under the TPM Configuration tab in the Windows Device Portal user can select the right communication protocol for the TPM module installed in the Raspberry Pi board. Then appropriate driver for the TPM module can be installed in the Device Manager tab.

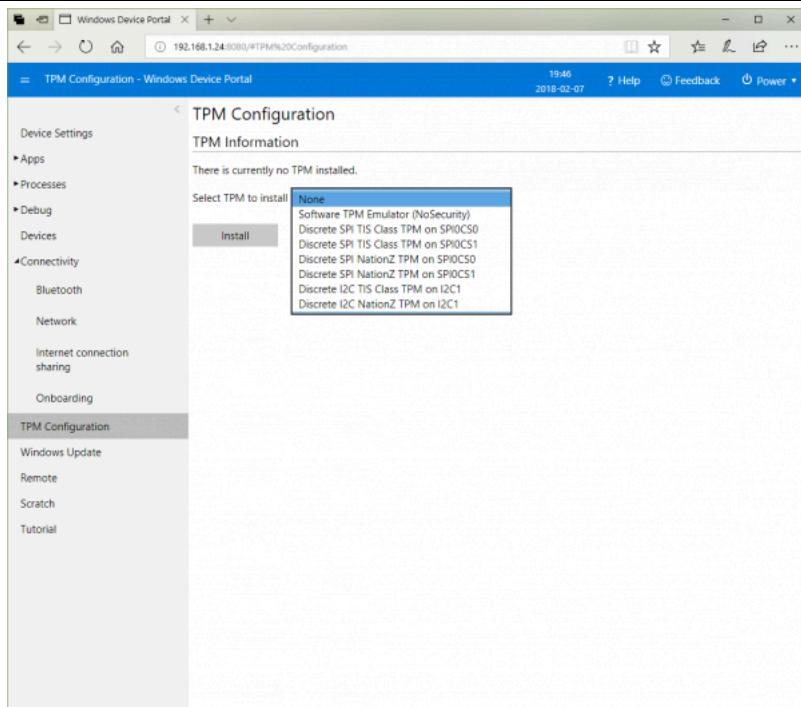


Figure 244: *TPM Configuration view*

RPI Windows 10 IoT sample project

Create simple Hello World application for Raspberry Pi board

To create simple Hello Word application under Windows 10 IOT Core the Visual Studio 2015 or higher is needed. Visual Studio must be installed with the Universal Windows Platform development extension.

4. IoT Hardware overview

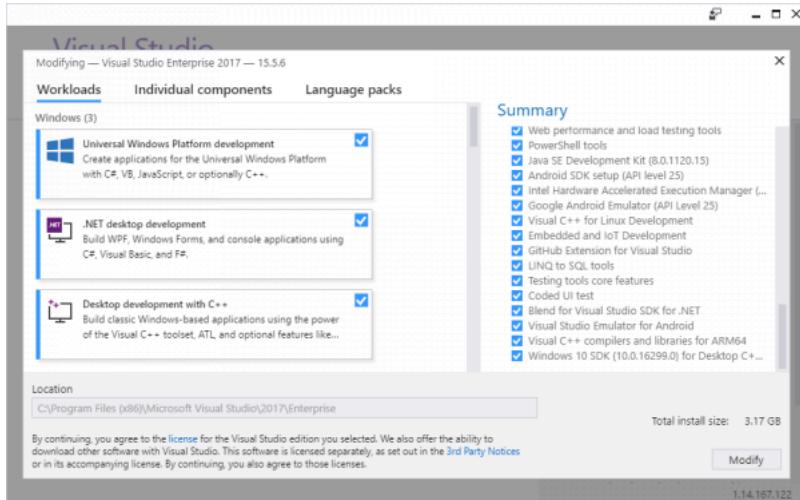


Figure 245: *Visual Studio UWP development view*

Step 1

Create new UWP project choosing the Windows Universal/Blank App Project.

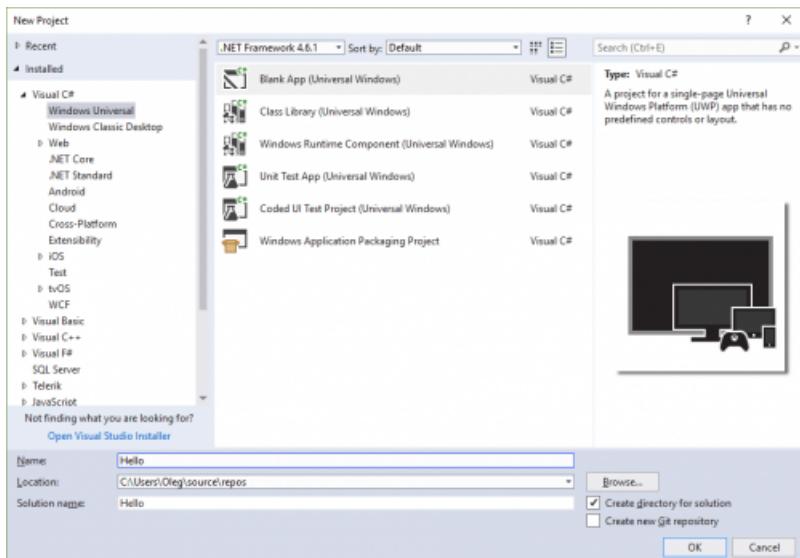


Figure 246: *Create new UWP App*

Step 2

Select target version (according to Raspberry Pi Windows 10 IOT Core build version)

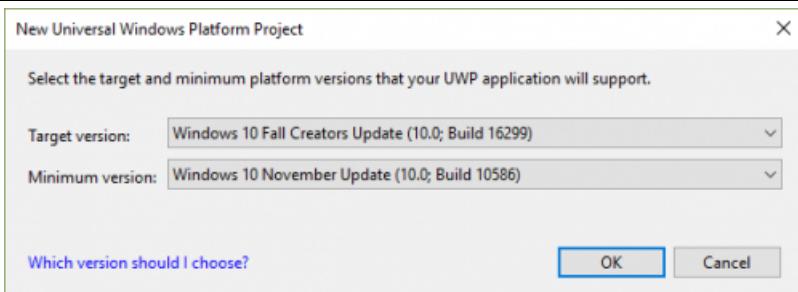


Figure 247: Select target version

Step 3

Create Hello solution.

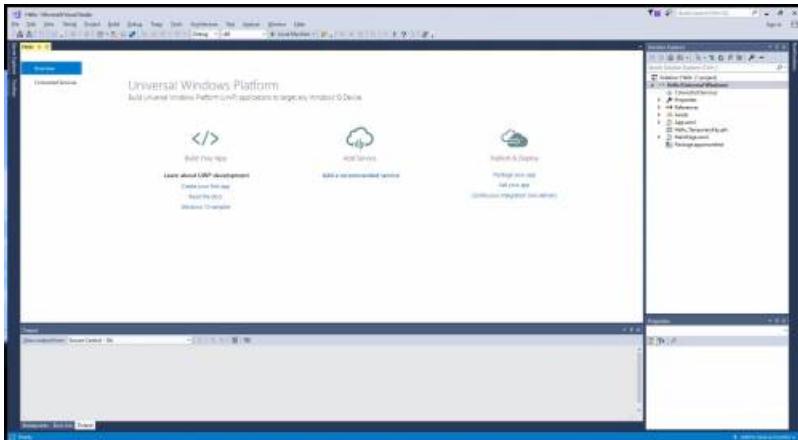


Figure 248: Select target version

Step 4

Design the application screen modifying the MainPage.xaml file. To add different screen features use the Toolbox/All XAML controls.

4. IoT Hardware overview



Figure 249: Add TextBlock control

Step 5

Modify the MainPage.cs file content if you need controls events programming.

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using Windows.Foundation;
7  using Windows.Foundation.Collections;
8  using Windows.UI.Xaml;
9  using Windows.UI.Xaml.Controls;
10 using Windows.UI.Xaml.Controls.Primitives;
11 using Windows.UI.Xaml.Data;
12 using Windows.UI.Xaml.Input;
13 using Windows.UI.Xaml.Media;
14 using Windows.UI.Xaml.Navigation;
15 
16 // The Blank Page item template is documented at https://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409
17 
18 namespace Hello
19 {
20     /// <summary>
21     /// An empty page that can be used on its own or navigated to within a Frame.
22     /// </summary>
23     public sealed partial class MainPage : Page
24     {
25         public MainPage()
26         {
27             this.InitializeComponent();
28         }
29 
30         private void TextBlock_SelectionChanged(object sender, RoutedEventArgs e)
31         {
32         }
33     }
34 }
35 
36 
```

Figure 250: Add TextBlock control

Step 6

Compile and run Hello solution. Choosing the Solution Platform to the x86 user will be able to debug and run Hello application on the computers desktop emulator. This step is useful for program touchscreen design but is not capable of testing the sensors and controls programming due to software emulator restrictions.

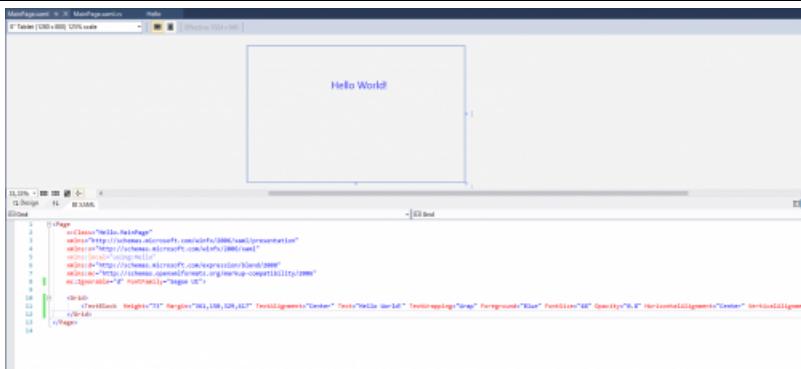


Figure 251: Run Hello solution

Software emulators aren't capable of simulating their behaviour. To use sensors and controls instead, the Solution Platform must be changed to the ARM platform in the VC Solution Configuration property. To debug it application package must be then transferred to the real IoT device.

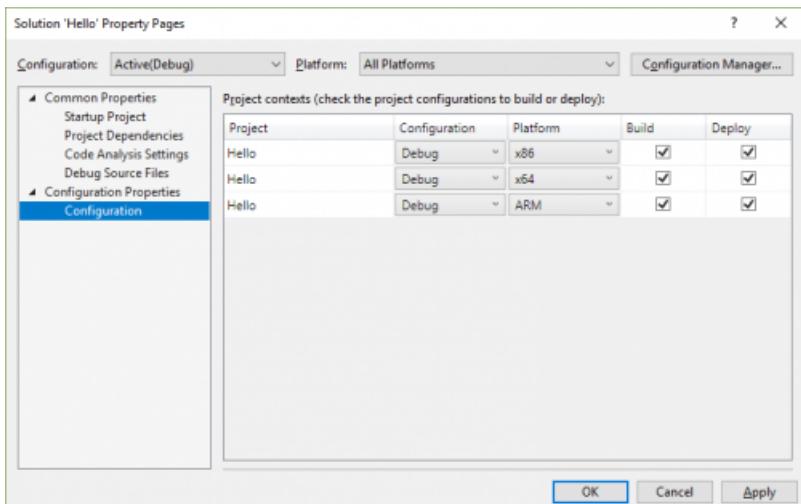


Figure 252: Select hardware platform for the solution

Step 7

To deploy and debug the application package on the real IoT device user must configure the debug application settings. In the Debug property page user must enter the proper Remote IoT device IP number.

4. IoT Hardware overview

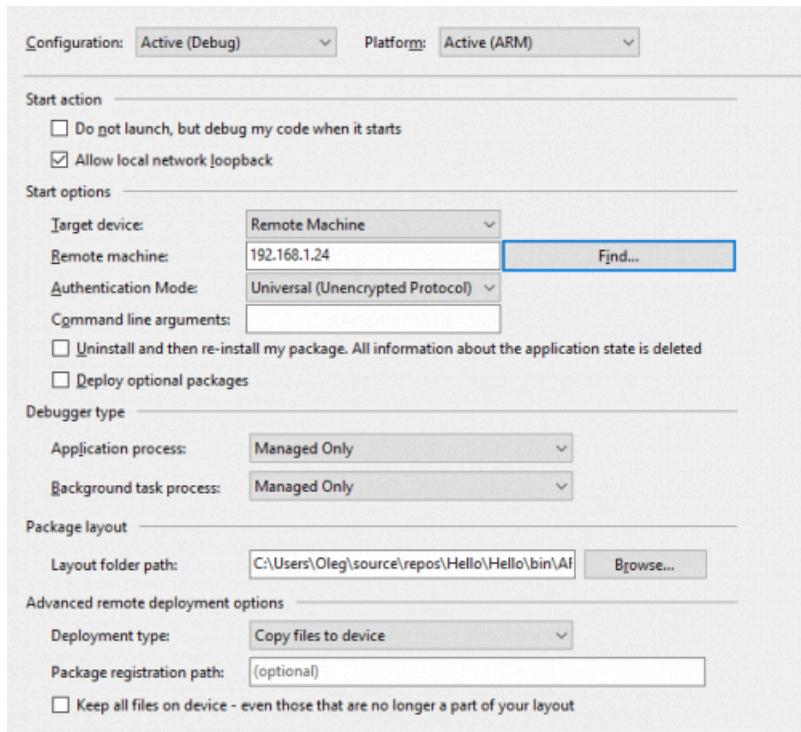


Figure 253: Set the Remote machine IP number

Step 8

Start debugging application and after deploy application package to the board SD card it will be displayed on the monitor.

Hello World!

Figure 254: Hello Application on the Raspberry Pi display

C# Variables and Data types

The C#¹⁴⁵⁾ variables are categorized into the following types:

1. Value types

-
2. Reference types
 3. Pointer types

Value Type Variables

Value type variables can assign a value directly. The class `System.ValueType` defines them.

The value types directly contain data. Value types may be: int, char and float, storing numbers, strings or floating point values. When an int type is declared, the system allocates memory to store its value.

The available value types list in C# is presented following:

Table 15: C# 2010 Value definitions.

| Type | Represents | Range | Default Value |
|---------|--|---|---------------|
| bool | Boolean value | True or False | False |
| byte | 8-bit unsigned integer | 0 to 255 | 0 |
| char | 16-bit Unicode character | U +0000 to U +ffff | '\0' |
| decimal | 128-bit precise decimal values with 28-29 significant digits | (-7.9 x 10E28 to 7.9 x 10E28) / 10E0 to 28 | 0.0M |
| double | 64-bit double-precision floating point type | (+/-)5.0 x 10E-324 to (+/-)1.7 x 10E308 | 0.0D |
| float | 32-bit single-precision floating point type | -3.4 x 10E38 to + 3.4 x 10E38 | 0.0F |
| int | 32-bit signed integer type | -2,147,483,648 to 2,147,483,647 | 0 |
| long | 64-bit signed integer type | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0L |
| sbyte | 8-bit signed integer type | -128 to 127 | 0 |
| short | 16-bit signed integer type | -32,768 to 32,767 | 0 |
| uint | 32-bit unsigned integer type | 0 to 4,294,967,295 | 0 |
| ulong | 64-bit unsigned integer type | 0 to 18,446,744,073,709,551,615 | 0 |
| ushort | 16-bit unsigned integer type | 0 to 65,535 | 0 |

Reference Type

The reference types don't contain the actual data stored in a variable. They contain a reference to the variables.

Using multiple variables, the reference types can refer to a memory location. If the variable changes the data in the memory location, the other variable automatically reflects this change in value. Built-in reference example types are: **object**, **dynamic**, and **string**.

4. IoT Hardware overview

==Object Type= The **Object Type** is an alias for the System.Object class. It is the ultimate base class for all data types in C# Common Type System (CTS). The object types can be assigned with values of any other types, value types, reference types, predefined or user-defined types. Before assigning values the type conversion is needed.

When a value type is converted to an object type, it is called **boxing** and when an object type is converted to a value type, it is called **unboxing**.

```
object obj;  
obj = 100; // this is boxing
```

Dynamic Type

Data type variable can store any value. But this type checking takes place at run-time.

Syntax for declaring a dynamic type is:

```
dynamic <variable_name> = value;
```

For example,

```
dynamic d = 20;
```

Dynamic types are similar to object types. That type checking for object type variables takes place at compile time. For the dynamic type variables checking takes place at run time.

String Type

The **String Type** allows to assign any string values to a variable. The string type is an alias for the System.String class and is derived from object type. The string type value can be assigned using string literals in two forms: quoted and @quoted.

For example,

```
string str = "Tutorials Point";
```

A @quoted string literal looks as follows –

```
@"Tutorials Point";
```

The user-defined reference types are: class, interface, or delegate.

Pointer Type

Pointer type variables store the memory address which is another type. Pointers in C# are similar to pointers in C or C++.

Syntax for declaring a pointer type is:

```
type* identifier;
```

For example,

```
char* cptr;
int* iptr;
```

C# Variables

Each variable in C# has a specific type, which determines the size and layout of the variable's memory.

The basic value types in C# can be categorized as following:

Table 16: C# 2010 Variables.

| Type | Example |
|----------------------|--|
| Integral types | sbyte, byte, short, ushort, int, uint, long, ulong, and char |
| Floating point types | float and double |
| Decimal types | decimal |
| Boolean types | true or false values, as assigned |
| Nullable types | Nullable data types |

Variable definitions

Variable syntax definition in C# is:

```
<data_type> <variable_list>;
```

Data_type must be a valid C# data type like char, int, float, double, or any user-defined data type. Variable_list may consist of one or more identifier names separated by commas.

Examples of valid variable definitions are shown below:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Variable can be initialized immediately during definition time –

```
int i = 100;
```

Variables Initialization

Variables are initialized with an equal sign followed by a constant expression. The general initialization form looks:

```
variable_name = value;
```

4. IoT Hardware overview

Variables can be initialized during their declaration. The initializer consists of an equal sign followed by a constant expression as:

```
<data_type> <variable_name> = value;
```

Some examples are:

```
int d = 3, f = 5;      /* initializing d and f. */
byte z = 22;           /* initializes z. */
double pi = 3.14159;   /* declares an approximation of pi. */
char x = 'x';          /* the variable x has the value 'x'. */
```

It is important to initialize variables properly, otherwise sometimes it may produce unexpected result.

The following example uses various types of variables:

```
using System;

namespace VariableDefinition {
    class Program {
        static void Main(string[] args) {
            short a;
            int b;
            double c;
            /* actual initialization */
            a = 10;
            b = 20;
            c = a + b;
            Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
            Console.ReadLine();
        }
    }
}
```

Output:

```
a = 10, b = 20, c = 30
```

C# Looping

C#¹⁴⁶⁾ provides following types of loop to handle looping requirements.

Table 17: C# 2010 Loops definitions.

| Sr.No. | Loop Type & Description |
|--------|---|
| 1 | while loop It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body. |
| 2 | for loop It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |

| Sr.No. | Loop Type & Description |
|--------|---|
| 3 | do..while loop It is similar to a while statement, except that it tests the condition at the end of the loop body. |
| 4 | nested loop You can use one or more loop inside any another while, for or do..while loop. |

The while loop

A while loop statement in C# repeatedly executes a target statement as long as a given condition is true.

```
while(condition) {
    statement(s);
}
```

Example:

```
using System;

namespace Loops {
    class Program {
        static void Main(string[] args) {
            /* local variable definition */
            int a = 10;

            /* while loop execution */
            while (a < 20) {
                Console.WriteLine("value of a: {0}", a);
                a++;
            }
            Console.ReadLine();
        }
    }
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

The for loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. The syntax of a for loop in C# is:

4. IoT Hardware overview

```
for ( init; condition; increment ) {  
    statement(s);  
}
```

Here is the flow of control in a for loop:

1. The init step is executed first, and only once. This step allows you to declare and initialise any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
2. Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not run, and flow of control jumps to the next statement just after the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
4. The condition is now evaluated again. If it is true, the loop executes, and the process repeats itself (body of the loop, then increment step, and then again testing for a condition). After the condition becomes false, the for loop terminates.

Example:

```
using System;  
namespace Loops {  
    class Program {  
        static void Main(string[] args) {  
  
            /* for loop execution */  
            for (int a = 10; a < 20; a = a + 1) {  
                Console.WriteLine("value of a: {0}", a);  
            }  
            Console.ReadLine();  
        }  
    }  
}
```

Output:

```
alue of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19
```

The C# do..while loop

The syntax of a do...while loop in C# is:

```
do {
    statement(s);
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false. Example:

```
using System;

namespace Loops {
    class Program {
        static void Main(string[] args) {
            /* local variable definition */
            int a = 10;

            /* do loop execution */
            do {
                Console.WriteLine("value of a: {0}", a);
                a = a + 1;
            }
            while (a < 20);
            Console.ReadLine();
        }
    }
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

C# nested for loop

C# allows using one loop inside another loop (loop nesting). The following section shows a few examples to illustrate the concept. The syntax for a **nested for loop** statement in C# is as follows:

```
for ( init; condition; increment ) {
    for ( init; condition; increment ) {
        statement(s);
    }
    statement(s);
```

4. IoT Hardware overview

```
}
```

The syntax for a **nested while loop** statement in C# is as follows:

```
while(condition) {  
    while(condition) {  
        statement(s);  
    }  
    statement(s);  
}
```

The syntax for a **nested do...while loop** statement in C# is as follows:

```
do {  
    statement(s);  
    do {  
        statement(s);  
    }  
    while( condition );  
}  
while( condition );
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa. Example:

```
using System;  
  
namespace Loops {  
    class Program {  
        static void Main(string[] args) {  
            /* local variable definition */  
            int i, j;  
  
            for (i = 2; i < 100; i++) {  
                for (j = 2; j <= (i / j); j++)  
                    if ((i % j) == 0) break; // if factor found, not prime  
                    if (j > (i / j)) Console.WriteLine("{0} is prime", i);  
            }  
            Console.ReadLine();  
        }  
    }  
}
```

Output:

```
2 is prime  
3 is prime  
5 is prime  
7 is prime  
11 is prime  
13 is prime  
17 is prime  
19 is prime  
23 is prime
```

```

29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime

```

Infinite loop

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop is required, you can make an endless loop by leaving the conditional expression empty.

Example:

```

using System;

namespace Loops {
    class Program {
        static void Main(string[] args) {
            for (; ; ) {
                Console.WriteLine("Hey! I am Trapped");
            }
        }
    }
}

```

When the conditional expression is absent, it is assumed to be true.

C# Program Control Structures

C#¹⁴⁷⁾ Specifying one or more conditions to be evaluated or tested by the program require decision making structures. If the condition is determined to be true or false the proper statements must be performed.

C# provides following types of decision making statements:

Table 18: C# 2010 Loops definitions.

| Sr.No. | Loop Type & Description |
|--------|---|
| 1 | An if statement consists of a boolean expression followed by one or more statements. |

4. IoT Hardware overview

| Sr.No. | Loop Type & Description |
|--------|--|
| 2 | if...else statement An if statement can be followed by an optional else statement, which executes when the boolean expression is false. |
| 3 | nested if statements You can use one if or else if statement inside another if or else if statement(s). |
| 4 | switch statement A switch statement allows a variable to be tested for equality against a list of values. |
| 5 | nested switch statements You can use one switch statement inside another switch statement(s). |
| 6 | The ? Operator. |

if Statement

An if statement consists of a boolean expression followed by one or more statements. The syntax of an if statement in C# is:

```
if(boolean_expression) {  
    /* statement(s) will execute if the boolean expression is true */  
}
```

If the boolean expression evaluates to true, then the block of code inside the if statement is executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) is executed.

Example:

```
using System;  
  
namespace DecisionMaking {  
    class Program {  
        static void Main(string[] args) {  
            /* local variable definition */  
            int a = 10;  
  
            /* check the boolean condition using if statement */  
            if (a < 20) {  
                /* if condition is true then print the following */  
                Console.WriteLine("a is less than 20");  
            }  
            Console.WriteLine("value of a is : {0}", a);  
            Console.ReadLine();  
        }  
    }  
}
```

Output:

```
a is less than 20;  
value of a is : 10
```

if..else Statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false. The syntax of an if...else statement in C# is:

```
if(boolean_expression) {
    /* statement(s) will execute if the boolean expression is true */
} else {
    /* statement(s) will execute if the boolean expression is false */
}
```

If the boolean expression evaluates to true, then the if block of code is executed, otherwise else block of code is executed.

Example:

```
using System;

namespace DecisionMaking {
    class Program {
        static void Main(string[] args) {
            /* local variable definition */
            int a = 100;

            /* check the boolean condition */
            if (a < 20) {
                /* if condition is true then print the following */
                Console.WriteLine("a is less than 20");
            } else {
                /* if condition is false then print the following */
                Console.WriteLine("a is not less than 20");
            }
            Console.WriteLine("value of a is : {0}", a);
            Console.ReadLine();
        }
    }
}
```

Output:

```
a is not less than 20;
value of a is : 100
```

Nested if Statement

It is always legal in C# to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s). The syntax for a nested if statement is as follows –

```
if( boolean_expression 1) {
    /* Executes when the boolean expression 1 is true */
    if(boolean_expression 2) {
        /* Executes when the boolean expression 2 is true */
    }
}
```

4. IoT Hardware overview

```
}
```

Example:

```
using System;

namespace DecisionMaking {
    class Program {
        static void Main(string[] args) {
            /* local variable definition */
            int a = 100;
            int b = 200;

            /* check the boolean condition */
            if (a == 100) {

                /* if condition is true then check the following */
                if (b == 200) {
                    /* if condition is true then print the following */
                    Console.WriteLine("Value of a is 100 and b is 200");
                }
            }
            Console.WriteLine("Exact value of a is : {0}", a);
            Console.WriteLine("Exact value of b is : {0}", b);
            Console.ReadLine();
        }
    }
}
```

Output:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

switch Statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

The syntax for a switch statement in C# is as follows:

```
switch(expression) {
    case constant-expression :
        statement(s);
        break; /* optional */
    case constant-expression :
        statement(s);
        break; /* optional */

    /* you can have any number of case statements */
    default : /* Optional */
        statement(s);
}
```

The following rules apply to a switch statement:

1. The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
2. You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
3. The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
4. When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
5. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
6. Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
7. A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example:

```
using System;

namespace DecisionMaking {
    class Program {
        static void Main(string[] args) {
            /* local variable definition */
            char grade = 'B';

            switch (grade) {
                case 'A':
                    Console.WriteLine("Excellent!");
                    break;
                case 'B':
                case 'C':
                    Console.WriteLine("Well done");
                    break;
                case 'D':
                    Console.WriteLine("You passed");
                    break;
                case 'F':
                    Console.WriteLine("Better try again");
                    break;
                default:
                    Console.WriteLine("Invalid grade");
                    break;
            }
            Console.WriteLine("Your grade is {0}", grade);
            Console.ReadLine();
        }
    }
}
```

4. IoT Hardware overview

Output:

```
Well done  
Your grade is B
```

nested switch Statement

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

The syntax for a nested switch statement is as follows:

```
switch(ch1) {  
    case 'A':  
        Console.WriteLine("This A is part of outer switch" );  
  
        switch(ch2) {  
            case 'A':  
                Console.WriteLine("This A is part of inner switch" );  
                break;  
            case 'B': /* inner B case code */  
        }  
        break;  
    case 'B': /* outer B case code */  
}  
  
Example:  
<code C>  
using System;  
  
namespace DecisionMaking {  
    class Program {  
        static void Main(string[] args) {  
            int a = 100;  
            int b = 200;  
  
            switch (a) {  
                case 100:  
                    Console.WriteLine("This is part of outer switch ");  
  
                    switch (b) {  
                        case 200:  
                            Console.WriteLine("This is part of inner switch ");  
                            break;  
                    }  
                    break;  
            }  
            Console.WriteLine("Exact value of a is : {0}", a);  
            Console.WriteLine("Exact value of b is : {0}", b);  
            Console.ReadLine();  
        }  
    }  
}
```

Output:

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

C# Classes

Defining a Class

A C#¹⁴⁸⁾ class definition starts with the keyword `class` followed by the class name, and the class body enclosed by a pair of curly braces. Following is the general form of a class definition:

```
<access specifier> class  class_name {
    // member variables
    <access specifier> <data type> variable1;
    <access specifier> <data type> variable2;
    ...
    <access specifier> <data type> variableN;
    // member methods
    <access specifier> <return type> method1(parameter_list) {
        // method body
    }
    <access specifier> <return type> method2(parameter_list) {
        // method body
    }
    ...
    <access specifier> <return type> methodN(parameter_list) {
        // method body
    }
}
```

Note:

1. Access specifiers specify the access rules for the members as well as the class itself. If not mentioned, then the default access specifier for a class type is internal. Default access for the members is private.
2. Data type specifies the type of variable, and return type specifies the data type of the data the method returns if any.
3. To access the class members, you use the dot (.) operator.
4. The dot operator links the name of an object with the name of a member

Example:

```
using System;

namespace BoxApplication {

    class Box {
        public double length;    // Length of a box
        public double breadth;   // Breadth of a box
        public double height;    // Height of a box
    }
}
```

4. IoT Hardware overview

```
class Boxtester {
    static void Main(string[] args) {
        Box Box1 = new Box(); // Declare Box1 of type Box
        Box Box2 = new Box(); // Declare Box2 of type Box
        double volume = 0.0; // Store the volume of a box here

        // box 1 specification
        Box1.height = 5.0;
        Box1.length = 6.0;
        Box1.breadth = 7.0;

        // box 2 specification
        Box2.height = 10.0;
        Box2.length = 12.0;
        Box2.breadth = 13.0;

        // volume of box 1
        volume = Box1.height * Box1.length * Box1.breadth;
        Console.WriteLine("Volume of Box1 : {0}", volume);

        // volume of box 2
        volume = Box2.height * Box2.length * Box2.breadth;
        Console.WriteLine("Volume of Box2 : {0}", volume);
        Console.ReadKey();
    }
}
```

Output:

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Member Functions and Encapsulation

A **member function** of a class is a function that has its definition or its prototype within the class definition similar to any other variable. It operates on an object of the class of which it is a member, and has access to all the members of a class for that object.

Member variables are the attributes of an object (from the design perspective), and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

Sample:

```
using System;

namespace BoxApplication {
    class Box {
        private double length; // Length of a box
        private double breadth; // Breadth of a box
        private double height; // Height of a box

        public void setLength( double len ) {
            length = len;
```

```

    }
    public void setBreadth( double bre ) {
        breadth = bre;
    }
    public void setHeight( double hei ) {
        height = hei;
    }
    public double getVolume() {
        return length * breadth * height;
    }
}
class Boxtester {
    static void Main(string[] args) {
        Box Box1 = new Box();      // Declare Box1 of type Box
        Box Box2 = new Box();
        double volume;

        // Declare Box2 of type Box
        // box 1 specification
        Box1.setLength(6.0);
        Box1.setBreadth(7.0);
        Box1.setHeight(5.0);

        // box 2 specification
        Box2.setLength(12.0);
        Box2.setBreadth(13.0);
        Box2.setHeight(10.0);

        // volume of box 1
        volume = Box1.getVolume();
        Console.WriteLine("Volume of Box1 : {0}" ,volume);

        // volume of box 2
        volume = Box2.getVolume();
        Console.WriteLine("Volume of Box2 : {0}" , volume);

        Console.ReadKey();
    }
}
}

```

Output:

```

Volume of Box1 : 210
Volume of Box2 : 1560

```

C# Constructors

A **class constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor has the same name as that of class, and it does not have any return type.

Example:

```
using System;
```

4. IoT Hardware overview

```
namespace LineApplication {
    class Line {
        private double length; // Length of a line

        public Line() {
            Console.WriteLine("Object is being created");
        }
        public void setLength( double len ) {
            length = len;
        }
        public double getLength() {
            return length;
        }

        static void Main(string[] args) {
            Line line = new Line();

            // set line length
            line.setLength(6.0);
            Console.WriteLine("Length of line : {0}", line.getLength());
            Console.ReadKey();
        }
    }
}
```

Output:

```
Object is being created
Length of line : 6
```

A default **constructor** does not have any parameter, but you can make one if you need to pass some setup values on the initialisation - such constructors are called parameterised constructors. This technique helps you to assign an initial value to an object at the time of its creation.

Example:

```
using System;

namespace LineApplication {
    class Line {
        private double length; // Length of a line

        public Line(double len) { //Parameterized constructor
            Console.WriteLine("Object is being created, length = {0}", len);
            length = len;
        }
        public void setLength( double len ) {
            length = len;
        }
        public double getLength() {
            return length;
        }
        static void Main(string[] args) {
```

```
Line line = new Line(10.0);
Console.WriteLine("Length of line : {0}", line.getLength());

// set line length
line.setLength(6.0);
Console.WriteLine("Length of line : {0}", line.getLength());
Console.ReadKey();
}

}

}
```

Output:

```
Object is being created, length = 10
Length of line : 10
Length of line : 6
```

C# Constructors

A **constructor** is a special member function of a class that is executed whenever an object of its class goes out of scope. A constructor has the same name as that of the class with a prefixed tilde (~), and it can neither return a value nor can it take any parameters. C# (.NET environment) has built-in memory management system that tracks unused objects and release memory automatically but in constrained memory systems like RPI, it is sometimes essential to manually notify this mechanism about the possibility to release memory once the object is no longer used. Here constructor helps much. Moreover, constructor brings a possibility to handle hardware-related issues, i.e. close connection, send a farewell message to the external device, etc. Constructors cannot be inherited or overloaded.

Example:

```
using System;

namespace LineApplication {
    class Line {
        private double length; // Length of a line

        public Line() { // constructor
            Console.WriteLine("Object is being created");
        }
        ~Line() { //destructor
            Console.WriteLine("Object is being deleted");
        }
        public void setLength( double len ) {
            length = len;
        }
        public double getLength() {
            return length;
        }
        static void Main(string[] args) {
            Line line = new Line();

            // set line length
            line.setLength(6.0);
```

4. IoT Hardware overview

```
        Console.WriteLine("Length of line : {0}", line.getLength());
    }
}
```

Output:

```
Object is being created
Length of line : 6
Object is being deleted
```

Static Members of a C# Class

We can define class members as static using the **static** keyword. When we declare a member of a class as static, it means no matter how many objects of the class are created, there is only one copy of the static member.

The keyword **static** implies that only one instance of the member exists for a class. Static variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it. Static variables can be initialised outside the member function or class definition. You can also initialise static variables inside the class definition.

Example:

```
using System;

namespace StaticVarApplication {
    class StaticVar {
        public static int num;

        public void count() {
            num++;
        }
        public int getNum() {
            return num;
        }
    }
    class StaticTester {
        static void Main(string[] args) {
            StaticVar s1 = new StaticVar();
            StaticVar s2 = new StaticVar();

            s1.count();
            s1.count();
            s1.count();

            s2.count();
            s2.count();
            s2.count();

            Console.WriteLine("Variable num for s1: {0}", s1.getNum());
            Console.WriteLine("Variable num for s2: {0}", s2.getNum());
            Console.ReadKey();
        }
    }
}
```

```
}
```

Output:

```
Variable num for s1: 6
Variable num for s2: 6
```

You can also declare a member function as static. Such functions can access only static variables. The static functions exist even before the object is created. Example:

```
using System;

namespace StaticVarApplication {
    class StaticVar {
        public static int num;

        public void count() {
            num++;
        }
        public static int getNum() {
            return num;
        }
    }
    class StaticTester {
        static void Main(string[] args) {
            StaticVar s = new StaticVar();

            s.count();
            s.count();
            s.count();

            Console.WriteLine("Variable num: {0}", StaticVar.getNum());
            Console.ReadKey();
        }
    }
}
```

Output:

```
Variable num: 3
```

C# Events

Events occurs when user makes actions like: key press, clicks, mouse movements, etc., or some other occurrence such as system-generated notifications. Applications must respond to events if they occur, i.e. handle interrupts. Events are used during inter-process communication.

Using Delegates with Events

The events are declared and raised in a class. They are associated with the event handlers using delegates within the same class or some other class. To publish the event the class containing it must be defined. It is called the **publisher** class. Some other

4. IoT Hardware overview

class that accepts this event is called the **subscriber** class. Events use the **publisher-subscriber** model.

The object containing definition of the event and the delegate is named **publisher**. The event-delegate association is also defined in this object. A publisher class object invokes the event, and it is notified to other objects.

A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Declaring Events

To declare an event inside a class, first a delegate type for the event must be declared. For example,

```
public delegate string MyDel(string str);
```

Next, the event itself is declared, using the event keyword:

```
event MyDel MyEvent;
```

The preceding code defines a delegate named *BoilerLogHandler* and an event named *BoilerEventLog*, which invokes the delegate when it is raised.

Example:

```
using System;

namespace SampleApp {
    public delegate string MyDel(string str);

    class EventProgram {
        event MyDel MyEvent;

        public EventProgram() {
            this.MyEvent += new MyDel(this.WelcomeUser);
        }
        public string WelcomeUser(string username) {
            return "Welcome " + username;
        }
        static void Main(string[] args) {
            EventProgram obj1 = new EventProgram();
            string result = obj1.MyEvent("Tutorials Point");
            Console.WriteLine(result);
        }
    }
}
```

Output:

```
Welcome Tutorials Point
```

5. Introduction to the IoT Communication and Networking

Mind, there is "I" in IoT !

In no doubt, IoT is network oriented - even the name IoT naturally relates to the Internet network. Communication is the essential part of IoT idea. Every IoT device must communicate somehow, even most simple, passive RFID tag – it responds with some data to the excitation. Communication is always performed with some rules known for both communicating parties. Like people have their different languages to use, devices have protocols. Communication protocol describes how to address the information to the remote device, how to encode the data, how to check the correctness of the incoming message. The physical layer of protocol description also tells how to transmit every bit of data, what is the frequency of radio waves, how fast we can send the data, what is the maximum range of the transmission.

Communication in IoT devices can be wired or wireless.

IoT networking is much different than typical, multilayered, stack-oriented TCP/IP or similar communication model; we know well while using our PC, MAC, server or Smartphone on a daily basis.

Indeed constrained IoT devices are usually unable to operate regular - full time on, ISO/OSI layered stack, because of constrained resources. In details it primary means, IoT devices are limited by the processor power, RAM and storage sizes and mainly because of limited power resources. IoT device is expected to be energy efficient, thus low powered, that in most cases excludes typical wireless connection standards like i.e. WiFi. On the other hand, IoT devices are expected to communicate over long distances - some couple or a dozen of kilometres - where wired infrastructure like Ethernet cables and related infrastructure is non-existent and most of the wired technologies, copper-based is out of range.

Also, IoT devices daily life-cycle is much different than, i.e. or PC life-cycle. We as humans used to switch on the notebook, work extensively on the web, then put it to the low power or off, making the machine to sleep, hibernate or just shutting it down. And we wake it up when needed. It barely makes network operation during sleep. IoT devices are expected to be sleeping providing low power mode whenever possible, and on the other hand, they're supposed to be fully operable, when only needed. Most performed IoT tasks related to the sensing have cyclical nature, i.e. measuring gases as a sensor-network node whereas period can be something like between seconds and months or even longer. Anyway, they're usually expected to trigger themselves to be awake from sleep, perform some operation and connect to the network.

Because of the existence of different IoT devices including those very constrained from 8-bit processors with some kB of the RAM to 32-bit multicore machines well-replacing PCs, IoT networking is very competitive on protocols, approaches and solutions. There are indeed some networking standards introduced by standardisation organisation like IEEE, yet they are competed by large manufacturers forcing their complex solutions including dedicated hardware, software and protocols. The third force driving this market are open solutions and enthusiasts, usually working with cheap equipment, providing de-facto standards for many hobbyists and also industry.

5. Introduction to the IoT Communication and Networking

Following chapters explain some most popular concepts about how to organise network fulfilling the above constraints on communication between IoT devices (Machine-2-Machine) and how to let them communicate with the Internet: including hardware, software and human-users.

We focus on the de-facto standards existing in the web, usually as open-source libraries and somewhat low-cost devices.

An interesting survey made by RS components¹⁴⁹⁾ shows 11 wireless protocols used in IoT. Some of them you can use free, without having any license to purchase, while some others are proprietary, some of them need the subscription plan.

Networking overview

Communication models

Media layers - Wired networking

Media layers - Wireless protocols

Host layer protocols

5.1. Networking overview

IoT devices are not separated from the global networking environment that nowadays is highly integrated, connecting various wired and wireless transmission standards into one network called the Internet. Indeed some networks are separated because of security and safety reasons and regulations, yet they usually share the same standards that global, Internet network.

XXI century brought wide acceptance of wireless connections. They became very popular even in so-called pico-networks, like your PAN (Personal Area Network)¹⁵⁰⁾, implemented using i.e. Bluetooth Connection, where your smartphone in the left pocket of your jeans is hosting a server and there are many wireless devices connected to it: your wireless headset/audio device that you're listening to the music ad the moment, wireless HID controller (i.e. MYO device)¹⁵¹⁾, your Smartwatch, AR glasses, etc. All those devices constitute a personal, wireless network cell, usually also routed via NAT (Network Adress Translation)¹⁵²⁾ to the Internet, through some other wireless connection like WiFi or mobile data (3G/4G/LTE). Naturally, many IoT devices share standard wireless protocols, models and ideas but some of them are not powerful enough to integrate with the Internet. On the other hand, wireless connections are somehow natural to the IoT devices where they are expected to be operating in remote destinations, usually without any wired infrastructure. Because of it, some Internet standards were adapted to their constraints or such networks of constrained devices are separated and gatewayed through some more powerful devices, where protocol translation occurs. Those models are discussed below, in following chapters.

The similar way to the regular Internet networking, IoT networking is implemented using (usually simplified) layered stack, similar to the regular ISO/OSI 7 layer networking stack well known to all IT students¹⁵³⁾, where the lowest 3 levels constitute so-called Media layers of the stack (recommendation X.200).

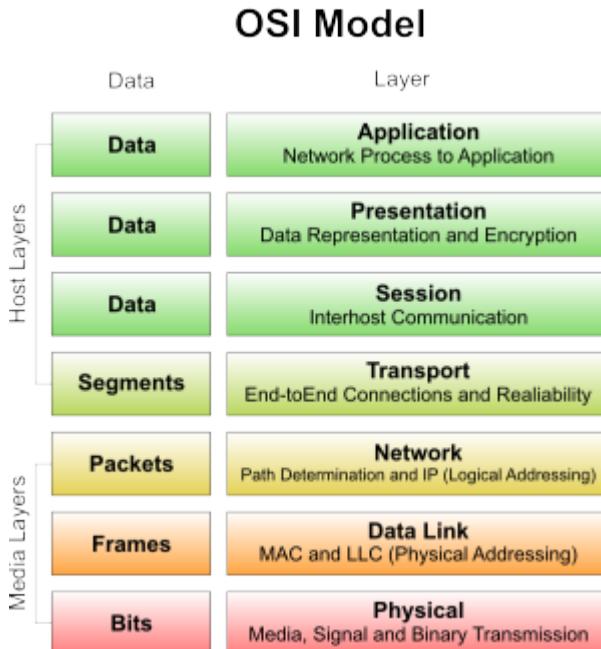


Figure 255: ISO/OSI multi-layer Internet networking protocol stack

Level 1 is a Physical Layer (PHY). On the top of it, there is level 2 is Data Link Layer with Media Access Control and Logical Link Control (MAC/LLC). Level 3 is the Networking layer (NET) where packets are formed and routed as presented in figure 255. ISO-OSI model was designed and implemented for dedicated network controller chips and powerful processors thus not all of the IoT devices are capable to fully implement this model, particularly because of constrained RAM and storage memory sizes. Also, this model requires an instant connection to the remote node (PHY dependent), so it has a strong impact on the battery drain in case of the low power devices.

A quick overview of popular communication technologies for the Internet is presented on the figure 256. Note wide distance range between nodes of the Wireless devices regarding protocol used (mostly because of the PHY nature) where it varies from some meters in case of piconets up to some 180-2000km when considering LEO (Low Earth Orbit) satellites like, i.e. communicating through Iridium network and even up to about 35 786 km in case of the use of the geostationary satellites¹⁵⁴⁾.

Another factor is the communication bandwidth. Fortunately, IoT devices usually do not require high bandwidth - some couple of kbps is enough; thus almost all protocols apply here.

5. Introduction to the IoT Communication and Networking

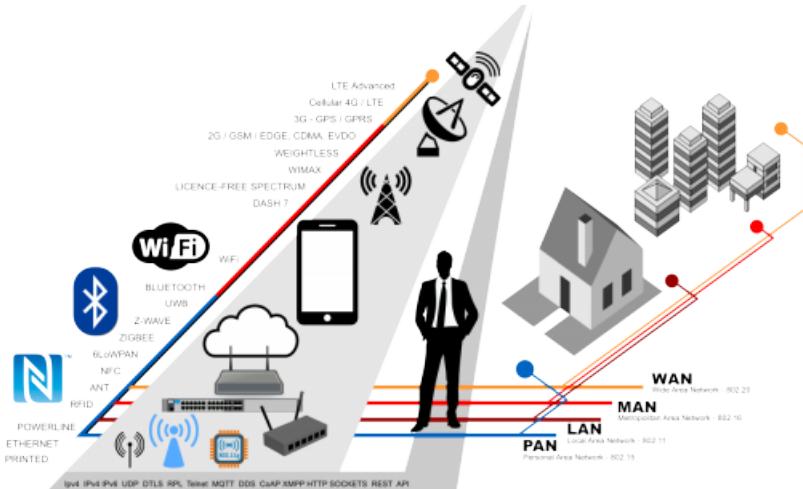


Figure 256: Popular wireless networking standards.

In many cases, IoT remote, distant nodes do not need constant communication, i.e. weather sensing would better communicate on datagram communication model (UDP rather than TCP)¹⁵⁵⁾. In such case, IoT devices utilize differently, simplified IoT stack as presented on image 257.

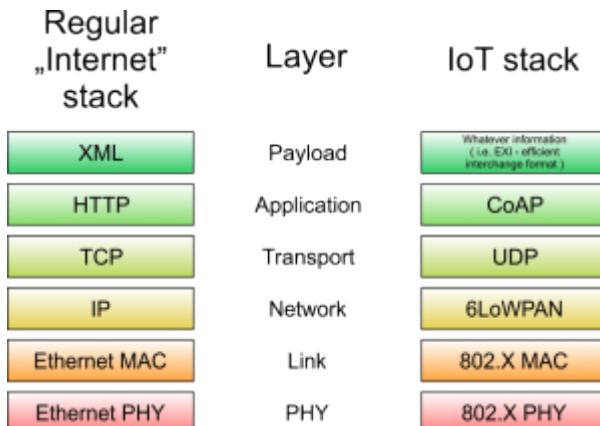


Figure 257: Simplified, IoT-oriented implementation of the protocol stack (using UDP)

5.2. Communication models

IoT Devices can be classified regarding their ability to implement full protocol stacks of the typical, Internet protocols like IPv4, IPv6, HTTP, etc.:

- Devices unable to implement full, protocol stack without external support, like, i.e. Arduino Uno (R3) with 32kb of the flash memory, 16MHz single core processor and 2kB of static ram, battery powered, consuming some couple of mW while operating.

- Devices able to implement full, protocol stack yet still limited by their resources, i.e. ESP8266 and ESP32 chips, battery powered, consuming some dozen or hundred of mW while operating.
- Devices that do can offer various, advanced network services, capable of implementing protocol stack with ease yet not servers, routers or gateways, i.e. Raspberry Pi and its clones. Usually, DC powered with power consumption far above 1-2W, usually up to 10-15W.
- Dedicated solutions for gateways, routers, usually with embedded, hardware-based implementations of the switching logic, utilising some 10-50W of power.
- Universal IoT computers (i.e. Intel IoT), using PC-grade processors (x86, but sometime ARM), using some about up to 100W of power.

Some IoT networks are also constrained by the number of IP addresses available regarding the number of IoT devices ones need to connect, so their topology is a priori prepared as NAT (Network Adress Translation) solution¹⁵⁶⁾ thus it requires automatically use of routers.

IoT devices are usually expected to deliver their data to some cloud for storage and processing while the cloud can send back commands to the actuators/outputs.

Finally, there are security concerns, which make the IoT devices to be put in some separate sub-network and guarded by the firewall.

All of it brings the three, main communication models, explained below.

5.2.1. Device to device and Industry 4.0 revolution

Device to device communication model, sometimes referenced as M2M (Machine to Machine communication model) used to be implemented between the homogeneous class of the IoT devices. Nowadays there is a need to enable heterogeneous systems to collaborate and talk one to another. In a device to a device model, communication is usually held simple, sometimes with niche, proprietary protocols, i.e. ANT/ANT+¹⁵⁷⁾, sometimes do employs heavy protocols like XML, so there is a need to provide common communication ontologies and semantics. Devices participating in such networks usually act as multimode, constituting self-organising networks, capable of exchanging the data through routing and forwarding as it appears in 6LowPAN networks where nodes may not only act as data producer/consumer but is also expected to act as message forwarder/router.

Device to device model is highly utilized in the Industrial Automation Control systems and recently very popular in developing Industry 4.0 (I4.0) solutions, where manufacturing devices, i.e. robots and other Cyber-Physical systems (CPS) communicate to set operation sequence for optimal manufacturing process (so-called Industry 4.0) thus providing elastic working zones along with manufacturing flexibility and self-adaptation of the processes. It happens because of the presence of various IoT devices (here sometimes referenced as Industrial IoT) and advanced data processing including Big and Small data. Such a device to device networks very frequently mimics popular P2P (peer to peer) networks, where one device can virtually contact any other to ask for information or deliver one. Comparing to the classical, tree-like topology, a device to device communication constitutes a graph of relations rather than a hierarchized tree. The figure 258 presents comparison between pre-I4.0 (Industry 3.0) and I4.0 data flow. Along with physical (real) devices participating in the manufacturing process there

5. Introduction to the IoT Communication and Networking

usually goes their virtual representation (Virtual Twin) to enable cognitive manufacturing based on data science. The detailed description of the data analysis and its use in I4.0 is out of the scope of this book, however.

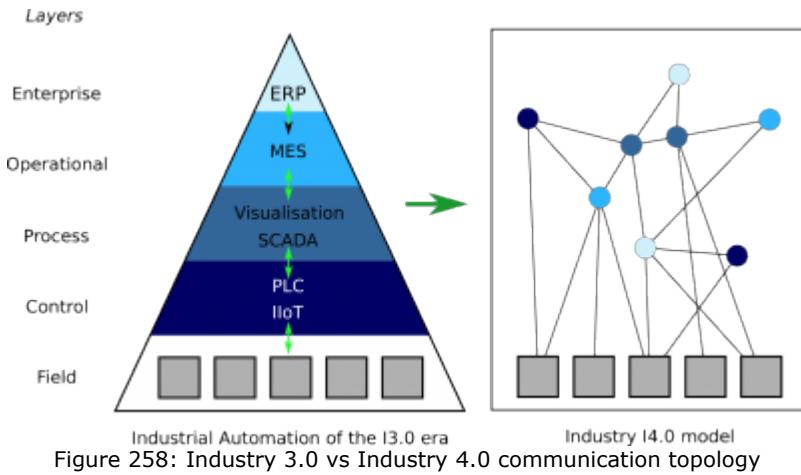


Figure 258: Industry 3.0 vs Industry 4.0 communication topology

The device to device communication assumes, participating devices are smart enough to talk one another, without the need of the translation nor advanced data processing, even if their nature is different (i.e. your intelligent door can inform your smart, IoT kettle to start boiling water once they get informed about poor weather condition by the Internet weather monitoring service, when you're back home after long day of work). Devices constituting mesh or scatter network communicate virtually one another similar way people do. Figure 259 briefly presents the data flow idea

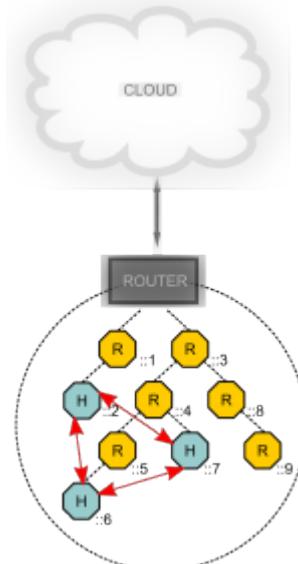


Figure 259: Device to device communication model

5.2.2. Device to gateway

Device to gateway communication occurs when there is a need to provide the translate information between different networks, i.e. some Zigbee¹⁵⁸⁾ network devices need to send data to the Internet to let the smart home be remotely monitored and managed. This model also appears when there is a need to transfer messages between IoT network implemented with constrained devices, so using some simplified protocol (i.e. LoRA, 6LowPAN) and the Internet network, using the full implementation of the protocols (i.e. IPv6). In this case, the gateway device (sometimes named here as Edge Router) needs to have knowledge about constrained devices constituting IoT network and it usually supplies some missing information instead of them, i.e. enriching message headers or addresses when passing packets from IoT constrained network to the Internet, but also translating Internet packets (i.e. by removing full address), when acting opposite, i.e. forwarding actuator requests to the IoT devices.

Gatewaying and protocol translation can also occur on the 6th and 7th level of the ISO/OSI model when implementation of high-level protocols overwhelms even more advanced IoT devices, i.e. simple MQTT textng can be converted to the XML, heavy messages or exposed as XHTML. Those solutions are mostly software-based, i.e. Node-RED¹⁵⁹⁾. Figure ref_device-to-gateway briefly presents data flow. Please note the protocol change: arrows of the different colours reflect it.

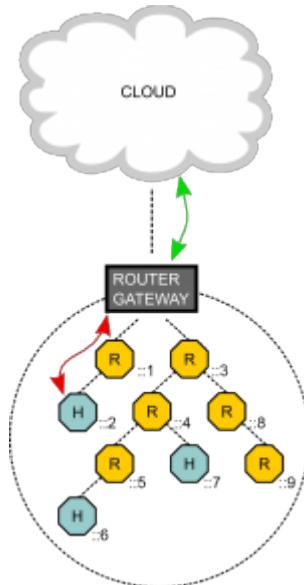


Figure 260: Device to gateway communication model

5.2.3. Device to cloud

As IoT devices are usually unable to constitute an efficient computation structure (as single IoT node or even their federation), most data is forwarded to the server, often a cloud-based solution, where it is stored and processed. This data processing in the cloud varies, depending on the type of information, their goal, etc. In any case, we usually face the problem of the visualisation, data analytics (statistics, data mining, knowledge discovery, big data processing). Those tasks are resource consuming, require huge processing capabilities thus utilising cloud solution is usually a good choice. Note, claiming "cloud" we consider not only public clouds like Amazon, Google or Microsoft but also dedicated solutions hidden somewhere in the separated, manufacturing networks. Eventually, there is a need to send back some actuation requests to the devices, from the cloud. Cloud services are usually PC based solutions, thus they extensively use rich protocols, providing their APIs via i.e. REST¹⁶⁰⁾, SoAP¹⁶¹⁾, HTTP GET/POST methods¹⁶²⁾, etc. It requires the IoT devices interfacing cloud to implement full communication stacks for the protocols needed. Some of the IoT devices can interface cloud services directly, but some of them are unable to do so due to the constraints, so it is necessary to use gateways as mentioned in the previous chapter.

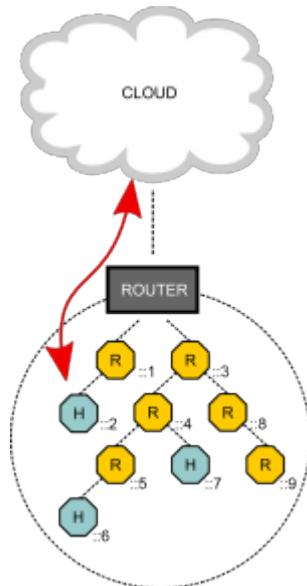


Figure 261: Device to cloud communication model

5.3. Media layers - Wired networking

While the IoT ecosystem is usually considered to be composed of wireless devices, it is still possible to connect IoT solutions using a wired connection. In this chapter, we do not present communication protocols that are short distant one designed to connect sensors to IoT device, like I2C, SPI, Serial, etc. Those are described in chapter Embedded Systems Communication Protocols.

When wireless-enabled SoCs where about to be delivered to the market (i.e. ESP8266), there were already available sorts of extension devices for popular Embedded systems, like i.e. Ethernet Shield for Arduino boards (figure 262):



5. Introduction to the IoT Communication and Networking



Figure 262: Ethernet shields for Arduino boards

Cooper based wired networks also bring an extra feature to the IoT designers - an ability to power the device via a wired connection, i.e. PoE (Power over Ethernet) - 802.3af, 802.3at, 802.3bt¹⁶³⁾. Long distance connections may be implemented using optic-based, fibre connections, but those require physical medium converters that are usually quite complex, pretty expensive and power consuming thus they apply only to the niche IoT solutions. Please note, mentioned optical connections do not cover so-called LiFi, as those are considered to be wireless¹⁶⁴⁾.

A non exhaustive list of some present and former wired networking solutions are presented in the table 19:

Table 19: A short review of the most popular wired networking standards

| Name | Communication medium | Max speed | Topology | Max range (single segment, passive) |
|--------------------|---|--|-------------------------------|---|
| Ethernet | Twisted pair: 10BaseT Coaxial: 10Base2/ 10Base5 Fibre: 10BaseF | 10 Mbps | Bus, Star, Mixed (Tree) | 10Base2: 0.5-200m(185m) 10Base5: 500m 10BaseT: 100m (150m) 10BaseF: 2km (multimode fibre) |
| Fast Ethernet | Twisted pair: 100BaseTx Fibre: 100BaseFx | 100 Mbps | Star | 100BaseTx: 100m (Cat 5) 100BaseFx: 2km |
| Gigabit Ethernet | Twisted pair: 1000BaseT Fibre: 1000BaseX (LX/ CX/SX) | 1000BaseT: 1 Gbps 1000BaseX: 4.268 Gbps | Star | 1000BaseT: 100m(Cat 5) 1000BaseLX: 5km |
| Local Talk (Apple) | Twisted pair | 0.23 Mbps | Bus, Star (PhoneNet) | 1000ft |

5.4. Media layers - Wireless protocols

| Name | Communication medium | Max speed | Topology | Max range (single segment, passive) |
|------------|----------------------|---|-----------------|-------------------------------------|
| Token ring | Twisted pair | 16 Mbps | Star wired ring | 22.5m/100m (cable dependent) |
| FDDI | Fibre | 100 Mbps (200 Mbps on two rings, but no redundancy) | Dual ring | 2km |

Nowadays, the most popular wired networks are 10/100/100BaseT - twisted pair with Cat 5, 5e and 6 cables. They require the IoT system to implement full TCP/IP stack to operate seamlessly with conventional Internet / Intranet / Extranet networks. Because it is usually out of the scope of standard Arduino Uno processor capabilities to implement full TCP stack, there are typically dedicated processors on the network interfaces that assist the central processor or even handle all networking tasks themselves.

5.4. Media layers - Wireless protocols

Wireless connections define core communication for IoT devices. Wast and growing amount of protocols, their variations and dynamic IoT networking market, all present non-solid situation where old "adult" Internet protocols coexist along with new ideas and IoT hardware and software platforms are more and more capable with every new generation, thus new ideas appear almost daily. Currently, there are many IoT networking protocols defined for various layers of the protocol implementation stack, some of them compatible while others are concurring. An image 263 presents some selected protocols existing for IoT. Please note, this covers only the most popular ones and presents non-exhaustive view. We discuss them more detailed below.

5. Introduction to the IoT Communication and Networking

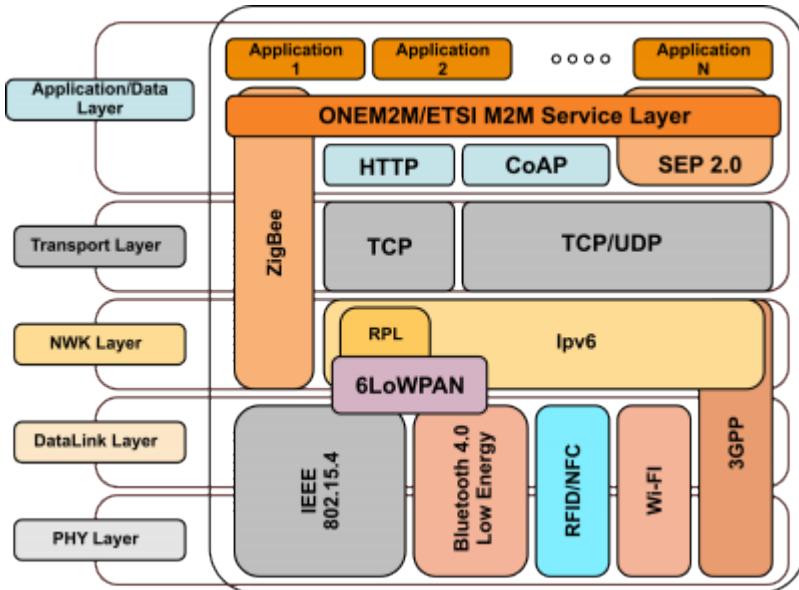


Figure 263: IoT Protocols

5.4.1. PHY+MAC+LLC layers

Below we present currently most popular, wireless protocols review for the lower ISO/OSI layers (1-2, some of them also implement layer 3 - Networking).

WiFi

WiFi is the set of standards for wireless communication using the 2.4GHz or 5GHz band, slightly different spectrum in different countries. The core specification of the 2.4GHz contains 14 channels with 20 MHz (currently 40 MHz) bandwidth. While there is no centralised physical layer controller, collisions frequently occur even more with a growing number of devices sharing the band. The collision is handled using CSMA-CA with a random binary exponential increase of repeating time.

With the high speed of transmission and range usually not exceeding 100m, it is widely used as the direct replacement of wired Ethernet in local area networks. It is a very good choice while the amount of data to be transferred is bigger, i.e. video streams or assembled IoT stream delivered by gateways. It is also possible to use it in direct connectivity for smart sensors, and other IoT elements, but the protocol itself is not designed to transmit small data packets. For many IoT applications, it is too much energy consuming, especially when it comes to the battery-powered devices. Moreover, WiFi itself offers only 1-to-1 or star-like models of connection, where the central point is a WiFi Access Point (1-to-many) and does not provide mechanisms for i.e. self-reorganizable, failure-tolerant mesh networks. WiFi becomes more and more popular choice for not-so-constrained IoT devices because they need to implement full TCP/IP stack and those devices that are also not so constrained by the power consumption.

Table 20: WiFi standards summary

| 802.11 standard | Transmission speed | Frequency |
|------------------------|---------------------------|------------------|
| 802.11b | 11Mbps | 2.4GHz |
| 802.11g | 54Mbps | 2.4GHz |
| 802.11n | 150Mbps | 2.4GHz |
| | 300Mbps | 5GHz |
| 802.11ac | 1Gbps | 5GHz |

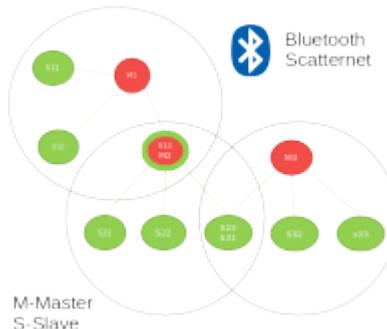
Bluetooth

Bluetooth is a prevalent method of connecting a variety of devices in short distance. Almost every computer and a smartphone have Bluetooth module built-in. Standard has been defined by Bluetooth SIG (Special Interest Group) founded in 1998. Bluetooth operates in the 2.4GHz band with 79 channels with automatic channel switching when interference occurs (hopping frequency). The single channel offers up to about 1Mbps (where around 700kbps is available for the user) bandwidth, and it provides communication within the range from up to 1m (class 3, 1mW) till up to 100m (class 1, 100mW). The most popular version is class 2 with 10m range (2.5mW). Every Bluetooth device has a unique, 48-bit MAC address.

Bluetooth offers various “profiles”, for multimedia, serial ports, packet transmission encapsulation (PAN), etc. The most useful for IoT devices is PAN (Personal Area Network) Profile and of course SPP (Serial Port) Profile.

Now Bluetooth covers two branches: BR/EDR (Basic Rate/Enhanced Data Rate) for high-speed audio and file transfer connections and LE (Low Energy) for short burst connections ¹⁶⁵.

Classical (prior to BLE and 4.0) Bluetooth networks can create ad-hoc, so-called WPAN (Wireless Personal Area Networks) sometimes referenced as Piconets. Bluetooth Piconet can handle up to 7+1 devices, where 1 device acts as Master, and it can contact up to 7 Slave devices. Only the Master device can initiate a connection. Fortunately for the IoT approach, much Bluetooth hardware can act as Slave and Master simultaneously, constituting this way a kind of router; thus devices can constitute a tree-like structure called Scatternet (figure 264).



5. Introduction to the IoT Communication and Networking

Figure 264: Bluetooth Scatternet

Bluetooth Low Energy (BLE) uses simplified implementation of the state machine thus is more constrained-devices friendly. It offers a limited range, and it is designed to expose the state rather than transmit streamed data. It provides a speed reaching up to about 1.4 Mbps (2 Mbps aerial throughput) if needed, however. It uses 2.4 GHz band but is designed to avoid interference with WiFi AP and clients. Communication is organised into 3 advertising channels (located “between” WiFi) and 37 communication channels.

Latest Bluetooth implementations (protocol version 5.0 and newer, implemented in mid-2017) offer a Bluetooth mesh network extending ubiquitous connectivity via many-to-many communication model, dedicated to IoT devices, lighting, Industry 4.0, etc. The Bluetooth mesh is layer-organised, and since there is no longer Master-Slave model used, but messages are relayed through the mesh, it is considered to be no longer the Scatternet because of its flat structure¹⁶⁶⁾. Sample Bluetooth Mesh Network is presented in the figure 265 (source: Bluetooth SIG, Mesh Profile Specification v1.0¹⁶⁷⁾).

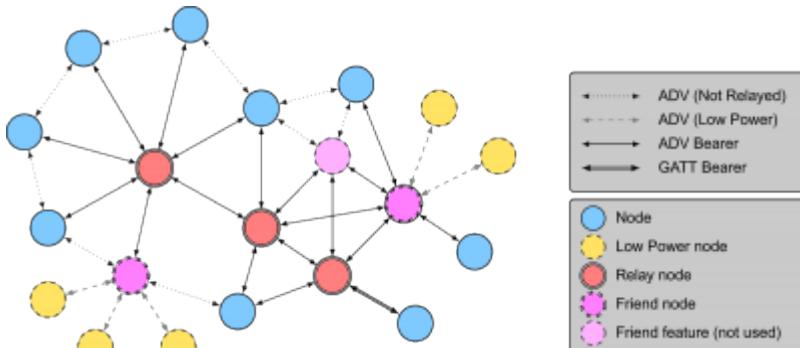


Figure 265: Example Topology of the Bluetooth 5 Mesh Network

Table 21: Bluetooth standard summary

| Bluetooth | Transmission speed | Remarks |
|-----------|-------------------------------|-----------------------|
| 1.0 | 21kbps | few implementations |
| 1.1 | 124kbps | |
| 1.2 | 328kbps | first popular version |
| 2.0 + EDR | 3Mbps | Extended Data Rate |
| 3.0 + HS | 24Mbps | High Speed |
| 3.1 + HS | 40Mbps | |
| 4.0 + LE | 1Mbps | Low Energy |
| 4.1 | designed for IoT | |
| 5.0 | one standard for all purposes | |

Cellular

Cellular (mobile/GSM) networks are one of the possible options because of its wide coverage and long range. Those network use orthogonality in frequency and time spaces. Cellular networks are presented by the subsequent generations (G) - currently up to 4.5G present on the market and 5G in the experimental phase (should be fully functional around the year 2020). Typical GSM network technology, sometimes referenced as an era, runs out within about 10-15 years. It is pretty close but still less than expected end-of-life for classes of IoT devices (15-25 years). GSM hardware used to be backwards compatible, enabling users to access older, even before 2G GSM networks with latest chips.

The picture 266 present GSM network evolution over time and generations. Cellular networks use different frequencies in different countries, yet available radio implementations nowadays are usually able to handle all of them.

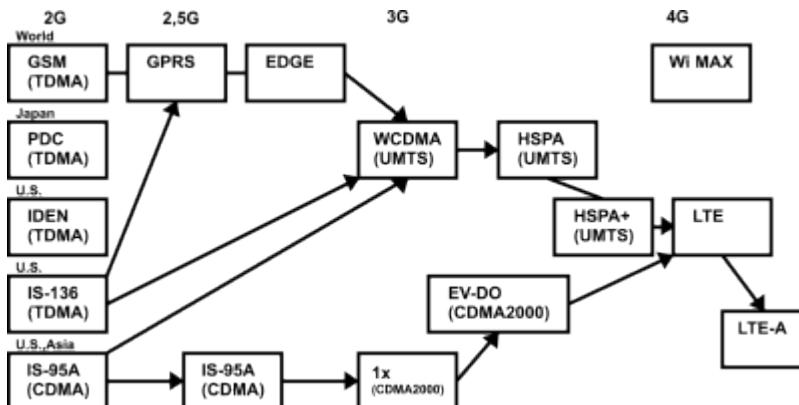


Figure 266: GSM network evolution and generations

The image 267 presents sample GSM hardware (separate module and ready shield for Arduino platform).



5. Introduction to the IoT Communication and Networking



Figure 267: Sample GSM hardware for IoT prototyping

GSM protocols are proprietary, quite complex (including advanced ciphering) and require dedicated hardware. A sort of documentation and standards is not publicly available because of security considerations (i.e. voice transmission ciphering details).

On the one hand, the GSM network seems to be a good solution for extended distant IoT networks, on the other, they have many disadvantages, however. First of all, they require the use of operators' infrastructure - as GSM bands are not free to use.

Professional operation requires licencing and connecting existing infrastructure involves a purchase of the unique identifier (phone ID and a number that is given by the SIM card, physical or virtual) and a service fee.

By the limited access constraints there do exist one more - GSM boards are using quite a significant amount of energy when establishing a connection because they need to broadcast their existence as far as possible, to gain a connection with a possibly distant-located base station. It requires tremendous power and drains the battery (even up to 10W peak) thus cellular solutions are not suitable for the IoT devices that use frequent data communication.

ZigBee

ZigBee protocol is so far very popular in Smart House but also in Industry appliances. Zigbee is a wireless technology developed as an open standard to address the needs of low-cost, low-power wireless machine to machine networks. It is more popular in the industry, however, but because of the relatively higher cost of equipment in comparison with WiFi, Bluetooth or other RF modules. The Zigbee standard operates on the radio bands 2.4 GHz for smart home applications, 915 MHz in US and Australia, 868 MHz in Europe and 784 MHz in China. The advantage of ZigBee is the possibility of forming the mesh networks where nodes are interconnected with others, so there are multiple paths connecting each pair of nodes. Connections are dynamically updated, so when one node turns off the path going through that node will be automatically rerouted via another path. Transmission speed is up to 250 kbps, theoretical range up to 100 m but usually to some 10-30 m. ZigBee does not provide direct, unique IP-addressing on the Networking layer like i.e. 6LowPAN or Thread do. Single ZigBee network can handle up to 65000 devices.

Z-Wave

Z-Wave is a protocol similar in principals to the ZigBee but hardware is cheaper thus it is

5.4. Media layers - Wireless protocols

more towards inexpensive home automation systems. Like in ZigBee, Z-Wave operates on different frequencies depending on the world region, usually between 865 and 926 MHz. Transmission speed is up to 200 kbps and the range is up to 100m. A single Z-Wave network is pretty limited on a number of concurrent devices in one network, that is only 232 devices. Each Z-Wave network has a unique ID and each node (device) in a network has a unique 8-bit identifier.

Thread

Another standard¹⁶⁸⁾ that works using the same 802.15.4 radio. There are some differences in the protocol like address allocation. In 6LowPAN it is done by nodes since in Thread addresses are obtained from DHCPv6 server.

NFC

NFC (Near Field Communication) is a technology that enables two-way interactions between electronic devices. What is important one of the devices does not have to be equipped with the power source – it is powered by the receiving radio signal. That's why NFC is used in contactless card technology enabling devices to exchange the data at a distance of less than 4cm. Transmission speed varies between 100–420kbps, range between both active devices is up to 10 cm, operating frequency 13.56MHz.

Sigfox

Sigfox¹⁶⁹⁾ is the idea to connect objects with sub 1GHz radio frequency. It uses the 900MHz frequency range from the ISM band. The range is about 30-50km (open space), 3-10km (urban environments). This standard uses a technology called Ultra Narrow Band (UNB). It has been designed to transmit data with deficient speed – from 10 to 1000 bps. Thanks to small data packets it consumes only 50 mW of power. It is intended to create the public networks only so using Sigfox requires the subscription plan. Many (but not all) European countries are covered with Sigfox.

LoRaWAN

LoRa (Long Range) is the technology for data transmission with relatively low speed (20 bps do 41 kbps) and the range about 2 km (new transceivers can transmit data up to 15km). It uses CSS (Chirp Spread Spectrum) modulation in the 433MHz ISM radio band. The cell topology is the star with the gateway placed at the central point. End-devices use one hop communication with the gateway, that is connected to the standard IP network with a central network server. The LoRa technology is supported as LoRa WAN by LoRa Alliance¹⁷⁰⁾ designed as Sigfox for public networks, but it can also be used in private networks that do not require a subscription.

5.4.2. NET (NWY) Layer

Traditionally, we use IP addressing (usually masked by DNS to be more user-friendly) when accessing Internet resources. IoT devices may also benefit from this approach. However, constrained devices do require special “editions” of the conventional protocols, that are lightweight. Networking layer implements the basic communication mechanisms on the packet level like routing, delivery, proxying, etc. Many IoT, lightweight implementations of the protocols presented below benefit or at least inherit ideas from

5. Introduction to the IoT Communication and Networking

regular “adult” implementations. Please note that some protocols implement more than one layer, as presented on image 263. We also provide a short reference of the IP v4 and IP v6, to show advantages and drawbacks.

IP v4

Internet Protocol v4 (1981) is perhaps the most widespread networking protocol. The predecessor of the IP v4 protocol originally called IP was introduced in 1974 and supported up to 2^8 hosts, organised in 2^4 subnetworks (RFC 675). In IP v4 (RFC 760/RFC791) the logical addressing space was extended to 2^{32} devices that seemed to be quite much in 1981, but now we struggle with lack of free addressing space. This number is less because some addresses are reserved for, i.e. broadcasting and due to the existence of different classes of addresses and their pools¹⁷¹⁾.

Sample IP v4 address is i.e.: 192.168.1.1 .

Some relief to suffocating Internet was brought as an ad-hoc solution with an introduction of the NAT (Network Address Translation). NAT-enabled subnetworks are those, where one public address represents a set of devices hidden behind the router, but that limits usability because of lack of direct access and unique identification in the global network of the devices sharing private address spaces. Even so, there are about 8.5 billion IoT devices expected to be connected to the Internet by the end of the 2017 year, according to the Gartner's report¹⁷²⁾. They all need to be uniquely addressed!

IP v6

IP v6 is the next generation of the IP v4 protocol. It is supposed to replace IP v4, but this process is somehow not so quick as there are many solutions still present on the Internet and Intranets that implement IP v4 only and would become inoperable if IP v4 would not be available anymore. IP v6 brings addressing space large enough to cover all existing and future needs. The number of possible addresses is 2^{128} . Addresses are presented by 8 groups of 4 hexadecimal values, i.e.,

2001:0db8:0000:0042:0000:8a2e:0370:7334

This brings the capability to uniquely identify any device connected to the Internet using its IP v6 address. Regarding IoT, implementations have many drawbacks (IP v4 also has them). IP v6 network is star-like whereas IoT networks can benefit from the mesh model. IP v6 network requires a controller providing free addresses (a DHCP server) - devices need to contact it to obtain the address. Every single IoT device needs to keep a list of devices it corresponds with (ARP) to resolve their physical address. Moreover, full IP v6 stack implementation requires large RAM, when used.

6LoWPAN

The name is the abbreviation of “IPv6 over Low-Power Wireless Personal Area Networks”¹⁷³⁾ and as it says is the IP based network. This protocol was introduced as a lightweight version of full IP v6, IoT-oriented. This feature allows connecting 6LoWPAN networks with other networks using so-called Edge Router. Thus every node can be visible on the Internet as states in IoT idea. This standard has been developed to operate on the radio channel defined in 802.15.4 (as ZigBee, Z-Wave). It creates the adaptation layer that allows using IPv6 over 802.15.4 link. 6LoWPAN has been adopted in Bluetooth Smart 4.2 standard as well.

5.4. Media layers - Wireless protocols

6LoWPAN supports two addressing models: 64bit and 16 bit (that, of course, limits the number of devices connected to one network to 64000 nodes). The primary frame size is just 127 bytes (comparing to full IP v6 where it is 1280 bytes at least). 6LoWPAN supports unicast and broadcast. It also supports IP routing and link-layer mesh (802.15.5) that enables the introduction of the fail-safe redundant, self-organising networks, because the link-layer mesh can have more than one Edge Router. 6LoWPAN uses autoconfiguration for neighbour devices discovery so does not require DHCP server. It also supports ciphered transportation using AES 128 (and AES 64 for constrained devices).

Sample 6LoWPAN network is presented on image 268.

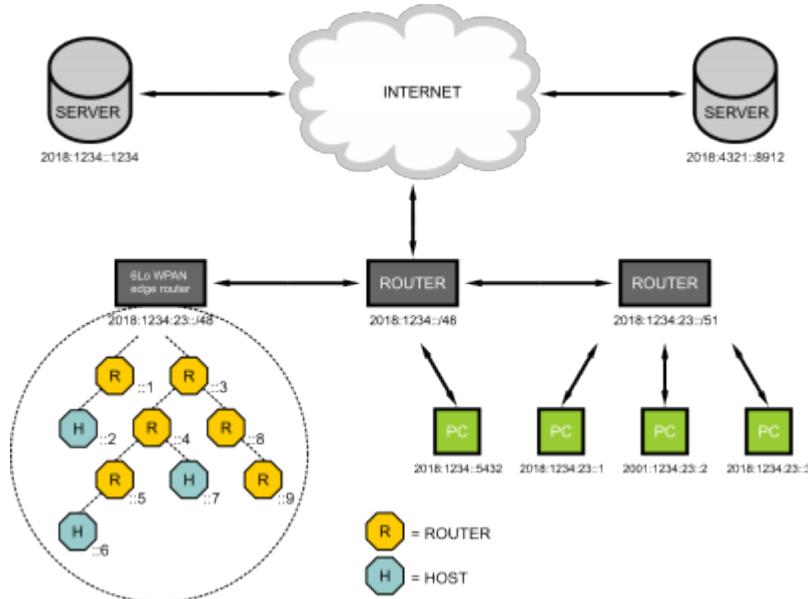


Figure 268: Sample 6LoWPAN and Internet

6LoWPAN devices can be just nodes (Hosts) or nodes with routing capability (Routers) as presented on the image 268.

A gateway between 6LoWPAN and regular IP v6 (IP v4) network is implemented by the Edge Router. Its purpose is to translate “compressed” IP v6 addresses to ensure bi-directional communication between the Internet and 6LoWPAN nodes. Note - the network structure of the 6LoWPAN is logically flat (star/mesh with single addressing space), and devices have unique MAC addresses to be recognisable by the Edge Router device.

When the 6LoWPAN network starts, there are three operations done, repeated consequently:

1. Commissioning - Establishes connectivity on the Data Link Layer level between nodes.
2. Bootstrapping - Performs address configuration, discovery and registration.

5. Introduction to the IoT Communication and Networking

3. Route Initiation - Executing routing algorithms to set up paths.

Typical IP v6 networking discovery won't work here because multicast / broadcast messages are not passable through 6LoWPAN routing nodes (Routes as on image 268).

An interesting procedure is performed when an IoT node (device) wants to connect to the existing 6LoWPAN network. As there is no central DHCP server broadcasting information, the device needs to discover the configuration and create 6LoWPAN address itself. It issues the network discovery process.

Network discovery (discovery of neighbour nodes) in 6LoWPAN uses 4 principals:

- NR - Node Registration
- NC - Node Confirmation
- DAD - Duplicate Address Detection
- Support for Edge Routers

The process is presented on the image 269

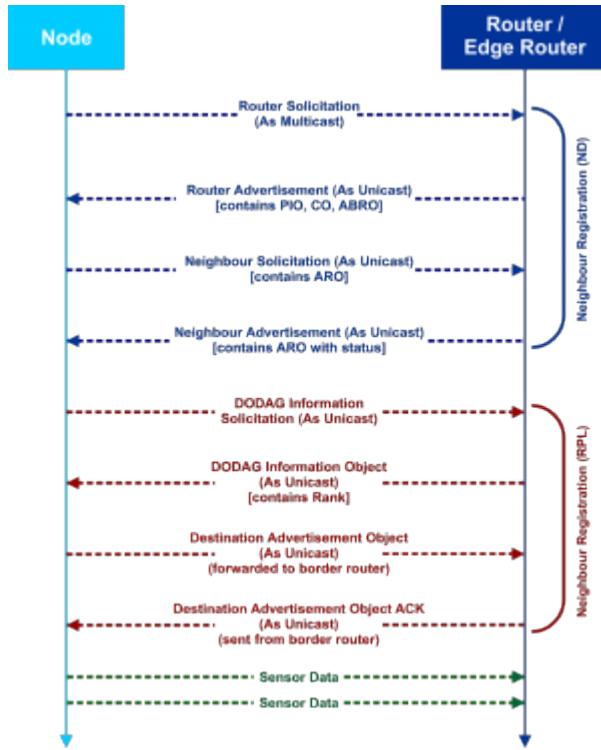


Figure 269: 6LoWPAN Network Automated Discovery demystified

Network Automated Discovery is composed of two, main sections:

- Part one (dark blue) – Neighbor Discovery (ND):

- New node sends RS multicast (SLLAO)
- All routers* respond unicast RA (PIO+6CO+ABRO+SLLAO)
- Node selects one router as default (usually first RA obtained) and derives global Ipv6 address based on prefix delivered (PIO)
- Node sends ARO (ARO+SLLAO) unicast to the selected router
- Router returns ARO with a status
 - Status is: OK, Duplicate Address, Cache Full, other (see RFC5226)
 - Assuming status OK, the router adds new neighbour node address into the cache.
- Node send periodically NS to inform that "it is alive" the router (so-called NUD (Neighbor Unreachability Detection))
- Process above may involve DAD (Duplicate Address Discovery) mechanism to be triggered.
 - On registration, router "asks" all Edge Routers if address offered by the node is unique (DAR/DAC messages).
 - DAD message is expected to "wake up" IoT device from standby mode!
- Part two (red) – Network Registration (NR):
 - Node sends DODAG Solicitation (DIS) unicast to the router.
 - Router responds with DODAG Information Object (DIO) and keeps broadcasting it periodically. DIO contains router rank (i.e. it presents, how far the router is from the Edge Router).
 - If node obtains DIO with better rank, it should re-register with other "better" router as a new default router.
 - Finally node sends Destination Advertising Object (DAO) to its default router that is forwarded to Edge Router.
 - Edge Router responds with DAO ACK.

This way the new 6LoWPAN node can join the new network seamlessly. Moreover, this mechanism enables 6LoWPAN mesh network to self-organise itself if needed, i.e. because of the failure of the router.

5.4.3. Host layer protocols

The host layers protocols include session (SES), presentation (PRES) and application (APP) level, particularly APP (application) layer in the regular Internet communication is dominated by the HTTP protocol and XML-related derivatives like i.e. SoAP. Also, ~~FTP~~ protocol for file transfer is ubiquitous; it exists since the beginnings of the Internet. Most of them are somehow related to the text. They're referenced as "WEB" protocols. Although these protocols are frequently used by advanced and more powerful IoT devices, this is problematic to be implemented in the constrained IoT devices world. I.e. even simplest HTTP header occupies at least 24+8+8+31 bytes without any payload!

There is also a problem to cross firewall boundaries when communication between subnetworks of the IoT devices is expected to occur. Some IoT designed protocols are reviewed below.

5. Introduction to the IoT Communication and Networking

MQTT

MQTT protocol¹⁷⁴⁾ was invented especially for the constrained IoT devices and low bandwidth networks. It is located in APP layer 7 of the ISO/OSI stack, but in fact, it covers all layers 5-7. It is text-based protocol yet very compact and efficient. Protocol stack implementation requires about 10kB of RAM/flash only.

MQTT uses TCP connection so requires open connection channel (this is in opposite to UDP connections, where communications work in a way: "send and forget"). It is considered a drawback of the original MQTT protocol, but there do exist MQTT variations for non-TCP networks, i.e. MQTT-SN. Protocol definition provides reliability and delivery ensure mechanisms.

The standard MQTT Message header is composed of just two bytes only (table 22)! There are 16 MQTT messages types. Some messages types require variable length header.

Table 22: MQTT standard message header

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------|------------------|----------|-----------|--------|---|---|---|---|
| byte 1 | Message Type | DUP flag | Qos level | RETAIN | | | | |
| byte 2 | Remaining length | | | | | | | |

MQTT requires for its operation a centralized MQTT broker that is located outside of firewalls and NATs, where all clients connect, send and receive messages via **publish/subscribe** model. The client can act as publisher and subscriber simultaneously. The image 270 presents publish-subscribe model idea.

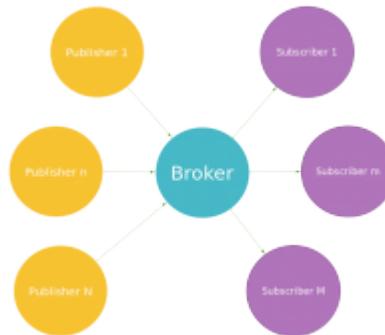


Figure 270: MQTT Broker, Publishers and Subscribers

MQTT Message

MQTT is a text-based protocol and is data-agnostic. A message is composed of a Topic (text) and a Payload (data). The topic is a directory-like string with / "slash" delimiter. Thus all Topics constitute (or actually may represent) a kind of tree-like folders, similar to those on the file system. The subscriber can subscribe to specific, single Topic, or to a variety of Topics using wildcards, where:

- # stands for entire branch,
- + stands for the single level.

5.4. Media layers - Wireless protocols

Example scenario.

Publishers deliver some air quality information data in separate MQTT messages and for various rooms at the department Inf of the Universities (SUT, RTU, ITMO) to the Broker:

| Topic (publishers): |
|--|
| SUT/Inf/Room1/Sensor/Temperature |
| SUT/Inf/Corridor/Sensor/Temperature |
| SUT/Inf/Auditorium1/Sensor/Temperature |
| RTU/Inf/Room1/Sensor/Temperature |
| ITMO/Inf/Room1/Sensor/Temperature |
| RTU/Inf/Room1/Sensor/Humidity |
| SUT/Inf/Room3/Sensor/Temperature |
| RTU/Inf/Room1/Window/NorthSide/State |

The subscriber 1 wills to get all sensor data for SUT university, Inf (informatics) department only, for any space:

| Topic (subscription): |
|------------------------------|
| SUT/Inf/+/Sensor/# |

The subscriber 2 wills to get only Temperature data virtually from any sensor and in any location in ITMO:

| Topic (subscription): |
|------------------------------|
| ITMO/#/Temperature |

The subscriber 3 wills to get any information from the sensors, but only for the RTU

| Topic (subscription): |
|------------------------------|
| RTU/# |

The payload (data) of the message is text as well, so in case one need to send binary data, it is necessary to encode it (i.e. Base64).

MQTT Broker

MQTT Broker is a server for both publishers and subscribers. The connection is initiated from client to the Broker, so assuming it is located outside of a firewall, it breaks firewall its boundaries. The Broker provides QoS (Quality of Service), and it can retain message payload. There are three levels of MQTT Broker QoS (supplied in the message level):

- Unacknowledged service: Ensures that MQTT message is delivered at most once to each subscriber.
- Acknowledged service: Ensures delivery of the message at least once to every subscriber. The broker expects acknowledgement to be sent from the subscriber.

5. Introduction to the IoT Communication and Networking

Otherwise, it retransmits data.

- Assured service: This is two-step delivery of the message, and ensures the message is delivered exactly once to every subscriber.

For Acknowledged and Assured services it is vital to provide unique packet IDs in MQTT frame.

The DUP flag (byte 1, bit 3) represents information sent by the publisher if the message is a "first try" (0) or a retransmitted one (1). It is mostly for internal purposes, and this flag is never propagated to the subscribers.

MQTT offers some limited set of features (options):

- Clean session flag for durable connections:
 - If set True, Broker removes all of the client subscriptions on client disconnect.
 - Otherwise Broker collects messages (QoS depending) and delivers them on client reconnecting; thus connections remain idle.
- MQTT "will": on connection lost, Broker will automatically "simulate" publishing of the pre-defined MQTT message (topic and payload). All clients subscribing this message (whether directly or via a wildcard) will be notified immediately. It is a great feature for failure/disaster recovery.
- Message retaining: it is a feature for regular messages. Any message can be set as retaining and in such case Broker will keep the last one. Once a new client subscribes topic, it will receive a retained message immediately even if the publisher is not publishing any message at the moment. This feature is **last known good value**. It is good to present publishers state (i.e. publisher sends retained message meaning "I'm going offline" and the disconnects. Any client connecting will be notified immediately about the device (client) state.

Interestingly MQTT is a protocol used by Facebook Messenger¹⁷⁵⁾. It is also implemented natively in Microsoft Azure and Amazon Web Services (among many others).

MQTT security is rather weak. MQTT Broker can offer user and password verification yet it is sent plain text. However, all communication between client and Broker may be encapsulated in SSL, encrypted stream.

A short comparison of MQTT and HTTP protocols are presented in the table 23:

Table 23: MQTT vs HTTP

| | MQTT | HTTP |
|-----------------------|----------------------------------|-------------------------------|
| Design | Data centric | Document centric |
| Pattern | Publish/Subscribe | Request/Response |
| Complexity | Simple | Complex |
| Message size | Small, with 2 byte binary header | Larger with text based status |
| Service levels | 3 QoS | none |

5.4. Media layers - Wireless protocols

| | MQTT | HTTP |
|---------------------------------|-------------------------------|-------------------------------------|
| Implementation | C/C++: 10-30kB Java ~100kB | Depends on application but hits >MB |
| Data distribution models | 1-to-1 1-to-N | 1-to-1 |

CoAP

CoAP protocol (RFC7252) originates from the REST (Representational State Transfer). CoAP does not use a centralised server as MQTT does, but every single device "hosts" a server on its own to provide available resources to the clients asking for service offering distributed resources. CoAP uses UDP (comparing to MQTT that uses TCP) and is stateless thus does not require memory for tracking the state. The CoAP implementation assumes every single IoT device has a unique ID and things can have multiple various representations. It is intended to link "things" together using existing standard methods. It is rather a resource-oriented (not document-oriented like HTTP/HTML), designed for slow IoT networks with a high degree of packet loss, also support devices to be periodically offline. CoAP uses URIs :

- "coap://" host[:port] path ["?" query] to access a service / resource,
- a secure, encrypted version uses "coaps" instead of "coap".

It supports various content types, can work with proxy and can be cached.

The protocol is designed to be compact and simple to implement. The stack implementation takes only about 10kB or RAM and 100kB of storage. The header is only 4 bytes.

CoAP protocol has a binary header to decrease overhead but payload depends on the content type. Initial, non-exclusive list of the payload types include:

- text/plain (charset=utf-8) (ID=0, RFC2046, RFC3676, RFC5147),
- application/link-format (ID=40, RFC6690),
- application/xml (ID=41 RFC3023),
- application/octet-stream (ID=42, RFC2045, RFC2046),
- application/json (ID=50, RFC7159).

CoAP endpoint services are identified by unique IP and port number. However, they operate on the UDP instead of TCP (like, i.e. HTML does). The transfer in CoAP is made using non-reliable UDP network so that a message can appear duplicated, disappear or it can be delivered in other order than initially sent. Because of the nature of datagram communication, messages are exchanged asynchronously between two endpoints, transporting Requests and Responses. CoAP messages can be (non-exhaustive list):

- CON (Confirmable, those requiring ACK Acknowledge),
- NON (Non-Confirmable, those that do not need ACK),
- ACK (an acknowledgement message),
- RESET (sent if CON or NON was received, but the receiver cannot understand the context, i.e. part of the communication is missing because of device restart, messages memory loss, etc.). Empty RESET messages can be used to "ping" the

5. Introduction to the IoT Communication and Networking

device.

Because of the UDP network characteristics, CoAP provides an efficient yet straightforward reliability mechanism to ensure successful delivery of messages:

- stop and wait for retransmission with exponential back-off for CON messages,
- duplicate message detection for CON and NON-messages.

Request-response pair is identified by a unique "Token". Sample request-response scenarios are presented in images below. Sample CoAP message exchange scenarios between client and server are presented on images (two per image) 271 and 272.

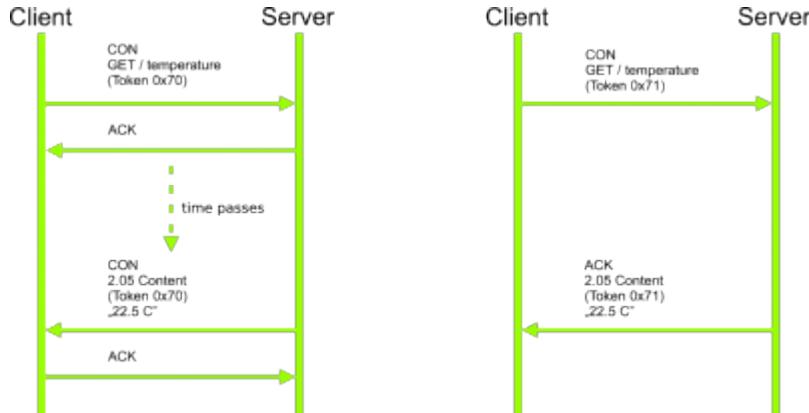


Figure 271: CoAP scenario 1: confirmable with time delay payload answer (0x70) and immediate payload answer (0x71)

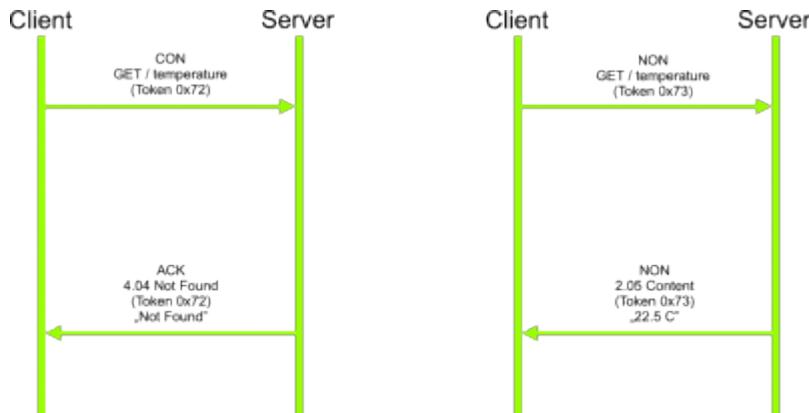


Figure 272: CoAP scenario 2 - unrecognized request (0x72) and non-confirmable request (0x73)

The scenario on the image 271 (left, with token 0x70) is executed in situation when a CoAP server device (a node) need some time to prepare data and cannot deliver information right away. The scenario on the image 271 (right, with token 0x71) is used,

5.4. Media layers - Wireless protocols

when a CoAP server can provide information to the client immediately. The scenario on the image 272 (left, with token 0x72) appears, when a CoAP server cannot understand the request. The scenario on the image 272 (right, with token 0x73) presents the situation where the request to the CoAP server was made with a non-confirmable request.

6. Data and Information Management in the Internet of Things

At the centre of the IoT ecosystem consisting of billions of connected devices is the wealth of information that can be made available through the fusion of data that is produced in real-time, as well as data stored in permanent repositories. This information can make the realisation of innovative and unconventional applications and value-added services possible and will act as an immense source for trend analysis and strategic business opportunities. A comprehensive management framework of data and information that is generated and stored by the objects within the IoT is thus required to achieve this goal.

Data management is a broad concept referring to the architectures, practices, and procedures for proper management of the data lifecycle requirements of a particular IT system. As far as the IoT is concerned, data management should act as a layer between the physical sensing objects and devices generating the data - on the one hand, and the applications accessing the data for analysis purposes and services - on the other.

The IoT data has distinctive characteristics that make traditional relational-based database management an obsolete solution. A massive volume of heterogeneous, streaming and geographically-dispersed real-time data will be created by millions of diverse devices periodically sending observations about certain monitored phenomena or reporting the occurrence of certain or abnormal events of interest. Periodic observations are most demanding regarding communication overhead and storage due to their streaming and continuous nature, while events present time-strain with end-to-end response times depending on the urgency of the response required for the event. Furthermore, in addition to the data that is generated by IoT entities, there is also metadata that describes these entities (i.e. "things"), such as object identification, location, processes and services provided. The IoT data will statically reside in fixed- or flexible-schema databases and roam the network from dynamic and mobile objects to concentration storage points. It will continue until it reaches centralised data stores. Communication, storage and process will thus be defining factors in the design of data management solutions for the IoT.

Traditional data management systems handle the storage, retrieval, and update of elementary data items, records and files. In the context of the IoT, data management systems must summarise data online while providing storage, logging, and auditing facilities for offline analysis. It expands the concept of data management from offline storage, query processing, and transaction management operations into online-offline communication/storage dual operations. We first define the data lifecycle within the context of the IoT and then discuss some of the phases to have a better understanding of the IoT data management.

IoT data lifecycle

IoT data management versus traditional database management systems

IoT data sources

Main IoT domains generating data

Infrastructure and architectures for IoT Data processing: Cloud, Fog, and Edge computing

IoT data storage models and frameworks

IoT data processing models and frameworks

IoT data semantics

IoT data visualisation

Sources

1. <https://www.cbronline.com/internet-of-things/10-of-the-biggest-iot-data-generators-4586937/>
2. <https://www.sam-solutions.com/blog/how-much-data-will-iot-create-2017/>
3. <http://www.enterprisefeatures.com/6-important-stages-in-the-data-processing-cycle/>
4. <https://pinaclsolutions.com/blog/2017/cloud-computing-and-iot>
5. <http://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Its-time-for-fog-edge-computing-in-the-internet-of-things>
6. <https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared>
7. Meng Ma, Ping Wang, and Chao-Hsien Chu. 2013. Data Management for Internet of Things: Challenges, Approaches and Opportunities. In Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (GREENCOM-ITHINGS-CPSCom '13). IEEE Computer Society, Washington, DC, USA, 1144-1151. DOI= <http://dx.doi.org/10.1109/GreenCom-iThings-CPSCom.2013.199>
8. M. Marjani et al., "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges," in IEEE Access, vol. 5, pp. 5247-5261, 2017. doi: 10.1109/ACCESS.2017.2689040 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7888916&isnumber=7859429>
9. C. W. Tsai, C. F. Lai, M. C. Chiang and L. T. Yang, "Data Mining for Internet of Things: A Survey," in IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 77-97, First Quarter 2014. doi: 10.1109/SURV.2013.103013.00206 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6674155&isnumber=6734839>

6.1. IoT data lifecycle

Data processing is simply the conversion of raw data to meaningful information through a process. Data is manipulated to produce results that lead to a resolution of a problem or improvement of an existing situation. Similar to a production process, it follows a cycle where inputs (raw data) are fed to a process (computer systems, software, etc.) to produce output (information and insights). Generally, organisations employ computer systems to carry out a series of operations on the data in order to present, interpret, or obtain information. The process includes activities like data entry, summary, calculation, storage, etc. A useful and informative output is presented in various appropriate forms such as diagrams, reports, graphics, etc.

6. Data and Information Management in the Internet of Things

The lifecycle of data within an IoT system proceeds from data production to aggregation, transfer, optional filtering and preprocessing, and finally to storage and archiving. Querying and analysis are the endpoints that initiate (request) and consume data production, but data products can be set to be “pushed” to the IoT consuming services. Production, collection, aggregation, filtering, and some basic querying and preliminary processing functionalities are considered online, communication-intensive operations. Intensive preprocessing, long-term storage and archival and in-depth processing/analysis are considered offline storage-intensive operations.

Storage operations aim at making data available on the long-term for constant access/updates, while archival is concerned with read-only data. Since some IoT systems may generate, process, and store data in-network for real-time and localised services, with no need to propagate this data further up to concentration points in the system, edge devices that combine both processing and storage elements may exist as autonomous units in the cycle. In the following paragraphs, each of the elements in the IoT data lifecycle is explained.

1. **Querying:** data-intensive systems rely on querying as the core process to access and retrieve data. In the context of the IoT, a query can be issued either to request real-time data to be collected for temporal monitoring purposes or to retrieve a certain view of the data stored within the system. The first case is typical when a (mostly localised) real-time request for data is required. The second case represents more globalised views of data and in-depth analysis of trends and patterns.
2. **Production:** Data production involves sensing and transfer of data by the edge devices within the IoT framework and reporting this data to interested parties periodically (as in a subscribe/notify model), pushing it up the network to aggregation points and subsequently to database servers, or sending it as a response triggered by queries that request the data from sensors and smart objects. Data is usually time-stamped and possibly geo-stamped and can be in the form of simple key-value pairs, or it may contain rich (unstructured) audio/image/video content, with varying degrees of complexity in-between.
3. **Collection:** The sensors and smart objects within the IoT may store the data for a certain time interval or report it to govern components. Data may be collected at concentration points or gateways within the network, where it is further filtered and processed, and possibly fused into compact forms for efficient transmission. Wireless communication technologies such as Zigbee, Wi-Fi and mobile networks are used by objects to send data to collection points. A collection is the first stage of the cycle and is very crucial since the quality of data collected will impact heavily on the output. The collection process needs to ensure that the data gathered are both defined and accurate so that subsequent decisions based on the findings are valid. This stage provides both the baseline from which to measure and a target on what to improve. Some types of data collection include census (data collection about everything in a group or statistical population), sample survey (collection method that includes only part of the total population), and administrative by-product (data collection is a byproduct of an organisation's day-to-day operations).
4. **Aggregation/Fusion:** Transmitting all the raw data out of the network in real-time is often prohibitively expensive, given the increasing data streaming rates and the limited bandwidth. Aggregation and fusion techniques deploy summarisation and merging operations in real-time to compress the volume of data to be stored and transmitted.

6.1. IoT data lifecycle

5. Delivery: As data is filtered, aggregated, and possibly processed either at the concentration points or at the autonomous virtual units within the IoT, the results of these processes may need to be sent further up the system, either as final responses or for storage and in-depth analysis. Wired or wireless broadband communications may be used there to transfer data to permanent data stores.
 6. Preprocessing: the IoT data will come from different sources with varying formats and structures. Data may need to be preprocessed to handle missing data, remove redundancies and integrate data from different sources into a unified schema before being committed to storage. Preparation is the manipulation of data into a form suitable for further analysis and processing. Raw data cannot be processed and must be checked for accuracy. Preparation is about constructing a dataset from one or more data sources to be used for further exploration and processing. Analysing data that has not been carefully screened for problems can produce highly misleading results that are heavily dependent on the quality of data prepared. This preprocessing is a known procedure in data mining called data cleaning. Schema integration does not imply brute-force fitting of all the data into a fixed relational (tables) schema, but rather a more abstract definition of a consistent way to access the data without having to customise access for each source's data format(s). Probabilities at different levels in the schema may be added at this phase to the IoT data items in order to handle uncertainty that may be present in data or to deal with the lack of trust that may exist in data sources.
 7. Storage/Update & Archiving: This phase handles the efficient storage and organisation of data, as well as the continuous update of data with new information as it becomes available. Archiving refers to the offline long-term storage of data that is not immediately needed for the system's ongoing operations. The importance of this step is that it allows quick access and retrieval of the processed information, allowing it to be passed on to the next stage directly when needed. The core of centralised storage is the deployment of storage structures that adapt to the various data types and the frequency of data capture. Relational database management systems are a popular choice that involves the organisation of data into a table schema with predefined interrelationships and metadata for efficient retrieval at later stages. NoSQL key-value stores are gaining popularity as storage technologies for their support of Big Data storage with no reliance on relational schema or strong consistency requirements typical of relational database systems. Storage can also be decentralised for autonomous IoT systems, where data is kept at the objects that generate it and is not sent up the system. However, due to the limited capabilities of such objects, storage capacity remains limited in comparison to the centralised storage model.
 8. Processing/Analysis: This phase involves the ongoing retrieval and analysis operations performed and stored and archived data in order to gain insights into historical data and predict future trends, or to detect abnormalities in the data that may trigger further investigation or action. Task-specific preprocessing may be required to filter and clean data before meaningful operations can take place. When an IoT subsystem is autonomous and does not require permanent storage of its data, but rather keeps the processing and storage in the network, then in-network processing may be performed in response to real-time or localised queries.
 9. Output and interpretation: This is the stage where processed information is now transmitted to the user. An output is presented to users in various visual formats like diagrams, infographics, printed report, audio, video, etc. The output needs to be interpreted so that it can provide meaningful information that will guide future
-

6. Data and Information Management in the Internet of Things

decisions of the company.

Depending on the architecture of an IoT system and actual data management requirements in place, some of the steps described above can be omitted. Nevertheless, it is possible to distinguish three main patterns for the IoT data flow:

- In relatively autonomous IoT systems, data proceeds from query to production to in-network processing and then delivery.
- In more centralised systems, the data flow that starts from production and proceeds to collection and filtering/aggregation/fusion and ends with data delivery to initiating (possibly global or near real-time) queries.
- In fully centralised systems, the data flow extends the production to aggregation further and includes preprocessing, permanent data storage and archival, and in-depth processing and analysis.

6.2. IoT data management versus traditional database management systems

Based on the IoT data lifecycle discussed earlier, we divide an IoT data management system into i) an online (i.e. real-time) front-end that interacts directly with the interconnected IoT objects and sensors, and ii) an offline back-end that handles the mass storage and in-depth analysis of the IoT data. The data management frontend is communication-intensive, as it involves the propagation of query requests and result to and from sensors and smart objects. The backend is storage-intensive, as it involves the mass storage of produced data for later processing and analysis and more in-depth queries. Although the storage elements reside on the back-end, they interact with the front-end on a frequent basis via continuous updates and are thus referred to as online. The autonomous edges in the IoT data lifecycle can be considered more communication-intensive than storage-intensive, as they provide real-time data to certain queries.

This envisioned data management architecture differs considerably from the existing database management systems (DBMSs), which are mainly storage-centric. In traditional databases, the bulk of data is collected from predefined and finite sources, and stored in scalar form according to strict normalisation rules in relations. Queries are used to retrieve specific "summary" views of the system or update specific items in the database. New data is inserted into the database when needed, also via insertion queries. Query operations are usually local, with execution costs bound to processing and intermediate storage. Transaction management mechanisms guarantee the ACID properties in order to enforce overall data integrity. Even if the database is distributed over multiple sites, query processing and distributed transaction management are enforced. The execution of distributed queries is based on the transparency principle, which dictates that the database is still viewed logically as one centralised unit, and the ACID properties are guaranteed via the two-phase commit protocol.

In the IoT systems, the picture is dramatically different, with a massive and ever-growing number of data sources that include sensors, RFIDs, embedded systems, and mobile devices. Contrary to the occasional updates and queries submitted to traditional DBMSs, data is streaming constantly from a multitude of edge devices to the IoT data stores, and queries are more frequent and with more versatile needs. Hierarchical data reporting and aggregation may be required for scalability guarantees as well as to enable more prompt processing functionality. The strict relational database schema and the relational normalisation practice may be relaxed in favour of more unstructured and flexible forms

that adapt to the diverse data types and sophisticated queries. Although distributed DBMSs optimise queries based on communication considerations, optimisers base their decisions on fixed and well-defined schemas. This may not be the case in the IoT, where new data sources and streaming, localised data create a highly dynamic environment for query optimizers. Striving to guarantee the transparency requirements imposed in distributed DBMSs on IoT data management systems is challenging, if not impossible. Furthermore, transparency may not even be required in the IoT, because innovative applications and services may require location and context awareness. Maintaining ACID properties in bounded IoT spaces (subsystems), while executing transactions can be managed, but is challenging for the more globalised space. However, the element of mobile data sources and how their generated data can be incorporated into the already established data space is a novel challenge that is yet to be addressed by the IoT data management systems.

6.3. IoT data sources

As the IoT gets more involved in various domains and types of operations, a countless number of its data sources generate immense volumes of data. All the sources can be roughly divided into three groups:

Passive sources

They are sensors that do not communicate actively and send the required information to the centralised management system only on demand. For instance, sensors that make atmospheric measurements produce data when API is activated. That does not mean that an application is also passive; on the contrary, the data from passive sensors requires proper management and processing, and it is what an application is purposed for.

Active sources

The main difference between passive and active sensors is that the latter transmit data continuously, not only by request. An example is jet engine sensors. Information comes in a real-time manner, which demands an application to provide its ongoing processing. As the data must be safe, the application must parse it from the stream and then place into a proper format for storage and processing.

Dynamic sources

These sources are most sophisticated, and also the most useful ones. Devices with dynamic sensors interact with respective applications bidirectionally and perform a wide range of capabilities, such as data format and frequency change, security issue fixing, update automation and more. Also, they are auto- and self-tuned. Dynamic sensors do not just produce rough information to an application that processes it but can also send ready data that meets the application's requirements.

6.4. Main IoT domains generating data

The emergence of new information sources indispensably affects the data centre market, and it has experienced structural changes in recent years. Indeed, information technology tends to go beyond processing data in traditional data centres and opt for of cloud-centric ones. In just a few years, only 8% of overall workloads will be handled by old-school data centres.

6. Data and Information Management in the Internet of Things

The IoT is predicted to generate 403ZBs of data a year by 2018, up from 113.4ZBs in 2013. But according to Cisco, not all generated data will be sent to data centres. In three years times, colo sites should be hosting 8.6ZBs, up from 3.1ZB in 2013. IDC has predicted that by 2020 one-tenth of the world's data will be produced by machines. The organisation forecast that in five years time the number of connected devices communicating over the internet will reach 32 billion and generate 10% of the world's data. CBR compiles a list of the top 10 critical areas set to foster data growth resulting from IoT connected solutions.

1. Air travel: Arming planes with smart sensors to prevent failures is already a reality. These sensors produce several terabytes of data per flight. For example, Cisco said that a Boeing 787 aircraft could generate 40 TBs per hour of flight. These IoT solutions in the air industry have several applications beyond preventing failure. They can also reduce fuel consumption, adjust speeds and reduce travel times.
2. Mining: For the mining industry, the main benefit of using the IoT is safety. By automating machines (M2M), humans are not required to stay close to the vehicles and risk their lives. Cisco predicts that mining operations can generate up to 2.4TBs of data every minute.
3. Cars: A smart IoT connected vehicle is a fountain of data. It is continuously transmitting data to manufacturers, to road operators, to its driver, to the authorities, etc. Data generated by smart cars could crash mobile networks with data surges by 2024. The company said connected vehicles are expected to total 2.3 billion by then, which will increase data traffic up to 97% during rush hour traffic at some cell points.
4. Utilities: the worldwide revenue opportunity presented by the IoT for the utility industry by 2018 is estimated to reach \$201 billion. Smart meters are just an example. According to the UK Department of Energy & Climate Change, by the end of 2014, there were a total of 20.8 million gas meters and 25.3 million electricity meters operated by the larger energy suppliers in British domestic properties. Smart meters collect data on how much energy is being used every 30 minutes, 24/7, 365. It sends to the cloud several TBs of information every year.
5. Cities: Smart cities will be made of everything out there. Street lamps talking to the grid, urban parks connecting to services and rivers sending out alerts on pollution levels. All this data is generated on a daily basis, and it's stored in the cloud. Millions of sensors, deployed in every city will continuously produce vast amounts of information.
6. Wearables: It is estimated that by 2019 more than 578 million wearables will be in use around the world. These solutions are continually collecting data on health, fitness and wellness. The amount of data produced by wearables varies according to the device being worn and the type of sensors it has included.
7. Sports: As sports adopt more wearables and smart clothing to improve performances, clubs are also looking at new ways to read the field and polish tactics using predictive analysis. For example, the NBA took on SAP to make its statistics accessible to fans, opening the clubs data to the world. SAP deployed its analytical software, primarily used in business environments, to create a database that records every single move players execute, players' stats, and much more.
8. Logistics: Until today, transportation of goods would be over once the supply chain shipped the products. But with the IoT, the service will be extended further beyond this, and smart goods will constantly produce more data. Some logistic companies

6.5. Infrastructure and architectures for IoT Data processing: Cloud, Fog, and Edge computing

are already collecting data from their suppliers, and also from their suppliers' suppliers. Most of this data will be RFID, giving logistic companies the ability to analyse it in real time and tackle any future problems that might happen in the chain.

9. Healthcare: Smart healthcare is already being adopted in several countries. Huge virtual platforms store patient data that can be accessed by health services anywhere else. The health sector will see tremendous benefits from the IoT, with sensors being deployed across all areas in a medical unit. Medical companies are using connectivity to prevent power surges in medical devices, including critical instruments used in surgeries. All this information is stored for future analysis.
10. Smart homes: Smart homes are already a reality and by 2020 consumers expect this ecosystem to be widely available. It is predicted that one smart connected home today can produce as much as 1GB of information a week.

6.5. Infrastructure and architectures for IoT Data processing: Cloud, Fog, and Edge computing

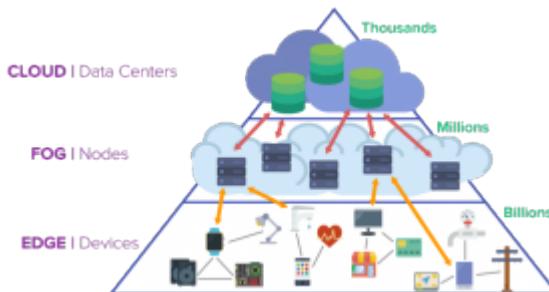


Figure 273: Cloud Edge Fog Computing

The IoT generates a vast amount of Big Data and this, in turn, puts a massive strain on Internet Infrastructure. As a result, this forces companies to find solutions to minimise the pressure and solve their problem of transferring large amounts of data. Cloud computing has entered the mainstream of information technology, providing scalability in the delivery of enterprise applications and Software as a Service (SaaS). Companies are now migrating their information operations to the cloud. Many cloud providers can allow for your data to be either transferred via your traditional internet connection or a dedicated direct link. The benefit of a direct link into the cloud will ensure that your data is uncontended and that the traffic is not crossing the internet and the Quality of Service can be controlled. As the IoT proliferates, businesses face a growing need to analyse data from sources at the edge of a network, whether they are mobile phones, gateways or IoT sensors. Cloud computing has a disadvantage here: It can't process data quickly enough for modern business applications.

Cloud computing and the IoT both serve to increase efficiency in everyday tasks and both have a complementary relationship. The IoT generates massive amounts of data, and cloud computing provides a pathway for this data to travel. Many Cloud providers charge on a pay per use model, which means that you only pay for the computer resources that you use and not more. Economies of scale is another way in which cloud providers can benefit smaller IoT start-ups and reduce overall costs to IoT companies. Another benefit of Cloud Computing for the IoT is that Cloud Computing enables better collaboration which is essential for developers today. By allowing developers to store and access data

6. Data and Information Management in the Internet of Things

remotely, developers can access data immediately and work on projects without delay. Finally, by storing data in the Cloud, this enables IoT companies to change direction quickly and allocate resources in different areas. Big Data has emerged in the past couple of years, and with such emergence, the cloud has become the architecture of choice. Most companies find it feasible to access the massive quantities of IoT Big Data via the Cloud.

The IoT owes its explosive growth to the connection of physical things and operational technologies to analytics and machine learning applications, which can help glean insights from device-generated data and enable devices to make “smart” decisions without human intervention. Currently, such resources are mostly being provided by cloud service providers, where the computation and storage capacity exists. However, despite its power, the cloud model is not applicable to environments where operations are time-critical, or internet connectivity is poor. It is especially true in scenarios such as telemedicine and patient care, where milliseconds can have fatal consequences. The same can be said about vehicle-to-vehicle communications, where the prevention of collisions and accidents can't afford the latency caused by the round-trip to the cloud server.

Moreover, having every device connected to the cloud and sending raw data over the internet can have privacy, security and legal implications, especially when dealing with sensitive data that is subject to separate regulations in different countries. IoT nodes are closer to the action, but for the moment they do not have the computing and storage resources to perform analytics and machine learning tasks. Cloud servers, on the other hand, have the horsepower, but are too far away to process data and respond in time.

The Fog/Edge layer is the perfect junction where there are enough compute, storage and networking resources to mimic cloud capabilities at the edge and support the local ingestion of data and the quick turnaround of results. Main benefits of Fog/Edge computing are the following:

- Increased network capacity: Fog computing uses much less bandwidth, which means it doesn't cause bottlenecks and other similar occupancies. Less data movement on the network frees up network capacity, which then can be used for other things.
- Real-time operation: Fog computing has much higher experience than any different cloud computing architecture we know today. Since all data analysis is being done at the spot, it represents a true real-time concept, which means it is a perfect match for the needs of IoT concepts.
- Data security: Collected data is more secure when it doesn't travel. It also makes information storing much more straightforward because it stays in its country of origin. Sending data abroad might violate specific laws.

Figure 273 pictorially depicts the Cloud Edge Fog computing scenario. The current trend shows that Fog computing will continue to grow in usage and importance as the IoT expands and conquers new grounds. With inexpensive, low-power processing and storage becoming more available, we can expect computation to move even closer to the edge and become ingrained in the same devices that are generating the data, creating even greater possibilities for inter-device intelligence and interactions. Sensors that only log data might one day become a thing of the past.

Fog/edge computing has the potential to revolutionise the IoT in the next several years. It seems evident that while Cloud is a perfect match for the IoT, we have other scenarios

6.6. IoT data storage models and frameworks

and IoT technologies that demand low-latency ingestion and immediate processing of data where Fog computing is the answer. Fog/Edge computing improves efficiency and reduces the amount of data that needs to be sent to the cloud for processing. But it's here to complement the cloud, not replace it. The cloud will continue to have a pertinent role in the IoT cycle. In fact, with Fog computing shouldering the burden of short-term analytics at the edge, cloud resources will be freed to take on the more cumbersome tasks, especially where the analysis of historical data and large datasets is concerned. Insights obtained by the cloud can help update and tweak policies and functionality at the fog layer. To sum up, it is the combination of Fog and Cloud computing that will accelerate the adoption of the IoT, especially for the enterprise.

6.6. IoT data storage models and frameworks

The increasing volumes of heterogeneous unstructured IoT data have also led to the emergence of several solutions to store these overwhelming datasets and support appropriate data management:

- NoSQL databases are often used for storing IoT Big Data. It is a new type of database which is becoming more and more popular among web companies today. Proponents of NoSQL solutions state that they provide more straightforward scalability and improved performance relative to traditional relational databases. These products excel at storing "unstructured data," and the category includes open source products such as Cassandra, MongoDB, and Redis.
- In-memory databases assume that data is stored in computer memory to make access to it faster. Representative examples are Redis and Memcached, both NoSQL databases, entirely served from memory. These products excel at storing "unstructured data," and the category includes open source products such as Cassandra, MongoDB, and Redis.

6.7. IoT data processing models and frameworks

Processing frameworks and processing engines are responsible for computing over data in a data system. While there is no authoritative definition setting apart "engines" from "frameworks", it is sometimes useful to define the former as the actual component responsible for operating on data and the latter as a set of elements designed to do the same. For instance, Apache Hadoop can be considered a processing framework with MapReduce as its default processing engine. Engines and frameworks can often be swapped out or used in tandem. For instance, Apache Spark, another framework, can hook into Hadoop to replace MapReduce. This interoperability between components is one reason that big data systems have great flexibility.

While the systems which handle this stage of the data lifecycle can be complex, the goals on a broad level are very similar: operate over data to increase understanding, surface patterns, and gain insight into complex interactions. To simplify the discussion of these components, we will group these processing frameworks by the state of the data they are designed to handle. Some systems handle data in batches, while others process data in a continuous stream as it flows into the system. Still, others can manage data in either of these ways.

Batch Processing Systems

Batch processing has a long history within the big data world. Batch processing involves

6. Data and Information Management in the Internet of Things

operating over a large, static dataset and returning the result at a later time when the computation is complete. The datasets in batch processing are typically bounded: batch datasets represent a limited collection of data persistent: data is almost always backed by some permanent storage large: batch operations are often the only option for processing extensive sets of data Batch processing is well-suited for calculations where access to a complete set of records is required. For instance, when calculating totals and averages, datasets must be treated holistically instead of as a collection of individual records. These operations require that state is maintained for the duration of the calculations. Tasks that need vast volumes of data are often best handled by batch operations. Whether the datasets are processed directly from permanent storage or loaded into memory, batch systems are built with large quantities in mind and have the resources to handle them. Because batch processing excels at managing large volumes of persistent data, it frequently is used with historical data.

The trade-off for handling large quantities of data is a longer computation time. Because of this, batch processing is not appropriate in situations where processing time is especially significant. Because this methodology heavily depends on permanent storage, reading and writing multiple times per task, it tends to be somewhat slow. On the other hand, since disk space is typically one of the most abundant server resources, it means that MapReduce can handle enormous datasets. MapReduce has incredible scalability potential and has been used in production on tens of thousands of nodes.

Apache Hadoop: Apache Hadoop is a processing framework that exclusively provides batch processing. Hadoop was the first big data framework to gain significant traction in the open-source community. Based on several papers and presentations by Google about how they were dealing with tremendous amounts of data at the time, Hadoop reimplemented the algorithms and component stack to make large-scale batch processing more accessible. Apache Hadoop and its MapReduce processing engine offer a well-tested batch processing model that is best suited for handling extensive datasets where time is not a significant factor. The low cost of components necessary for a well-functioning Hadoop cluster makes this processing inexpensive and useful for many use cases. Compatibility and integration with other frameworks and engines mean that Hadoop can often serve as the foundation for multiple processing workloads using diverse technology.

Stream Processing Systems

Stream processing systems compute over data as it enters the system. It requires a different processing model than the batch paradigm. Instead of defining operations to apply to an entire dataset, stream processors determine processes that will be used to each individual data item as it passes through the system. The datasets in stream processing are considered “unbounded”. It has a few important implications:

- The total dataset is only defined as the amount of data that has entered the system so far.
- The working dataset is perhaps more relevant and is limited to a single item at a time.
- Processing is event-based and does not “end” until explicitly stopped. Results are immediately available and will be continually updated as new data arrives.

Stream processing systems can handle a nearly unlimited amount of data, but they only process one (true stream processing) or very few (micro-batch processing) items at a

time, with a minimal state being maintained in between records. While most systems provide methods of maintaining some state, stream processing is highly optimised for more functional processing with few side effects.

Functional operations focus on discrete steps that have limited state or side-effects. Performing the same operation on the same piece of data will produce the same output independent of other factors. This kind of processing fits well with streams because state between items is usually some combination of challenging, limited, and sometimes undesirable. So while some type of state management is generally possible, these frameworks are much simpler and more efficient in their absence.

This type of processing lends itself to certain kinds of workloads. Processing with near real-time requirements is well served by the streaming model. Analytics, server or application error logging, and other time-based metrics are a natural fit because reacting to changes in these areas can be critical to business functions. Stream processing is a good fit for data where you must respond to changes or spikes and where you're interested in trends over time.

- Apache Storm
- Apache Samza

Hybrid Processing Systems

Some processing frameworks can handle both batch and stream workloads. These frameworks simplify diverse processing requirements by allowing the same or related components and APIs to be used for both types of data. The way that this is achieved varies significantly between Spark and Flink, the two frameworks we will discuss. It is mainly a function of how the two processing paradigms are brought together and what assumptions are made about the relationship between fixed and unfixed datasets. While projects focused on one processing type may be a close fit for specific use-cases, the hybrid frameworks attempt to offer a general solution for data processing. They not only provide methods for processing over data, but they also have their integrations, libraries, and tools for doing things like graph analysis, machine learning, and interactive querying.

- Apache Spark
- Apache Flink

6.8. IoT data semantics

With some 25 billion devices expected to be connected to the Internet by 2015 and 50 billion by 2020, providing interoperability among the things on the IoT is one of the most fundamental requirements to support object addressing, tracking, and discovery as well as information representation, storage, and exchange.

The lack of explicit and formal representation of the IoT knowledge could cause ambiguity in terminology, hinder interoperability and mostly semantic interoperability of entities in the IoT world. Furthermore, lack of shared and agreed semantics for this domain (and for any domain) may easily result to semantic heterogeneity - i.e. to the need to align and merge a vast number of different modelling efforts to semantically describe IoT entities, efforts conducted by many different ontology engineers and IoT vendors (domain experts). Although there are tools nowadays to overcome such a problem, it is not a fully automated and precise process, and it would be much easier to do so if there is at least a partial agreement between the related stakeholders - i.e. a

6. Data and Information Management in the Internet of Things

commonly agreed IoT ontology.

In these circumstances, an ontology can be used as a semantic registry for the facilitation of the automated deployment of generic and legacy IoT solutions in environments where heterogeneous devices also have been deployed. Such a service can be delivered by IoT solution providers, supporting the interoperability problems of their clients/buyers remotely when buying third-party devices or applications. Practically, this will require the existence of a central point - e.g. a web service/portal for both end users (buyers of the devices) and the IoT solution providers (sellers of the applications) to register their resources, i.e. both the devices and the IoT solutions, in an ontology-based registry.

Sensor Web Enablement and Semantic Sensor Networks

The Sensor Web Enablement (SWE) standards enable developers to make all types of sensors, transducers and sensor data repositories discoverable, accessible and usable via the Web. Sensor technology, computer technology and network technology are advancing together while demand grows for ways to connect information systems with the real world. Linking diverse technologies in this fertile market environment, integrators are offering new solutions for plant security, industrial controls, meteorology, geophysical survey, flood monitoring, risk assessment, tracking, environmental monitoring, defence, logistics and many other applications. The SWE effort develops the global framework of standards and best practices that make linking of diverse sensor-related technologies fast and practical. Standards make it possible to put the pieces together in an efficient way that protects earlier investments, prevents lock-in to specific products and approaches, and allows for future expansion. Standards also influence the design of new component products. Business needs drive the process. Technology providers and solutions providers need to stay abreast of these evolving standards if they are to stay competitive.

Semantic Web technologies have been proposed as a means to enable interoperability for sensors and sensing systems in the context of SWE. Semantic Web technologies could be used in isolation or in augmenting SWE standards in the form of the Semantic Sensor Web (SSW). Semantic technologies can assist in managing, querying, and combining sensors and observation data. Thus allowing users to operate at abstraction levels above the technical details of format and integration, instead of working with domain concepts and restrictions on quality. Machine-interpretable semantics allows autonomous or semi-autonomous agents to assist in collecting, processing, reasoning about, and acting on sensors and their observations. Linked Sensor Data may serve as a means to interlink sensor data with external sources on the Web.

One of the primary outcomes of the SSW research is the Semantic Sensor Network (SSN) ontology (by [W3C Semantic Sensor Network Incubator Group](#)). This IoT ontology provides all the necessary semantics for the specification of IoT devices as well as the specifications of the IoT solution (input, output, control logic) that is deployed using these devices. These semantics include terminology related to sensors and observations, reusing the one already provided by the SSN ontology, and extended to capture also the semantics of devices beyond sensors - i.e. actuators, identity devices (tags), embedded devices, and of course the semantics of the devices and things that are observed by sensors, that change their status by actuators, that are attached to identity tags, etc. Furthermore, the ontology includes semantics for the description of the registered IoT solutions - i.e. input, output, control logic - regarding aligning and matching their requirements with the specifications and services of the registered devices.

6.9. IoT data visualisation

One of the challenges for the IoT industry is data analysis and interpretation. Big Data generated by the IoT devices is impractical if it cannot be translated into a language that is easy to understand, process and present a visual language. For this reason, IoT data visualisation is becoming an integral part of the IoT. Data visualisation provides a way to display this avalanche of collected data in meaningful ways that clearly present insights are hidden within this mass amount of information. This can assist us in the making fast, informed decisions with more certainty and accuracy than ever before. It is thus vital for business professionals, developers, designers, entrepreneurs and consumers alike to be

6. Data and Information Management in the Internet of Things

aware of the role that Visualization will and can play shortly. It is crucial to know how it can affect the experience and effectiveness of the IoT products and services.

7. IoT security and privacy

Concept of information security and its importance.

There are two approaches to the determination of the concept of "information security":

1. Information security — the status of the safety of information resources and the protection of the legitimate rights of the personality and society in the information sphere.

2. Information security is a process of support for confidentiality, integrity and accessibility of information.

Confidentiality: Ensuring access to information only to authorised users.

Integrity: Support of reliability and completeness of information and processing methods.

Accessibility: Ensuring access to information and related assets of authorised users as required.

The properties given above are fundamental bases in the sphere of protection and safety of information.

Safety of information — a status of the security of data in the case of which their confidentiality, accessibility and integrity are provided.

Safety of information is defined by absence of the unacceptable risk connected to information leakage on technical channels, unauthorised and inadvertent impacts on data and (or) on other resources of an automated information system used in the automated system.¹⁷⁶⁾

To understand what activities for the support of information security consist of, it is necessary to understand the value of three major concepts clearly: risk, threat and vulnerability.

The risk of information security - a possibility that this threat will be able to use the vulnerability of an asset or group of assets and by that will cause damage to the organisation.

The threat is a potential or real-life danger of making of any act (actions or inactivities) directed against the subject to protection (information resources) causing damage to the owner or user, which is shown it is in danger of distortion and losses of information.

Vulnerability is a shortcoming, the error in implementation which does possibly the unforeseen impact on system attracting failures in system operation is more often. Vulnerabilities are classified by a set of signs. One of the most important signs — harm which can be caused by the system, using vulnerability. Most often understand the specific mistake made in case of design or coding of the system as a vulnerability.

In case of the appearance of new information technologies and furthermore the whole information branches, there is a vast number of potential threats and vulnerabilities

7. IoT security and privacy

which shall be probed correctly. Indeed, the Internet of Things did not become an exception.¹⁷⁷⁾

The recent report of Gartner predicts that by 2020 20,4 billion devices will be connected to IoT, and at the same time will be joined every day by 5,5 million new devices. Besides, by 2020 more than a half of sizeable new business processes and systems will include the IoT component.

These digits are suprising and assume that standard protection the PC and anti-virus solutions will not be able to resist to future threats of cybersecurity on the attached devices IoT.

For the last few years, many widespread cyber attacks showed risks of the inadequate safety of IoT. Perhaps, the attack of "Stuxnet" aimed at the industrial programmable logic controllers (PLC) at the Iranian uranium enrichment plant became the most known. Experts read that Stuxnet destroyed up to 1000 centrifuges connected through broadband networks to the PLCs devices working under control of the Windows operating system at the PCs standard platforms.

In 2016 was many serious attacks directed to IoT devices. Mirai botnet became one of such attacks. This specific a bot network infected numerous IoT devices (first of all old routers and IP cameras) and then used them for superimposing of Dyn DNS provider utilising the DDoS-attack. The botnet of Mirai destroyed Etsy, GitHub, Netflix, Shopify, SoundCloud, Spotify, Twitter and some other the large websites. This piece of the malicious code used the devices using outdated versions of a kernel of Linux and relied on the fact that most users do not change names users/passwords by default on the devices.

Many companies reduce the costs of production, not including sufficient space for storage on the devices to provide updating of a kernel Linux. Because of it, kernels which include vulnerabilities work on many IoT devices. Vendors need to learn this lesson and to allow each device to update regularly kernels. Until this problem is solved, IoT devices will still suffer from the weight of exploits.

In November 2016¹⁷⁸⁾ cybercriminals closed heating of two buildings in the city of Lappeenranta, Finland. It was the DDoS-attack; in this case, the attack allowed heating controllers to reboot the system permanently, so heating was not made. As the temperature in Finland fell below zero at this time, this attack caused very unpleasant consequences.

Even if you take reasonable measures of the safety of IoT, your connected gadgets can be compromised by criminals. Last fall the DSN Dyn-Internet service provider got under the attack which broke access to favourite websites. Attackers could take under control a large number of the devices connected to the Internet, such as video recorders and cameras. These devices than were used for carrying out the attack.¹⁷⁹⁾

IoT gives the almost infinite opportunities for connection of our devices and the equipment. From the point of view of creativity, this field is widely opened, with the endless set of methods "to connect devices". It can become the ample platform for people with the innovative ideas, but also it concerns also malefactors. Therefore, IoT offers both new opportunities for development, and potential security concerns.

7.1.1. Types of vulnerabilities of IoT

As it was already told in the previous point, IoT is the problematic platform giving very ample opportunities not only for direct users but also and for violators. As well as by operation with any other information technologies, IoT includes the range of utterly different vulnerabilities, beginning from a human factor (inadvertent errors of maintenance), finishing with shortcomings of the firmware of devices. Indeed, to provide the due protection level, it is necessary to define and whenever possible to eliminate the highest possible number of such vulnerabilities.¹⁸⁰⁾

The first question concerns the problems connected to the safety of the web interfaces which are built in IoT devices which allow the user to interact with the device, but at the same time can allow the malefactor to get illegal access to the device. Specific vulnerabilities of safety which can lead to this problem include:

1. Feeble registration data by default - logins and passwords
2. The registration data displayed in a network traffic
3. Cross-Site Scripting (XSS)
4. SQL injection
5. Careless control of a session
6. Feeble settings of lock and deleting accounting entry.

It is also possible to select the specific area of vulnerabilities considering the ineffective mechanisms authenticating users of IoT/or bad mechanisms of authorisation. Specific vulnerabilities of safety which can lead to these problems include:

1. The absence of an optimum password policy
2. The absence of two-factor authentication
3. Unprotected recovery of the password
4. The absence of monitoring of the access by roles.

Vulnerabilities in network services which are used for access to the IoT device allowing the malefactor to get illegal access to the device or the related data should not be underestimated. Specific vulnerabilities of safety which can lead to this problem include:

1. Vulnerable services
2. Buffer overflow
3. Open ports through UPnP
4. Operational services of UDP
5. DoS and DDoS of the attack

The insufficient configuration of safety is relevant when users of the device have limited opportunities or cannot change the controls safety. The poor shape of security is apparent when the web interface of the device has no possibility of the creation of detailed user permissions or, for example, for forced use of reliable passwords. The risk with it is that the IoT device could be attacked easier, allowing illegal access to the device or data. Specific vulnerabilities of safety which can lead to this problem include:

1. The absence of the granular model of permission
2. The absence of monitoring of safety
3. The absence of journalizing of events of safety

7. IoT security and privacy

There is also a set of the vulnerabilities using shortcomings of mobile and cloudy interfaces. The range of these vulnerabilities is truly wide because it includes all types of vulnerabilities: the human factor expressing in carelessness and inadvertent errors, negligent attitude to a configuration of IoT devices, etc.¹⁸¹⁾

Weaknesses of physical security are relevant when the malefactor can get physical access to the data media and any data which are stored on this carrier. Deficiencies are also present when USB ports or other external ports can be used for access to the device with use of the functions intended for setup or service. It can lead to illegal access to the device or data.

7.2. Monitoring of vulnerabilities

Monitoring of information security became more and more comprehensive task. Thanks to trust relationships policies, administrators shall track a considerably large number of devices and platforms. And certainly also face the growing flow of the devices coming to a network through IoT. Temperature sensors and the connected engines, refrigerating aggregates and modules of energy management - all this as already more than once it was mentioned above, is a source of new and also well-known threats of information security and a receptacle of various vulnerabilities.

The more becomes potential vulnerabilities with IoT distribution, the more function of monitoring and detection of threats becomes essential. One of the most severe areas in this sphere is the growing use of cloud resources, at the same time the enterprises sometimes maintain several varieties of the public and private environment which generate own datasets of monitoring of safety. The growing number of the ending points connected to IoT ecosystems also has an adverse effect on safety.

The data volume of safety, added to the system, is only one side of a coin. The same level of readiness for safety, as the normal ending points controlled by the user does not always have those devices which make IoT. It can lead to the fact that IoT devices will cause suspicious traffic, having caused still a big need for monitoring of vulnerabilities. These data arrays mean the bigger number of the logs of safety necessary for scanning and activation of different security protections and also sources of assessment and processing.¹⁸²⁾

Today the zone of monitoring of network safety looks different than what it was only a few years ago. Administrators once knew where their territories begin and come to an end. There was a firewall, and there were accurately specific ending points. "Now the technology promoted to such an extent that you have a mobile phone, you have the virtual applications, you have the virtual machines, and you have cloud applications", - Chris Thomas, the strategist of Tenable Network Security told. "You have no set of certain networks any more with accurately certain boundaries for the administrator of safety. These problems, he added, "over time will only worsen because we have an Internet of things". Whenever network edges extend to envelop new locations, the new environments and new machines, the volume of the log of safety this, requiring revising, also grow.

When several separate networks suddenly get into one big interdependent group, administrators and specialists in information security should understand how safety is ensured and as the standard template of traffic looks. It also often means that monitoring of safety and vulnerabilities shall be quickly applied to places which were not under

7.3. Malware detection in IoT

attention earlier.¹⁸³⁾

In the previous subject of lectures types of the vulnerabilities widespread in IoT were considered, in same it is necessary to list the primary methods of preventing of threats of information security considering relevant vulnerabilities.

For neutralisation of the vulnerabilities connected to problem aspects of network interfaces I exist the following methods:

1. The initial setup, including the mandatory change of the password by default.
2. Support of reliable mechanisms of recovery of the password and information security about the user.
3. Support of the web interface is insensitive to XSS, SQLi or CSRF.
4. The password policy, regulating the complexity of passwords.
5. Support of lock of the accounting entry after a certain number of abortive attempts to log in.

To provide the necessary reliability of authentication and authorisation, used on IoT devices, the following methods will help:

1. Configuring of reliable password policies.
2. Support of granular monitoring of access if necessary.
3. Support for appropriate protection of registration data.
4. Implementation of two-factor authentication.
5. Safety of mechanisms of recovery of the password.
6. The organisation repeated authentication for sensitive functions of devices

Methods of the safety of network services:

1. Organization of access to necessary ports.
2. A configuration and use of services, not vulnerable to buffer overflow and the similar attacks.
3. A configuration and use of services, not vulnerable to DoS and DDoS - the attacks which exhaust resources of the system.
4. Use of UPnP and similar technologies for ensuring access to network ports or services.

The following measures are applied to the elimination of the vulnerabilities connected to use of cloud computing:

1. The organisation of the system of change of the password by default for new users of services.
2. Support of lock of accounting entries after a certain number of abortive attempts to log in.
3. Use of the cloudy web interfaces steady against XSS, SQLi or CSRF.
4. The organisation of the absence of leakage of registration data through cloud services.
5. Use of two-factor authentication if necessary.

For the elimination of threats from physical vulnerabilities the following methods are recommended:

1. Support of impossibility of easy deleting data media.
2. Support for encoding of the saved data.
3. Support of protection of USB ports or others of external ports.
4. Minimizing the number of external ports, such as USB, for the operation of a product.

7.3. Malware detection in IoT

In 2016 there was a row of incidents which attracted keen interest in the safety of IoT.

7. IoT security and privacy

Among them, there were record DDoS-attacks against the French hosting provider OVH and Dyn DNS provider of the USA. It is known that these attacks are launched using the massive botnet consisting of routers, IP cameras, printers and other devices.

Last year the world also learned about enormous a botnet, consisting of nearly five million routers. The German telecommunication giant Deutsche Telekom also faced hacking of the router after the devices used by clients of the operator were infected Mirai. Cracking did not stop on a network equipment: security concerns were also found in intelligent dishwashers of Miele and AGA furnaces. "Frosting on cake" is the BrickerBot worm who not just infected vulnerable devices as most of his "peers", but actually rendered them completely unserviceable.¹⁸⁴⁾

As representatives of the Kaspersky company told, they fixed not only the attacks arriving from the network equipment classified as home devices but also the hardware of an enterprise-grade without the knowledge of corporate owners.

"Even more disturbing is the fact that among all IP addresses from which there were attacks there were some which placed monitoring systems and/or device managements with corporate and protective affairs", - researchers say.¹⁸⁵⁾

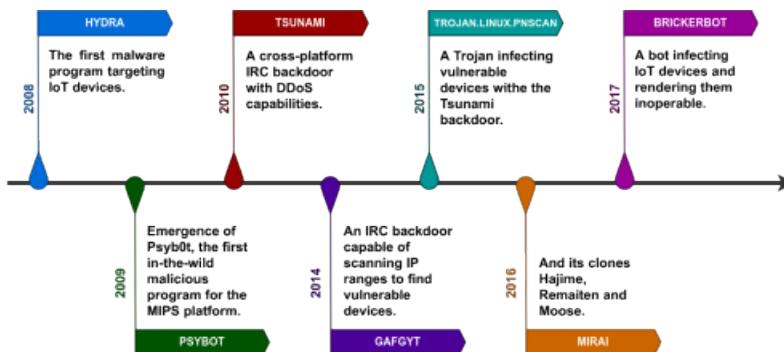


Figure 274: Timeline of the most famous malware for IoT.

Devices for sale in outlets, restaurants and gas stations belong to analysable; systems of digital TV broadcasting; systems of physical security and monitoring of access; and devices of monitoring of the environment.

Researchers also found malicious software infecting a monitoring system at the seismic station in Bangkok and also industrial programmable microcontrollers and management systems a supply in other places.

In baits, the attacks from China, Vietnam, Russia, Brazil and Turkey are found.

"The increasing number of the malicious applications intended for IoT devices and the related incidents of safety shows serious security of smart devices. 2016 I showed that these threats not only are conceptual but also are very real", - researchers say. "The existing competition in the market of DDoS induces cybercriminals to look for new resources for start of more and more powerful attacks".

Kaspersky recommends that devices did not allow access because of limits of their

7.3. Malware detection in IoT

local area network if it especially is not required for the use of the device. All network services which are not necessary also shall be disconnected. Passwords shall be by default changed and if they cannot be, then network services shall be disconnected if these passwords are used, or access to devices out of a local area network shall be turned off.

For detection of the aberrant behavior happening in the existing mobile environment (malicious software, a virus, a worm, etc.) were executed detection on the basis of signatures, detection on the basis of behavior and detection on the basis of the analysis. Tendencies of researches are generalized in the table below on the basis of their methods of detection and collected data:

Table 24: Malware detection techniques.

| Detection technique | Collected data | Description |
|----------------------------|--------------------------|--|
| Signature-based technique | Executable file analysis | Uses the readelf command to carry out static analysis on executable files using system calls |
| | Source code analysis | Uses the Android sandbox to carry out static/dynamic analysis on applications |
| | Packet analysis | Uses functions such as packet-preprocessing and pattern-matching to detect malware |
| | API call history | Collects system events of upper layers and monitors their API calls to detect malware |
| Behavior-based technique | System log data | Detects anomalies in terms of Linux kernels and monitors traffic, kernel system calls, and file system log data by users |
| | SMS, Bluetooth | Lightweight agents operating in smartphones record service activities such as usage of SMS or Bluetooth, comparing the recorded results with users average values to analyze whether there is intrusion or not |
| | Battery consumption | Monitors abnormal battery consumption of smartphones to detect intrusion by newly created or currently known attacks |
| | System call | Monitors system calls of smartphone kernel to detect external attacks through outsourcing |
| | Process information | Continuously monitors logs and events and classifies them into normal and abnormal information |
| Dynamic analysis technique | Data marking | Analyzes malware by carrying out static taint analysis for Java source code |
| | Data marking | Modifies stack frames to add taint tags into local variables and method arguments and traces the propagation process through tags to analyze malware |

Detection on the basis of signatures is the traditional method used for detection of malicious software in the environment of the PC. For determination of the signature static and dynamic methods are at the same time used. Static analysis is aimed at a code

7. IoT security and privacy

of a source and an object and analyzes codes without the actual start of the program. It decompiles the source code of malicious software for detection of the vulnerabilities arising in commands, instructions, etc. Dynamic analysis is a method of search of certain templates in memory leak, a traffic flow and a data stream in case of the actual start of the program. However application of this method to the mobile environment requires the large volume of memory, and service data of productivity are high for compliance of templates.

Signatures on the basis of technologies monitor the known threats. In case of computation all objects have attributes which can be used for creation of the unique signature. Algorithms can quickly and effectively scan an object to define its sign-code signature.¹⁸⁶⁾ When the solution provider for protection against malicious applications identifies an object as harmful, its signature is added to the database of the known malicious applications. These records may contain hundreds of millions signatures which identify harmful objects. This method of identification of harmful objects was the main method used by harmful products and remains the basic approach used by the latest firewalls, mail and network gateways.

The technology of detection of malicious applications on the basis of signatures has a row of advantages from which main thing is that it is well-known and clear - the very first anti-virus programs used this approach. It is also fast, simple in control and widely available. First of all, it provides good protection against many millions old, but nevertheless the active threats.

Check that the new file is harmful can be difficult and labour-consuming, and often the malicious application already develops by then. In the Annual report on the cyber security of Cisco 2017, it was set that 95% of files of malicious applications which they analysed were not even 24-hour that specifies fast "time for development". The time delay in the detection of new forms of malicious software does corporations vulnerable to severe losses.

The modern malicious software often strikes directly, being reduced for a short period. For example, "Puzzle" begins deleting files within 24 hours. HDDcryptor infected 2000 systems in San Francisco of the municipal transport agency before it was found. Therefore vulnerable to infection waiting for the signature to be very risky.

The other problem is that the modern malicious software can change the signature to avoid detection; signatures are created by a study of internal components of an object, and authors of malicious applications change these components, saving at the same time functionality and behaviour of an object.

There is a set of methods of conversion, including a swap of a code, renaming of registers, extension and abbreviation of code and also an insertion of a code of garbage or other constructions.

Detection on the basis of behavior is a method of detection of the status of invasion by the comparative analysis of the predetermined templates of the attack and behavior of process which occur in system. It is one of researches which receives the greatest attention because of limited detection of harmful behavior on the basis of signature detection recently. To find the abnormal templates, it generally monitors information on events which arises in such functions of the smartphone as memory use, contents of the SMS and consuming of the battery. Are often used detection on the basis of a

7.3. Malware detection in IoT

host (for direct monitoring of information in the device) and detection on a network (for information collection on a network). As detection on the basis of a host increases use of the battery and memory of the smartphone, the method of detection of data collection in the device and data transfer on the external analytical server is generally used. Besides, for fall forward of the analysis of dynamic data the method of machine learning is used. Therefore it is very important to select suitable functions for collection and to select a suitable algorithm of machine learning for exact detection.

Assessment of the malicious code and its behaviour in the process of execution is called dynamic analysis. The threat or malicious intention can also be estimated by static analysis which looks for dangerous opportunities in code and structure of an object.

In spite of the fact that the decision is not completely reliable, behavioral detection still allows technologies to reveal new and unknown threats in real time. Some examples of when the technology based on behavior succeeds when the systems of the signature do not work:

1. Protection against new and unimaginable types of the harmful attacks
2. Detection of a separate copy of the malicious software aimed at the person or the organization
3. Determination of malicious software in a certain environment when opening files
4. Obtaining exhaustive information on malicious software

There are several essential restrictions about which it is necessary to know. If the malicious application defines that it is launched in an isolated software environment, it will try to avoid detection, having reduced harmful actions. It is critical that "sandbox" remained undetectable, and most of them such is not.

It also requires time for the analysis of the behaviour of an object; while static analysis can be made in real time, dynamic analysis can enter latency while an object is implemented. Besides, many behavioural decisions are exclusively cloudy that can be a problem for some organisations. Ordinary technologies of "sandbox" have limited visibility and can estimate only interaction between an object and an operating system. Watching for 100% of actions which a harmful object can make, even when it delegates these actions to an operating system or other programs, OHO can estimate not only communication of malicious software with an operating system but also each command processed by the processor.

Expanded solutions for the detection of malicious applications watch and evaluate each code line executed by malicious software in a context. They analyse all requests for access to specific files, processes, connections or services. It includes each command executed at the level of an operating system or other programs which were caused including the low-level code hidden by rootkits.

The technology identifies all harmful or, at least, suspicious actions which when combining do very clear that the file is harmful before it is released in a network actually to execute any potentially dangerous behavior. ¹⁸⁷⁾

As well as detection of malicious applications according to the signature, and the behavioral method is important and has advantages. The best safety will be ensured due to use of both technologies. Too many security service specialists are misled by the sellers advertising firewalls of the next generation and other "modern" security protections. They do not understand that these "latest" products rely only on the ten

7. IoT security and privacy

years' approach based on signatures to detection of malicious applications which will miss evasive malicious applications and the attacks with zero-day.

7.4. IoT security protocols

As it was already repeatedly marked, IoT grows before the eyes, and together with it and the number of problems in the sphere of safety. For support of last, the complex of protocols of security is always necessary for the support of due protection of IoT devices.

The IoT protocols will play a vital role in the complete finite implementation of technology. They form a basis for a data stream between sensors and the outside world. They are necessary also for effective system operation and practical use of the MAC protocol and the appropriate routing protocol. For different domains several MAC protocols with access patterns, available to the user, to TDMA (without the conflicts) were offered, to CSMA (a low performance of traffic) and FDMA (without the conflicts, but requires additional diagrams in nodes). Any of them is not accepted as the standard, and with a large number of available devices search of a method of uniform functioning of all devices will be more and more problematic that of course will demand further researches.¹⁸⁸⁾

The personal sensor can fall out of an operation for several reasons therefore the network shall be self-setting up and allow routing with several ways. Routing protocols with several transitions are used on the mobile ad hoc networks and in terrestrial WSN. They are generally subdivided into three categories: oriented on the data-oriented on location and hierarchical, and again based on different application domains. The electric power is a pacing factor for the existing routing protocols. In case of IoT it is necessary to mark that the trunk will be available, and the number of transitions to scenarios with several transitions will be restricted. In such scenario the existing routing protocols shall be sufficient in implementation with little changes.¹⁸⁹⁾

There are some obvious things which can be made for integration of safety into IoT: the most obvious is support of the web interface of the device. Simple things, such as check that names of users and passwords were by default changed during initial setup very much help. And changes shall not allow uses of weak passwords. Perhaps, it is necessary to consider such measures as lock of the accounting entry after three-five abortive attempts of login. Attention shall be paid to passwords outside original changes of settings. Check of a network traffic for support of the fact that codes of an input do not go to cleartext is a reasonable step, and it also belongs to any diagrams of recovery of the password. Besides, for sensitive areas, such as accounting entries of the administrator, two-factor authentication can be required.

The web interface study for protection against the general attacks, such as Cross-Site Scripting, cross-website fake of requests and SQL injection also shall be carried out by the development of reliable protocols. These attacks are harmful if they are successful, but they are also relatively unaffected by the nature that does them preventable in case of successful configuring of protocols.¹⁹⁰⁾

Many protocols were developed at all levels of a stack of International Organization for Standardization (ISO) to ensure the functioning of IoT devices. From exchange protocols messages, such as protocol of restrictions (CoAP), to highly expanded routing protocols, such as routing protocol for low-power and Lossy Networks (RPL). The importance for the understanding of these protocols is that they were developed taking into account

saving the electric power and also with low requirements to computation and memory that, certainly, is extremely important by operation with IoT.

Solutions of safety by IP

Internet exchange of keys (IKEv2)/IPsec and the protocol of identification of a host (HIP) is at the level or higher than the level of a network in the OSI model. Both protocols can execute the authenticated exchange of keys and set up the IPsec conversion for a safe delivery of payload capacity.

The expanded authentication protocol (EAP) represents the authentication framework supporting several methods of authentication. EAP works directly on the level of the transmission channel and, therefore, does not require IP deployment. It supports repeated detection and repeated transmission but does not allow fragmentation of packets. The protocol for the support of authentication for network access (PANA) represents the transport layer of the network layer for EAP which provides authentication of network access between clients and network infrastructure. In the terms, EAP PANA is the bottom level of EAP by UDP which is executed between the peer-to-peer EAP node and an authenticator of EAP.

On the Internet and, therefore, in IoT, safety at the network layer is ensured by a set of IP safety (IPsec). IPsec in the transport mode provides open protection utilizing services of authentication and repeated protection in addition to confidentiality and integrity. By operation at the network layer of IPsec, it can be used with any transport layer protocol, including TCP, UDP, HTTP and CoAP. IPsec provides confidentiality and integrity of payload capacity of IP with use of the Encapsulated Security Payload (ESP) protocol and integrity of title of IP plus payload capacity with use of the protocol of title of authentication (AH). IPsec is mandatory in the IPv6 protocol that means that already IPv6 devices by default have the support of IPsec which can be included at any time. Being the solution of the network layer, security polices of IPsec are shared by all applications started by the specific machine.

However, being mandatory in IPv6, IPsec is one of the most suitable options of the safety of E2E in IoT¹⁹¹⁾:

1. The limited node uses 6LoWPAN for addressing and CoAP as protocol application layer.
2. The easy node uses IPv6 for addressing and HTTP as the protocol of level of application.
3. The limited node is already authenticated through the gateway (GW).
4. There is a trust relationships policy providing safe communication in the limited network.
5. The gateway is an authorised representative.

It is possible that the finite node will set up a connection of a transmission mode of IPsec-ESP with the IoT device, moving processes of generation of the main session and authentication from the IoT node on the entrusted gateway. The ESP mode which provides data encryption and authentication allows setting open safe connection between two peer-to-peer nodes by encoding of payload capacity, having left IPv6 titles untouched. Cryptographic keys are generated and exchange according to the IKE protocol with use of the diagram of exchange of keys of Elliptic Curve Diffie Hellman. Employing these mechanisms of the logician of key generation and authentication moves from the IoT node to the relevant GW, thereby exempting the IoT device from the computing loading connected to the generation of cryptography data.

7. IoT security and privacy

WirelessHART

WirelessHART is rather safe protocol and provides several protection levels. All traffic is protected, payload capacity is ciphered, and all messages undergo authentication, as from single-hop-basis, and at the end. WirelessHART requires that all devices were supplied with a secret key of Join and also the network identifier to join a network.

WirelessHART, though is limited by a resource, represents a bidirectional network of rather powerful devices and the central manager of a network and the controller has. WirelessHART, now the single WSN standard developed first of all for automation of industrial production and control is well developed for other aspects, except safety. The provided safety extends according to specifications of WirelessHART.

For support of safe communication the set of different security keys is used. The new device is supplied with Join key before he tries to join a wireless network. The key of combining is used for authentication of the device for the specific WirelessHART network. After the program successfully joined a network, the manager of a network will provide it the correct keys of a session and a network for further communication. The actual creation and key management is processed by the manager of safety of "Plant wide" who is not specified to WirelessHART, but keys are distributed to network devices by means of the manager of a network. The key of a session is used by the network layer for authentication of open communication between two devices (for example, the field device and the gateway). Different keys of a session are used for each conjugate communication (for example, the Field device for the gateway, the Field device for the manager of a network, etc.). At the level of transmission channel the network key for authentication of messages on the basis of one transition is used. The known network key is used when the device tries to be connected to a network that is before it received the correct network key. Keys are turned on the basis of procedures of safety of installation of automation of process.

Three types of keys are used: Master key, a key of Link and Network key. We will compare a Master key with a connection key in WirelessHART, and it is necessary for the association to a network.¹⁹²⁾ The key of the link is used for open encoding and will provide the maximum level of safety at the price of higher requirements to storage. The network key is distributed between all devices and, thus, provides a lower level of safety though taking into account smaller requirements to storage in devices. All keys can be set at the plant or are transmitted from the trust centre (which is in the network coordinator) or by air, or through the physical interface. For commercial application the confidential center can control association of new devices and periodically update a network key.

6LoWPAN

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) is the name of a concluded working group in the Internet area of the IETF. The 6LoWPAN concept originated from the idea that "the Internet Protocol could and should be applied even to the smallest devices, and that low-power devices with limited processing capabilities should be able to participate in the Internet of Things".

The 6LoWPAN group has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4 based networks. IPv4 and IPv6 are the work horses for data delivery for local-area networks, metropolitan area

7.4. IoT security protocols

networks, and wide-area networks such as the Internet. Likewise, IEEE 802.15.4 devices provide sensing communication-ability in the wireless domain. The inherent natures of the two networks though are different. Requirements to safety 6LoWPAN RFC4919 defines the requirements list of safety for 6LoWPAN which are generally directed to protection of messages against ultimate users to a network of sensors. Requirements list:

1. Confidentiality: only authorized users can get information access
2. Authentication: data undertake only from the untrusted sources
3. Integrity: data retrieved remain invariable in transmission time
4. Freshness: consider like this, and a key not to reproduce old messages
5. Accessibility: guarantees that data can be available by transmission
6. Reliability: operation support, despite abnormal conditions
7. Fail safety: provides the acceptable security level even in case some nodes
8. Energy efficiency: reduce control expenses to increase network service life.
9. Support: possibility of dissemination of different information.

These requirements require a combination of different systems of protection.

Cryptography methods

Ciphering messages before transmission, cryptographic solutions are aimed at three-fold protection: only of the authenticated user who has the correct key can decrypt and read messages; contents of integrity shall not change in transmission time and confidentiality. Nobody can understand the message without the key.

Cryptography techniques for 6LoWPAN shall be developed more for adaptation to the prevailing restrictions in devices 6LoWPAN, such as low power and low calculation capacity. It is connected to the fact that not optimized mechanisms of cryptography will consume more resources and, therefore, to reduce lifetime of a network. The key used in cryptography techniques also shall not be too short; otherwise malefactors will be able easily to break. As 6LoWPAN is a combination of WSN and Internet, it is natural to use these two network cryptography mechanisms to safety of this network. WSN uses AES (Advanced Encryption Standard) for support of level of communication link with several operation modes, the majority of which do not provide integrity. For protection of open safety of the network layer of IPsec (Internet Protocol Security) it is used with the transport and tunnel modes. Earlier the cryptography mechanism with public key was read too heavy for application in WSN. Nevertheless, the last researches showed how to combine RSA (asymmetrical encoding of Rivest-Shamir-Adelman) and ECC (an elliptic curve of cryptography) with several modes for adaptation to network scenarios.

Exchange of a key - one more problem which shall be considered in case of implementation of protocols. For exchange of keys on a network exchange of keys of the Internet with IPsec (protection of the internet protocol) is offered. Nevertheless, exchange of keys of the Internet is not considered the acceptable decision because of its large messages of signaling that is unsuitable for small packet size 802.15.4 and the requirement to energy efficiency. WSN used several key methods of distribution, such as redistribute and key pool; however, they lack scalable opportunities. It is also necessary to analyse threat for a key at the time of boot strap loading when the opponent is among other nodes, without requiring authentication.

Though researches show the considerable improvement in the use of cryptography for 6LoWPAN, the network still should overcome a set of problems. The cryptography is

7. IoT security and privacy

also used only in case of protection 6LoWPAN from the external attacks but has no opportunity to find and eliminate the internal attacks. It is connected to the fact that the cryptography cannot find malefactors using legal keys but behaves with malice aforesight. Thus, the network safety using only cryptography is feeble in case of the attacks directed to network productivity, such as DoS or the battery, and the resource attacks, such as jamming and switch-off.

Therefore the cryptography cannot ensure complete safety for 6LoWPAN. It is necessary to implement IDS for monitoring of any harmful behaviour of a network to prevent the early attacks of safety to reduce its consequences. IDS - an effective method of detection of any malefactor which bypasses the line of protection of cryptography and ensures normal network functioning.

The cryptography combination as a first line and IDS as protection of the second line can protect a network from the majority of threats. The missions of IDS consist of tracing and giving the alarm about any possible risks and transferring them to cryptography to restart the process of manipulation for the elimination of malefactors. IDS can cope with all threats.

Problems of detection of invasions of IPv6.

IDS from IPv6 protects the boundary router from any threats sending packets from IPv6 to WSN to begin WSN attack. It is easy to solve the majority of problems in parts of WSN regarding IPv6 because the boundary router usually is implemented with the strong by protection and not - resource restriction, and, besides, the threats proceeding from the IPv6 network it is, much less, than threats in a sensor network, For Example, the boundary router is the most suitable line item to define, there is a network where exactly. The problem with extraction of functions is also not restricted, as regarding WSN, because of the high throughput of the boundary router.¹⁹³⁾ The single problem on which it is necessary to concentrate is to select the suitable IDS methods for early detection and detection of threats.

Again three types of methods can be applied: misuse, anomaly and specification. The direction of misuse is still not favorable because attack signatures are not defined. There are three methods: Anderson Darling algorithm, an algorithm of an entropy and the PAT calculator (the predetermined types of attacks) for detection of abnormal behavior. The selected function of data are cards with disks from overflowing preventing algorithms when queues are filled. To reduce the speed of false alarm, they bring the found these anomalies in the qualifier of templates which checks the predetermined attack type on the saved buffer. The threshold is also selected for safety warning generation as soon as it is found which will be transferred by the qualifier. This system requires many computing loadings with three checking modules and other appropriate part to reduce detection speed. The author did not explain why they decided to analyze only data which are discarded from the buffer. Doing it, they probably assumed that data which were transferred to the buffer are harmless while in fact there is no warranty. Nevertheless, the main architecture of this system can be applied with use of different methods of detection to the best decision.

7.5. IoT privacy

Protecting consumer privacy becomes increasingly tricky as the IoT becomes more prevalent. More devices are connected to different types of devices and this increase in

connectivity and data collection results in less control. Both controls of data and control of the very devices that are connected are at stake.

Control can be lost if someone hacks into the smartphone or computer acting as a remote for the other devices. In the case of computers and smartphones, this hacking can be done remotely and often undetected. Smartphones, just like computers, carry an enormous amount of personal information about their owners. They often link to bank accounts, email accounts, and in some cases household appliances. Stolen data can result in serious problems. Vehicles contain many computers that control their function. Initially, these computers could not be hacked into. With the increased connectivity of the IoT, however, vehicles are now at risk due to being connected to the Internet.

In another sense, control can be lost as more and more companies collect data about users. This data often paints a detailed picture of individual users through the collection of activities online. Everything you search, all of your activities online, are being tracked by companies that use that data.¹⁹⁴⁾ These companies often use the data to improve the user's experience, but they also use this data to sell users products or sell to other companies who sell users products.

Innovation in this realm means that companies must alter the privacy policies that are in place as well as how they interact with these devices. Companies will need to take another look at the policies that they have in place to ensure that consumers are offered opportunities to access and control their data. Consumers will become increasingly aware of the privacy implications of this level of connectivity through interaction with the IoT and exposure to the policies that companies provide to them.

Frank Pasquale, law professor and EPIC advisory board member¹⁹⁵⁾ discusses privacy concerns related to the IoT in a May 2014 Pew Research Report. Pasquale states that the expansion of the IoT will result in a world that is more "prison-like" with a "small class of 'watchers' and a much broader class of the experimented upon, the watched." In another article, he reinforces the idea that the IoT "will be a tool for other people to keep tabs on what the populace is doing."

EPIC President, Marc Rotenberg, explains in the Pew Research Report that the problem with the IoT is that "users are just another category of things," and states that this "is worth thinking about more deeply about in the future."

There are many actual issues with IoT privacy, and all of them must be must be in detailed explored, but here are general ways **IoT developers can improve IoT privacy**¹⁹⁶⁾:

Minimize data acquisition: Software architects should look at the frequency and type of data collected in the context of the application and should not collect more data than the task requires. The platform should control which data an application receives.

Minimize the number of data sources: Aggregation of data from multiple sources allows malicious parties to identify sensitive personal information of an individual that could lead to privacy violations.

Minimize raw data intake: Raw data could lead to secondary usage and privacy violation. Therefore, IoT platforms should consider converting or transforming raw data into secondary context data.

7. IoT security and privacy

Minimize knowledge discovery: IoT applications should discover only the knowledge necessary to achieve their primary objectives. For example, if the objective is to recommend food plans, the app should not attempt to infer users' health status without their explicit permission.

Minimize data storage: Raw data should be deleted once a secondary context is derived.

Minimize the data retention period: More extended retention periods give malicious parties more time to breach and exfiltrate data.

Support hidden data routing: To make it more difficult for internet activities to be traced back to the users, this guideline suggests that IoT applications should support and employ an unknown routing mechanism.

Anonymize data: Remove personally identifiable information (PII) before the data gets used by IoT applications so that the people described by the data remain anonymous.

Encrypt data communications: Typically, device-to-device communications are encrypted at the link layer using special electronic hardware included in the radio modules. Gateway-to-cloud communication is generally secured through HTTPS using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

Encrypt data during processing: Sometimes the party processing the data should not be able to read the data or the computational results. Process data while they are in encrypted form. For example, homomorphic encryption is a form of encryption that allows computations to be carried out on cypher-text, thus generating an encrypted result that, when decrypted, matches the result of operations performed on the plain-text.

Encrypt data in storage: Encrypted data storage reduces any privacy violations due to malicious attacks and unauthorised access.

Reduce data granularity: IoT applications should request the minimum level of granularity that is required to perform their primary tasks. A higher level of granularity could lead to secondary data usage and eventually privacy violations. For example, location can be coarse, based on cell tower location or fine, based on the address.

Query answering: Raw data can lead to identification and privacy violations due to secondary usage. Instead of providing a numeric response to a query a relative scale, e.g. 1 - 5 should be used.

Block repeated queries: Query responses should block multiple queries that maliciously could discover knowledge that violates user privacy, such as analysing intersections of multiple results.

Distribute data processing: Distributed data processing avoids centralized large-scale data gathering and exfiltration.

Distribute data storage: Distributed data storage reduces any privacy violation due to malicious attacks and unauthorised access. It also reduces privacy risks due to unconsented secondary knowledge discovery.

Knowledge discovery based on aggregated data: New knowledge, such as the visitors to the park were young students during a time period, is sufficient for a gift shop to perform

7.6. Privacy preservation

time series sales analysis. But the exact timing of their movement is not necessary.

Aggregate geography-based data: Geographic data should be aggregated within boundaries. For example, how many electric vehicles are in use in each city should not store details about individual vehicles.

Aggregate data based on the time period: Energy consumption of a given house can be acquired and represented in aggregated form as 160 kWh per month instead of gathering energy consumption daily or hourly.

Aggregate data based on category: Aggregating based on a category that meets the needs of the analysis rather than exact data prevents secondary use. For example, categorising a household's energy use in the range of 150 - 200 kWh instead of exact usage.

Disclose information to users: Data subjects should be adequately informed whenever data they own is acquired, processed or disseminated.

Apply controls: It is the software architects' responsibility to consider what kind of controls are useful to data owners, especially when data owners are not knowledgeable. Some of the considerations: 1) data granularity; 2) anonymisation technique; 3) data retention period; 4) data dissemination.

Log events: Logging of events during all phases will allow both internal and external parties to examine what happened in the past to make sure a given system performed as promised.

Perform regularly audits: Systematic, independent audits and examination of the logs, procedures, processes, hardware and software specifications should regularly be performed. Outside parties should be bound by non-disclosure agreements.

Make apps open source: Wherever possible IoT applications should be made available under an open-source license so that outside parties can review the code and compliance demonstrated.

Use data flow diagrams: Data flow diagrams used by unified modelling language will allow interested parties to understand the data flows of a given IoT application and how data is treated for a demonstration of compliance.

Get IoT apps certified: Certifications given by a neutral authority will add trustworthiness to IoT applications.

Use industry standards: Industry-wide standards such as AllJoyn and the AllSeen Alliance typically inherit security measures that would reduce some privacy risks.

Comply with policies and regulations: Adherence to policies, laws, and regulations such as ISO 29100, OECD privacy principles and the European Commission's rules on the protection of personal data will reduce privacy risks.

7.6. Privacy preservation

Privacy legislation

Today, primarily the principles of notice, consent, access and security are enforced,

7. IoT security and privacy

e.g. in e-commerce and online advertising. Privacy legislation also touches some mature technologies which are part of the IoT evolution: RFID and camera networks have received much attention in the past. Recent legislation efforts have focussed on data protection in cloud computing and adequate protection of web users against tracking. 197)

However, already today the level of privacy protection offered by legislation is insufficient, as day-to-day data spills and unpunished privacy breaches indicate. The Internet of Things will undoubtedly create new grey areas with ample space to circumvent legislative boundaries.

First, most pieces of legislation centre around the fuzzy notion of Personally Identifiable Information (PII). However, efforts towards a concise definition of what constitutes PII (e.g. by enumerating combinations of identifying attributes) are quickly deprecated as new IoT technologies unlock and combine new sets of data that can enable identification and make it increasingly difficult to distinguish PII from non-PII.

Second, timeliness of legislation is a constant issue:

E.g. tracking of web-users has been used for many years before the European Commission passed a law against it in early 2011. With the IoT evolving fast, legislation is bound to fall even farther behind. An example is Smart Meter readings, which already allow inferring comprehensive information about people's lifestyle.

Third, already today, many privacy breaches go unnoticed. In the IoT, awareness of privacy breaches among users will be even lower, as data collection moves into everyday things and happens more passively. The legislation, however, is often the only response to public protests and outcries that require awareness of incidents in the first place.

Finally, the economics of privacy are still in favour of those in disregard of privacy legislation. On the one side, development of PETs, enforcement and audits of privacy-protection policies is expensive and can limit business models. On the other hand, violations of privacy legislation either go unpunished or result only incomparably small fines, while public awareness is still too low to induce excessive damage to public reputation. Thus, disregard of privacy legislation, as, e.g. Google deliberately circumventing Safari's user tracking protection, seems profitable. Over this incident, Google paid a record fine of \$22.5 Million in a settlement with the Federal Trade Commission (FTC), but it is conceivable that the earnings more than compensated.

It will be a significant challenge to design a unified enduring legislative framework for privacy protection in the Internet of Things, instead of passing quickly outdated pieces of legislation on singular technologies. Success will undoubtedly require a comprehensive knowledge of the technologic basis of the IoT and its ongoing evolution. The key, however, will be a deep understanding of existing and lingering new threats to privacy in the IoT – these threats are what legislation needs to protect against, ultimately.

7.7. IoT privacy preservation threats

Identification

Identification denotes the threat of associating a (persistent) identifier, e.g. a name and address or a pseudonym of any kind, with an individual and data about him. The threat thus lies in associating an identity to specific privacy violating context, and it also enables

7.7. IoT privacy preservation threats

and aggravates other threats, e.g. profiling and tracking of individuals or combination of different data sources.

The threat of identification is currently most dominant in the information processing phase at the backend services of our reference model, where vast amounts of information are concentrated in a central place outside of the subject's control.

First, surveillance camera technology is increasingly integrated and used in non-security contexts, e.g. for analytics and marketing. As facial databases (e.g. from Facebook) become available also to non-governmental parties like marketing platforms, automatic identification of individuals from camera images is already a reality.

Second, the increasing (wireless) interconnection and vertical communication of everyday things, opens up possibilities for identification of devices through fingerprinting. It was recognised already for RFID technology that individuals can be identified by the aura of their things.

Third, speech recognition is widely used in mobile applications, and vast databases of speech samples are already being built. Those could potentially be used to recognise and identify individuals, e.g. by governments requesting access to that data. With speech recognition evolving as a powerful way of interaction with IoT systems and the proliferation of cloud computing for processing tasks, this will further amplify the attack vector and privacy risks.

Identity protection and, complementary, protection against identification is a predominant topic in RFID privacy but has also gained much attention in the areas of data anonymisation, and privacy-enhancing identity management. Those approaches are difficult to fit the IoT: Most data anonymisation techniques can be broken using auxiliary data, that is likely to become available at some point during the IoT evolution. Identity management solutions, besides relying heavily on expensive crypto-operations, are mostly designed for very confined environments, like enterprise or home networks and thus tricky to fit the distributed, diverse and heterogeneous environment of the IoT. Approaches from RFID privacy due to similarities in resource constraints and numbers of things are the most promising. However, those approaches do not account for the diverse data sources available in the IoT as e.g. camera images and speech samples.

Localization and Tracking

Localization and tracking is the threat of determining and recording a person's location through time and space. Monitoring requires identification of some kind to bind continuous localisations to one individual. Already today, tracking is possible through different means, e.g. GPS, internet traffic, or cell phone location. Many concrete privacy violations have been identified related to this threat, e.g. GPS stalking, the disclosure of private information such as an illness, or generally the uneasy feeling of being watched. However, localisation and tracking of individuals is also an important functionality in many IoT systems.¹⁹⁸⁾ These examples show that users perceive it as a violation when they don't have control over their location information, are unaware of its disclosure, or if the information is used and combined in an inappropriate context.

In the immediate physical proximity, localisation and tracking usually do not lead to privacy violations, as, e.g. anyone in the immediate surrounding can directly observe the subject's location. Traditionally, localization and monitoring thus appear as a threat mainly in the phase of information processing, when locations traces are built at

7. IoT security and privacy

backends outside the subject's control. However, the IoT evolution will change and aggravate this threat in three ways:

First, we observe an increasing use of location-based services (LBS). IoT technologies will not only support the development of such LBS and improve their accuracy but also expand those services to indoor environments, e.g. for smart retail.

Second, as data collection becomes more passive, more pervasive and less intrusive, users become less aware of when they are being tracked and the involved risks. Third, the increasing interaction with smart things and systems leaves data trails that not only put the user at risk of identification but also allow to track his location and activity, e.g. to swipe an NFC-enabled smartphone to get a bus ticket or using the cities' smart parking system. With these developments, the threat of localization and tracking will also appear in the interaction phase, making the subject trackable in situations where he might falsely perceive physical separation from others, e.g. by walls or shelves, like privacy.

Research on location privacy has proposed many approaches that can be categorized by their architectural perspective into client-server, trusted third party, and distributed/peer-to-peer. However, these approaches have been mostly tailored to outdoor scenarios where the user actively uses an LBS through his smartphone. Thus, these approaches do not fit without significant modifications to the changes brought about by IoT. The main challenges we identify are awareness of tracking in the face of passive data collection, control of shared location data in indoor environments, and privacy-preserving protocols for interaction with IoT systems.

Profiling

Profiling denotes the threat of compiling information dossiers about individuals to infer interests by correlation with other profiles and data. Profiling methods are mostly used for personalization in e-commerce (e.g. in recommender systems, newsletters and advertisements) but also for internal optimization based on customer demographics and interests. Examples, where profiling leads to a violation of privacy violation, are price discrimination, unsolicited advertisements, social engineering, or erroneous automatic decisions, e.g. by Facebook automatic detection of sexual offenders. Also, collecting and selling profiles about people as practiced by several data marketplaces today is commonly perceived as a privacy violation. The examples show that the profiling threat appears mainly in the dissemination phase, towards third parties, but also towards the subject itself in the form of erroneous or discriminating decisions.

Existing approaches to preserve privacy include client-side personalization, data perturbation, obfuscation and anonymization, distribution and working on encrypted data. These approaches can be applied to IoT scenarios but must be adapted from the usual model that assumes a central database and account for the many distributed data sources which are expected in the IoT. It will require considerable efforts for recalibration of metrics and redesign of algorithms, as, e.g. recent work in differential privacy for distributed data sources shows. After all, data collection is one of the central promises of the IoT and the primary driver for its realization. We thus see the biggest challenge in balancing the interests of businesses for profiling and data analysis with users' privacy requirements.

Privacy-violating interaction and presentation

7.7. IoT privacy preservation threats

This threat refers to conveying private information through a public medium and in the process disclosing it to an unwanted audience. It can be loosely sketched as shoulder-surfing but in real-world environments.

Many IoT applications, e.g. smart retail, transportation, and healthcare, envision and require substantial interaction with the user. In such systems, it is imaginable that information will be provided to users using smart things in their environment, e.g. through advanced lighting installations, speakers or video screens. Vice versa, users will control systems in new intuitive ways using the things surrounding them, e.g. moving, touching and speaking to smart things. However, many of those interaction and presentation mechanisms are inherently public, i.e. people in the vicinity can observe them. It becomes a threat to privacy when private information is exchanged between the system and its user. In smart cities, e.g. a person might ask for the way to a specific health clinic. Such a query should not be answered, e.g. by displaying the way on a public display nearby, visible to any passers-by. Another example is recommendations in stores that reflect private interests, such as specific diet food and medicine, movies or books on precarious topics. Due to its close connection to interaction and presentation mechanisms, the threat of privacy-violating interactions and presentation appears primarily in the homonymous phases of our reference model.

Since such advanced IoT services are still in the future, privacy-violating interactions have not received much attention from researchers. Interaction mechanisms are however crucial to usable IoT systems and privacy threats must consequently be addressed. We identify two specific challenges that will have to be solved:

First, we need means for automatic detection of privacy-sensitive content. It is easily imaginable that the provisioning of content and rendering it for the user are handled in two steps by two different systems: E.g. company A generates recommendations for customers of a store, which are then delivered to the customer by company B's system: either by special lighting and the use of speakers or through a push to his smartphone.

Second, with the previous point in mind, scoping will be necessary, i.e. how can we scope public presentation medium to a specific subgroup of recipients or a specific physical area? This approach would prove useful to support users, which have no smartphone (or any other device providing a private channel for interactions and presentations). However, it will be difficult to determine the captive audience of a particular presentation medium accurately, separate the intended target group and adjust the scope accordingly. E.g. what if the target user is in the midst of a group of people?

Applications for privacy-preserving pervasive interaction mechanisms are, e.g. smart stores and malls, smart cities and healthcare applications. Here, it would indeed be an achievement to provide similar levels of privacy as people would expect in the contexts of their everyday conversations, i.e. interactions with their peers.

Lifecycle transitions

Privacy is threatened when smart things disclose private information during changes of control spheres in their lifecycle. The problem has been observed directly about compromising photos and videos that are often found on used cameras or smartphones – in some cases 'disturbing' data has even been spotted on 'new' devices. Since privacy violations from lifecycle transitions are mainly due to the collected and stored information, this threat relates to the information collection phase of our reference model.

7. IoT security and privacy

Two developments in the IoT will likely aggravate issues due to the lifecycle of things. First, smart things will interact with some persons, other things, systems, or services and amass this information in product history logs. In some applications, such data is highly sensitive, e.g. health-data collected by medical devices for home-care. But also the collection of simple usage- data (e.g. location, duration, frequency) could disclose much about the lifestyle of people. Already today, detailed usage logs are maintained for warranty cases in TV sets, notebooks or cars. Second, as exchangeable everyday things such as light bulbs become smart, the sheer numbers of such things entering and leaving the personal sphere will make it increasingly difficult to prevent disclosure of such information.

Despite obvious problems with the lifecycle of today's smartphones, cameras, and other storage devices this threat has not been adequately addressed. The lifecycle of most consumer products is still modelled as buy-once-own- forever, and solutions have not evolved beyond a total memory wipe (e.g. before selling a phone) or physical destruction (e.g. before disposal of a hard drive). Smart things could, however, feature a much more dynamic lifecycle, with things being borrowed, exchanged, added and disposed of freely.

We thus identify the requirement for flexible solutions that will undoubtedly pose some challenges: Automatic detection of lifecycle transitions of a smart thing will be required to implement suitable privacy lifecycle management mechanisms. E.g. a smart rubbish bin could automatically cleanse all items in it from private information, such as medical prescriptions on a smart pillbox. It will be difficult, though, to automatically distinguish between different lifecycle transitions as, e.g. lending, selling or disposing of an item and taking the appropriate action. Specific lifecycle transitions, e.g. borrowing a smart thing, will require locking private information temporarily, e.g. the readings of a vital signs monitor. Once the device has returned to its original owner, the private data can be unlocked, and the original owner can continue to use it seamlessly.¹⁹⁹⁾

Inventory attack

Inventory attacks refer to the unauthorised collection of information about the existence and characteristics of personal things. One evolving feature of the IoT is interconnection. With the realisation of the All-IP and end-to-end vision, smart things become queryable over the Internet. While things can then be queried from anywhere by legitimate entities (e.g. the owner and authorised users of the system), non-legitimate parties can query and exploit this to compile an inventory list of things at a specific place, e.g. of a household, office building, or factory. Even if smart things could distinguish legitimate from illegitimate queries, a fingerprint of their communication speeds, reaction times and other unique characteristics could potentially be used to determine their type and model. With the predicted proliferation of wireless communication technology, fingerprinting attacks could also be mounted passively, e.g. by an eavesdropper in the vicinity of the victim's house.

The impact of new technologies on this threat is not yet clear. On the one hand, we expect the diversification of technologies in the IoT as more and more different things become smart. Diversification increases the attack vector for fingerprinting, as, e.g. observed with the many diverse configurations of web browsers. On the other hand, at some point in time, we expect the establishment of specific standards for communication and interaction that could reduce such differences.

Manifold concrete privacy violations based on inventory attacks are imaginable or have

7.7. IoT privacy preservation threats

happened. First, burglars can use inventory information for targeted break-ins at private homes, offices and factories, similar to how they already use social media today to stake out potential victims. Note that a comprehensive inventory attack could then also be used to profile the anti-burglar system down to every last presence sensor. Second, law enforcement and other authorities could use the attack to conduct (unwarranted) searches. Third, private information is disclosed by the possession of specific things, such as personal interests (e.g. books, movies, music) or health (e.g. medicine, medical devices). Fourth, efforts for industrial espionage can be complemented through an inventory attack, as noted by Mattern.

Radomirovic and Van Deursen have recognised the danger of profiling through fingerprinting in the context of RFID. However, with RFID the problem is at a much more local scope as RFID tags can be read only from a close distance and queries are mostly restricted to reading the tag's identifier. As analysed above, the problem will aggravate in the IoT evolution as the attack vector is significantly increased by increasing proliferation of wireless communications, end-to-end connectivity, and more sophisticated queries. To thwart inventory attacks in the IoT, we identify the following two technical challenges: First, smart things must be able to authenticate queries and only answer to those by legitimate parties to thwart active inventory attacks through querying. Research in lightweight security provides useful approaches for authentication in resource-constrained environments. Second, mechanisms that ensure robustness against fingerprinting will be required to prevent passive inventory attacks based on the communication fingerprint of a smart thing. Inventory attacks will undoubtedly be difficult to counter. The fact that the use of PETs, though meant to protect privacy, can make fingerprinting even easier, leaves hiding in the (privacy-ignorant) masses currently as the most viable but suboptimal solution. However, an IoT system that discloses comprehensive information about its owner's possessions is not likely to gain acceptance.²⁰⁰⁾

Linkage

This threat consists in linking different previously separated systems such that the combination of data sources reveals (truthful or erroneous) information that the subject did not disclose to the previously isolated sources and, most importantly, also did not want to reveal. Users fear poor judgement and loss of context when data that was gathered from different parties under different contexts and permissions is combined. Privacy violations can also arise from bypassing privacy protection mechanisms, as the risks of unauthorised access and leaks of private information increases when systems collaborate to combine data sources. A third example of privacy violations through linkage of data sources and systems is the increased risk of re-identification of anonymized data. A common approach towards protecting privacy is working on anonymized data only, but the act of combining different sets of anonymous data can often enable re-identification through unforeseen effects. The examples show that the threat of linkage primarily appears in the information dissemination phase.

The threat of linkage will aggravate the IoT evolution for two main reasons. First, horizontal integration will eventually link systems from different companies and manufacturers to form a heterogeneous distributed system- of-systems delivering new services that no single system could provide on its own. Successful collaboration will above all require an agile exchange of data and controls between the different parties.²⁰¹⁾ However, as horizontal integration features more local data flows than vertical integration it could provide a way to enhance privacy. Second, the linkage of

7. IoT security and privacy

systems will render data collection in the IoT even less transparent than what already is expected from the predicted passive and unintrusive data collection by smart things.

Threats from linking different systems and information sources are not entirely new. They can already be observed in the domain of online social networks (OSN) and their applications. However, this involves only two parties (i.e. the OSN and the third party application), while the IoT is expected to feature services that depend on the interaction and collaboration of many coequal systems. Here, we identify three technical challenges for privacy-enhanced systems-of-systems: First, transparency about what information system-of-systems shares with whom is crucial to gain user acceptance. Second, permission models and access control must be adapted to the plurality of stakeholders collaborating in linked systems. Third, data anonymisation techniques must work on linked systems and be robust against a combination of many different sets of data. E.g. it will be interesting how concepts like differential privacy can be fitted to such multi-stakeholder multi-systems scenarios

7.8. Support of confidentiality and methods of authentication

Support of confidentiality becomes harder and harder as IoT becomes more widespread. More devices are connected to different types of devices, and this increase in opportunities for connection and data collection leads to smaller monitoring. Both the monitoring of data and monitoring of the attached devices are staked.

Control of support of confidentiality can be lost if someone cracks the smartphone or the computer acting as the panel for other devices. In the case of computers and smartphones, this cracking can be remotely and often not found. Smartphones, as well as computers, contain a vast number of personal information on their owners. They often refer to bank accounts, e-mail accounts and in some instances to household appliances. The stolen data can lead to severe problems. Vehicles contain many computers which control the function. Initially, these computers could not be cracked. Nevertheless, in case of the increase in a possibility of connection of IoT vehicles are exposed to risk because of connection to the Internet.

In other sense, monitoring can be lost as more and more companies collect data on users. These data often draw a detailed pattern of certain users by means of the collection online of data. Everything that you look for, all your actions on the Internet are monitored by the companies using these data. These companies often use data for improvement of the user experience, but they also use these data for sale of products of users or for sale to other companies which sell products of users.

Innovations in this sphere mean that the companies shall change privacy policy which exists and also how they interact with these devices.²⁰²⁾ The companies will look at a policy which they have to provide to users a possibility of access and monitoring of own data once again. Customers everything will realize more consequences for confidentiality of this interoperability layer by means of interaction from IoT and susceptibility to policies which provide them with the companies.

Now first of all the principles of the notification message, consent, access and safety are applied, for example. in electronic commerce and online advertising. The legislation on confidentiality also affects some mature technologies which are a part of the evolution of IoT: last RFID and a network of cameras paid much attention. Recent legislative efforts focused on data protection in cloud computing and support of appropriate protection of

7.8. Support of confidentiality and methods of authentication

web users against tracing. However already today the protection level of personal privacy offered by the legislation is insufficient as demonstrate data leakages and unpunished violations of confidentiality day by day.

However already today the protection level of personal privacy offered by the legislation is insufficient as demonstrate data leakages and unpunished violations of confidentiality day by day. The Internet of Things, undoubtedly, will create new grey zones with sufficient space to bypass legislative barriers.

First, the majority of acts is concentrated around an indistinct concept of Personally identified information (PII). Nevertheless, efforts on receiving a short determination of what represents PII (for example, by listing of combinations of the identifying attributes) quickly become outdated as new IoT technologies are unblocked and integrate new data sets which can provide identification and make more difficult to distinguish PII from a He-PII,

Secondly, the timeliness of the legislation is a constant problem:

For example, tracing of web users was used for many years before the European Commission adopted the law against it at the beginning of 2011. With the fast development of IoT, the legislation will be inevitable to fall further away. An example is indications of Smart Meter which already allow making exhaustive information on the life of people.

Thirdly, already today many violations of confidentiality remain unnoticed. In IoT realization of violations of confidentiality among users will be even lower as data collection moves to daily things and happens more passively. The legislation, however, often is only the response to public protests and shouts which require the realization of incidents first of all.

At last, the economy of private life still appears for those who ignore the legislation on confidentiality. On the one hand, development of PET, ensuring compliance and audit of policies of protection of private life are expensive and can restrict business models. On the other side, violations of the law about personal privacy either remain unpunished or lead only to rather small penalties while awareness of the public still too low to cause unacceptable damage to public reputation. Thus, ignoring the legislation on personal privacy as, for example, Google intentionally bypasses protection of tracing of users of Safari, seems profitable. In this regard, an incident of Google paid a record penalty in the amount of 22,5 mln. \$ of the USA in the settlement with the Federal trade commission (FTC), but it is entirely possible that profit more than is compensated.²⁰³⁾

Will be a serious problem to draft the uniform strong legislative base for privacy protection in IoT instead of quickly revising outdated acts for singular technologies. Success, undoubtedly, will demand all-round knowledge of a technological basis of IoT and its current evolution. The key, however, will consist in a deep understanding of the existing and remaining new threats of confidentiality in IoT - these threats are what the legislation shall protect from, eventually.

Authentication methods

Implementation of intelligent devices created the incalculable potential both for customers, and for business, but thanks to it there was an opportunity for hackers to abduct valuable information from personal data in intellectual property which does the

7. IoT security and privacy

company or a product unique. In the wider context of IoT this idea of authentication of users or devices becomes more and more widespread. For example, when we go to unblock our connected car by means of our mobile phone, we want to be sure that only we, owners, are authorized to do it to which successful "authentication" precedes. It means that users of the device (and/or the accounting entry) are those whom they speak, and have the authorized registration data for an information access after that that helps to create the main basis for support of communication and with the device on these expanded networks.

Nevertheless, presence only of one allowed user also creates problems or restrictions. For example, that if the defect in the attached device is found? The supplier, most likely, will demand access to the device far off to provide updates of the software for the solution of these problems. It is obvious in updates of the software of iPhone, therefore, the device receives the software far off, but is set only after you agree with conditions and you allow loading to begin.²⁰⁴⁾

If Apple had no original powers on sending to you the software, you will not be able to approve loading and to maintain operability of the device effectively or effectively.

One more practical example from the courageous new world of IoT is a concept of the virtual keys for cars which you can "wear with yourself" on the mobile phone, but also you can share with other family members or service personnel in a garage and resolve them (for example, during limited time) to use your car (of course, after successful authentication).

It is necessary to set the trust level according to which the public shall be sure that correspondence arrives directly from the specified source, but not for this purpose who creates a security risk for a network.

Thanks to several recent loud attacks in the field of cybersecurity, such as TalkTalk and Ashley Maddison, is more and more important that the enterprises assured the clients that these growing networks will be safe and will allow the user to control the data.

One of the methods of the solution of this problem of false authentication of users is the use of biometric data, that is the use of unique "biology" of individuals for access to their data. It includes unique means of identification, such as fingerprints and scanning of an iris of the eye of an eye which is incredibly difficult for reproducing.

Use of biometry and behavioural biometry (gestures, retina, etc.) creates the unique level of identification of users - indeed attributing feeling "personal" between the user and the device. It considerably increases registration this safety of the device and acts as the main barrier between hackers and their data access. When "things" communicate in IoT, registration data which are in the protected elements protected from illegal access which are built in devices can not only protect network access and communication but also support the protected services, such virtual private area networks, for example. for updates of the software.

When the attached devices IoT/M2M (for example, the built-in sensors and the executive mechanisms or ending points) need access to IoT infrastructure, trusting relationships are initiated on the basis of the device identifier. The method of storage and provision of the identification information can significantly differ for IoT devices. Pay attention that on typical corporate networks ending points can be identified by means of registration data of the person (for example, username and the password, a token or biometry). Ending

7.8. Support of confidentiality and methods of authentication

points of IoT/M2M shall be printed by fingers by means of means which do not require interaction with the person. Such identifiers include radio frequency identification (RFID), the general secret, certificates of X.509, the MAC address of an ending point or some type of the invariable trust based on hardware.

Establishment of authenticity through certificates of X.509 provides a reliable authentication system. However in the IoT domain memory for storage of the certificate cannot be enough for many devices or even not have the required CPU power for execution of cryptography operations of verification of certificates of X.509 (or any type of operations with the public key).

The existing identification traces, such as 802.1AR and protocols of authentication as IEEE 802.1X is defined, can be used for those devices which can control loading and memory of the CPU for storage of the strong registration data. Nevertheless, problems of new shape factors and also new modalities create an opportunity for further researches in the determination of more small-sized account types and fewer cryptography constructions and authentication protocols with intensive computation.

The second level of a framework is the authorization controlling access of the device on all network. This level is based on the main authentication level, using the identification information of an object. With components of authentication and authorization, the trusting relationship between IoT devices for exchange of the relevant information is established.

For example, the car can set the confidential union with another car at the same supplier. Nevertheless, these trusting relationships are able to allow cars to exchange the opportunities of safety. When an entrusted alliance is set between the same car and a network of his dealer, the car can have the right to share additional information, such as indications of the odometer, the last protocol of maintenance, etc.

Fortunately, the existing policy mechanisms for control and monitoring of access to consumer and corporate networks very well reflect needs of IoT/M2M.²⁰⁵⁾ The big task will consist in the creation of architecture which can be scaled for processing of billions of IoT/M2M devices with different relations of trust in structure. Policies of traffic and the appropriate controls will be applied on all network to segmentation of traffic of data and establishment of open communication.

Different medical devices shall be authenticated on the local gateway at the left when sending state-of-health data of health. Then the gateway shall be authenticated in an ending point of a cloud in case of transfer of these data. Applications with the rights which will analyze and display data of working capacity also shall be authenticated in a cloud in case of a request of data. The single scalable model for all above-mentioned authentications are tokens of safety - one actor is authenticated on another, including earlier received token in the messages. This token serves for identification of the first actor allowing the second actor to accept the appropriate permission.

It is important for data on health and other personal information that the appropriate users controlled as their data on health are collected, shared and analyzed. The powerful mechanism providing such monitoring is the requirement of the active involvement of the user in the process when to different characters is given safety tokens used for the subsequent interactions. Without the consent of the user, tokens are not given and there are no authenticated interactions. Thus, state-of-health data of health cannot proceed.

7. IoT security and privacy

OAuth 2.0 and OpenID Connect 1.0 are the two standardized frames for authentication and authorization which obviously support the model stated above. Both allow the user to participate obviously in the release of tokens for the applications requiring user data - health or otherwise, - and, thus, can provide significant monitoring of confidentiality. Besides, Connect provides the built-in mechanisms of detection and registration which are extremely important for scaling of any architecture to the number of the participants created by IoT.

One of the problems is that OAuth and Connect are still connected to HTTP. Experts in safety read that HTTP is not enough for many interactions in IoT, especially between things/devices and other participants. There was a new class of protocols which promises to be more suitable, than HTTP, for such interactions, including MQ Telemetry Transport and Constrained Application Protocol. There were early researches of binding OAuth and Connect with this new category of protocols with IoT optimization, but operation remains.

The task to invent new mechanisms and standards for authentication of participants of IoT is not all history. The possibility of authentication in IoT consists in acknowledging the possibility of switching on of new methods of authentication of users via devices and things which surround us. Use of the smartphone for two-factor authentication is an early manifestation of this tendency. Opportunities which do the smartphone by a powerful authentication factor are the same that will allow our hours, bracelets and thermostats to have a judgement on our identity - and ability to approve this judgement.

A phone does powerful coefficient of authentication because for most of the users it always with them - a factor "what you have" matters a little if you cannot assume that the user has it at their instruction. But this quality which is tightly connected to the user is even fairer concerning a new class of the carriers used for monitoring of suitability of the person, a dream and other personal indices.

We will consider a bracelet of Fitbit which gives to users reviews of the daily activities. Fitbit is a tiny connected computer which is tightly connected to the specific user. Thus, Fitbit and other similar devices can facilitate authentication of the user in case of access to applications, devices or cloud services. The Nymi device accepts the idea on one step, having added biometric authentication of the user; it will not do keys which it saves for authentication, before confirmation of the electrocardiogram of the user from the saved template.

Authentication with use of the infrastructure of OAUTH over the simple level of authentication and safety (sasl) in iot-devices

OAuth is the open standard structure of authorization and the authentication protocol providing to third-party applications the limited delegated access to private resources by establishment of interaction between the third-party application and the owner of a resource and determination of a certain process to which the owner of a resource provides authorization of access to third-party applications to server resources, without tearing off their registration information (a user id, passwords, etc.).²⁰⁶⁾

On the other hand, the authentication level of level and the security level (SASL) is an authentication basis for data protection in the environment of the application layer. During the provision of access to the client (for example, Facebook application, application of Twitter, etc.). For protected resources (the user account of Facebook, the accounting entry of Twitter, etc.) Originally the permissions on access to resources

7.8. Support of confidentiality and methods of authentication

executed over Plain OAuth 2.0 were requested, but the last stage of authentication the client for access to resources from the owner of a resource is implemented with the use of the infrastructure of the OAuth protocol through the structure of authentication of SASL. Systematic transmissions of requests from the client for the provision of permission are given below:

Use of Plain OAuth:

A step of I: The client's request for provision of permission from the owner of a resource two methods: i) The owner of a resource receives the request sent per the client, directly. ii) A request is sent per the client via the intermediate server of authorisation.

Step of II: authorization is provided to the client in the form of registration data. This permission depends on whether the client for receiving a grant directly or indirectly requested.

Step of III: Access to the resource server is possible only by means of a certain token of access. They are requested by the client, at first verifying authenticity with the authorization server, and then redirecting the authorization permission got directly from the owner of a resource or indirectly via the authorization server.

Step of IV. If the client is authenticated on the servers, the server of authorization checks the permission of authorization and then gives an access token.

Use of OAuth over SASL:

Step of I: after receiving a token of access the client requests access to private resources from the server of resources, authenticating himself by means of an access token.

Step of II: The server of resources checks an access token. In case of success, the client is authenticated for access to resources on behalf of the owner of a resource.

All steps given above are provided schematically in the following figure:

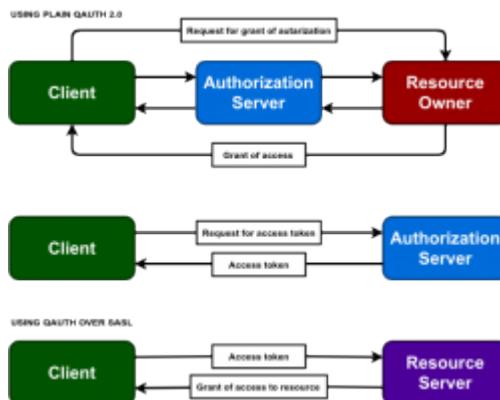


Figure 275: Block diagram for OAuth.

8. Introduction to the IoT Energy consumption

The IoT energy consumption technology market is difficult to define because the discovery of energy optimisation is ongoing and most of the producers do not define it as the main priority. The competitiveness of European technologies could strengthen the European position in the global market for energy efficient solutions. The EU is making rapid progress towards establishing Europe-wide energy consumption standards; however, most European countries have developed and follow their own set of regulations. Although IoT energy consumption effective solutions are currently not available, none represent a complete solution with low cost-performance parameters. Energy efficiency consists of reducing the energy consumption, keeping the same energy services but using sustainable methods that protect the environment. Those come from renewable resources.

The energy and environment benefits of the energy efficiency implementation are the CO₂ reductions in the atmosphere. It is essential to be conscious of the use of resources and to contribute to sustainability. Boost the renewable energy and the importance of making use of it.

The socio-economic factors that are related to the energy efficiency are the saving on the energy bill, reducing climate change impact and also the reduction of the external energy dependency.

The IEA (International Energy Agency) promotes energy efficiency policy and technology in buildings, transport, appliances and industry and at lighting applications. They developed the document "25 energy efficiency policy recommendations" which identify best-practice, for energy efficiency improvements and implements full potential of it in each sector. IEA is an autonomous organisation founded in 1974 which works to ensure reliable, affordable and clean energy for its 29 member countries and beyond. It has four main areas of focus:

- Energy security: Promoting diversity, efficiency and flexibility within all energy sectors.
- Environmental awareness: Analysing policy options to offset the impact of energy production and use on the environment, especially for tackling climate change.
- Economic development: Supporting free markets to foster economic growth and eliminate energy poverty.
- Engagement worldwide: Working closely with partner countries, especially major economies, to find solutions to share energy and environmental concerns.

Power efficiency in IoT

Minimum Energy Performance Standards (MEPS)

Electronic components and their power requirements: motors, sensors, microcontrollers

IoT software platform

IoT Battery management systems

8.1. Power efficiency in IoT

8.2. Minimum Energy Performance Standards (MEPS)

The primary indicator of IoT, which provides a practical network application, is the overall life expectancy of a network and the tasks that are related to modelling the life expectancy as well as the practical application. The concept of an IoT provides that any network element uses an independent power supply. Any network element performs specific tasks in the common IoT network, which according to the network scenario, are pre-programmed.

Looking at the life cycle of the IoT, it is essential to pay attention to the device energy efficiency. The energy efficiency of each device is dependent on the characteristics of the applied standards, the algorithm used and the network protocol. While developing the project of a network, one of the most important factors is increasing energy efficiency and the operating cycle of the network depends on it.

Often when actions are related to IoT the term “network energy efficiency” is equated to the operating cycle of the autonomous devices and it is believed that longer operating time of each IoT device ensures a higher energy efficiency of the network.

It is understandable that IoT devices can be seen as active for as long as they are capable of reading the information from the sensors correctly as well as transferring it to the coordinator node within a network. IoT energy consumption depends on several factors:

- The technical parameters of the node battery capacity as well as the parameters of the processor, the transmitter, the sensors and the other elements.
- Data collection frequency, which may depend on the situation and the environment in which the network operates.
- Physical and channel level protocols that determine the control of access mechanism in the environment.
- Network topology that defines the amount of the transmitted information within each node including the flow of technical information.
- The use of routing protocols that complement the network with additional service data flow.

Any IoT network has three node types – the terminals, the routers and the coordinator or data collector. Data collectors do not affect the overall life expectancy of a system because they are provided with a permanent power supply or are equipped with a much more powerful autonomous power supply.

8.2. Minimum Energy Performance Standards (MEPS)

MEPS specify the maximum energy or power demand of devices, which manufacturers must ensure in their models of a regulated category when using the test method that accompanies the MEPS. MEPS may define two measures: modal power, expressed in watts W, and total annual energy consumption (TEC), shown in kWh. The modal power specifies the maximal power consumption for one or more low power modes. TEC provides the annual estimate of energy consumption across various modes, based on an assumed use profile.

8.2.1. Vertical MEPS

Vertical MEPS are set on a device category basis. Vertical MEPS advantages:

- Can be tailored to each specific device category.

8. Introduction to the IoT Energy consumption

- Energy saving can be defined.

Vertical MEPS disadvantages:

- Complex to develop.
- Costly to support and update.
- Unable to define one structure for multiple devices.

Considering the need for a quick implementation of policies to keep up with technology developments, Vertical MEPS should focus on well-defined, high impact product categories. The following Vertical MEPS policies are proposed for the prioritized IoT categories:

- Vertical MEPS for network standby of home security cameras, smart home gateways, smart LED lamps and network-connected audio products.
- Tightening of Vertical MEPS for external power supplies.

8.2.2. Horizontal MEPS

Horizontal MEPS cover a range of different device categories. Horizontal MEPS advantages:

- Covers a broad range of devices.
- The energy saving plan can be defined.

Horizontal MEPS disadvantage is some specific devices require unique structure and design. Horizontal MEPS don't consider specific device characteristics. Therefore, the defined performance limits are typically a compromise between the best possible values and broad product coverage. Given the rapid evolution of the IoT market and the many and diverse product categories, it may be an excellent trade-off to use Clustered MEPS.

8.2.3. Clustered MEPS

Clustered MEPS combine a few device categories with similarities in main functions, network interactions and energy demand. An example of horizontal MEPS is the European Union's Standby Regulation 1275/2008/EC amended by Regulation EU/801/2013 to include network standby).

8.2.4. Electronic components and their power requirements: motors, sensors, microcontrollers

Power consumption is one of the most significant challenges for IOT. Today IOT device need to be able to sustain longer battery lifespan, especially in cases such as outdoor deployments, to shorten hardware maintenance and prevent the breakdown of communication. In many deployment cases, to prolong the usability of the equipment in the field, large battery sources have to be attached to the sensors, making the sensor setup bulky and cumbersome. IOT supports the pervasive connectivity of sensors and the need for them to interact with each other, i.e., act as both tags and interrogators. To support such connectivity and communications, the design and use of low-power chipsets will create a significant impact and consideration on power consumption for future sensors. Ultra-low power designs for chipset circuits have been an ongoing research area, with techniques moving from single gate to multi-gate transistors and carbon

8.2. Minimum Energy Performance Standards (MEPS)

nanotube designs.

Energy harvesting technologies that convert energy out of physical energy sources such as temperature differences and applied pressure have been researched to explore their capability to replace conventional batteries. Two examples of power scavenging technologies are photovoltaic technology which generates electric power by using solar energy and piezoelectric technology that creates charges on stress or shape change on the voltage applied. Newer forms of battery technologies, e.g., polymer battery, fuel cell and paper batteries will support increasing functionality and longer battery lifetime. Paper and smart label batteries have shown promising use cases in warehousing usage as they allow containers to perform two-way communications with the reader.

8.2.5. IoT software platform

The IoT hardware requires operating systems and communication protocols to interact with a user and other devices. Some components facilitate communication and exchange of information between devices. In IoT architectures, integration layers play an important role in combining and integrating information acquired from thousands of devices and presenting this information to users. In this section, we review the general software structure inside of an IoT system. In the design of an IoT software platform, scalability, the extensibility and interoperability between heterogeneous devices and their business models should be considered. Also, IoT enabling technologies (hardware) may move geographically hence need to communicate with others in a real-time mode. This kind of operation necessitates decentralised and event-driven software architecture. Service-oriented-architecture (SoA) ensures the scalability and interoperability of heterogeneous technologies in one platform. In a generic SoA four layers are defined:

- sensing layer uses integrated hardware to sense things' statuses;
- network layer which connects the things together and collects the data from hardware infrastructure,
- service layer creates and manages services requested by users or applications;
- interface layer enables the interaction methods with applications or users.

In a SoA for an IoT middleware, the software between objects (things which are equipped with sensors) and applications should provide object abstraction, service management and service composition through a secure network.

Each IoT software main task is device identification in the network. For object identification, different addressing methods are used based on internet protocols (IPs) such as IPv4, IPv6, and 6LoWPAN. For the identification, it should be notified that an object's identification and address are different. While an object can be identified locally, for example inside a 6LoWPAN network, the object within the global network uses public IPs as the address. Identification methods aim to make a clear identity for an object inside the network. Communication link technologies should provide the infrastructure for the connection of smart devices (sensor nodes). The IoT sensor nodes should work normally under severe designs specifications including low-power consumption, and operation in the noisy environment. Currently, there are different communication protocols which can be used for IoT applications, which have been covered in Introduction to the IoT Communication and Networking

8. Introduction to the IoT Energy consumption

8.2.6. IoT Battery management systems

The term battery management system (BMS) refers to a structure that allows a battery pack to be kept in a safe operating area. A relatively advanced battery management system is expected to provide the following functions:

- Keeping the battery pack between specific voltage values, preventing overcharging and discharging. It is particularly important in case of use of the Lithium-based batteries (LiPo, LiIon, LiFe).
- To keep the battery pack within certain temperature limits and to intervene in the system to provide these limits when necessary.
- Measuring and limiting the charge and discharge current.
- Reducing voltage imbalances between cells to increase the productive use capacity of the battery pack.
- Remaining Useful Life (RUL), State of Charge (SoC) and State of Health (SoH) estimation.

Battery System Security Issues on physical layer

The physical layer of a battery system includes the battery cells, the surrounding circuitry, and the connections with the BMS. The battery cells have certain limits (lower and upper voltage/current bounds) and demonstrate specific behaviour towards different power requests. The BMS and the circuitry components are responsible for monitoring the battery cells and protecting them from overvoltage, under voltage, overcurrent, overloading, and also overheating.

Battery System Security Issues on Management System Layer

The BMS can be any system that manages the battery. As discussed, batteries are sensitive to overcharging and deep discharging because they may damage the battery, therefore shortening its lifetime and even causing hazardous situations. This requires the adoption of a proper BMS to maintain the states of each cell of the battery within its safe and reliable operating range. In addition to its primary function of battery protection, a BMS should estimate the battery status to predict the actual amount of energy that can still be delivered to the load.

9. Emerging technologies in IoT

Since IoT as a paradigm for developing systems of the next generation is being actively advanced by the IoT community, new technologies are arriving every year. This chapter is devoted only to few of them to show the direction of the current efforts:

ROS - a new framework in IoT - introduction to Robot Operating system

Autonomous transport systems - Introduction to autonomous transport systems

Blockchain - Introduction to BlockChains

9.1. ROS - a new framework in IoT

9.1.1. What is ROS?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocros, CARMEN, Orca, MOOS, and Microsoft Robotics Studio²⁰⁷⁾. The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. All these things would be explained a bit later.

ROS is not a real-time framework, though it is possible to integrate ROS with real-time code.

9.1.2. ROS features:

Distributed computation

Many modern robot systems rely on software that spans many different processes and runs across several different computers. For example:

- Some robots carry multiple computers, each of which controls a subset of the robot's sensors or actuators.
- Even within a single computer, it's often a good idea to divide the robot's software into small, stand-alone parts that cooperate to achieve the overall goal. This approach is sometimes called "complexity via composition."
- When multiple robots attempt to cooperate on a shared task, they often need to communicate with one another to coordinate their efforts.
- Human users often send commands to a robot from a laptop, a desktop computer, or mobile device. We can think of this human interface as an extension of the robot's software.

Software reuse

9. Emerging technologies in IoT

The rapid progress of robotics research has resulted in a growing collection of useful algorithms for common tasks such as navigation, motion planning, mapping, and many others. Of course, the existence of these algorithms is only truly useful if there is a way to apply them in new contexts, without the need to reimplement each algorithm for each new system. ROS can help to prevent this kind of a pain in at least two important ways.

- ROS's standard packages provide stable, debugged implementations of many important robotics algorithms.
- ROS's message passing interface is becoming a de facto standard for robot software interoperability, which means that ROS interfaces to both the latest hardware and to implementations of cutting-edge algorithms are quite often available. For example, the ROS website lists hundreds of publicly-available ROS packages. This sort of uniform interface greatly reduces the need to write "glue" code to connect existing parts.

As a result, developers that use ROS can expect—after, of course, climbing ROS's initial learning curve—to focus more time on experimenting with new ideas, and less time reinventing wheels.

Rapid testing

One of the reasons that software development for robots is often more challenging than other kinds of development is that testing can be time-consuming and error-prone. Physical robots may not always be available to work with, and when they are, the process is sometimes slow and finicky. Working with ROS provides two effective workarounds to this problem.

- Well-designed ROS systems separate the low-level direct control of the hardware and high-level processing and decision making into separate programs. Because of this separation, we can temporarily replace those low-level programs (and their corresponding hardware) with a simulator, to test the behaviour of the high-level part of the system.
- ROS also provides a simple way to record and playback sensor data and other kinds of messages. This facility means that we can obtain more leverage from the time we do spend operating a physical robot. By recording the robot's sensor data, we can replay it many times to test different ways of processing that same data. In ROS parlance, these recordings are called "bags" and a tool called *rosbag* is used to record and replay them.

A crucial point for both of these features is that the change is seamless. Because the real robot, the simulator, and the bag playback mechanism can all provide identical (or at least very similar) interfaces, your software does not need to be modified to operate in these distinct scenarios, and indeed need not even "know" whether it is talking to a real robot or not. Of course, ROS is not the only platform that offers these capabilities. What is unique about ROS, at least in the author's judgment, is the level of widespread support for ROS across the robotics community. This "critical mass" of support makes it reasonable to predict that ROS will continue to evolve, expand, and improve in the future.

The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (Nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS also supports

a federated system of code Repositories that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools. In support of this primary goal of sharing and collaboration, there are several other goals of the ROS framework:

- **Thin:** ROS is designed to be as thin as possible – we won't wrap your main() – so that code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks: ROS has already been integrated with OpenRAVE, Orocos, and Player.
 - **ROS-agnostic libraries:** the preferred development model is to write ROS-agnostic libraries with clean, functional interfaces.
 - **Language independence:** the ROS framework is easy to implement in any modern programming language. We have already implemented it in Python, C++, and Lisp, and we have experimental libraries in Java and Lua.
 - **Easy testing:** ROS has a builtin unit/integration test framework called roster that makes it easy to bring up and tear down test fixtures.
 - **Scaling:** ROS is appropriate for large runtime systems and large development processes.
-

9.1.3. Operating systems

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms.

While a port to Microsoft Windows for ROS is possible, it has not yet been fully explored.

9.1.4. ROS architecture

ROS has three levels of architecture: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts are summarized below and later sections go into each of these in greater detail.

In addition to the three levels of concepts, ROS also defines two types of names – Package Resource Names and Graph Resource Names – which are discussed below.

ROS Filesystem Level

The filesystem level concepts mainly cover ROS resources that you encounter on disk, such as:

- **Packages.** Packages are the main unit for organising software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organised together. Packages are the most atomic build item and release the item in ROS. Meaning that the most granular thing you can build and release is a package.
- **Metapackages:** Metapackages are specialised Packages which only serve to represent a group of related other packages. Most commonly meta packages are used as a

9. Emerging technologies in IoT

backwards compatible placeholder for converted rosbuild Stacks.

- **Package Manifests.** Manifests (`package.xml`) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages. The `package.xml` package manifest is defined in REP-0127.
- **Repositories.** A collection of packages which share a common VCS system. Packages which share a VCS share the same version and can be released together using the catkin release automation tool bloom. Often these repositories will map to converted rosbuild Stacks. Repositories can also contain only one package.
- **Message (msg) types.** Message descriptions, stored in `my_package/msg/MyMessageType.msg`, define the data structures for messages sent in ROS.
- **Service (srv) types.** Service descriptions, stored in `my_package/srv/MyServiceType.srv`, define the request and response data structures for services in ROS.

Computation Graph level

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

These concepts are implemented in the `ros_comm` repository.

- **Nodes:** Nodes are processes that perform a computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localisation, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as `roscpp` or `rospy`.
- **Master:** The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server:** The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages:** Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).
- **Topics:** Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it on a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a particular kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to various topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.

- Services: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system. Request/reply is done via services, which are defined by a pair of message structures: one for the request and one for the response. A providing node offers a service under a name, and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.
- Bags: Bags are a format for saving and playing back ROS message data. Bags are an essential mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

The ROS Master acts as a name service in the ROS Computation Graph. It stores topics and services registration information for ROS nodes. Nodes communicate with the Master to report their registration information. As these nodes communicate with the Master, they can receive information about other registered nodes and make connections as appropriate. The Master will also make callbacks to these nodes when this registration information changes, which allows nodes to dynamically create connections as new nodes are run.

Nodes connect to other nodes directly; the Master only provides lookup information, much like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic, and will establish that connection over an agreed upon connection protocol. The most common protocol used in a ROS is called TCPROS, which uses standard TCP/IP sockets.

This architecture allows for decoupled operation, where the names are the primary means by which more extensive and more complex systems can be built. Names have a crucial role in ROS: nodes, topics, services, and parameters all have names. Every ROS client library supports command-line remapping of names, which means a compiled program can be reconfigured at runtime to operate in a different Computation Graph topology.

ROS Community Level

The ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include:

- Distributions: ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.
- Repositories: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.
- The ROS Wiki: The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.
- Bug Ticket System: Please see Tickets for information about file tickets.
- Mailing Lists: The ros-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software.

9. Emerging technologies in IoT

- ROS Answers: A Q&A site for answering your ROS-related questions.
- Blog: The ros.org Blog provides regular updates, including photos and videos.

9.1.5. Introduction to ROS programming

What does IoT need?

It's clear that in the future we will have many devices that need the Internet for their work in our regular life. While devices are becoming more intelligent, the degree of automatisation is increasing. Today devices use the Internet to find or store some data. However, frequently it's hard to integrate different IoT-devices in one system. The market needs standardisation.

In this chapter, we are going to introduce you to the Robot Operating Systems (ROS) libraries, which could be used in IoT tasks. The ROS is not just the most popular platform for academic researches in robotics, but a powerful open-source set of software libraries and tools not only for robotics but also autonomous systems.

Why ROS?

Firstly, the ROS is the most popular education platform for world universities. ROS is widely used in different researches. Secondly, ROS is based on Linux, so you can use all libraries from ROS to stop worrying about the physical side of your device and write code to it or even your own Linux core. So far, Linux is the best OS for embedded systems.

ROS Installation

The latest version of ROS is **Kinetic Kame**, which is available for Ubuntu Wily (15.10) and Ubuntu Xenial (16.04 LTS). If you have older Ubuntu version install ROS **Indigo Igloo**.

Instructions for installing ROS **Kinetic Kame**:

1. Configure your Ubuntu repositories to allow restricted, universe, and multiverse
2. Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt
```

3. Make sure your Debian package is up-to-date

```
sudo apt-get update
```

4. Install ROS Desktop version. If you want to work with navigation and 2D/3D perception packages, make Desktop-full install

```
sudo apt-get install ros-kinetic-desktop
```

```
apt-cache search ros-kinetic
```

5. To make all system dependencies for source you want to compile, run `rosdep`. Also it

is required to run some core components in ROS.

```
sudo rosdep init  
rosdep update
```

6. Environment setup

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

7. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. To install this tool and other dependencies for building ROS packages, run:

```
sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

First steps

To be good at ROS programming, you need to understand ROS filesystem concepts and be able to use `roscd`, `rosls`, and `rospack` command line tools.

After that, let's instal `turtlesim`. This package illustrates how different things work. Use the following command:

```
sudo apt-get install ros-kinetic-turtlesim
```

Now, there is a need to configure `rosdep` systemwide. Just execute the command:

```
rosdep
```

The goal of this command is to initialise `rosdep`, which is a tool for checking and installing package dependencies in an OS-independent way. On Ubuntu, for instance, `rosdep` acts as a front end to `apt-get`.

Account Configuration

Doesn't matter you install ROS yourself or you're using a computer with pre-installed ROS, there are two important configuration steps that must be done within the account of every user that is going to ROS using.

At start, initialize the `rosdep` system in your account:

```
rosdep update
```

This command stores some files in your home directory, in a sub-directory named `.ros`. This action must be done one time.

9. Emerging technologies in IoT

Before using this command be sure you are acting into your account, not using `sudo`.

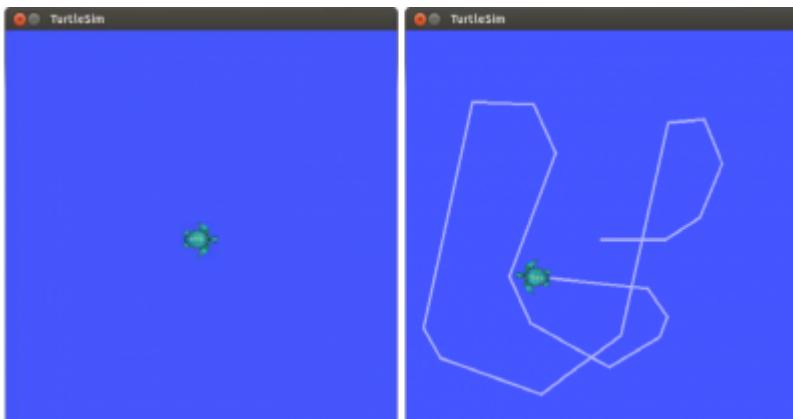
Turtlesim example

Let's start ROS programming with a simple example. This exercise serves a few purposes: it provides an opportunity to check that ROS is installed correctly, introduce turtlesim simulator and it is used in a lot of tutorials published on the web.

Starting turtlesim

```
roscore  
rosrun turtlesim_node  
rosrun turtlesim turtle_teleop_key
```

By running separate command terminals, we can run three commands simultaneously. If the configuration is OK you can see a graphical window:



This window shows a simulated turtle robot that operates in a squad.

Note, the appearance of your turtle may be different. It depends on the version of ROS.

If you choose the terminal one which is executing `turtle_teleop_key` and give the inputs by pressing *Left, Right, Up, Down*, the turtle should move to your commands. You should keep these three terminals open because of the examples in the following sections will show some additional ways to interact with this system. Before we start to write programs, we should create a workspace to save there our packages, and then create the package.

Creating workspace Packages we are creating should be situated together in a directory called a **workspace**. For instance: `/home/$username$/ros`. You can name your workspace ever you want and save the directory anywhere in your account. To create a directory, use the `mkdir` command. If you are working on many projects, we recommend you to create several independent workspaces. Now, lets set up the workspace. Create an `src` subdirectory inside the workspace directory. This directory will contain the source

code of your packages.

A package creation To create a new ROS package use the following command:

```
catkin_create_pkg //package-name//
```

This command should be run from the `src` directory of initial workspace. This command creates a directory to hold the package and creates two configuration files inside that directory.

- `package.xml` is the manifest. This file defines some details about the package, such as its name, version, maintainer, and dependencies. The directory containing `package.xml` is called the `package` directory.
- `CMakeLists.txt` is a script for an industrial-strength cross-platform build system `CMake`. It contains a list of build instructions including what executables should be created, what source files to use to build each of them, and where to find the include files and libraries needed for those executables. `CMake` is used internally by `catkin`.

Hello, ROS!

Now, after the workspace configuration we can create ROS programs. Let us consider a version of the "Hello".

```
#include <ros / ros<h>
int main(int argc, char **argv) {
ros::init(argc, argv, "hello_ros");
ros::NodeHandle nh;
ROS_INFO_STREAM("Hello, ROS!");
}
```

- `ros/ros.h` is declaration of the ROS classes. It is necessary to connect it in every ROS program you are creating.
- `ros::init` It is a function which initialises the library of the ROS client.
- `ros::NodeHandle` object is the main mechanism that your program will use to interact with ROS system. By creating this object provides registration with your program as a node to the ROS master.
- `ROS_INFO_STREAM` this command generates an information message. This message is going to be sent to several various locations, including the console.

Arduino IDE setup

The Arduino and Arduino IDE are great tools for quickly and easily programming hardware. Using the `rosserial_arduino` package, you can use ROS directly with the Arduino IDE. `rosserial` provides a ROS communication protocol that works over your Arduino's UART. It allows your Arduino to be a full-fledged ROS node which can directly publish and subscribe to ROS messages, publishes TF transforms, and get the ROS system time. Our ROS bindings are implemented as an Arduino library. Like all Arduino libraries, `ros_lib` works by putting its library implementation into the `libraries` folder of your sketchbook. If there is not already a `libraries` folder in your sketchbook, make one. You can then install the library using the instructions below. In order to use the `rosserial` libraries in your own code, you must first put:

9. Emerging technologies in IoT

```
#include <ros.h>
```

First, should be done.

```
#include <std_msgs/String.h>
```

Otherwise the software will not be able to locate header files.

Setting up the software

At first we should install `rosserial` for Arduino by the following command

```
sudo apt-get install ros-kinetic-rosserial-arduino  
sudo apt-get install ros-kinetic-rosserial
```

If you use another ROS version just replace *kinetic* to your release, e.g. *hydro*

Installing from Source onto the ROS workstation

Source build instructions are different for *groovy+* (*catkin*) than for earlier (*rosbuild*) releases. Select the build system based on your release to see appropriate instructions. Rosserial has been *catkin*-ized since the *groovy* release, and the workflow is a bit different from *Fuerte* and earlier versions. Rather than running the library generator over each package you want to use, you run it once and generate libraries for all installed messages. In the instructions below, `<ws>` represents your *catkin* workspace.

```
cd <ws>/src  
git clone https://github.com/ros-drivers/rosserial.git  
cd <ws>  
catkin_make
```

These commands clone rosserial from the github repository, generate the `rosserial_msgs` needed for communication, and make the `ros_lib` library in the `<ws>/install` directory.

Note: currently you HAVE to run the `catkin_make install`. Otherwise, portions of the `ros_lib` directory will be missing. It will hopefully be fixed in the future release.

In case if you use previous ROS versions, use:

```
hg clone https://kforge.ros.org/rosserial/hg rosserial  
rosmake rosserial_arduino
```

These commands clone rosserial from the *kforge* repository using mercurial, generate the `rosserial_msgs` needed for communication and make the `ros_lib` library.

ros_lib installing to the Arduino Environment

The preceding installation steps created `ros_lib`, which must be copied into the Arduino build environment to enable Arduino programs to interact with ROS.

9.1. ROS - a new framework in IoT

In the steps below, <sketchbook> is the directory where the Linux Arduino environment saves your sketches. Typically this is a directory called sketchbook in your home directory. Alternately, you can install into a Windows Arduino environment.

The `ros_lib` installation instructions are different for the *groovy* source (*catkin*) than for earlier (*rosbuild*) or binary releases. Be sure you've selected the correct build system above to see appropriate instructions - *catkin* for a *groovy* source build, *rosbuild* otherwise.

Now that you've installed either from source or debs, all you have to do is to copy the `rosserial_arduino/libraries` directory into your Arduino sketchbook:

```
rosscd rosserial_arduino/src  
cp -r ros_lib <sketchbook>/libraries/ros_lib
```

If you are building Arduino on Windows, copy the `ros_lib` directory from Linux to the Windows system's `sketchbook/libraries` folder (typically found in My Documents).

Note: Currently you can install the Arduino libraries directly in the Arduino IDE. Just open the Library Manager from the IDE menu in *Sketch* → *Include Library* → *Manage Library*. Then search for "rosserial". It is useful if you need to work on an Arduino sketch but doesn't want to set up a full ROS workstation.

After restarting your IDE, you should see `ros_lib` listed under examples.

Publisher example

We'll start our exploration into `rosserial` by creating a "hello world" program for our Arduino.

Note: the Arduino community often calls source code for programs a "sketch", we will use the same convention below.

If you have followed the Arduino IDE Setup tutorial, you'll be able to open the sketch below by choosing `ros_lib` → *HelloWorld* from the Arduino examples menu.

It should open the following code in your IDE:

```
1 /*  
2  * rosserial Publisher Example  
3  * Prints "hello world!"  
4 */  
5  
6 // Use the following line if you have a Leonardo or MKR1000  
7 //#define USE_USBCON  
8  
9 #include <ros.h>  
10 #include <std_msgs/String.h>  
11  
12 ros::NodeHandle nh;  
13  
14 std_msgs::String str_msg;  
15 ros::Publisher chatter("chatter", &str_msg);
```

9. Emerging technologies in IoT

```
16
17 char hello[13] = "hello, ROS!";
18
19 void setup()
20 {
21   nh.initNode();
22   nh.advertise(chatter);
23 }
24
25 void loop()
26 {
27   str_msg.data = hello;
28   chatter.publish( &str_msg );
29   nh.spinOnce();
30   delay(1000);
31 }
```

The code compiling and running

To compile the code, use the compile function within the Arduino IDE. It is no different from compiling any other sketch. Once the compilation is completed, you will receive a message about program storage space and dynamic memory usage. To upload the code to your Arduino, use the upload function within the Arduino IDE. It is no different from uploading any other sketch.

Now, launch roscore:

```
roscore
```

Next, run the rosserial client application that forwards your Arduino messages to the rest of ROS. Make sure to use the correct serial port:

```
rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

Alternatively, if you want the ability to programmatically reset your Arduino, run using:

```
rosrun rosserial_arduino serial_node.py _port:=/dev/ttyUSB0
```

It will automatically provide a service endpoint at `~reset_arduino` that you can call which will have the same effect as pressing the Arduino's reset button.

Finally, watch the greetings come in from your Arduino by launching a new terminal window and entering:

```
rostopic echo chatter
```

9.1.6. IoT bridge

The `iot_bridge` provides a connection between ROS and any smart device of the control system. Since 2017, the package offers to bridge with openHAB, one of the most actively developed and used a framework for home automation.

9.1. ROS - a new framework in IoT

- ROS is an extremely powerful open source set of libraries and tools that help you build robot applications - providing drivers and state-of-the-art algorithms for vision, movement, etc. See the official website ²⁰⁸⁾ for more info.
- openHAB is an open source system that connects to virtually any intelligent device, such as smoke detectors, motion detectors, temperature sensors, Amazon Alexa, security systems, TV/audio, Chromecast, fingerprint scanners, lighting, 1-Wire, Bluetooth, MQTT, Z-Wave, telephony, weather sensors, and web services such as Twitter, etc. openHAB also provides a basic Web GUI and iPhone / Android app for setting and dynamically viewing values. To access the full list of supported features go to ²⁰⁹⁾ for more info.

How can we use it? *

- A motion detector in OpenHAB triggers and ROS dispatches the robot to the location.
- ROS facial recognition recognises a face at the door, and OpenHAB turns on the lights and unlocks the door.
- A Washing Machine indicates to OpenHAB that the load is complete and ROS dispatches a robot to move the laundry to the dryer.
- A user gives a voice smart home command to Alexa. Using the OpenHAB/Amazon Alexa integration then forwarded to ROS via the iot_bridge.
- The OpenHAB MQTT location binding indicates that Sarah will be home soon and a sensor indicates that the temperature is hot. ROS dispatches the robot to bring Sarah's favourite beer. OpenHAB turns on her favourite rock music and lowers the house temperature.
- A user clicks on the OpenHAB GUI on an IPAD and selects a new room location for the robot. The message is forwarded by the iot_bridge to ROS, and ROS dispatches the robot. ²¹⁰⁾

Using iot-bridge technology, any OpenHAB device can be easily setup to publish updates to the iot_updates topic in ROS, giving a ROS robot knowledge of any Home Automation device. ROS can publish to the iot_set topic (or iot_command), and the device in OpenHAB will be set to the new value (or act on the specified command).

Installing iot_bridge

At first, we should install and configure openHAB. For details and guides see the OpenHAB official website. Many people find that the simplest way to experiment with openHAB is to get a Raspberry Pi and install openHABian - the "hassle-free openHAB setup". While openHABian offers a streamlined and simplified way to get up and running quickly, it is a complete openHAB home automation system capable of automating your entire home ²¹¹⁾.

Just in a few words, let's consider the OpenHAB installation. Configure the repository in your Ubuntu OS:

```
 wget -qO - 'https://bintray.com/user/downloadSubjectPublicKey?username=openhab' |  
 sudo apt-key add - echo "deb http://dl.bintray.com/openhab/apt-repo stable main" | sudo tee  
 /etc/apt/sources.list.d/openhab.list
```

After this, we can install the runtime:

9. Emerging technologies in IoT

```
sudo apt-get install openhab-runtime
```

After installation, you should configure the runtime. See the following guide Configuring-the-openHAB-runtime.

Now, we should install iot-bridge on the ROS system:

1. Go to iot bridge and find the GIT clone address
2. Ordered List Item In the ROS console:

```
cd catkin_ws/src  
git clone address-from-above  
cd ..  
catkin_make
```

3. Edit iot_bridge/config/items.yaml
 - * Unordered List Item update host address and port to match your OpenHAB server).
 - Ordered List ItemEdit Openhab's item file
 - * Create the ROS group: Group ROS (All)
 - * Add the (ROS) group to each item that should send status updates to ROS.

Note that the items must be directly in a (ROS) group, not in a sub-group of the (ROS) group. Note, this is only needed for status updates to go from OpenHAB to ROS - you can send commands from ROS to any OpenHAB item regardless of what group it is in. 212)

```
add ROS_Status to openhab's item file
```

. Sample Open_HAB item definition:

NOTE: you must have two or more items defined in the ROS group.

```
Group ROS (All)  
String ROS_Status "ROS [%s]"  
Switch Light_GF_Corridor_Ceiling "Ceiling" (GF_Corridor, Lights, ROS)  
Switch Light_GF_Bathroom (GF_Bathroom, Lights, ROS)
```

Running

1. Run the iot-bridge by

```
roslaunch iot_bridge iot.launch
```

2. Testing.

Follow steps to check to receive from OpenHAB.

- In your browser, go to <http://localhost:8080/rest/items/ROS>
- We can see an XML response with the state of the items you have put in the ROS

group

- In the console, type

```
rostopic echo /iot_updates
```

- Bring up the OpenHAB demo site in your browser and change an item in the ROS group
- Now we can see the new state in the rostopic echo console

Follow next steps to check sending to OpenHAB:

```
cd catkin_ws/src/iot_bridge/scripts  
./iot_test item_name item_value
```

- You should see the message logged where you did the rosrun.
- If you have the OpenHAB demo site in your browser, you should see the item you named changed to your specified value.
- The value must be valid for that device (number, or ON/OFF, or OPEN/CLOSED). See openhab/items for a summary of valid values.

1. Statistics

- You can place the ROS_Status item in your OpenHAB sitemap. It will display the time of the last update (to the nearest minute) and counts of messages and errors.
- Place the following in demo.sitemap:

```
Text item=ROS_Status label="ROS [%s]"
```

Set status for an OpenHAB Item ROS topic subscribed by iot_bridge:

```
//iot_set (diagnostic_msgs/KeyValue)//
```

When the iot_bridge receives a name/value pair from the ROS iot_set topic, it publishes those to OpenHAB, and OpenHAB updates the status for the item specified (e.g. indicate that a switch is now ON). For instance: A ROS program running Facial Detection detects that Sarah is present. It publishes the following to the iot_set topic:

- message type: diagnostic_msgs/KeyValue
- key: "facedetection" - key must match openhab's .items file
- value: "Sarah" (string)

This will set the facedetection item in OpenHAB to Sarah, indicating Sarah has been detected. See sample code in iot_test. **Receive item updates from OpenHAB** ROS topic published by iot_bridge: *iot_updates (diagnostic_msgs/KeyValue)* The IoT bridge receives updates from OpenHAB and publishes those as name/value pairs to the iot_updates ROS topic.

To see updates from the command line, type:

9. Emerging technologies in IoT

```
rostopic echo iot_updates
```

For example, A motion detector is triggered in OpenHAB. The openhab bridge will publish the following to the iot_updates topic in ROS

- message type: diagnostic_msgs/KeyValue
- key: "office_motion" - key matches openhab's .items file
- value: "1" (string)

NOTE - device states are ONLY published to ROS when they change (or when iot_bridge is started). If you need the current state for ALL items in the ROS group, use the Request Refresh interface ²¹³⁾.

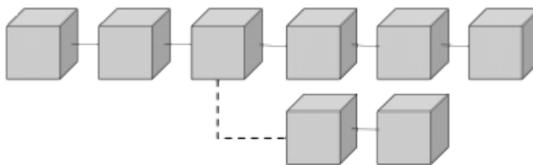
openhab will send the current status of every item in the (ROS) group to the iot_updates topic. The following status will continue to send only when they change.

9.2. Autonomous transport systems

In the future cities, IoT will play a crucial role in connecting things. The new paradigm in transportation is going to be happening in the near future. Mobility as a Service (MaaS) or Transportation as a Service (TaaS) and connected vehicles are primary aspects of a new paradigm. It means that instead of using the car like today, but, as with existing ride-hailing or car-sharing services, vehicles will be booked or hailed through an app. But not only personal transportation but also cargo transportation is changing significantly. The leading technology behind this is the software-driven autonomous systems, which has been come to intelligent enough to start replacing the human driver or operator. The future of the automotive ecosystem is heading toward the integration of multiple domains, such as the Internet of Things (IoT), connected vehicles, autonomous vehicles, and V2X communication. The integration and shift in focus will make software-defined cars to be the essence of connected vehicles in the future. Moreover, as the ecosystem expands, the market would notice extreme competition from both proprietary and open source solution providers.

9.3. Blockchain

News about the launch of new cryptocurrencies has been circulating for several consequent years already. And while the core of debates is the most promising, effective and economically viable model for reaching mutual understanding of all parties, blockchain based systems have been actively developing for some years. Nevertheless, in the Russian-language literature, and surprisingly, on the Internet (video conferences do not count), the information on blockchain in general. And articles on examples of its application in particular boil down to a dry description of the differences between the new protocol and the most famous one - Bitcoin. Sometimes this information is inclined to own advertising product or trying to describe an individual part of a decentralised network working on the blockchain. At the same time, most readers who are not familiar with the basic principles and concepts of this rather widely applied field, are often bewildered by the very talk about blockchain and the possibilities of its application outside the cryptographic currencies. Comprehensive articles aimed at explaining the principles of blockchain and its application started appearing only recently.



Using blockchain technology in IoT provides some advantages:

- Decentralized systems
- Opportunity to create a net of different IoT devices (e.g. sensory market). We can at any time attach a new device to the created net.

Moreover, any person could attach his sensors to the created net without any additional authentication and to share its data. This data could be bought by someone who is interested in.

- Therefore, attaching the IoT device, we can become an economic agent and a node of the net.

The story of blockchain development is inextricably linked with the story of cryptocurrencies' mining. In many respects, Bitcoin is to be thanked for fundamentally new systems that started appearing and operating on the blockchain. The climax of this development at the moment is the emergence of the Ethereum system, which revolutionized views on blockchain itself.

Not so long ago Habrahabr.ru (the most significant Russian online IT community - Ed.) has featured an article with a brief description of blockchain and its comparison with the usual database²¹⁴⁾. In this series of articles we will briefly, but more thoroughly than in article attempt to describe the story of blockchain and the systems based on it, the principles on which they operate, and the possible areas of application of this technology, which are plenty.

9.3.1. In search of consensus

There were attempts to create an unregulated currency long before the appearance of Bitcoin. The first prototypes of electronic money were proposed back in the 1980's by David Chaum. He is most famous in the world of cryptography for inventing the so-called "blind signature", which enabled creating an electronic digital signature for a message without getting to know its contents. It was the algorithm on which the electronic currency he created was based.

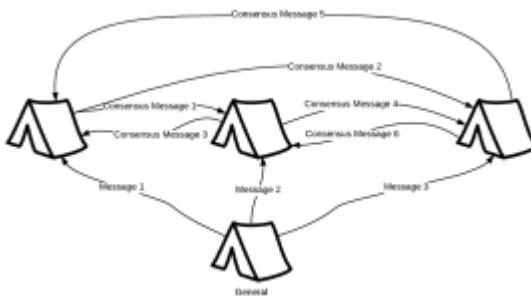
9. Emerging technologies in IoT

Another significant impact on the world was made by Adam Back's work on denial of service counter-measure. Thanks to this work the first mechanism for reaching consensus was elaborated: Proof-of-Work, which was named Hashcash. Further on we will tell about it in details.

Studies conducted by Wei Dai and Nick Szabo are just as important here. These studies aimed at identifying true information when interacting in an unreliable environment where there is no reason to trust any remote nodes with which the connection was established. The situation described above got the name "The Byzantine generals' problem". It was formulated by the well-known American scientist Leslie Lamport. And it was he who proposed an ad hoc solution to this problem.

The essence of this problem simmers down to the following: once the Byzantine army was about to enter into a great battle with the enemy. The whole army was divided into N legions. A general was appointed as head of every legion. All generals are receiving commands from the commander-in-chief. The night before the battle, each of the generals gets a message from the commander-in-chief, which indicates what should be done at 10 am the next day: the order was either to attack or to retire. The problem lies in the fact that by that time Byzantium was in decline - and anyone, including the commander-in-chief, could turn out to be a traitor. In particular, if the commander-in-chief was the traitor, he could give different orders to different generals. So basically, there are three possible outcomes of the battle:

1. If all the legions attack, then Byzantium will win;
2. If all the legions retire, then Byzantium will, at least, keep its army;
3. If some part of the legions retires, and the other attacks, then the Byzantine army will be defeated - and this is precisely what the traitor was striving for.



Therefore, the generals are facing the problem of making a mutually agreeable decision. That is, reaching a consensus in an unreliable environment where it is impossible to trust unconditionally any of the interacting parties. The problem solution could be seen in Byzantine fault tolerance

The emergence of Bitcoin in 2008 signalled a full-on revolution in the field of electronic cash. Someone who uses the pseudonym Satoshi Nakamoto is the author of this system. However, his or their true identity (in case if it is a group of people) has not been established so far. By and large, as it was rightly noted, Bitcoin was built on the foundation of work done for a quarter of a century in the field of cryptography and absorbed all the best ideas it could at that time. Nevertheless, this system is extremely

limited in some respects, as it will be repeatedly stipulated further. For instance, it does not allow creating complex financial transactions (contracts in Ethereum's terminology), since its internal programming language does not allow the creation of cycles, and is rather limited in general. Later on, the MasterCoin system emerged on top of Bitcoin. It allowed users to create their currencies (tokens). Nevertheless, the very idea of this system is still a subject of discussion, and besides, it has not been fully implemented. However, Bitcoin's feasibility raises many issues. The fact is that the number of this currency's units that have ever been created (we will discuss in detail how it is done further on) is limited to 21 million. Fears of possible deflation are indeed justified: if the demand for electronic money grows, and the offer cannot be increased because of the particular characteristics of the protocol itself, the cost of each unit of this cryptocurrency will increase. This will lead to the unwillingness of crypto-currencies' holders to part with their savings in the hope of making even greater profits due to further rising costs of each crypto-currency unit. It will even further reduce the supply - and so on. The thoughts cited above are far from being a full-fledged economic rationale, but they express one of the possible ways of future developments.

In a fairly short period after the creation of Bitcoin, a large number of alternative cryptographic currencies were created based on its free implementation. Some cryptocurrencies have made minimal changes to the structure of Bitcoin - increasing the maximum number of coins ever created or completely removing this restriction. Others made significant changes. Peercoin, for example, went this way, creating alternative methods for protecting the internal data structure (blockchain). It started using the mechanism of Proof-of-Stake on a par with Proof-of-Work. Primecoin uses the search for Cunningham chains as Proof-of-Work, which, according to the author of the system, has exceptional scientific value. Namecoin creates a distributed database of matches "IP address - network name" (a DNS-server analogue) based on the blockchain.

There are many examples of systems based on Bitcoin. Nevertheless, the most promising system based on blockchain does not stem from Bitcoin. We are talking about the Ethereum system, proposed by Vitalik Buterin in 2013 and formally described by Gavin Wood a year later. Changes in the concept from the developer's point of view can only be described after considering a specific protocol. From the user's point of view, the ability to embed fragments of the program code for the Ethereum virtual machine (reminiscent of the Java virtual machine) in the blockchain was the innovation. And this innovation is subsequently executed by all nodes in the network when accessing from outside (this is how contracts are created and executed). In addition to that, Ethereum allows creating DAOs (Decentralized Autonomous Organizations), which are represented by a set of contracts in the system that implements the logic of the organization in the network: starting from the creation of inner currency, fundraising through the sale of shares and ending with the work of the elected board of directors.

We should briefly mention that Ethereum developers are currently discussing the prospects of transitioning to the so-called Web3 - a new Internet, built by blockchain and implementing the interaction between non-trusted nodes in an unreliable network, as it was repeatedly stated earlier. There are some individual applications based on such a system: namely cryptographic currency, Whisper - a messenger based on such P2P network, as well as Swarm - an application for storing data in the blockchain.

9.3.2. Mechanisms of reaching consensus

We should separately focus on mechanisms of reaching consensus in an unreliable

9. Emerging technologies in IoT

network with unfamiliar nodes. Earlier we mentioned that long before Bitcoin and any other electronic currency sustainable over time emerged, the Hashcash scheme was created, which determined the Proof-of-Work mechanism. Currently, this mechanism is the most common and widely spread one. It is used in parallel with the Proof-of-Stake mechanism in some electronic currencies (Peercoin). And we may fully switch to it in the future if it can overcome the restrictions imposed on it now and defend itself against attacks.

First of all, let us discuss Proof-of-Work because this is the mechanism used in Bitcoin. It is directly related to the mechanism of the cryptographic hash-sum, which was discussed here earlier. We have mentioned that one of the main structural principles of hash-sums was and remains as follows: there should be no computational possibility (other than a complete enumeration) to restore the original message by a certain hash-sum. Roughly speaking, this fact is used in Proof-of-Work.

Each node of the network that creates blocks (or mines them in Bitcoin and other cryptocurrencies' terminology) creates a new block and fills its body - generally, we are not interested in how exactly it does it. The main thing is that the other nodes consider its content correct from the point of view of a particular protocol. Then the header is filled in, in which one of the fields (nonce) essentially contains a random value. The task of the node is to select such a header hash-sum value that it is less than a certain predetermined value. This value is called complexity and varies over time. For example, in Bitcoin, the complexity is maintained at the level necessary and sufficient for a new block to appear once every 10 minutes. The calculation is extremely simple: it is enough to calculate the number of new blocks appearing in a certain period and divide it by this time interval, thus receiving the speed of new blocks' appearance. Comparing this value with the required one you can "adjust" the complexity to increase or decrease the speed. Since all nodes in the network perform such an action, the system remains consistent: blocks with an incorrect value of the achieved complexity will simply be discarded.

Further on, the work of the node is basically an enumeration of hash-sums by changing the nonce field. We should mention that the drawback of the Bitcoin system and its derivatives are that the speed of computation has a significant influence on the speed of generating blocks to a great extent. At the same time, users of the system used and continue using various tricks to increase the likelihood of success. At first, they transitioned to calculating hash-sums on video-cards (thanks to technologies like NVIDIA CUDA). Next transition was to the use of specialized ASIC boards, and later on - to creating data centres. Presently, an ordinary user can not qualify for successful mining in the Bitcoin system and its derivatives, simply using his home PC.

The Ethereum system offers one obvious way to solve this problem. A special structure weighing about 2 GB called DAG is used to generate new blocks. This structure is created in advance. Its advantage is that once it is installed the use of ASIC boards becomes impossible without their significant improvement, which can not pay off if we keep in mind the current cost of RAM modules and all the required changes. It also makes no sense to create full-fledged pools consisting of "weak" machines with insufficient RAM capacity, when each node checks nonce, starting with a specific value so that the speed of calculating the hash-sum increases in proportion to the number of participants.

Proof-of-Stake is an even more promising, but still not entirely safe way to abandon Proof-of-Work. It appeared as a response to the public's displeasure with the costs of electricity and equipment required for the new blocks' mining. The idea is the following.

Each node on the network (in this case, we are dealing with a cryptographic currency) has accumulated certain savings. You can reduce the cost needed to generate a new block by the node, allowing it to include a transaction that transfers its funds to its account in the blockchain. Reduction of complexity, in this case, depends on the "age" of the funds used (the moment when they were received) and their quantity. And as a result of all this we get the following consequences:

1. The node that created the block is not interested in blockchain being addressed since it receives a fee from the transferred funds;
2. The cost of mining is reduced, as complexity is reduced, and hence the number of hash-sums' enumerations is also reduced;
3. A node can not use all the same tools for mining of sequential blocks, as their age after the transaction is less than the age of system's other participants' savings.

There is one problem that is not obvious at first sight. The attacker can purchase private keys for some UTXO (Unspent Transaction Output) from the users of the system, namely, keys used by them in the past. As a result of all this, there may arise a situation when the node will be in possession of absolutely all private keys at some point in the past. Then it will be very easy for it to quickly generate an absolutely new blockchain, which will replace the original blockchain and will be accepted by the system. And all of it can be done just by using its own means. As a protection against such an attack, a time frame is established for the age of the tools used in all systems implementing Proof-of-Stake.

9.3.3. Ethereum: the new generation of Internet

Ethereum takes a special place in the array of technologies based on the blockchain. Vitalik Buterin invented this system in 2013. Several important features distinguish it from all the previous systems:

1. Each block stores not only the sequence of transactions but also the current state of the system in the form of links;
2. Mining requires DAG generation;
3. A fully featured programming language, Turing complete (able to create cycles) and the Ethereum Virtual Machine for executing the bytecode are introduced;
4. Thanks to the existence of its own programming language, it is possible to create "smart contracts" - pieces of program code that "live" in the blockchain and is consistently executed by nodes when called from transactions;
5. Creation of DAOs (Decentralized Autonomous Organizations) on the basis of blockchain is also possible due to the availability of contracts. This way it becomes possible to implement the logic of their financial interaction: issues of shares, elections of the board of directors, etc.

Based on Ethereum, it is possible to create a new generation of Internet - Web3 in the future. It entails development of three areas: Ethereum - a cryptographic currency, Whisper - a chat based on Ethereum P2P-network and Swarm - a P2P-system of decentralised data storage. They are used to create Dapps - decentralised applications that use the Ethereum API to interact with blockchain.

9. Emerging technologies in IoT

9.3.4. Conclusions

Decentralized technologies are one of the most promising areas for the development of contemporary networks. And blockchain does take a special place among such technologies. As we have mentioned here more than once, Ethereum remains the most promising system based on the blockchain. Perhaps it will be blockchain that will allow us to create an absolutely anonymous Internet, the interaction where yet will remain safe, but protected from tracking. The Internet community strives for that. Nevertheless, the issue of security in the real world remains vital: terrorist organisations can also interact, using these technologies - and because of their resistance to hacking and absolute uncontrollability, the special services' work aimed at preventing terrorist acts will become more complicated. Therefore, the question of applying blockchain in its original form is controversial. One thing we can be certain of, though: blockchain changed the notion of a decentralised interaction and created the basis for its future development.

- 1) "ITU Internet Reports 2005: The Internet of Things." <http://www.itu.int/osg/spu/publications/internetofthings/>
- 2) "Special Report: The Internet of Things", in "the institute", IEEE 2014, <http://theinstitute.ieee.org/static/special-report-the-internet-of-things>
- 3) "Towards a definition of the Internet of Things (IoT)", IEEE 2015
- 4) Standard for an Architectural Framework for the Internet of Things (IoT) <http://grouper.ieee.org/groups/2413/>
- 5) ,⁸⁾ Ovidiu Vermesan, Peter Friess (eds.): Digitizing the Industry, Internet of Things Connecting the Physical, Digital and Virtual Worlds, River Publishers Series in Communications, 2016
- 6) Vision and Challenges for Realising the Internet of Things, CERP-IoT 2010, http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf
- 7) Salim Elbouanani, My Ahmed El Kiram, Omar Achbarou: "Introduction To The Internet Of Things Security. Standardization and research challenges", 2015 11th International Conference on Information Assurance and Security (IAS), IEEE 2015
- 9) Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, Moussa Ayyash: Internet of Things: A Survey on Enabling Technologies, Protocols and Applications, IEEE Communications Surveys & Tutorials, Volume: 17, Issue: 4, 2015
- 10) ,¹¹⁾ Arslan Munir, IFCIoT: Integrated Fog Cloud IoT Architectural Paradigm for the Future Internet of Things, IEEE Consumer Electronics Magazine, Vol. 6, Issue 3, July 2017
- 12) S.Matthews at <http://www.ibmbigdatahub.com/blog/what-cognitive-iot>, Cited: 11.06.2018.
- 13) <https://www.arduino.cc/>
- 14) ,⁴⁶⁾ <https://create.arduino.cc/projecthub>
- 15) <https://www.raspberrypi.org/>
- 16) <https://www.coursera.org/>
- 17) <https://www.edx.org/>
- 18) <https://eu.udacity.com/>
- 19) <https://www.udemy.com/>
- 20) <https://www.skillshare.com/>

- 21) <https://www.electronics-tutorials.ws/>
- 22) <https://www.instructables.com/howto/iot/>
- 23) <https://www.tinkercad.com/circuits>
- 24) Cloudonomics: The Business Value of Cloud Computing
- 25) IoT data semantics
- 26) Top 10 IoT security challenges
- 27) IOTA: A permissionless distributed ledger for a new economy
- 28) https://www.researchgate.net/publication/273389706_Towards_a_smart_city_based_on_cloud_of_things_a_survey_on_the_smart_city_v
- 29) , 31) , 33) <https://hal.archives-ouvertes.fr/hal-01581127/file/2016-TE2016-Taxonomy-for-IoT-Sensors.pdf>
- 30) https://www.w3.org/WoT/IG/wiki/Use_cases_across_application_domains#Use_Cases_and_Applications
- 32) <http://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/IoT-as-a-solution-for-precision-farming>
- 34) <https://news.panasonic.com/global/topics/2015/44009.html>
- 35) <https://www.fitbit.com>
- 36) <http://www.businessinsider.com/wearable-technology-iot-devices-2016-8>
- 37) Internet of Things: Security Vulnerabilities and Challenges Ioannis Andrea, Chrysostomos Chrysostomou, George Hadjichristofi, The 3rd IEEE ISCC 2015 International Workshop on Smart City and Ubiquitous Computing Applications, <https://doi.org/10.1109/ISCC.2015.7405513>
- 38) Rajan Arora, I2C Bus Pullup Resistor Calculation, Texas Instruments Application Report
- 39) <https://www.maximintegrated.com/en/products/digital/one-wire.html>
- 40) <https://www.arduino.cc/en/Guide/Introduction>
- 41) <http://forum.arduino.cc/>
- 42) , 112) http://www.electronics-tutorials.ws/io/io_3.html
- 43) , 114) <https://www.engineersgarage.com/articles/humidity-sensor>
- 44) , 116) <http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/>
- 45) <http://wiki.seedstudio.com/Grove-GPS/>
- 47) https://create.arduino.cc/projecthub/jose-cruz/arduino-home-controller-activated-by-alexa-1a5fbc?ref=platform&ref_id=424_trending___&offset=11
- 48) <https://www.amazon.com/b/?node=14047587011>
- 49) https://create.arduino.cc/projecthub/aaronkow/iot-home-security-model-71e48e?ref=platform&ref_id=424_popular___&offset=1
- 50) https://create.arduino.cc/projecthub/carmelito/plant-monitoring-system-using-aws-iot-6cb054?ref=platform&ref_id=424_popular___&offset=21
- 51) https://create.arduino.cc/projecthub/ahmedismail3115/arduino-based-amazon-echo-using-1sheeld-84fa6f?ref=tag&ref_id=iot&offset=4
- 52) https://www.hackster.io/arduino/projects?category_id=13
- 53) <https://www.hackster.io/order-of-the-bolt-nix/harry-potter-weasleys-clock-using->

9. Emerging technologies in IoT

bolt-iot-a7d04a

- 54) <https://www.hackster.io/LightPro/phonelocator-dfef67>
- 55) <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>
- 56) <https://learn.sparkfun.com/tutorials/how-to-solder-through-hole-soldering>
- 57) ,⁸⁶⁾ <https://www.arduino.cc/en/Main/Software>
- 58) <http://arduino.cc/en/Guide/Environment#languages>
- 59) <http://arduino.cc/en/Guide/Troubleshooting>
- 60) <http://arduino.cc/en/Tutorial/HomePage>
- 61) <http://arduino.cc/en/Reference/HomePage>
- 62) <https://www.arduino.cc/reference/en/language/functions/time/delay/>
- 63) <https://www.arduino.cc/reference/en/language/functions/time/millis/>
- 64) <http://harteware.blogspot.com/2010/11/protothread-and-arduino-first-easy.html>
- 65) <https://www.arduino.cc/en/Tutorial/DigitalPins>
- 66) https://www.espressif.com/en/media_overview/news/espressif-achieves-100-million-target-iot-chip-shipments
- 67) ,⁷⁸⁾ <https://www.espressif.com>
- 68) ,⁷⁰⁾ <https://en.wikipedia.org/wiki/ESP8266>
- 69) http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
- 71) <https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family>
- 72) ,⁷⁷⁾ <https://www.wemos.cc/>
- 73) ,⁷⁴⁾ ,⁷⁵⁾ ,⁷⁶⁾ <https://en.wikipedia.org/wiki/ESP8266>
- 79) https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- 80) https://espressif.com/sites/default/files/documentation/esp32-picod4_datasheet_en.pdf
- 81) <https://platformio.org/>
- 82) <https://code.visualstudio.com/>
- 83) <https://atom.io>
- 84) <https://micropython.org/>
- 85) http://nodemcu.com/index_en.html
- 87) <https://github.com/esp8266/Arduino>
- 88) <https://www.python.org/downloads/>
- 89) <https://github.com/espressif/arduino-esp32>
- 90) <https://git-scm.com/download/win>
- 91) <https://esp32.net/#Development>
- 92) <http://randomnerdtutorials.com/esp8266-web-server-with-arduino-ide/>
- 93) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>
- 94) https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-Zero-V1.3-Schematics.pdf
- 95) https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-A-Plus-V1.1-Schematics.pdf

- 96) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-B-Plus-V1.2-Schematics.pdf>
- 97) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-2B-V1.2-Schematics.pdf>
- 98) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-3B-V1.2-Schematics.pdf>
- 99) <https://www.raspberrypi.org/documentation/hardware/camera/README.md>
- 100) <https://www.raspberrypi.org/documentation/hardware/computemodule/cmio-camera.md>
- 101) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/README.md>
- 102) <https://www.raspberrypi.org/documentation/hardware/computemodule/cmio-display.md>
- 103) <https://www.raspberrypi.org/documentation/hardware/raspberrypi/usb/README.md>
- 104) <https://www.sparkfun.com/products/9190>
- 105) <http://www.tradesparq.com/products/1459651/Microswitch-SC7303-1-manufacturers>
- 106) <http://www.trossenrobotics.com/store/p/6445-5-Inch-Force-Sensing-Resistor-FSR.aspx>
- 107) <https://www.geekbuying.com/item/Digital-Capacitive-Touch-Sensor-v2-0-Switch-Module-for-Arduino-AVR-STM32-343414.html>
- 108) <https://www.digibay.in/hc-sr04-ultrasonic-proximity-sensor>
- 109) <http://hub360.com.ng/shop-2/pir-motion-sensor/>
- 110) <http://www.hotmcu.com/gy521-mpu6050-3axis-acceleration-gyroscope-6dof-module-p-83.html>
- 111) <https://www.sparkfun.com/tutorials/301>
- 113) <https://learn.adafruit.com/thermistor/overview>
- 115) <https://www.inventelectronics.com/product/dht11-temperature-humidity-sensor/>
- 117) <https://www.makerlab-electronics.com/product/digital-sound-sensor-detector-module/>
- 118) <http://www.circuitstoday.com/interfacing-mq2-to-arduino>
- 119) <https://www.sparkfun.com/products/retired/9891>
- 120) <https://www.2r-bg.com/blog/what-you-need-to-know-about-led-lights>
- 121) <https://electronics.stackexchange.com/questions/2574/arduino-hooking-up-lcd-without-pot>
- 122) <https://banlinhkien.shop/oled-0-96-inch-2>
- 123) <https://www.smart-prototyping.com/E-ink-E-paper-Display-module-3.3V-2.04-inch-177x72.html>
- 124) <http://www.vslot-europe.com/home/94-1-channel-relay-module-interface-board-shield-for-arduino.html>
- 125) <https://www.tlxtech.com/solenoids/long-stroke-latching-solenoid>
- 126) <https://www.amazon.in/Robo-India-DCMHBY-Hobby-Manual/dp/B00WOCYQT8>
-

9. Emerging technologies in IoT

- 127) <https://www.omc-stepperonline.com/hybrid-stepper-motor/nema-23-bipolar-18deg-126nm-1784ozin-28a-25v-57x57x56mm-4-wires-23hs22-2804s.html>
- 128) <https://www.raspbian.org/>
- 129) <http://ubuntu-mate.org/>
- 130) <https://www.ubuntu.com/core>
- 131) <https://developer.microsoft.com/en-us/windows/iot>
- 132) <https://osmc.tv/>
- 133) <https://libreelec.tv/>
- 134) <http://pinet.org.uk/>
- 135) <http://www.riscos.com/>
- 136) <https://www.raspberrypi.org/learning/hardware-guide/>
- 137) <https://uk.rs-online.com/web/c/computing-peripherals/data-storage-memory/secure-digital-cards/?searchTerm=noobs>
- 138) <https://thepihut.com/products/raspbian-preinstalled-sd-card>
- 139) <https://www.raspberrypi.org/downloads/noobs/>
- 140) <https://www.raspberrypi.org/downloads/raspberry-pi-desktop/>
- 141), 144) <https://www.microsoft.com/en-us/software-download/windowsiot?wa=wsignin1.0>
- 142) <https://etcher.io/>
- 143) https://www.sdcard.org/downloads/formatter_4/index.html
- 145), 146), 147), 148) <https://www.tutorialspoint.com/csharp/>
- 149) 11 Internet of Things (IoT) Protocols You Need to Know About, DesignSpark, <https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about>
- 150) <http://www.ieee802.org/15/>
- 151) <https://www.myo.com/>
- 152) https://en.wikipedia.org/wiki/Network_address_translation
- 153) <https://support.microsoft.com/en-gb/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained>
- 154) <https://earthobservatory.nasa.gov/Features/OrbitsCatalog/>
- 155) <http://web.mit.edu/modiano/www/6.263/lec22-23.pdf>
- 156) RFC 1631: <http://www.faqs.org/rfcs/rfc1631.html>
- 157) <https://en.wikipedia.org/wiki/ANT%2B>
- 158) <http://www.zigbee.org/>
- 159) <https://nodered.org/>
- 160) <http://www.restapitutorial.com/lessons/whatisrest.html>
- 161) <https://www.w3.org/TR/soap/>
- 162) https://www.w3schools.com/tags/ref_httpmethods.asp
- 163) https://en.wikipedia.org/wiki/Power_over_Ethernet
- 164) <https://www.techworld.com/data/what-is-li-fi-everything-you-need-know-3632764/>
- 165) What is Bluetooth, <https://www.bluetooth.com/what-is-bluetooth-technology/how>

it-works

- 166) <https://blog.bluetooth.com/introducing-bluetooth-mesh-networking>
167) https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=429633&_ga=2.211169244.858695694.1511814243-2000857715.1
168) Thread Stack Fundamentals, Thread group, 2015
169) <https://www.sigfox.com/en>
170) <https://www.lora-alliance.org/>
171) <https://en.wikipedia.org/wiki/IPv4>
172) <https://www.gartner.com/newsroom/id/3598917>
173) Jonas Olsson, „6LoWPAN demystified”, 2014, Texas Instruments
174) <https://www.hivemq.com/mqtt/>
175) <https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/>
176) R. Minerva, and A. Biru, “Towards a Definition of the Internet of Things,” in IEEE IoT Initiative White Paper
177) H. Reza Ghorbani, M. Hossein Ahmadzadegan, “Security challenges in the Internet of Things: a survey”, Wireless Sensors (ICWiSe) 2017 IEEE Conference on, pp. 1-6, 2017.
178) <https://www.welivesecurity.com/2016/12/30/biggest-security-incidents-2016>
179) Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, S. Shieh, “IoT security: Ongoing challenges and research opportunities”, Proc. IEEE 7th Int. Conf. Service-Oriented
180) A. C. Sarma, and J. Girão, “Identities in the Future Internet of Things,” in Wireless Personal Communications 49.3, 2009, pp. 353-363.
181) R. Roman, P. Najera, J. Lopez, “Securing the Internet of Things,” Computer, vol.44, no.9, 2011
182) https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project
183) H. Reza Ghorbani, M. Hossein Ahmadzadegan, “Security challenges in the internet of things: a survey”, Wireless Sensors (ICWiSe) 2017 IEEE Conference on, pp. 1-6, 2017.
184) <https://www.helpnetsecurity.com/2018/01/16/internet-of-things-security-issues-2018/>
185) <https://securelist.com/honeypots-and-the-internet-of-things/78751/>
186), 205) R. Khan, S. U. Khan, R. Zaheer, S. Khan, “Future Internet: The Internet of Things architecture possible applications and key challenges”, Proc. 10th Int. Conf. FIT, pp. 257-260, Dec. 2012.
187) H. Kumarage, I. Khalil, A. Alabdulatif, Z. Tari, X. Yi, “Secure data analytics for the cloud-integrated Internet of Things applications”, IEEE Cloud Comput., vol. 3, no. 2, pp. 46-56, Mar. 2016.
188) Se-Ra Oh, Young-Gab Kim, “Security Requirements Analysis for the IoT”, Platform Technology and Service (PlatCon) 2017 International Conference on, pp. 1-6, 2017.
189) H. Reza Ghorbani, M. Hossein Ahmadzadegan, “Security challenges in the internet of things: survey”, Wireless Sensors (ICWiSe) 2017 IEEE Conference on, pp. 1-6, 2017.
190) Amna Shifa, Mamoon N. Asghar, Martin Fleury, “Multimedia security perspectives in IoT”, Innovative Computing Technology (INTECH) 2016 Sixth International Conference on, pp.

9. Emerging technologies in IoT

- 191) Montenegro, G., Kushalnagar, N., Hui, J., & Culler, D. (2007). Transmission of IPv6 packets over IEEE 802.15.4 networks. *RFC 4944*, September 2007.
- 192) F. Skarmeta, J. L. Hernandez-Ramos, M. Moreno, "A decentralized approach for security and privacy challenges in the internet of things", Proceedings of the IEEE World Forum on Internet of Things (WF-IoT), March 6-8, 2014.
- 193) T. A. Alghamdi, A. Lasebae, M. Aiash, "Security analysis of the constrained application protocol in the Internet of Things", proceedings of 2nd IEEE International Conference on Future Generation Communication Technology (FGCT), Nov 12-14, 2013
- 194) Arijit Ukil, Soma Bandyopadhyay, Arpan Pal, "Privacy for IoT: Involuntary privacy enablement for smart energy systems", Communications (ICC) 2015 IEEE International Conference on, pp. 536-541, 2015, ISSN 1550-3607.
- 195) <https://epic.org/privacy/internet/iot/>
- 196) <https://www.networkworld.com/article/3221474/internet-of-things/30-ways-to-improve-iot-privacy.html>
- 197) Mary R. Schurgot, David A. Shinberg, Lloyd G. Greenwald, "Experiments with security and privacy in IoT networks", World of Wireless Mobile and Multimedia Networks (WoWMoM) 2015 IEEE 16th International Symposium on a, pp. 1-6, 2015.
- 198) F. Skarmeta, J. L. Hernandez-Ramos, M. Moreno, "A decentralised approach for security and privacy challenges in the internet of things", proceedings of the IEEE World Forum on Internet of Things (WF-IoT), March 6-8, 2014.
- 199) R. Roman, J. Zhou, J. Lopez, "On the Features and Challenges of Security and Privacy in the Distributed Internet of Things", Computer Networks, vol. 57, no. 10, pp. 2266-2279, 2013.
- 200) N. Li et al., "Privacy Preservation in Wireless Sensor Networks: A State-of-the-Art Survey", Ad Hoc Networks, vol. 7, no. 8, pp. 1501-1514, 2009.
- 201) R.H. Weber, "Internet of Things—New Security and Privacy Challenges", Computer Law and Security Rev., vol. 26, no. 1, pp. 23-30, 2010.
- 202) Soma Bandyopadhyay, Arijit Ukil, Chetanya Puri, Rituraj Singh, Tulika Bose, Arpan Pal, "SensIPro: Smart sensor analytics for the Internet of things", Computers and Communication (ISCC) 2016 IEEE Symposium on, pp. 415-421, 2016.
- 203) Jongmin Lee, Michael Stanley, Andreas Spanias, Cihan Tepedelenlioglu, "Integrating machine learning in embedded sensor systems for Internet-of-Things applications", Signal Processing and Information Technology (ISSPIT) 2016 IEEE International Symposium on, pp. 290-294, 2016.
- 204) Arijit Ukil, Soma Bandyopadhyay, Chetanya Puri, Arpan Pal, "IoT Healthcare Analytics: The Importance of Anomaly Detection", Advanced Information Networking and Applications (AINA) 2016 IEEE 30th International Conference on, pp. 994-997, 2016, ISSN 1550-445X.
- 206) A. Capossele, V. Cervo, G. De Cicco, C. Petrioli, "Security as a CoAP resource: an optimized DTLS implementation for the IoT", Proceedings of the IEEE International Conference on Communication, June 8-12, 2015.
- 207) wiki.ros.org
- 208) ros.org
- 209) openhab.org/features
- 210) , 212) , 213) http://wiki.ros.org/iot_bridge
- 211) <https://www.openhab.org/docs/>

²¹⁴⁾ <https://habrahabr.ru/post/313212/>



ITT GROUP
INNOVATSIOON - TEADUS - TEHNika





ITT GROUP
INNOVATSIOON - TEADUS - TEHNIKA



ITMO UNIVERSITY



RIGA TECHNICAL
UNIVERSITY

