System collaboration and information sharing through Internet of Things

Mälardalens Högskola

School of Innovation, Design and Engineering

Maja Grubisic
mgc14001@student.mdh.se

Tina Marsic
tmc14001@student.mdh.se

Thesis for the Degree of Master of Computer Science – Software Engineering
30.0 credits

June, 2015.

Examiner: Ivica Crnkovic

Supervisors: Hongyu Pei-Breivold
Mälardalens Högskola
Kristian Sandström
ABB AB, Corporate Research
Larisa Rizvanovic
ABB AB, Corporate Research
Abstract

The focus of this thesis is realization of system collaboration and information sharing between devices through Internet of Things. Internet of Things is a network of things, where a thing can be any device capable of acquiring an IP address. Internet of Things has been discussed in many domains. Companies are exploring the full potential of it, with the purpose of automating their services and optimizing their productivity. In this thesis we have conducted a systematic research review to investigate the existing Internet of Things solutions with respect to system collaboration and information sharing through Internet of Things. We have also implemented a prototype, using Arduino microprocessors and NUCs, to illustrate system collaboration and information sharing between the two systems.
# Table of Contents

1 Introduction .......................................................................................... 1  
1.1 Internet of Things ............................................................................ 1  
1.2 Motivation and research questions .................................................. 2  
1.3 Summary of research results ......................................................... 3  
1.4 Thesis outline .................................................................................. 3  
2 Related work ......................................................................................... 4  
2.1 System collaboration ......................................................................... 4  
2.2 Information sharing .......................................................................... 8  
2.2.1 Messaging patterns ..................................................................... 9  
2.2.2 DDS ....................................................................................... 10  
2.3 Cloud computing ............................................................................ 11  
2.4 Big data .......................................................................................... 12  
2.5 Internet of Things architecture ....................................................... 12  
2.6 Key characteristics of IoT ............................................................... 14  
2.7 Differences between IoT and M2M .................................................. 14  
2.8 Industrial IoT platforms ................................................................... 14  
2.8.1 mbed (ARM) ........................................................................... 15  
2.8.2 Oracle .................................................................................... 16  
2.8.3 Predix (General Electric) .......................................................... 17  
2.8.4 Vortex ..................................................................................... 17  
2.8.5 Intel IoT developer kit ............................................................ 18  
2.8.6 IoT SYS ................................................................................. 19  
2.8.7 Ayla Networks .......................................................................... 20  
2.8.8 Kombridge Things .................................................................... 20  
2.8.9 SeeControl ............................................................................ 20  
2.8.10 SiteWhere ............................................................................ 21  
2.8.11 IzoT platform (Echelon) .......................................................... 22  
2.8.12 Axeda ................................................................................ 22  
2.8.13 Arduino ............................................................................... 23  
2.8.14 Conclusions regarding Industrial IoT platforms .................... 23  
3 Research method .................................................................................. 25  
3.1 Identifying objectives ....................................................................... 25  
3.2 Identifying relevant publications .................................................... 25  
3.3 Assessing the quality of each publication ....................................... 28  
3.4 Validity discussion .......................................................................... 29  
4 Research analysis .................................................................................. 30  
4.1 Synthesizing the findings ............................................................... 30  
4.2 General statistics ........................................................................... 30  
4.3 System collaboration related publications analysis ........................ 31  
4.3.1 ROA based publications ......................................................... 31  
4.3.2 SOA based publications .......................................................... 38  
4.3.3 Ungrouped publications ........................................................... 46  
4.3.4 Quality attributes .................................................................... 48  
4.4 Information sharing related publications analysis .......................... 49  
4.4.1 Computer networking .............................................................. 50  
4.4.2 Protocols ............................................................................. 53  
4.4.3 Network service ...................................................................... 58  
4.4.4 Security ............................................................................... 59
## List of figures

Figure 1.1: Graphic illustration of IoT ................................................................. 1  
Figure 1.2: Gartner’s Hype Cycle, August 2014 .................................................. 2  
Figure 2.1: DIKW Pyramid [P4] ................................................................... 4  
Figure 2.2: TCP/IP model ............................................................................. 5  
Figure 2.3: Avahi Discovery Browser ............................................................... 8  
Figure 2.4: Client-server messaging pattern [28] .......................................... 9  
Figure 2.5: Publish-subscribe messaging pattern [29] ................................... 9  
Figure 2.6: How cloud computing works ....................................................... 11  
Figure 2.7: Generic IoT architecture ............................................................... 13  
Figure 2.8: IoT architecture [8] ..................................................................... 13  
Figure 2.9: mbed OS architecture [40] .......................................................... 15  
Figure 2.10: mbed Device Server’s Architecture [41] .................................... 16  
Figure 2.11: Oracle’s IoT platform [44] ........................................................ 17  
Figure 2.12: Vortex features [49] ................................................................. 18  
Figure 2.13: SeeControl supported protocols [63] ....................................... 21  
Figure 2.14: SiteWhere System Architecture Diagram [69] ......................... 22  
Figure 3.1: Graph visualization of number of publications ......................... 28  
Figure 4.1: Interaction between geotracker, its clients and the IoT gateway [P2] 32  
Figure 4.2: The service description (a) and the service artifacts (b) model [P3] 32  
Figure 4.3: Resource discovery by Registry Manager [P3] .......................... 33  
Figure 4.4: Smart Object system architecture [P5] ...................................... 34  
Figure 4.5: System architecture [P9] ............................................................. 36  
Figure 4.6: Conceptual diagram of device sociality [P10] ............................ 37  
Figure 4.7: Comprehensive device collaboration model [P13] ..................... 38  
Figure 4.8: DIAT architecture [P14] .............................................................. 39  
Figure 4.9: IoTCloud Architecture [P15] ....................................................... 40  
Figure 4.10: Multi-agent system architecture [P16] ....................................... 41  
Figure 4.11: Functional architecture [P19] .................................................... 42  
Figure 4.12: Implemented prototype, devices and architecture [P19] .......... 43  
Figure 4.13: MobilityFirst architecture [P20] ............................................ 44  
Figure 4.14: MVC system architecture [P21] ............................................. 45  
Figure 4.15: RaaS in cloud environment [P22] ........................................... 45  
Figure 4.16: Technical architecture [P23] .................................................... 46  
Figure 4.17: O2O architecture [P26] ........................................................... 47  
Figure 4.18: Models of channel operation ................................................... 51  
Figure 4.19: Network topologies [16] .......................................................... 52  
Figure 4.20: 6LoWPAN protocol stack diagram [22] .................................. 54  
Figure 4.21: MQTT-based IoT Architecture [30] ........................................ 56  
Figure 5.1: Schema of collaboration ............................................................. 61  
Figure 5.2: Implemented scenario .............................................................. 62  
Figure 5.3: Intel NUC ................................................................................. 63  
Figure 5.4: Picture of the Arduino Starter Kit components .......................... 64  
Figure 5.5: Arduino Uno with and without Ethernet shield ............. 64  
Figure 5.6: Patched kernel ...................................................................... 66  
Figure 5.7: Successful ping .................................................................... 67  
Figure 5.8: Unsuccessful ping ................................................................. 68  
Figure 5.9: Pidgin Bonjour account .......................................................... 68
List of tables

Table 2.1: Entities of DDS [30] ........................................................................................................10
Table 2.2: Examples of connections ...............................................................................................14
Table 2.3: Key benefits of OpenSplice [53] ....................................................................................18
Table 3.1: Results of the initial search ............................................................................................26
Table 3.2: Search results after limiting the year range ...................................................................26
Table 3.3: Results of title and abstract screening ...........................................................................26
Table 3.4: Number of duplicate publications selected by both persons ......................................27
Table 3.5: Results after title and abstract screening .......................................................................27
Table 3.6: Final number of publications for full text screening .....................................................27
Table 3.7: Final number of included publications .........................................................................29
Table 4.1: General statistics for the literature review ....................................................................30
Table 4.2: Comparison of IPv4 and IPv6 protocols [23] .................................................................55
Table 4.3: Comparison of TCP and UDP protocol [24] ..................................................................55
Table 4.4: CoAP architecture .........................................................................................................57
Table 4.5: Comparison of protocols [19] .......................................................................................58
1 Introduction

This master thesis was conducted at ABB Corporate Research Center in Västerås, Sweden. ABB, as a company operating within the automation domain, among others, has recognized the rapid change of devices and a significant increase in the number of devices connected to the Internet. Nowadays, devices are becoming smarter, affordable, more mature and more autonomous. Not so long ago, the only way to gain access to the Internet was through a computer - now we do it every day with our phones, tablets, even TVs and gaming consoles. Even though the number of devices connected to the Internet is growing by the day, the process of discovery and configuration of the devices is still very complex, and what the world strives to achieve is to reduce the complexity behind devices so that the new technology can be used to improve our lives and as a solution for overcoming daily challenges.

The goal of the thesis work is to investigate how to realize the collaboration and information sharing between devices, which should take us closer to the future possibility of replacing the traditional centralized control of field devices with distributed intelligent devices that are connected to the cloud – concept nowadays popularly called Internet of Things (IoT). These intelligent devices are supposed to communicate among themselves and with the environment by exchanging data and information.

This Chapter gives us an introduction to Internet of things concept in Section 1.1, then it explains the motivation behind this research – both research and businesswise in Section 1.2. In Section 1.3 summary of research results is presented, and Section 1.4 gives us an outline of the remainder of this thesis report.

1.1 Internet of Things

European Research Cluster on the Internet of Things defines Internet of Things as: “A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual “things” have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network” [1]. Basically the goal of the Internet of Things is to enable things to be connected anytime, anyplace, with anything and anyone using any path/network and any service [2].

We could observe IoT as a logical next step in the evolution of communication over the Internet. With the rise of wearable technology and intelligent devices and rapid increase of things connected to the internet, the term Internet of Things has gone viral. Nowadays, more and more devices are connected to the Internet, including devices we never would think of, such as TV sets, fridges, washing machines, etc., as demonstrated in Figure 1.1.

"On the Internet, no one knows you’re a toaster."

Figure 1.1: Graphic illustration of IoT
Prerequisites for communication of IoT devices are existing internet connection and unique identifier. Since every device that wants to communicate over the Internet has to have an IP address, IPv6 was launched in 2012 for the purpose of solving the exhaustion of IP addresses within the current IPv4 system. IPv6 is the common network layer and provides the required addressing space, allowing all devices to be accessible in the Internet [3]. For comparison, IPv4 uses 32-bit addresses and provides approximately 4.3 billion addresses while IPv6 uses 128-bit addresses making the total number of addresses we can get with IPv6 at least $7.9 \times 1028$ times more than with IPv4, which will be more than enough even with the predictions regarding the number of IoT devices in 2020 reaching as much as 50 billion devices [4].

It is estimated by Gartner that by 2020 there will be around 26 billion connected things that are communicating on the Internet, and by that time IoT product and service suppliers will generate incremental revenue exceeding $300 billion, mostly in services [5]. And as we can see from Figure 1.2, Gartner’s Hype Cycle which traces how different technologies progress from their innovation to their plateau of productivity, Internet of Things is at its peak of inflated expectations. Once the market settles into a reasonable set of approaches for IoT it will move beyond the peak.

IDC predicted in its Worldwide and Regional Internet of Things 2014-2020 Forecast that the global IoT market revenue would hit $7 trillion by 2020, and that the installed base of IoT units will grow to 28.1 billion in 2020 [6]. According to the worldwide survey conducted by The Economist in June 2013, of 779 senior business leaders around the world, 96% of them believe that in the next two years their business will be using IoT in some respect and 75% of them are already exploring the IoT. 61% of them believe that companies slow to integrate IoT will fall behind the competition, which is why many enterprise companies are preparing for IoT [7]. The Internet of Things is a global concept, not a single technology, and it provides solutions based on the synergy of hardware and software. Some of the main enabling technologies behind IoT are cloud computing and big data, and we will talk about them more in Chapter 2.

1.2 Motivation and research questions
The possible applications of Internet of Things are vast, justifying IoT as one of the key technology trends for the next 5 years. Some of the possible applications are [8]:

![Gartner's Hype Cycle, August 2014](Image)
• Environmental sensing that utilizes sensors to assist in environmental protection,
• Building and home automation,
• Management of manufacturing equipment,
• Manufacturing process control,
• Remote health monitoring.

After looking at the numbers regarding IoT future growth, and the possible applications of IoT, we can infer why ABB, as a multinational corporation operating mainly in power and automation domain, would like to have its share in the Internet of Things revolution. ABB’s Corporate Research Center in Sweden provides fundamental research, new technologies and innovative solutions for future products of the operating companies. Among others, it has a focus on software architecture, embedded systems, automation networks and wireless technologies. ABB as an organization has to be reactive and have an efficient, reliable and a global solution. IoT for ABB Industrial Automation can mean a simpler integration, deployment, monitoring and control of units on site which results in a faster response time and reduced down time for the customers therefore optimizing business process flow. IoT can also provide the interconnectivity and collaboration of units, so automating the processes while at the same time storing the data collected by the machines for analysis and real-time tracking of data.

IoT in the industry holds huge potential recognized by ABB but also its competitors like Siemens, Alstom, Rockwell, General Electric, and others. That being said, the motivation for this work is to investigate the possibility of using the Internet of Things to enable collaboration between distributed intelligent devices.

While conducting this thesis our research questions were:
• What are the existing IoT solutions from other domains and how are they applicable in industrial context?
• How to realize discovery and system collaboration between networking devices?
• How to realize information sharing between connected devices?

1.3 Summary of research results
1) We investigated IoT solutions from other domains and analysed their applicability in industrial context, refer to subsection 2.8.
2) A systematic literature review and analysis was performed in two areas, refer to subsections 4.3 and 4.4.
3) We defined system architecture for collaboration and information sharing.
4) We prototyped a demonstrator.

1.4 Thesis outline
The remainder of this thesis is structured as follows. In Chapter 2 related work is discussed. In Chapter 3 the research method is described, the literature review is conducted and the research results are provided. In Chapter 4 we conduct a research analysis of the research results stated in Chapter 3. Chapter 5 describes the process and tools used for the implementation part of this thesis. In Chapter 6 we outline ideas for future work, which is followed by Chapter 7 where we give our final conclusions.
2 Related work

In this Chapter we will reflect on the subjects relevant to this thesis. First we will start with system collaboration and information sharing in general. More details can be found in subsections 2.1 and 2.2. Then we will talk about cloud computing and big data. Big data and cloud computing can be used to store and analyse the data received from devices, since they are both well-established commercially to analyse large amounts of data in order to make timely decisions [P1]. Exchange of information is becoming faster, so the volume of data grows exponentially every day, and the main problem to be solved by big data and cloud computing is the challenge of treating their large volumes of data [9]. After that we will talk about the architecture of IoT, key characteristics of IoT, and the subsection after that will be about the difference between IoT and M2M, since they are often confused with one another. We will conclude this chapter with the comparison of some of the existing IoT platforms.

2.1 System collaboration

According to the BusinessDictionary\(^1\), collaboration is: “Cooperative arrangement in which two or more parties (which may or may not have any previous relationship) work jointly towards a common goal”, while system is: “an organized, purposeful structure that consists of interrelated and interdependent elements (components, entities, factors, members, parts etc.). These elements continually influence one another (directly or indirectly) to maintain their activity and the existence of the system, in order to achieve the goal of the system”. When talking about collaboration, it usually refers to collaboration among people, be it a collaboration over computer networks or face-to-face collaboration.

In this thesis, we are focusing on system collaboration, by which we mean collaboration of devices in the same network, where devices are able to discover one another without manual operator intervention, and are able to collaborate among themselves without the interference of people. There are different levels of collaboration, and this work will focus on the first level – getting the devices to be aware of the presence of the devices around them. To achieve the top level of collaboration the devices should be cognitive, able to assess the situation in which they are in, know how to solve the situation in the right way, and know how to behave. [P4] suggests that things in the Internet of Things should be presented as virtual entities with key features of a social intelligent entity. In Figure 2.1 we can see a Data Information Knowledge Wisdom (DIKW) pyramid presented by the same authors, where they introduce the knowledge management cycle for IoT devices, which, as stated before, are expected to generate a huge amount of data.

\[\text{Figure 2.1: DIKW Pyramid [P4]}\]

\(^1\) http://www.businessdictionary.com
To explain the pyramid in Figure 2.1:

- Data level: at this point things know nothing and are collecting raw data and sharing them through their IoT services.
- Information level: here things should already be situational-ware, and this level includes information gathered by analysing raw data.
- Knowledge level: at this level a problem is included or a situation that needs to be resolved. Here the things have the advantage of learning from previous experiences, be it their own or from other things.
- Wisdom level: this level includes reasoning techniques such as case-based reasoning and rule-based reasoning, which gives the things in IoT the ability to understand their situation and make decisions on their own.

This concept should give things in IoT the ability for automatic configuration and understanding high-level information. So, if we can see the devices, why can’t they see each other? For communication on the Internet we use Transmission Control Protocol and the Internet Protocol (TCP/IP)\(^2\). Combined, they provide end-to-end connectivity organized into four layers shown in Figure 2.2.

![Figure 2.2: TCP/IP model](image)

As we can see from Figure 2.2, the link layer of the TCP/IP model contains communication technologies, internet layer provides the basis for connecting hosts across networks, transport layer is responsible for host-to-host communication, while application layer provides data exchange. To be able to communicate over TCP/IP you have to have the four layers established. But what we want to do is connect things even when there is no working DHCP or DNS server, or when there is no connection to the Internet. This is where zero-configuration networking comes in [10].

**Zero-configuration networking**

Zero-configuration networking defines a group of technologies that automatically recognizes connected computers or network peripherals and creates a TCP/IP based usable computer network, without requiring any prior configuration – plug the device, turn it on, and it works. This kind of networking does not use central services like

---

\(^2\) "TCP is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent."

Domain Name Service (DNS)\(^3\) or Dynamic Host Configuration Protocol (DHCP)\(^4\), so it does not require special configuration servers or manual operator intervention.

The initial goal of zero-configuration networking was to improve network ease-of-use, that is to enable the communication over IP of two or more devices (e.g. laptops, mini PCs, etc.) just by connecting them with a crossover Ethernet cable – a type of Ethernet cable used to connect devices of the same type directly together. Apple, as a pioneer of this technology, managed to do that in the 1980s by connecting a group of MACs with a LocalTalk cable which represents the physical layer of the AppleTalk networking system. In the 1990s, it was done by connecting two computers via Ethernet, and now they are using built-in AirPort networking so there is no need to use cables anymore [11].

Zero-configuration networking seems like a rather convenient way to allow communication between everyday devices such as home appliances, laptops, mobile phones, cameras, etc. Why connect your smartphone with a USB cable or via Bluetooth to your computer just so you could transfer photos, music or videos? Why would you have to remove your digital photo frame from the wall and connect it to your computer so that you could change the picture on it? Why do you need to have appropriate software drivers, working DHCP server, working DNS server, or an Internet connection so you could connect two devices? Why not just have the devices in the same network, without any laborious configuration? And that is what we want to accomplish in this thesis, zero-configuration system collaboration and information sharing. Existing major implementations of zero-configuration networking are:

- **Bonjour**\(^5\) which is an implementation of Apple Inc. that comes built-in with Apple’s OS and iOS systems and can also be installed on computers running Microsoft Windows,
- **Avahi**\(^6\) is a free zero-configuration implementation for Linux and BSDs that implements IPv4 link-local, mDNS and DNS-SD. It comes built-in with most of the Linux distributions,
- **MS Windows Embedded CE**\(^7\) includes Microsoft’s implementation of Link-Local multicast name resolution protocol.

Zero-configuration networking is built on top of three core technologies [10]:

- **Link-local addressing** – assigning numeric network addresses for devices in the network,
- **Multicast DNS name resolution** – automatic distribution of computer hostnames,
- **DNS service discovery** – automatic location of network services (printers, cameras, speakers, gateways, etc...).

**Link-local addressing** can be compared with DHCP. We usually use DHCP to assign numeric network addresses to the devices which we want to use to access Internet. But,

---

\(^3\) DNS is an Internet service that translates domain names into IP addresses. Because domain names are alphabetic, they are easier to understand. [http://www.webopedia.com/TERM/D/DNS.html](http://www.webopedia.com/TERM/D/DNS.html)

\(^4\) DHCP is a protocol for assigning dynamic IP addresses to devices on a network. With dynamic addressing, a device can have a different IP address every time it connects to the network. [http://www.webopedia.com/TERM/D/DHCP.html](http://www.webopedia.com/TERM/D/DHCP.html)


\(^6\) [http://www.avahi.org/](http://www.avahi.org/)

when DHCP is not available, we have link-local addressing which lets the computer generate an address for itself so it can communicate with other computers on the network, even when there is no Internet connection. Since IPv4 addresses are still widely used we will use them in our examples. So let’s say that we want our device to have an IP address. We can assign it manually, use a DHCP server, or your device could self-assign it. When assigning addresses manually, we enter the 32-bit value in the form of, for example 192.168.1.15 and enter its subnet mask which indicates which portion of the address has information about the network and which about the host. The most common subnet mask is 255.255.255.0 and it indicates us that the network information is in the first 24 bits of the address. We could also merge the IP address and subnet mask in one field by writing 192.168.1.15/24, which indicates that the first 24 bits of the address are reserved for the network information. Of course, before choosing the IP address you should make sure that the chosen IP address is not already in use. When using a DHCP to assign addresses the device sends a request for an address and the DHCP server assigns him one from his pool of addresses, usually on a first-in first-out basis and allocating sequential IP addresses. When using zero-configuration networking, every device chooses its own IP address and verifies that the selected address is not already in use. Link-local address range is usually 169.254.0.1 – 169.254.255.254. Since there is no administrator to handle the list of available addresses, the devices must be able to handle possible conflicts and so the devices should choose the same address they chose the last time they connected to the network, unless it is already in use. To tell if an address is already in use we can use Address Resolution Protocol (ARP) which checks if an address is already in use by broadcasting ARP probe packets. After probing, the device claims the chosen IP address.

Multicast DNS (mDNS) is the step after link-local addressing. This technology is supposed to assign our numeric network addresses corresponding names. Names are easier for the common user to remember than a numeric address, and also the IP addresses of devices can change, so having a persistent and an unique name is important. Claiming a name for the device is similar to claiming an address for it – you choose a name, check if it is in use, if it is not you can claim it, and if it is already in use you choose another name and repeat the process. So now, to our 192.168.1.15 IP address we can give a name, let’s say Marvin, and our device will be known on the network as Marvin. Since Marvin is a node in our local network and we want him to be accessible from all of the devices in that network, we can see if he is alive by typing into our Linux terminal or Windows command prompt:

```
ping Marvin.localhost
```

Even when there is no underlying DHCP service you can give your device a name with mDNS and reach it from other devices using that name.

So now that we can refer to the devices by their names, how do we discover the devices in our network so that we are able to communicate with them without knowing their numeric address or name? For that we use DNS service discovery (DNS-SD). If we think about it in our everyday life we also use some sort of service discovery protocols, like for

---

instance Bluetooth Service Discovery Protocol and DHCP. DNS service discovery lets you discover other services in the network that are also implementing zero-configuration networking. DNS service discovery does not necessarily have to be done with mDNS; we can also use it with the traditional DNS.

Considering the fact that there are more and more devices plugged in and connected to the Internet, the goal is to have the simplest possible set up and use of those devices, otherwise people will not invest in them because they will not be willing, or have the possibility, to put the time and effort into making those devices work. The focus of DNS service discovery is not in discovering devices; it is in discovering services with which we can communicate, because in the end point you want to see a list of resources you can use for your benefit – be it printing, fetching music, photos, etc. So when you do a lookup of services in your domain the result you get will also contain service type name in the form of ServiceInstance.ServiceType.Domain, where ServiceType can be Transmission Control Protocol (TCP), User Datagram Protocol (UDP)\(^\text{11}\), HyperText Transfer Protocol (HTTP)\(^\text{12}\), etc. In Figure 2.3 we can see an example of DNS Service Discovery in Linux.

![Figure 2.3: Avahi Discovery Browser](image)

### 2.2 Information sharing

Broad definition of information sharing says that it is an activity through which information and knowledge are exchanged among entities (people, organizations and technologies) \(^\text{12}\). In computer networks, data exchange is allowed to computers along data connections, which can be wired or wireless, in the form of packets. Technologies that allow communication among devices of the same type are called Machine to Machine (M2M). Information sharing can be selected into four groups which are one-to-one, exchange between a sender and a receiver; one-to-many where one sender exchanges data with several receivers; many-to-one in which data from many senders.


has only one target, and many-to-many which implies that number of senders and receivers is greater than one [12]. Two methods of communication or messaging patterns can be considered when talking about sharing information between various devices. One is request-response, in which one device is the client and the other one server, and the second one is publish-subscribe model. Service which implements publish-subscribe model is data distribution service (DDS) and it is used for sending data between nodes and one of its features is enabling real-time distribution.

2.2.1 Messaging patterns

Request-response messaging pattern is a method of communication in which repair processes request received from the requestor. This method allows two-way conversation and is common in client-server architectures [27]. Client-server model of communication is based on tasks between clients who request some service and servers who provide that service. Clients in this model initiate communication while servers wait for incoming requests. The language and the rules of communication are defined in communication protocol and they are vital for enabling client-server communication. One server can communicate with one or more clients as it is shown in Figure 2.4, and once the request from some client is fulfilled the connection is terminated [27].

![Figure 2.4: Client-server messaging pattern [28]](image)

Another messaging pattern in networking is the publish-subscribe pattern, and Figure 2.5 shows its composition. Publishers (senders) and subscribers (receivers) in this method are not aware of mutual existence and publishers are not sending messages directly to receivers. Published messages are divided into classes and subscribers subscribe to one or more classes of interest [P50]. There are two ways of filtering messages which subscribers are supposed to receive, one is topic-based and the other content-based. In the topic-based way publishers publish messages under some topics, and subscribers will receive all messages under the topic they subscribed to. In the content-based way of filtering subscribers will only receive messages if their attributes or content match constraints subscribers defined [P50].

![Figure 2.5: Publish-subscribe messaging pattern [29]](image)
Between publishers and subscribers message brokers can be found, which are in charge of performing the filtering between two communication sides. Publishers post messages and subscribers register their subscriptions to message brokers which store and forward messages and can even prioritize messages in a queue before routing them.

2.2.2 DDS
Data distribution service is an Object Management Group (OMG) machine-to-machine data-centric middleware standard, which allows real-time data exchanges between publishers and subscribers for embedded systems. It simplifies network programming and implements publish-subscribe model for sending and receiving data among the nodes. Any node can be a publisher, subscriber or both [30]. In DDS, message brokers between publisher and subscriber is not used like it is shown in Figure 2.5, but instead they route messages based on the discovery of each other locally. DDS is used for applications in the field of financial trading, smart grid management, air-traffic control, and many other big data applications. “DDS is both language and OS independent. The DCPS APIs have been implemented in a range of different programming languages, including Ada, C, C++, C#, Java, JavaScript, CofeeScript, Scala, Lua, and Ruby. Using standardized APIs helps ensure that DDS applications can be ported easily between different vendor’s implementations” [51].

DDS provides scalability, performance, and Quality of Service required for supporting IoT. Real-time for DDS is measured in microseconds because devices for work and mutual communication need data much faster [31]. It implements device-to-device “bus” communication which controls data access and updates by big number of simultaneous users. DDS presents the technology that delivers reliability, necessary speed and flexibility to build real-time applications which are characterized by a high degree of complexity. The architecture of DDS consists of several entities which interoperate so that the system can work. In the Table 2.1 main entities of the DDS and their roles are presented [30].

<table>
<thead>
<tr>
<th>Entities of DDS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher</td>
<td>Responsible for distribution of publications</td>
</tr>
<tr>
<td>Subscriber</td>
<td>Responsible for receiving the distributed data</td>
</tr>
<tr>
<td>DomainParticipantFactory</td>
<td>Entry point to DDS</td>
</tr>
<tr>
<td>TopicDescription</td>
<td>Basic definition of the data</td>
</tr>
<tr>
<td>Topic</td>
<td>Specialized TopicDescription produced by Publisher</td>
</tr>
<tr>
<td>MultiTopic</td>
<td>Specialized TopicDescription, allows combining the data from several Topics</td>
</tr>
<tr>
<td>ContentFilteredTopic</td>
<td>Specialized TopicDescription, allows content-filtered subscriptions</td>
</tr>
<tr>
<td>DataWriter</td>
<td>Sets the values of the data to be published</td>
</tr>
<tr>
<td>DomainParticipant</td>
<td>Represents the participation of an application in one DDS Domain</td>
</tr>
<tr>
<td>DataReader</td>
<td>Permits the application to declare which data it wishes to receive and to access the data received by Subscriber</td>
</tr>
</tbody>
</table>

Features of DDS:

- Low overhead and efficient use of transport bandwidth
- Dynamically scalable and highly flexible. It can receive notification about all sorts of meta-events, such as new topics, publishers, subscribers, etc.
• Supports one-to-one, one-to-many, many-to-one and many-to-many communication
• Large number of configuration parameters and QoS policies that give developers control of message transmission and reception
• DDS is not well suited to the request-reply services, file transfer and transaction processing
• The data-distribution paradigm is not optimized for sending a request and waiting for a reply
• Lack of interoperability between different vendor’s implementations

2.3 Cloud computing
When looking at cloud computing from the IoT point of view we can consider it an infrastructure that is dynamic enough to support the data coming from IoT. Cloud for IoT can be the provider of storage, computing power, and networking. Did I leave my flat iron plugged in? Or my gas stove on? Maybe I could open the application on my smartphone or a web browser and check. Maybe I could even turn it off through the application! By combining technologies such as Cloud Computing, Big Data and robotics with Internet of Things we reveal its new potential. How cloud computing actually works can be seen from Figure 2.6.

In Figure 2.6 client computer represents the front end of cloud computing architecture, and everything else belongs to the back end, or the “cloud”. The simplest example of using cloud computing is checking your email online without having to install an email program on your computer. Also, services that offer cloud storage like Dropbox, OneDrive, GoogleDrive and Apple iCloud are becoming more and more popular. In Cloud Computing there are four different deployment models, each used depending on the organizational needs [34]:

1) Public cloud – “one in which the cloud infrastructure and computing resources are made available to the general public over a public network. A public cloud is
owned by an organization selling cloud services, and serves a diverse pool of clients.”

2) Private cloud – “gives a single Cloud Consumer’s organization the exclusive access to and usage of the infrastructure and computational resources. It may be managed either by the Cloud Consumer organization or by a third party, and may be hosted on the organization’s premises (i.e. on-site private clouds) or outsourced to a hosting company (i.e. outsourced private clouds).”

3) Community cloud – “serves a group of Cloud Consumers which have shared concerns such as mission objectives, security, privacy and compliance policy, rather than serving a single organization as does a private cloud. Similar to private clouds, a community cloud may be managed by the organizations or by a third party, and may be implemented on customer premise (i.e. on-site community cloud) or outsourced to a hosting company (i.e. outsourced community cloud).”

4) Hybrid cloud – “a composition of two or more clouds (on-site private, on-site community, off-site private, off-site community or public) that remain as distinct entities but are bound together by standardized or proprietary technology that enables data and application portability.”

2.4 Big data
Data is growing more and more every day. In year 2000 only the biggest and wealthiest companies had a terabyte of data in databases, now we have that much at home [35]. Big Data is a generally accepted term for any set of data so large or complex that it can’t be processed using traditional data processing applications. The idea is to bring computation to the data without having to move possibly even petabytes of data. When thinking about IoT and its use cases we want to have a solution that will be able to perform autonomous data-driven actions. Most of the IoT devices will generate some sort of data, be it machine data or sensor data, and that data is usually too large and complex for storing into traditional relational database and we need a way to figure out what all that generated data is telling us. IoT is able to collect that data, but it still needs to somehow convert the data into information, information to knowledge, knowledge to wisdom, which requires analytical processing skills. That is why being able to store big data in the cloud is a perfect solution for data that needs to be analysed – all of the analytical processing can be done in the cloud without worrying about the physical limits of computer infrastructure (e.g. bandwidth, disk space, etc.).

2.5 Internet of Things architecture
When speaking of IoT architecture the most common perception is visualized in Figure 2.7, but a more detailed architecture is given in Figure 2.8.
Figure 2.7 shows us a generic IoT architecture where IoT connects all sorts of objects that usually have different types of sensors and actuators and are able to collaborate over the Internet.

Figure 2.8 shows us the architecture of IoT divided into five layers:

1) Physical layer – consists of intelligent devices, sensors and actuators. Sensors can for example be thermometer, voltage detector, gas meter, gyroscope, tilt sensor, photo detector, etc. Actuators are the types of motors that can make a device act upon an environmental trigger. This layer collects the information from the sensors and forwards it to the Transmission layer.

2) Network layer – its task is to securely transfer information received from the Physical layer to Middleware layer, be it over wired or wireless technology.

3) Middleware layer – this layer receives the information from the Transmission layer and stores it, processes it and takes decisions based on the results of processing. It is important to emphasize that the devices that want to communicate over IoT have to have the same service type.
4) Application layer – this layer consists of smart applications like smart health, smart parking, smart lighting, smart grid, intelligent shopping applications, indoor air quality, etc. These applications are here to support management, automate tasks and act proactively.

5) Business layer – this layer manages the entire IoT system and builds business models based on the data received from the Application layer.

2.6 Key characteristics of IoT
We can distinguish five main characteristics of IoT [36]:

1) Sensing – things in IoT are able to sense changes and provide an output regarding those changes therefor recreating the experience of the physical world and facilitating the understanding of the real world.

2) Intelligence – devices, together with the accompanying software, provide the needed intelligence to make IoT product smart. For the devices to be able to act autonomously first they need to have the intelligence to act depending on the given circumstances, context, or data received from the environment.

3) Autonomy – indicates the ability of the devices to act without the intervention of third parties, based on the conclusions they get from the processed data.

4) Collaboration – in IoT collaboration is the ability of devices to work with one another to achieve shared goals.

5) Safety – it means the safety of the data transmitted in IoT and safety of the devices used in IoT.

2.7 Differences between IoT and M2M
IoT is a wider term and it includes M2M but also person-to-person and person-to-machine connections. Examples of each of these connections you can see in Table 2.2.

<table>
<thead>
<tr>
<th>Machine to machine</th>
<th>Person to machine</th>
<th>Person to person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart factories</td>
<td>Smart parking</td>
<td>Connected learning</td>
</tr>
<tr>
<td>Smart buildings</td>
<td>Mobile marketing</td>
<td>Telework</td>
</tr>
<tr>
<td>Smart grid</td>
<td>Video surveillance</td>
<td>Social media</td>
</tr>
</tbody>
</table>

M2M is defined as: “a communication paradigm that enables machines (sensors, actuators, robots and smart meter readers) to communicate with each other with little or no human intervention” [37]. Basically M2M enables IoT to extract the information from different devices so that it could then use that information and connections of M2M to provide a solution. And while M2M represents things connecting to things, with IoT things can also connect to people and different systems.

2.8 Industrial IoT platforms
To accomplish the development of the Internet of Things, in which network connectivity is enabled to different devices so they can exchange data and communicate, several requirements have to be met. Incorporating the IP protocol stack into smart objects and choosing the right communication protocols are some of them, but choosing the right IoT platform is also crucial.
IoT platforms provide a set of functionalities for building IoT projects. While comparing different platforms for IoT, their architecture, used protocols, and their different capabilities have to be taken into consideration. Security mechanisms, degree of innovation of a platform and the ability to enable real-time communication are also some of the criteria that should be satisfied. We have looked into the following platforms:

- mbed (ARM)
- ORACLE
- Predix (General Electric)
- OpenSplice (Vortex)
- INTEL IoT developer kit
- IoTSyS
- SeeControl
- Ayla Networks
- Kombridge Things
- SiteWhere
- IzoT (Echelon)
- Axeda

For the evaluation of the platforms, twelve features were taken into considerations. All of them could be divided into three groups which are:

- Security,
- Flexibility,
- Data requirements.

Additional information about platforms and a table with their selected features can be found in the Appendix 2. Additionally, the Arduino platform for developing Internet of Things projects will be presented.

### 2.8.1 mbed (ARM)

ARM, British multinational software design company, developed mbed, platform, and OS for Internet of Things. mbed OS is proprietary [43] full-stack operating system whose architecture can be seen in Figure 2.9. mbed OS is suitable for devices well-suited to run in energy-constrained environments. It provides a C++ Application Framework, and contains security, communication, and device management features that enable the development of energy-efficient IoT devices [40]. Some of the key benefits of mbed OS include connectivity protocol stack support for key standards like 2G, 3G, Bluetooth Smart, Ethernet, CDMA, and LTE cellular technologies, Zigbee IP, Zigbee NAN, Wi-Fi, and some others. It is supporting all of the key open standards for connectivity and device management, and a wide range of ARM Cortex-M based hardware platforms (32-bit micro-controllers). “mbed OS provides a C++ API Framework and component architecture that is used to create device applications, eliminating much of the low-level work normally associated with MCU code development” [40].

![Figure 2.9: mbed OS architecture [40]](image-url)
Another ARM product is mbed Device Server. It is a licensable software product that provides required technologies to connect and manage devices in a secure way and is characterized by built-in security management for constrained devices and networks [42]. “It utilizes open source protocols like CoAP, HTTP, MQTT, TLS/TCP, DTLS/UDP and OMALWM2M for data communication and device management” and connects protocols used by IoT devices and APIs that are used by web developers. Access to a big number of ARM IoT devices is enabled and device and application data management are unified using the same technology for client and server infrastructure [41]. mbed Device Server’s architecture can be seen in the Figure 2.10.

![Figure 2.10: mbed Device Server’s Architecture [41]](image)

### 2.8.2 Oracle
Oracle delivered an “integrated, secure, and comprehensive platform for the entire IoT architecture” [44] and its architecture can be seen in Figure 2.11. It uses Oracle Java Embedded programming language for developing IoT applications which enables code reuse and interoperability [46], and Oracle Identity Management to enable identity governance, access management and directory services with strong care of security. Oracle’s database security solutions are also used along with Oracle's platform technologies and infrastructure, as well as Oracle Mobile Platform, which simplifies enterprise mobility, allowing secure development, connection and deployment of applications across any data source.

ProSyst Software is designed as middleware for IoT which managed to operate around 4 million online Java embedded devices simultaneously on a four-node Oracle Exalogic Elastic Cloud. With Oracle IoT platform development of the applications became faster, with lower costs and enablement of managing and analysing big amounts of data. Main benefits of the Oracle IoT platform are using end-to-end security mechanisms and compatibility, enabling of real-time response capabilities among devices, and the possibility to integrate with IT systems and applications (from ERP and CRM to specialized custom and vertical applications) [45].
2.8.3 Predix (General Electric)
General Electric’s Software Center gave its contribution to the IoT area by developing Predix, platform for industrial internet of things which is scalable, supports high-volume analytics, industrial data and operational management, across individual machines and entire networks, on premise, in the cloud, or in a hybrid environment. Predix is marked as machine-centric, which means it connects, makes machines intelligent and optimizes them from anywhere in the network. Its architecture quickly delivers customer-grade industrial internet applications with a cloud-agnostic big data platform, and its security mechanisms protect data and control access to machines, networks and systems [48]. The key capabilities of big data software platform are:

- Data collection and aggregation from the wide range of industrial devices and software systems,
- Advanced analytics,
- Cloud-agnostic, which means that it runs on machines, in data centres, or in public clouds,
- Flexible deployment architecture,
- Extensibility and customizability which allow it to adapt to new capabilities, devices, data types and resources,
- Orchestration of all the components which need to interoperate effectively [47].

2.8.4 Vortex
PrismTech made its intelligent data sharing platform Vortex for business critical IoT applications which is suitable for areas such as healthcare, smart cities, industrial automation, transportation, and energy [49]. It represents leading implementation of the DDS standard, in commercial and Open Source field. Its data distribution service (DDS) enables real-time, scalable, reliable and interoperable data exchanges between publishers and subscribers [50], and it is language (Ada, C, C++, C#, Java, JavaScript, CoffeeScript, Scala, Lua, Ruby) and OS independent. One of the key benefits of Vortex platform is providing efficient, scalable, reliable, secure and interoperable real-time D2D, Device to Cloud and Device to Device via the Cloud data sharing. It is built on Open standards what enables application portability, interoperability and component reuse. It offers platform independent interoperable solutions for mobile, embedded and...
enterprise environments, and LAN or WAN deployment over private or public networks and clouds [49]. Features of the Vortex platform are shown in Figure 2.12.

![Figure 2.12: Vortex features [49]](image)

**OpenSplice**

OpenSplice DDS is a product developed by PrismTech which includes a middleware, an API and protocol for interoperability, and implements publisher and subscriber model. Key benefits of OpenSplice can be seen in Table 2.3.

<table>
<thead>
<tr>
<th><strong>Data Centric</strong></th>
<th>Enables applications to be designed around extensible data model, including end-to-end safety and efficiency.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real-time</strong></td>
<td>If the right information is not delivered at the right time and place it can lead to business or life threatening situations.</td>
</tr>
<tr>
<td><strong>Dependable</strong></td>
<td>Ensures availability, reliability, safety and integrity not related to hardware and software failures.</td>
</tr>
<tr>
<td><strong>High-performance</strong></td>
<td>Able to distribute very high volumes of data with very low latencies.</td>
</tr>
<tr>
<td><strong>Scalable</strong></td>
<td>Ability to function on simple systems and on very large scale system-of-systems, and from smart sensors to high end servers.</td>
</tr>
<tr>
<td><strong>Secure</strong></td>
<td>Maintains confidentiality, integrity and authenticity of exchanged data.</td>
</tr>
</tbody>
</table>

"It enables seamless, timely, scalable and dependable data sharing between distributed applications and network connected devices" [52]. OpenSplice is fast, scalable and reliable Open Source integration technology which can interoperate with other DDS implementations and support mobile, web and cloud data sharing. It has the richest set of QoS policies for controlling efficiency, determinism and fault-tolerance. OpenSplice enables data to be shared and integrated across a wide spectrum of operating systems and platforms [53].

### 2.8.5 Intel IoT developer kit

Intel launched IoT Developer Kit, which is a set of hardware and software resources developed for creating IoT solutions. It consists of hardware components which include
Intel® Galileo and Intel® Edison boards, shields, sensors, actuators, and software image and/or packages [39]. Various IDEs are supported allowing to program in [54]:

- C/C++ - Eclipse -> for Windows, Mac or Linux,
- JavaScript /Intel XDK IoT Edition -> using Node.js on-board apps and companion apps,
- Arduino,
- Wyliodrin - Visual programming (Galileo Only).

Cloud services for a data management and analytics are offered, along with the additional tools and solutions for optimization and performance which are:

- Intel System Studio for IoT is used for speeding development, testing and optimization. It supports Intel Quark (Galileo) and Intel Atom (Edison) with features such as debuggers (JTAG, using open OCD), Intel C++ compiler and performance libraries/tools. Intel System Studio is available with a non-commercial use trial license [55].
- Wind River VxWorks for Makers (Galileo Only) is modular and optimized operating environment for embedded devices and different types of applications. Medical, safety, security and industrial profiles can be added to core VxWorks. It is available with a limited non-commercial use trial license [39].

2.8.6 IoTSyS

IoTSyS is an integration middleware for the Internet of Things that provides communication stack for embedded devices based on IPv6, Web services and Open Building Information Exchange (oBIX) [56]. It enables information distribution and multi-protocol interoperability among heterogeneous devices. IoTSyS contributes novel protocol bindings for oBIX including CoAP, JSON and EXI, so the use of Web centric technologies for embedded devices can be more efficient in sensor and actuator systems [57]. For existing home and building automation technologies it is provided IPv6 based Web service interface which provides:

- OSGi based oBIX server,
- Protocol bindings for oBIX using CoAP, SOAP or HTTP for message transport and offering XML, EXI, schema informed EXI, JSON and oBIX binary encodings,
- IPv6 address for each device,
- Protocol bundles for KNX, BACnet, Wireless M-Bus.

IoTSyS is using end-to-end security mechanisms and data encryption, and can create IPv6 based control and monitoring solutions with existing automation systems [58]. IoT related standards are:

- Constrained RESTful environments (CoAP),
- ETSI M2M,
- OMA Lightweight M2M Web objects,
- IPSO application profile,
- Extensible Messaging and Presence Protocol (XMPP) and Message Queuing Telemetry Transport (MQTT),
- IPSO application profile,
- Extensible Messaging and Presence Protocol (XMPP) and Message Queuing Telemetry Transport (MQTT).
2.8.7 Ayla Networks

Ayla Networks IoT is a cloud-based platform for connecting devices to the Internet. Its main components are:

- Ayla embedded agents,
- Ayla cloud services,
- Ayla application libraries [59].

With Ayla Networks, secure connectivity and data intelligence can be integrated into any device without significant modifications. It provides software agents, which enable end-to-end support, embedded in connected devices and mobile device applications [60]. Ayla Embedded Agents can be integrated in designs developed using Linux or other OS to enable cloud capabilities over any type of network media. All communications with Ayla Cloud Services use industry-standard, public-private key encryption and SSL. RESTful APIs enable Ayla-enabled products to be managed and controlled by trusted third-party applications. Ayla Networks is a scalable platform that enables managing data, devices, and multiple user types with role-based permissions. It combines software and networking technologies with cloud-based platforms and digital information services. The main functionalities of the platform are:

- No need to re-architect host processor code,
- Develop connected devices and apps in parallel,
- User behaviour analytics,
- Cloud-cloud and cloud-enterprise APIs,
- Multiple integration options,
- Device mapping,
- Rules-based automation [60].

2.8.8 Kombridge Things

Kombridge AB is company which develops IoT software solutions for companies whose goal is remotely monitor and control their connected devices. Their product, Kombridge Things, is a secure, scalable and robust platform for Internet of Thing [61]. It is a managed cloud service for connected products, with standards for security, reliability and simplicity, which can be monitored and controlled via the web, smartphone apps or business systems. It is proprietary scalable and flexible IoT service delivery platform provided as Platform as a Service (PaaS). It can work with wired (Ethernet, xDSL) and wireless (GSM, WiFi, ZigBee, Satellite) technologies and different operating systems [62]. Key features of the platform are:

- Monitoring – overview, map and detailed view of equipment status,
- Alarms – notifications via apps, SMS and/or email,
- Remote control – distance equipment control,
- History – list of events, alerts and actions.

2.8.9 SeeControl

SeeControl is the only no-coding Software-as-a-Service Internet of Things platform. It is a cloud agnostic platform, and it also supports many M2M and IoT protocols. For scaling each client’s connected service, SeeControl provides optional wireless connection, ERP modules and business management tools. Drag-and-drop approach is used to define connected data, its analysis and visualization [63]. SeeControl enables creating alarms, statistic and analysis for real-time and historical data. It delivers transactional and
unstructured data from any device using several technologies. Some of the other features of SeeControl are: using open APIs, and scalability and security which are tested for demanding environments [64]. SeeControl works with many protocols and different types of devices what is simplified in the Figure 2.13.

![SeeControl supported protocols](image)

**Figure 2.13: SeeControl supported protocols [63]**

### 2.8.10 SiteWhere

SiteWhere is an open source M2M integration platform which allows managing and organizing connected devices. It is integrated with MongoDB for smaller installations which do not require extreme scalability, and Apache HBase which supports big amounts of device data. SiteWhere depends on many supporting open source technologies for accomplishing its goals [65].

The information is protected by limiting access to it based on user management system and historical device event data is never deleted while location data is stored so it can be geospatially indexed and accessed [66]. The system is based on a core set of interfaces that allow new communication protocols and encoding schemes to be added and configured easily. HTML5 administrative application is provided which enables simple access and manipulation of the data. Core platform Technologies are Apache Tomcat 7, Spring Framework and Security (for user access to REST services), MongoDB, Apache HBase and Solr, and Hazelcast. Mule, enterprise service bus that makes it easy to integrate various technologies is supported [66]. Mule has the concept of flows that pull in data from inbound endpoints, perform processing and decision logic, and route the resulting data to outbound endpoints. Mule Studio is an Eclipse-based user interface that allows graphical creation of Mule flows and their local execution. SiteWhere provides a plugin for Mule Studio that allows Mule flows to pull in SiteWhere data and integrate it with all of the other technologies Mule connects to [67]. SiteWhere system architecture diagram is shown in Figure 2.14.
American company Echelon developed IzoT platform that enables development of devices for the industrial Internet of Things. It is the open source platform which provides source code for a LonTalk/IP protocol stack for devices based on 32 and 64-bit processors, and server stack with the RESTful API for software development [68]. It supports communication with IzoT router and it supports wired and wireless communication. IzoT is compatible with Raspberry Pi with Raspbian Linux, and BeagleBone Black with Debian Linux, and supports complex controllers with support for up to 32,767 address table entries and simultaneous transactions [70]. Echelon provides three types of products for the IzoT Platform [71]:

- Chips, software stacks, and modules that communicate with the LonTalk/IP protocol. Options are provided for both wired and wireless connectivity, providing the flexibility for each application to choose the best media or a combination of media.
- Routers for interconnecting different IP links, and for serving Web pages and interfacing to Web applications.
- Software for providing management, monitoring, and control of IzoT-enabled devices for complex networks.

### 2.8.12 Axeda

The Axeda Platform is a complete M2M and IoT data integration and application development platform with infrastructure delivered as a cloud-based service. It is characterized with a high level of scalability and security, powerful development tools and flexible APIs. Axeda Platform enables communication with any wired or wireless asset and managing connected products for enterprise scale [72]. The Axeda Machine Cloud Service includes M2M and IoT connectivity services, software agents, and toolkits that enable establishing connectivity between devices or assets and the Axeda Platform. That results in connecting any product using any device over any communication channel for any application.
Axeda M2M Connectivity Services include three types of solutions depending on the class of device or asset that needs to be connected to [73]:

- **Firewall-Friendly Agents** - Software agents provide the core services to establish and maintain secure communication between Axeda platform and devices.
- **Wireless Agent Toolkits and protocols** - A Java or ANSI C library for embedding Axeda connectivity into devices that can be compiled into your own software and executed on a wide array of computing hardware and platforms.
- **Device Protocol Adapter** - A device communication server that connects to any M2M message protocol and can be extended with coder/decoders that translate the device’s native communication format into a form that the Axeda Platform can understand and process.
- **Policy Server** – It is a server-based application that provides set of permission settings that governs Axeda Agent behaviour for all devices at their locations.
- **MQTT** - The Axeda Machine Cloud ® supports the open source IoT protocol standard MQTT.

### 2.8.13 Arduino

Arduino is hardware and software open-source electronic platform intended for developing IoT projects on a simple microcontroller board. It is a tool for making computers with sensors which can sense and control the physical world [78]. A wide range of different kind of boards, shields, kits and accessories make the hardware part of Arduino platform. For programming, the Arduino platform provides an integrated development environment (IDE). Arduino Starter Kit is intended for everybody who wants to create his own project and it is also suitable for those who want to learn basics of programming and using microcontrollers. It comes with the book which includes 15 projects, Arduino board and wide range of accessories [77]. Features of Arduino board are:

- Inexpensiveness,
- Cross-platform,
- Easy-to-use programming environment,
- Open source,
- Extensible hardware [78].

### 2.8.14 Conclusions regarding Industrial IoT platforms

For evaluating different IoT platforms several features were taken into consideration. Since IoT is relatively new field, it is expected that platforms for developing IoT applications would also be newly developed or in the process of developing. According to that, for some platforms it is not possible to take into consideration former use cases as well as related successes or failures. While evaluating, impression about availability of customer support and assistance provided also played an important role. While using chosen platform users can come across a problem which requires help from customer support, and if help is not well-timed it can be an obstacle.

For more information about platforms, please refer to Appendix 2. In the security group of explored features almost all twelve platforms had satisfactory results. End-to-end security mechanisms and data encryption, as well as equal protection across multiple communication protocols are common to all platforms. In the flexibility group, which includes supported hardware, protocols and languages, and cloud agnosticism, various results were obtained. While platforms like Oracle, Predix, Axeda or OpenSplice support wide range of different hardware, others like mBed and Intel support only their own
devices what can be considered quite limiting, but can also have some benefits. Cloud agnosticism is confirmed for every platform except Axeda and SiteWhere, and some platforms support a wide range of programming languages while, for example, Oracle supports Oracle Java Embedded, and SeeControl stands out as a non-coding platform. Third and the last group of chosen features are data requirements in which big amount of information could not be obtained for specific platforms due to the lack of information on their web pages or from their customer support.

During the research some platforms have shown to be more or less adequate for the future work on a project. Intel, mBed, OpenSplice, SeeControl, Ayla and SiteWhere satisfied the most requirements. At the end of the research OpenSplice platform was chosen for the continuation of the work. Due to the fact that the 30 days free evaluation trial with the fast and reliable customer support was provided, OpenSplice was considered to be a good choice. Arduino platform was also chosen because of its features and ability to interact with the physical world.
3 Research method

One of the expected outcomes of this thesis was to perform a literature review [38] to investigate how to realize the system collaboration and information sharing between devices. To get an unbiased overview of the existing research in the field of Industrial Internet of Things a systematic research of the existing studies was done by two students, and it consisted of five steps:

1. Identifying our objectives and sources,
2. Identifying relevant publications,
3. Assessing the quality of each publication,
4. Synthesizing the findings,
5. Analysing the findings.

The following subsections explain the first three above listed steps of conducting a systematic research synthesis. Steps number four and five are explained and carried out in Chapter 4.

3.1 Identifying objectives

In this phase, we identify the need for doing the research, as well as the objective of the research itself, which is:

- How to realize the system collaboration and information sharing between devices through Internet of Things?

The resources used for conducting systematic research are:

- ABI/INFORM Global (http://search.proquest.com/abiglobal/)
- ACM Digital Library (http://dl.acm.org/)
- Google Scholar (http://scholar.google.se/)
- IEEE Xplore (http://www.ieee.org/web/publications/xplore/)
- ScienceDirect (http://www.sciencedirect.com)
- Scopus (http://www.scopus.com)

Mostly everything today is digitalized so we used only Internet-based information sources and they were chosen based on the subject area – Computer and Engineering Sciences. Also, only the primary studies written in English were chosen.

3.2 Identifying relevant publications

In this phase we have identified relevant publications from all of our sources based on meta-data screening; that is basing our search on title, abstract and indexing terms screening. In this step we have included publications regardless of their year of publishing, document type, source type or region. The keywords used to find relevant publications were:

- “Internet of Things” AND “device”,
- “Internet of Things” AND “M2M”,
- “Internet of Things” AND “Machine to Machine”.

Abbreviations like IoT (Internet of Things) or IIoT (Industrial Internet of Things) were not used because they were most often misinterpreted for Input-Output Tools, Input-Output Tables or Input-Output Techniques.
Since the initial search returned 3105 results, as seen in Table 3.1., we decided to narrow the search based on the year of publication. The oldest publication year in our initial search was 2002, but the year span we chose was from 2008 to present. Results of this step you can see in Table 3.2.

Table 3.2: Search results after limiting the year range

<table>
<thead>
<tr>
<th></th>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial search</td>
<td>102</td>
<td>656</td>
<td>54</td>
<td>534</td>
<td>113</td>
<td>1646</td>
<td>3105</td>
</tr>
<tr>
<td>Year limited</td>
<td>99</td>
<td>649</td>
<td>52</td>
<td>529</td>
<td>113</td>
<td>1628</td>
<td>3070</td>
</tr>
</tbody>
</table>

After narrowing the search based on the year of publication we can see that the difference in total search results is only 35 publications less, which confirms the assumption that before the year 2008 there wasn’t so much talk about the Internet of Things, for additional explanation please refer to Chapter 3.4. Next step in this phase was title screening. Title screening was done based on several exclusion and inclusion criteria. Inclusion criteria:

- Publications issued from 2008-present,
- Publications written in English,
- Publications related to research question.

Exclusion criteria:

- Publications not related to research question,
- Publications written in languages other than English,
- Duplicate publications,
- Publications available only in the form of abstracts or PowerPoint presentations.

Before the title and abstract screening all of the references have been imported to EndNote tool and screened for duplicates. 21 duplicates were found in ACM library and 8 duplicates in Scopus library, which is total of 29 publications less. While conducting the title and abstract screening all six digital libraries were screened separately in EndNote. Both title and abstract screening were done twice by two people separately to reduce bias, and the number of results for each person is shown in Table 3.3.

Table 3.3: Results of title and abstract screening

<table>
<thead>
<tr>
<th></th>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person 1</td>
<td>6</td>
<td>21</td>
<td>8</td>
<td>25</td>
<td>8</td>
<td>62</td>
<td>130</td>
</tr>
<tr>
<td>Person 2</td>
<td>13</td>
<td>23</td>
<td>12</td>
<td>41</td>
<td>23</td>
<td>58</td>
<td>170</td>
</tr>
</tbody>
</table>
The number of duplicate publications selected by both persons per library you can see in Table 3.4.

Table 3.4: Number of duplicate publications selected by both persons

<table>
<thead>
<tr>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicates</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>19</td>
</tr>
</tbody>
</table>

After completion of these steps the results were merged and then revised by two persons together. The publications that were selected by both persons were left for full text screening. Abstract screening also followed the listed inclusion and exclusion criteria. Results of both the title and abstract screening are listed in Table 3.5.

Table 3.5: Results after title and abstract screening

<table>
<thead>
<tr>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial search</td>
<td>102</td>
<td>656</td>
<td>54</td>
<td>534</td>
<td>113</td>
<td>1646</td>
</tr>
<tr>
<td>Year limited search</td>
<td>99</td>
<td>649</td>
<td>52</td>
<td>529</td>
<td>113</td>
<td>1628</td>
</tr>
<tr>
<td>Title screening</td>
<td>42</td>
<td>208</td>
<td>39</td>
<td>245</td>
<td>62</td>
<td>381</td>
</tr>
<tr>
<td>Abstract screening</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>32</td>
<td>9</td>
<td>57</td>
</tr>
</tbody>
</table>

After merging all the libraries in one EndNote library 17 more duplicates were found across libraries, so the total number of publications selected for full text screening was 117. The duplicates were found in Scopus which was expected considering that Scopus search also covers ACM and IEEE publications, with an exception of one duplicate between IEEE Xplore and Google Scholar.

Table 3.6: Final number of publications for full text screening

<table>
<thead>
<tr>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial search</td>
<td>102</td>
<td>656</td>
<td>54</td>
<td>534</td>
<td>113</td>
<td>1646</td>
</tr>
<tr>
<td>Year limited search</td>
<td>99</td>
<td>649</td>
<td>52</td>
<td>529</td>
<td>113</td>
<td>1628</td>
</tr>
<tr>
<td>Title screening</td>
<td>42</td>
<td>208</td>
<td>39</td>
<td>245</td>
<td>62</td>
<td>381</td>
</tr>
<tr>
<td>Abstract screening</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>32</td>
<td>9</td>
<td>57</td>
</tr>
<tr>
<td>After removing duplicates</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>31</td>
<td>9</td>
<td>41</td>
</tr>
</tbody>
</table>
### 3.3 Assessing the quality of each publication

In this phase a full text screening was conducted by two reviewers, also following the already listed inclusion and exclusion criteria. To get an idea of the general purpose of the article an overview was done by focusing on the introduction, first few paragraphs and the conclusion of each article. Notes were taken using Microsoft Excel. During the full text screening the quality of each publication was assessed. The quality of the publication itself relates to the extent to which the article minimizes bias and maximizes validity. The criteria used for assessment were:

- Is the purpose of the publication clearly presented?
- Are the results of the publication clearly presented?
- Is the publication focused on the area of interest/Does it relate to the research question?
- Was the publication written objectively?
- Does the publication include all relevant data?
- Is the publication well written and unambiguous?

A graph visualization of number of publications left after quality analysis is shown in Figure 3.1. The graph visualizes the total number of publications per library, at the same time visualizing the number of publications per library for information sharing and for system collaboration separately.

![Graph visualization of number of publications](image)

**Figure 3.1:** Graph visualization of number of publications

The final numbers of publications as well as results of every phase of our systematic research are visible in Table 3.7.
Table 3.7: Final number of included publications

<table>
<thead>
<tr>
<th></th>
<th>ABI/INFORM Global</th>
<th>ACM Digital Library</th>
<th>Google Scholar</th>
<th>IEEE Xplore</th>
<th>Science Direct</th>
<th>Scopus</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial search</td>
<td>102</td>
<td>656</td>
<td>54</td>
<td>534</td>
<td>113</td>
<td>1646</td>
<td>3105</td>
</tr>
<tr>
<td>Year limitation</td>
<td>99</td>
<td>649</td>
<td>52</td>
<td>529</td>
<td>113</td>
<td>1628</td>
<td>3070</td>
</tr>
<tr>
<td>Title screening</td>
<td>42</td>
<td>208</td>
<td>39</td>
<td>245</td>
<td>62</td>
<td>381</td>
<td>977</td>
</tr>
<tr>
<td>Abstract screening</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>32</td>
<td>9</td>
<td>57</td>
<td>134</td>
</tr>
<tr>
<td>After removing duplicates</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>31</td>
<td>9</td>
<td>41</td>
<td>117</td>
</tr>
<tr>
<td>Full text screening</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>16</td>
<td>4</td>
<td>15</td>
<td>51</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>51</td>
</tr>
</tbody>
</table>

3.4 Validity discussion

Naturally, there are always some concerns when it comes to identification of relevant publications. Did we choose the right sources? Was our search scope wide enough to ensure discovery of relevant information? Was it narrow enough to exclude irrelevant information? There will always be some doubts when it comes to answering these questions.

Why did we limit the search to publications written from 2008 onwards? Although the Internet of Things concept is not new, the year 2008 is broadly mentioned as the birth year for the Internet of Things. It is also the founding year of IPSO Alliance – group of companies deciding to promote the use of Internet Protocol in networks of “smart objects” and to enable the Internet of Things, including Bosch, Cisco, Ericsson, Intel, SAP, Sun, Google and Fujitsu. It is also the same year in which an increase in publications relevant for this research has been noticed, the year IoT got recognition from EU and the first European IoT Conference was held [75].

Since this master thesis was written by two students, the research conducted was unbiased – the initial search, title screening, abstract screening and full text screening with quality assessment were done separately by both students. In case of any disagreements regarding included literature, a discussion was held and an agreement has been made whether to include the publication in question or not. Despite this there might be some missed articles relevant to the research, but that articles maybe used different terms in their studies.

Our search terms, defined in subsection 3.2, were chosen so that they would be broad enough to capture all the relevant publications related to the Internet of Things but also system collaboration and information sharing. At the same time the search terms had to be narrow enough so we wouldn’t spend unnecessary time and effort on irrelevant publications. The databases chosen for the literature review, defined in subsection 3.1, were chosen because they cover the field of computer science in general.
4 Research analysis
The list of publications selected after the full text screening and the quality assessment is provided in the Appendix 1. In this chapter, we will show statistics of our selected publications and also a content analysis will be done after splitting the selected publications on the ones related to system collaboration and the ones related to information sharing.

4.1 Synthesizing the findings
Synthesis of findings was done after applying inclusion and exclusion criteria and conducting the quality assessment. It helps in getting a better overview of chosen publications and easier statistics extraction. Synthesis of the findings involves information extraction from all the primary studies selected. For the purpose of this step a table has been created with the following columns:

- Publication code – since listing the names of each publication and its authors would take a lot of space we have used a primary key P{1, 2, …. } and listed the publications in the Appendix 1,
- Publication type – book, workshop paper, journal paper, conference paper, etc.,
- Year of publication,
- Country,
- Affiliation – community within which the paper was published,
- Publication results – results after the study execution,
- Publication problem – e.g., limitations with which the author met,
- Notes.

After the full text screening phase the publications were divided into publications related to system collaboration and the ones related to information sharing.

4.2 General statistics
Table 4.1 shows us general statistics about the selected publications, for both system collaboration and information sharing publications. The statistics will observe the type of publication, country of origin of the publication, year of the publication, the conferences from whose proceedings we have found multiple publications, or the journals from which we have found multiple publications.

| Table 4.1: General statistics for the literature review |
|---|---|---|---|
| **Type of paper** | Conference paper | Workshop paper | Journal paper | Book chapter |
| | 29 | 6 | 15 | 1 |
| **Country** | China | France | Germany | USA | Italy | Other |
| | 4 | 4 | 4 | 4 | 3 | 32 |
| | 22 | 7 | 10 | 7 | 3 | 1 | 1 |
4.3 System collaboration related publications analysis

The publications selected for the system collaboration part of the analysis were selected with the criteria that the publication reflects on the problem of system collaboration and/or service discovery in the Internet of Things and proposes a solution. The selected publications were divided into two groups – publications based on resource-oriented architectural style of the system, and publications based on service-oriented architecture (SOA) of the system. Two of twenty six publications were not classified into any of those two groups.

4.3.1 ROA based publications

Resource-oriented architecture is a style of software architecture that enables collaboration between resources, where a resource is any entity that can be assigned a uniform resource identifier (URI). Resource-oriented architecture is considered Representational State Transfer (REST) architecture. REST is a software architectural style and systems based on REST usually communicate over the HTTP protocol [76]. In the following subsections an overview of the publications that are based on ROA will be given.


This paper proposes a set of requirements for achieving a pervasive, integrated information system of wireless sensor networks (WSN) and associated services. It also presents a set of architectural requirements, a resulting layered architecture and abstractions for the data exchange roles taken by services on WSN nodes and in the Cloud. It also evaluates an initial implementation of the architecture. The paper evaluates the proposed architecture by implementing it on the Contiki operating system using the CoAP protocol and HBase table for the storage of the data about sensors. An exchange of sensor information from the node to CoAP to HBase was also shown, independent of the underlying technology. Their implementation showed that the proposed architecture can be relatively easily implemented on both constrained WSNs and on more capable devices and applications.

[P2] A network architecture for the web of things

This paper presents the implementation of a gateway-based architecture for the Internet of Things. In the paper, a Geotracker prototype was built by attaching the SunSPOT sensor nodes to GPS and GPRS modules with the goal of querying the position of the Geotracker any time, even when the device is moving, sleeping, or when it is connected to the Internet via a private IP address. For the clients to dynamically discover the resources that are not known in advance, they propose the use of mDNS, and for the resources that are already known the approach is to use DNS-Based Service Discovery to get a list of resources available on a device or search for a specific resource. When the device becomes active it contacts the gateway and that way notifies it of its availability and establishes a channel for reaching the device if it is behind a Firewall. The gateway then forwards any queued requests to the device. The example of the interaction is shown in Figure 4.1.
This paper presents a REST inspired architecture to provide an end-to-end integration of front-end IoT devices with the back-end business process applications. As shown in Figure 4.2, the service is a set of resources that expose their functionality via interfaces. Service descriptions within the proposed architecture are written in a Web Services Description Language (WSDL) or Web Application Description Language (WADL6). A service description consists of a human-understandable name and a base URI containing host name and port number of the network resource in which the service endpoints are contained.

![Figure 4.1: Interaction between geotracker, its clients and the IoT gateway][P2]

**[P3]** A resource oriented integration architecture for the Internet of Things: a business process perspective

![Figure 4.2: The service description (a) and the service artifacts (b) model][P3]
Web services (WS) of this architecture were developed on the Contiki-based Tmote Sky nodes, and each one of those services provides access to resources through a RESTful API. After the deployment of RESTful WSs, the next step was to expose them so that they can be discovered at the Internet scale. Registry Manager, developed in Java, enables a REST server to be exposed as RESTful service. This is done in two steps: resource discovery and then automatic generation of service description. The former is achieved through Contikis built-in REST engine which implements the CoAP standard resource discovery mechanism: each REST server implements a default resource called .well-known/core (WC). A list of hosted REST resources is returned upon making a GET request to the WC resource. As depicted in Figure 4.3, RM enables resource discovery by first making an HTTP GET request to the WC resource of the 6lowPAN border router, which in response returns the list of attached devices. RM then sends a unicast GET request to WC resource hosted by each neighbour device to obtain the list of REST resources that the underlying REST server provides. The received response is further parsed by the RM to generate a service description in the form of a WADL file [P3].

![Figure 4.3: Resource discovery by Registry Manager [P3]](image)


This paper presents the approach that the COSMOS project\(^\text{13}\) introduces in order to enable Things in the IoT to act in a more autonomous way. The COSMOS Data Information Knowledge Wisdom Pyramid is already presented in subsection 2.1. COSMOS presents Things as Virtual Entities (VEs) that are described by ontologies and provided with social characteristics, and as such social entities they incorporate also social relations and interactions between themselves and that way via Social Network Analysis the patterns and network dynamics can be discovered and examined. The paper does not describe in details what protocols should be used to implement the described approach.

[P5] Adding sense to the Internet of Things

Event-driven Smart Objects architecture is presented in this paper, shown in Figure 4.4. The proposed architecture should be able to encapsulate radio-frequency identification (RFID), sensor technologies, embedded object logic, object ad-hoc networking, and Internet-based information infrastructure. The Smart Objects networking design enables the discovery of additional sensor sources and the automatic selection of the most appropriate data source from multiple data providers. The nodes presented in this paper are wireless sensor network nodes that are able to communicate with each other and form a ad-hoc network for better management of resources and data capture. The information infrastructure design was evaluated in terms of flexibility and was implemented with XML and JSON-based web services, REST and binary distributed

\(^\text{13}\) [http://iot-cosmos.eu/]
object paradigms. Smart Objects were implemented using the ANTS sensor network platform and each object/node featured a 8bit micro-controller, a 2.4Ghz transceiver, 128kb of Flash memory, 4kb of SRAM and a variety of sensors. The network created from these nodes communicated with the infrastructure via a gateway node.

![Smart Object system architecture](image)

**Figure 4.4: Smart Object system architecture [P5]**

**P6] An IoT Gateway centric Architecture to provide novel m2m services**

This paper proposes an IoT architecture that allows real-time interaction between mobile clients and smart things via a wireless gateway (WG). The proposed architecture should provide dynamic discovery of M2M devices, manage connection with non-smart things connected over Modbus, associate metadata to sensor and actuator measurements using Sensor Markup Language (SenML) representation and extend the current capabilities of SenML to support actuator control from mobile clients. The gateway is implemented using RESTful web services and authors implemented it to run on a Google App Engine (GAE).

The architecture proposed in this paper is composed of three layers: sensing layers that contain the M2M devices and endpoints, gateway API layer and application layer. The M2M devices and endpoints have to register themselves in the gateway to be available to the mobile clients. If they have no prior knowledge of the things they initiate the discovery phase by establishing a connection to the WG. The API necessary for the discovery phase is implemented in the gateway. After the discovery phase, the user can choose desired sensors to receive the measurements which are represented using SenML. The mobile clients are equipped with an application named “Connect and
Control Things” (CCT) that receives the measurements from sensors and can control actuators via the GAE.

**[P7] Adaptable service composition for very-large-scale internet of things systems**
The concepts of service orchestration and choreography are described in this paper, and an architecture that enables efficient integration of services by locally orchestrating distributed web-enabled services in very large scale (VLS) IoT systems and globally choreographing Web-based applications is presented. One of the challenges that authors identified was enabling the resource discovery for a large scale system and dealing with the interoperability issues of smart devices. They propose the use of service orchestration to integrate sensor data and use of choreography module of external services that provides a global view of the system and has the ability to invoke services to extend the cooperation between virtual and physical worlds. The orchestration schema is defined with Business Process Modeling Notations and translated into Business Process Execution Language code using Eclipse SOA Tools Platform, and the REST paradigm was used to exchange request/response messages between gateway node and smart devices.

**[P8] IoT interoperability: A Hub-based Approach**
A hub-based approach is presented in this paper with the purpose of aggregating things using web protocols. The authors also present the HyperCat specification which is a specification for lightweight hypermedia catalogue designed for querying and representing catalogues of resources (URIs) on the web. Because of its simplicity, applications can easily query the things they are interested in. In this architecture, each hub is responsible for interacting with applications and possibly other hubs. This paper presents two tools for achieving interoperability among things in IoT - Comprehensive Knowledge Archive Network (CKAN) data management system, and WoTKit which is focused on managing things that exhibit real-time behaviour. No details about the protocols used in the implementation are provided. With this architecture, the authors have implemented a Smart Streets IoT hub that, at that time, managed 64000 time-series sensor feeds and a variety of static data sets. After several hubs came online the authors found that the resource catalogues item semantics will have to be standardized.

**[P9] Keeping you in the loop: Enabling Web-based Things Management in the Internet of Things**
In this paper a IoT prototype system, that enables easy monitoring and aggregation of services and resources offered by things, was introduced. It implements a layered architecture, as shown in Figure 4.5.
The system has two ways to identify physical objects and connect them to the Web. The first one is to use RFID technology, where the physical objects are attached with RFID tags and interrogated by RFID readers. The second one is to attach sensors to objects for transferring the raw data to the network. A Sensor Hive module of the architecture manages sensors and tags, establishes connections and works in a plug-and-play manner. Virtual Resources module maps a collection of classes to their corresponding physical devices. The Event Processing module has three phases of processing the event: event detection, contextual information retrieval and event aggregation. The Event Detector captures and decides whether a physical thing is in-use. In the implementation presented in the paper, there are two ways to detect usage of things: sensor-based for detecting state changes and RFID-based for detecting mobility. In the sensor-based detection, the use of an object is reflected by changes of the objects status, while in the RFID-based detection, the movement of an object indicates that the object is being used. The Event Aggregator indexes and stores all the events and services, together with their related information. The rule engine module allows users to control a device automatically by setting up a series of basic rules.

**Towards zero-configuration in device collaboration using device sociality**

A zero-configuration method in device collaboration via self-constructing social relationships of devices is presented in this paper. A conceptual diagram of the proposed model is shown in Figure 4.6.
In device social network, devices may publish their profile and status, establish valuable and trustworthy relationships to satisfy tasks of the user, and are able to detect groups of common interests and the corresponding leader, that way creating social relations with one another.

**[P11] True self-configuration for the IoT**

This paper aims to show how we can include IoT devices in systems that combine local and remote data from different sources. It also describes the SPITFIRE project by the same authors. For the communication they are using 6LoWPAN and CoAP based protocol, semantic web technologies for meaningful data exchange, autonomous sensor correlation to learn about the environment, and software built around the Linked Data principles to be open for novel and unforeseen applications. For the machine-understandable and domain-independent representation of data they use Resource Description Format, and they link the data with Linked Open Data cloud. To avoid pre-configuration and hard-wiring of device addresses, they utilize automatic sensor correlation and smart self-annotation of devices. The proposed solution was shown on a building automation demo application. The installation of new devices in the room is done by placing new devices in the rooms after which they self-configure to find their location, form a network and appear in the graphical user interface. The demonstration will allow the visitor to understand how applications can listen to the environmental conditions, mine the Web for relevant data, develop an understanding of the real conditions and act upon them.

**[P12] A Scalable and self-configuring architecture for service discovery in the Internet of Things**

A scalable and self-configuring peer-to-peer (P2P)-based architecture for large scale IoT networks was proposed in this paper, aiming at providing automated service and resource discovery mechanisms, which require no human intervention for their configuration [P12]. The service discovery architecture proposed in this work relies on the presence of an IoT Gateway. The gateway provides nodes with service and resource discovery, interacts with nodes using CoAP protocol, and is naturally aware of the presence of the nodes and the available services and resources. When the IoT Gateway detects that a CoAP node has joined its IP network, it can query the CoAP node asking...
for the list of provided services which is then used to route incoming requests to proper resource nodes. Authors also propose the Zeroconf-based service discovery for service and resource discovery within local networks. The performance evaluation of the Zeroconf-based service discovery was conducted using Zolertia Contiki nodes, simulated in the Cooja simulator.

4.3.2 SOA based publications

Service-oriented architecture is a style of architecture where components of software provide services to other components via network. Protocols that SOA uses are usually Web Service Description Language (WSDL) to describe the services and Simple Object Access Protocol (SOAP) for the communication. In the following sub-sections an overview of the publications that are based on SOA will be given.

[P13] A comprehensive device collaboration model for integrating devices with web services under internet of things

This paper proposes a comprehensive device collaboration model, shown in Figure 4.7. The proposed model is based on service oriented architecture and includes device layer, device oriented web service layer which encapsulates device functions, compatible resource layer which abstracts compatible devices functions and process layer which describes the collaboration process.

![Figure 4.7: Comprehensive device collaboration model [P13]](image)

[P14] A scalable distributed architecture towards unifying IoT applications

A layered and distributed architecture was proposed in this paper, shown in Figure 4.8, called Distributed Internet-like Architecture for Things (DIAT). The three layers of the proposed architecture, Virtual Object Layer (VOL), Composite Virtual Object Layer (CVOL) and Service Layer (SL), are put together as a stack and named IoT Daemon. VOL is responsible for a virtual representation of the objects, and CVOL is responsible for communication and coordination among Virtual Objects (VOs). The discovery mechanism for locating suitable VOs can be in both VOL and CVOL. SL is responsible for the creation and management of the services. Observer is used for automation of M2M communication and is spread across CVOL and SL. According to the authors, the presence of the proposed IoT daemon in every object can ensure the scalability and zero-configuration for the object.

[P15] Architecture and measured characteristics of a cloud based Internet of Things

Authors of this publication propose a sensor-centric framework called the IoTCloud which supports an extensible set of sensor-types and large numbers of possibly geographically distributed smart objects. The architecture of IoTCloud, shown in Figure 4.9, consists of IoTCloud Controller that is responsible for managing the other system components and providing SOAP Web Services for sensor registration, discovery, subscription and control, the Message Broker which handles the low level details of message routing, the combination of the Controller and the Message Broker provides a middleware layer for the system, Sensors may be smart objects or computational services. Clients subscribe to sensor data for any application specific purpose. The authors also included examples of currently implemented sensor modules, described the experimental test-bed for cloud development and conducted a preliminary study to analyse the performance characteristics of their middleware in the context of high end real-time video sensors.
Bottom-up approach based on Internet of Things for order fulfillment in a collaborative warehousing environment

In this paper an IoT architecture based on RFID, ambient intelligence and multi-agent system was proposed, shown in Figure 4.10. Multi-agent systems provide this architecture with spontaneous configuration, toleration of partial failures and management of complex systems. The discovery protocol used in this architecture was Universal Plug and Play (UPnP) ambient middleware layer that interfaces with the devices, and is used for the information exchange between the devices, ERP and the multi-agent system.
[P17] **CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System**

This paper proposes an architecture of service system based on Cloud Computing and Internet of Things that uses service-oriented technologies. Their presented model is a computing and service-oriented manufacturing model which has three users: providers, an operator who controls the platform and consumers. The running principle of the cloud manufacturing service system is divided into three phases: in the first phase various services are intelligently sensed and connected into a wired network and automatically managed and controlled using IoT, in the second phase the data collected from the services is aggregated and processed and different pools of manufacturing services are constructed, and in the last phase different manufacturing services can be invoked to help in the manufacturing task.

[Figure 4.10: Multi-agent system architecture [P16]]

[P18] **Intelligent device-to-device communication in the Internet of Things**

This paper has a focus on establishing device to device (D2D) communication with routing algorithms. To be able to communicate with each other, according to the authors, devices have to have the awareness of the environment in which they are operating and be able to collect data from their environment and use it to perform an activity. In this paper two approaches for D2D communication are given – network infrastructure-dependent D2D communication and network-infrastructure independent D2D communication approach. When it comes to the infrastructure-dependent D2D communication the main task for realizing it is link-discovery which is a process used by devices to find other neighbouring device within its communication range. The network infrastructure coordinates link discovery by informing devices about other devices within their proximity and allocating the time and frequency for sending beacons. Since
the network knows about the devices’ locations, link discovery is less time consuming and more energy efficient. In the infrastructure-independent communication link discovery is simple and is done by the devices themselves and they communicate over an unsecured frequency. This paper also discusses some routing algorithms/protocols for D2D communication and divides them into stochastic/probabilistic, bio-inspired, hierarchical, and context-aware algorithms.

**[P19] Interoperability of security-enabled Internet of Things**

A layered architecture for the IoT was proposed in this paper, shown in Figure 4.11. In this architecture a semantically enhanced overlay interlinks the other layers and facilitates secure access provision to Internet of Things enabled services. The main element of semantic overlay is security reasoning through ontologies and semantic rules. Finally the interoperability of security aspect is addressed through ontology and a machine-to-machine platform [P19]. The communication and real world access layer have the task of discovering the nodes, receiving events from the nodes and forwarding the received events. This layer also provides different adapters to communicate with different types of nodes.

![Figure 4.11: Functional architecture [P19]](image)

In this paper a prototype was implemented, as shown in Figure 4.12, on the Shepherd M2M platform, using also SunSPOT sensors and embedded Linux System. A two way communication was established between the SunSPOT sensors and their base station, and also between the embedded Linux system and the Shepherd platform.
[P20] Supporting efficient machine-to-machine communications in the future mobile Internet

This paper presents the discussion about the MobilityFirst architecture, shown in Figure 4.13, that supports sensor and M2M scenarios through the use of an identity naming system for all network attached objects using the concept of a “globally unique identifier” GUID. The use of GUID results in a “flat” network where sensors and other embedded devices are visible to all applications and devices on the Internet. MobilityFirst delivers data by dividing it into chunks, where each chunk has a destination GUID which can be resolved with a global name resolution service (GNRS) which makes it possible for the data to be distributed to an unknown network location [P20].
The authors have implemented a prototype network of the MobilityFirst network architecture that consists of Click-based MobilityFirst routers that route GUID-addressed packets to network entities using a reliable hop-by-hop data transport, and enables network presence and reachability for sensors using GUIDs. Each router interacts with a GNRS to enable dynamic GUID-to-location binding for mobile entities. Network entities run a corresponding protocol stack and API to allow applications to interact with the network.

[P21] Augmented service in the factory of the future
This paper presents a model-view-controller based architecture, as shown in Figure 4.14, which enables the decoupling of information presentation, data aggregation and application control. In this scenario it is a wireless sensor network, and the architecture comprises of a web-service gateway and an augmented reality user interface. The sensor network interface is based on the device profile for web services (DPWS). Using multicast announcements the application can use ad-hoc discovery to connect to all devices that are in the proximity. The authors have implemented the proposed architecture using two different approaches for the on-site Field Service Engineer that
collects data on site and establishes a link between the business backend and the real world. One approach is using the ARtoolkit that can identify and track 2D tags, and the other approach is using active infrared LED tags that can be coupled with nodes.

**Figure 4.14: MVC system architecture [P21]**

[P22] **Internet of intelligent things and robot as a service**

This paper discusses the architecture of intelligent devices connected to the cloud. The goal of the paper is to further extend centralized cloud computing into a decentralized system through autonomous physical services by forming a pool of intelligent devices that can make decisions without communicating to the cloud. The architecture proposed in this paper is based on the device profile for web services (DPWS) which enables secure messaging, discovery, description and event notifications on resource-constrained devices. Robot-as-a-service (RaaS) architecture should have all functions of service oriented architecture - it should implement a cloud unit as a service provider, as shown in Figure 4.15, a service client and a service broker. For the decentralization of behaviours in RaaS units the authors are using Context-aware Community Service Dissemination Technology and Autonomous Decentralized Community System (ADCS) framework which applies a ripple-based multi-target context-cognizant service discovery technique to enable selection of right services to the right user at a right place. The RaaS was implemented in two robots – a hex crawler and an autobot. Arduino boards were used for both robots, Atom N270 processor for hex crawler and Intel i5 for the autobot, and USB to I2C communications modules were used for both robots.

**Figure 4.15: RaaS in cloud environment [P22]**
[P23] Novel Consumer-To-Product Interactions with Context-Aware Embedded Platforms

A prototype of a product sleeve and IoT architecture was presented in this paper, shown in Figure 4.16. The architecture was designed to show the information flow between sensor, actuators and the Internet of Things. The sleeve prototype is used on products that have RFID chip, and is initially made to carry bottles and show information about the product in an elderly-people friendly way. Over web service discovery multiple sleeves in proximity can know about the existence of one another and generate notifications based on code snippets they have automatically downloaded.

![Figure 4.16: Technical architecture [P23]](image)

[P24] When devices become collaborative

This paper presents an IoT based devices collaboration solution. According to the authors of this paper, the main IoT technologies used for communication between computers and network-enabled devices are UPnP and DPWS. In this paper they develop a DPWS add-on based on WS-Management that brings WS-Management capabilities to a DPWS server. Device interoperability is assured through a bridge between device technologies and a sensor gateway. For device collaboration across specific workflows the authors developed a choreography based solution, where the device autonomy is assured through the behaviours assigned to them, and the collaboration workflows are dynamically reconfigured according to the currently available devices.

4.3.3 Ungrouped publications

The two of the remaining publications were not grouped into any of the before mentioned groups, but we still found that the results or ideas of these publications might be relevant and therefore they are included in our study.


In this paper middleware architecture was proposed. The proposed architecture uses context information of sensors to supply a plug-and-play gateway and resource management framework for heterogeneous wireless sensor networks. The sensor node context is the information which is not related to the real sensing operation and
information. For the implementation of their solution authors used SunSPOT architecture, TinyOS and 6LoWPAN protocol. The architecture is divided into three main layers: connectivity, information processing and service layer. The connectivity layer establishes the connection with the sensor networks. When a new node is activated, the information about the node and the association process is stored in the device registry designed in the middleware component. The Information processing layer uses the data from the connectivity layer and from the knowledge layer. In this layer intelligent data analysis is done. Different algorithms and mechanisms can be deployed in this layer to discover different patterns and/or events from sensing data or correlate information. This information will be made available through the service provision layer which allows Publish/Subscribe service where users can be informed about the network. Before a device is able to exchange information it has to be registered in at least one gateway. The gateway will send a beacon signal every few seconds, which the devices can use to register themselves to the gateway. Each gateway knows its neighbours. If a gateway cannot satisfy a request by the user, the query is sent to the connected gateways. The approach reduces the configuration of discovery and registration of sensor nodes by introducing platform depended management modules in a platform independent middleware solution that automatically establish a connection to the nearest gateway [P25].

[P26] Scalable Object-to-Object communication over a dynamic global network
In this paper an object to object (O2O) architecture, shown in Figure 4.17, was described. The described architecture addresses the problem of end-to-end communication between arbitrary classes of objects while employing a set of Attachment Registers (AR) of which each is associated with a network entity or a networked object. A networked object or network entity registers the neighbours to which it is directly attached in its AR. The neighbour registrations in the ARs serve as a distributed topology data base which is used by the LCS to construct topologically significant locators for networked objects.

Figure 4.17: O2O architecture [P26]
4.3.4 Quality attributes

For industrial IoT quality attributes such as availability, reliability, performance and latency are an important factor when it comes to device collaboration. Some of the forementioned publications refer to those attributes.

In [P17] authors refer to reliability as a problem to be solved, but at the same time mention that underlying technologies of their Cloud Manufacturing Service system are reliable, as well as their virtual environment. In [P12] authors conclude that the reliability of a large-scale architecture for efficient and self-configurable service and resource discovery in IoT networks can be shown with a transparent integration of two different types of overlays. Authors of [P10] state that the decision making step involves reliability among others, and so, with their proposed concept, they allow devices to identify the credibility of other devices.

[P1] says that a data-centric approach such as directed diffusion has the potential of relatively high performance but is tightly coupled to a query on demand data model. The same publication also states that when large numbers of WSNs are deployed, treating them as peripheral devices and connecting them to the Internet via proxies or sinks will limit performance and scalability, so they propose a solution for that with their holistic architecture. [P3] gives us an evaluation of the latency of their proposed architecture. To provide performance comparison they used both CoAP and HTTP protocols on a single sensor device, with or without the Event Manager (EM), and compared the results. They also provided test results when using more than one sensor device. In [P14] group members are programmed so they can communicate with each other for coordination, that way optimizing their performance. [P7] states that the service orchestrations orchestrator node may become a performance bottleneck, especially when there is a large number of participants involved, so they wonder how to keep their model highly scalable and adaptable so it does not become an overhead for the system performance. They believe they can solve that problem by combining service orchestration and choreography. In [P5] authors plot and compare latency and delivery ratio inside Smart Objects and Zigbee networks and conclude that: „ZigBee presents a very large trade-off between delivery ratio and latency, where delivery ratios comparable to those of SC produce large latencies, and where small latencies comparable to those of SC produce very low delivery ratios. Judging by these results, we can conclude that SO networks generate very low network latencies, outperforming the well-established routing protocol ZigBee for the same delivery ratios“. The authors also state that the infrastructure they presented in the paper was also evaluated for data access performance by pushing the sensor data towards subscribers. The test performed showed that the architecture proposed in this publication had good performance in terms of network lifetime, overhead, scalability, and latency. [P15] presents IoTCloud – Cloud Based Internet of Things, and claims that their API enables developers to create scalable high performance IoT and sensor-centric applications. Authors also present measurements of system performance while conducting a simulated real-time video chat application, implemented using the IoTCloud framework and deployed with a single message broker. Their results showed latency of 600ms when there are 250 clients, 400ms when there is 200 clients. They conclude that a single broker is capable of supporting approximately 200 clients for a simulated video conferencing application, and about 150 clients for a real-time video conferencing application. [P12] carried out experiments to validate feasibility of their solution and evaluate its performance for
both local and large-scale service discovery mechanisms. Their results showed that: “the time required for service resolution in the Zeroconf-based approach for local SD is linearly dependent on the number of hops in the path between the client and server node. Considering large-scale SD, the adoption of a P2P overlay provides scalability in terms of time required to perform the basic publish/lookup operations”. In [P26] authors evaluate the performance of the object-to-object architecture, focusing on scalability and latency. Their worst case handover latency is 800ms, and the locator construction latency is 350ms.

[P1] states that cloud services and big data can be used to analyse data and improve scalability and availability of devices in IoT, but they do not state any further details. In publication [P2] it is stated that the need of device collaboration accentuates problems with availability, reachability and stability, and that a sleep proxy can ensure availability of URIs associated with an IoT device even when the device is “sleeping”. [P3] states that, compared to traditional WSs, services provided by IoT devices have more volatile availability. In [P19] it is said that: “it is extremely important that IoT services are available from anywhere at any time in order to provide information (i.e., measured data, sensor alarm, etc.) continuously. There is no single security protocol that can satisfy this property. However, different pragmatic measures can be taken to ensure the availability. For example, in the aforementioned smart home if the attacker knows the consumption monitoring service, he can launch the denial-of-service (DoS) attack by just trying to send false service requests and the sensor nodes are incapable of handling a huge number of requests due to resource limitations. Since any transmission (i.e., receiving or sending) consume power, the node will eventually run out of its battery”. In [P12] authors explain how the availability of mechanisms that minimize the need for external human intervention for configuration and maintenance of deployed objects is crucial, so they propose a scalable self-configuring architecture for service discovery in IoT. They also state that centralized approaches for service discovery, such as the CoAP protocol, suffer from availability limitations, and that the presence of a proxy at the border of IoT network can ensure high availability through caching.

After conducting our research study, we could notice that most of the publications didn’t address the quality attributes and that not much research has been done regarding that topic.

4.4 Information sharing related publications analysis

Analysis of the publications related to the information sharing will be presented in the following paragraphs. Chosen publications are giving insight to the idea of information sharing, and possible solutions and issues of network communication. Most attention was given to computer networking and different communication protocols, with the significant review of the security issues.

In the studies [P44] and [P50] it is mentioned how for efficient end-to-end delivery of data several requirements have to be met. Those requirements are optimal network design, which enables a high number of devices to communicate while minimizing the cost of infrastructure. The second requirement is security and privacy against a large variety of security threats. And the last one is governance and proper use of data with assigning proper rights to users.
In the [P51] it is indicated how three standardized communication interfaces are used for data exchange between the different M2M entities in the different domains and those interfaces are:

- dIa interface which mediates the interactions between applications in the M2M network area;
- mId interface which mediates interactions between xSCL;
- mIa interface which mediates the interactions between applications in the application domain [P35].

Study [P44] is based on a work on developing a middleware toolkit. Also, the development of machine type communication systems was presented. OpenMTC platform optimized for the development of IoT applications enables integration of various devices and applications into a single local testbed while lowering development costs. Information sharing can be realized in two distinct sharing modes, real-time and sharing of control applications across multiple solutions. Mentioned sharing modes were discussed in [P37] where the main goal was proposing a new cloud platform IoT PaaS for scalable and efficient IoT service delivery which is suitable for industrial IoT cloud application. Real-time sharing between different devices was discussed in [P38] and the new approach was presented which provides “peer-to-peer distribution of sensor information in a reliable way among connected entities, which also has an extensible information model that enables intelligent application behaviour.” This approach deals with real-time communication between devices, and uses MediaSense platform for evaluating and testing. Autonomy in IoT is a concept that refers to self-configuration of the network which is able to adapt dynamically and adjust the offered services [P30]. Comparison of concepts of autonomy can be found in [P30] along with the design of the “protocol for registration of end nodes that addresses the issues of autonomic behaviour.”

4.4.1 Computer networking
Communication process of exchanging information and sharing resources among various devices or computer systems in a network is studied by engineering discipline named computer networking [13]. It depends on the theoretical application and practical implementation of several fields, including computer science and engineering, telecommunication and information technology. Study [P33] gave detailed insight in data management in the field of IoT, from the production of the data, processing, delivering and archiving through different layers of the network.

In distributed systems, components located on connected computers communicate and coordinate their actions by sending messages in order to accomplish common tasks. For enabling communication we must have a network, and for building a network, a network card, routers and protocols must be available. Computer networks differ in modes of channel operation, transmission media, the size of the network, topology and the communication protocols [P13].

**Modes of channel operation**
In computer networks devices pass data to each other along channels in the form of packets. The communication channel is defined over a transmission medium and makes the path that connects the nodes. There are three basic configurations of modes of channel operation and they can be seen in the Figure 4.18:
Figure 4.18: Models of channel operation

- Simplex communication, representing communication channel which sends information in only one direction.
- Half-duplex communication which means that communication goes in both directions, but not simultaneously, only one direction at a time.
- Full-duplex communication which requires two simplex channels operating in opposite directions so information can be sent in both directions at the same time [15].

Size of the network
The connection among devices is established using cable or wireless media, and best known computer network is the Internet. Networks can be divided into several types according to their size as well as their purpose, and we will mention some of them [14]:

- Personal area network or PAN is a network used for connecting devices that are very close to each other (up to 10 meters) and it includes wired and wireless devices. Wireless PAN (WPAN) uses Bluetooth, IrDA, ZigBee and other, while wired one uses USB and FireWire. PAN is often used in IoT networks [P49].
- Local Area Network or LAN is a small network which connects devices on the same geographical space, like an office. Its size varies from two to thousands of devices. It supports peer-to-peer or client-server networking methods, and is used when resources need to be shared among people on the network and not with the outside world.
- Metropolitan Area network or MAN is spread across a college campus, small region or entire city. It is often used to connect several LANs to form a bigger network.
- Wide Area Network or WAN is formed to connect a computer with its peripheral resources across a large geographical area.
- Wireless Local Area Network (WLAN) and Wireless Wide Area Network (WWAN) are formed without physical media or wires to connect hosts with the server. One of the most widely used versions of the WLAN is Wi-Fi. WWAN often differs from WLAN by using mobile network technologies for transferring data.

Topology
Arrangement of elements of a computer network is called topology and it can be physical or logical. Physical topology refers to the placement of different components of a network, while logical is connected to network protocols and shows the data flows within a network, regardless of its physical design. There are currently seven basic
topologies recognized, and they are point-to-point, bus, star, ring, mesh, tree and hybrid [16]. Basic topologies can be seen in the Figure 4.19 followed by an explanation and description for each of them.

Figure 4.19: Network topologies [16]

- Point-to-point is a simple topology with a link between two endpoints.
- Bus topology represents a network whose nodes are connected to a single cable. A signal from the source travels in both directions until it finds intended recipient while other machines ignore data.
- Star is a topology in LAN in which each node or client is connected to a central hub or switch, which represents the server, with a point-to-point connection.
- Ring topology is set up as a circle in which data flows through nodes in one direction, and there is no server present but all nodes act as a server repeating the signal.
- Mesh network is topology in which all nodes cooperate in distributing data over the network. It can be formed as a fully connected network where every node is connected to every other node, or partially connected where some nodes are connected to more than one other node.
- Tree topology is combination of bus and star topologies which creates hierarchical form.
- Hybrid networks combine two or more existing topologies.

Computer networks use packets, which are formatted unit of data, to send information to the destination point. Packets consist of two parts. The first part is control information which provides needed data like source and destination network addresses, sequencing information and error detection codes used to deliver packages, and the second part is user data. When the packets arrive they are reassembled into the original message.

**Transmission media**

Transmission media used for connecting devices can be electrical cable, optical fibre or radio waves, and they are defined in physical and data layer of the OSI model. Depending on transmission media there can be separated wired and wireless technologies [17]. Wired technologies include:

- Coaxial cable usually used for television systems with speed range from 200 to 500 million bits per second,
- G.hn technology for creating LAN with speed up to 1 Gigabit per second,
• Twisted pair wire which is the most widely used communication medium with the transmission speed from 2 to 10 million bits per second.

Wireless technologies include:
• Terrestrial microwave communication uses Earth-based transmitters and receivers similar to satellite dishes,
• Communication satellites which communicate via microwave radio waves,
• Radio and spread spectrum technologies used by WLAN,
• Cellular systems which use several radio communications technologies,
• Free-space optical communication which uses visible or invisible light for communication and transmitting data for telecommunications or computer networking.

**Bluetooth** is a wireless technology standard for transmitting data over small distances by using short-wavelength UHF radio waves. It is used for wireless personal area network, low-bandwidth applications, identification and tracking of object positions with the real-time location system, and others [18]. Bluetooth 4.0 (Bluetooth Smart) is an optimized version of the Bluetooth technology standard which is compatible with classic Bluetooth-capable devices [18]. It is convenient for devices with low-power consumption and can be included in sensor devices like heart rate and blood pressure monitors and other applications in healthcare, as well as in sports, fitness, etc. Bluetooth Smart, because of its features, makes an excellent choice for developing Internet of Things.

4.4.2 **Protocols**
In the Internet of Things communication between various devices plays the main role, and in order to achieve it, some requirements have to be met. Formal description of rules and formats of the data to be communicated is called communication protocol. Before successful transmission of messages between devices, physical aspects of the data have to be defined. The size of the packet, the speed of its transmission, types of the error correction, address mapping, flow control, routing and address formatting are some of the aspects that have to be defined before communication can be established.

For handling the transmission, systems use several protocols which cooperate with each other. They are structured using a layering scheme for their basis, and the protocol on each layer has clearly defined responsibilities and functionalities which are not overlapping with the ones of the protocols on other layers [P29]. Framework for designing standard protocols and services that is widely mentioned is Open Systems Interconnection reference model MR/OSI. It consists of several layers which, from lowest to highest, are physical, link, transport, session, presentation, and application layer. Architecture of layers and addressing and identifying of devices was described in [P29], and the importance of flexibility of device networks connecting clusters of objects in technology and topology was highlighted. New approach of moving identification layer to network level was proposed in [P32] which would enable “avoiding overlay solutions and offering caching if data for energy saving, multicast of services and ID/location separation”. With given experiments and calculated results, this approach was proved to be feasible and makes strong basis for the future research.
The most interesting layer that has to be considered in IoT is the highest, application layer, needed for handling the communication. It provides identification of the possible communication partners, their availability and authentication, agreement on privacy mechanisms, error handling and recovery and data flow over the network [20]. This layer provides application services for the file transfers, and other network software services.

In the following text 6LoWPAN will be described; IPv4 and IPv6 will be compared, as well as UDP and TCP protocols. Most noticeable application layer protocols are mentioned, and those are MQTT, XMPP, CoAP, and AMQP.

**6LowPAN** (IPv6 over Low-Power Wireless Personal Area Networks) is an open standard used to support low-power wireless networks over Bluetooth Smart, low-power Wi-Fi, Sub-1 GHz low-power and others.

![6LoWPAN protocol stack diagram](Image)

Architecture of the 6LoWPAN protocol stack consists of several layers which are compared with standard TCP/IP protocol stack what can be seen in Figure 4.20. For the physical layer of 6LoWPAN IEEE 802.15.4 is used, and the data link includes low-power multi-hop media access layer (MAC), better suitable for industrial communication requirements [P46], and 6LoWPAN adaptation layer which provides adaptation from IPv6 on the network layer to IEEE 802.15.4 [P50] [P39]. 6LoWPAN compresses the IP stack by removing a number of fields from the IPv6 and UDP headers which contain known values, or because they can be found in the header fields of IEEE 802.15.4 [P27]. As it is noted in [P48] “6LoWPAN is defining a set of protocols that can be used to integrate sensor nodes into IPv6 networks.”

Transport layer protocols are ICMP, and UDP which is used for devices which cannot use TCP because of their low power consumption. For the last, application layer, CoAP is developed which runs over UDP, and there is also MQTT which runs over TCP. In the study [P46], preliminary simulation results were provided based on the research about the properties and performance of IEEE 802.15.4e TSCH MAC, 6LoWPAN and RLP stack. Study [P27] offers comparison from various aspects between link, proxy and full-IP model. Full-IP model which is using 6LoWPAN showed to be the “only connectivity model that is able to bring end-to-end solutions” but it requires high capability hardware.
IP address is an identifier assigned to a device in a network to which it is connected. The main vision behind IoT is making all the devices IP enabled [P30]. The addressing system used in networking is **IPv4** or Internet Protocol version 4 which provides $2^{32}$ addresses, but the number of its available addresses is decreasing because of the big number of connected devices nowadays, so it will soon become insufficient [P48]. For the growth of the Internet, **IPv6** was developed. IPv6 is the version of communication protocol that provides identification and location system for devices on networks so they can communicate, and it enables $2^{128}$ addresses. It provides several new functionalities, has big addressing capabilities, and provides security of the data which also contributes to its use in the constrained devices of IoT [P41]. In the Table 4.2 comparison between two addressing systems can be found.

<table>
<thead>
<tr>
<th></th>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard since</td>
<td>1974</td>
<td>1998</td>
</tr>
<tr>
<td>Developed by</td>
<td>IETF</td>
<td>IRTF</td>
</tr>
<tr>
<td>Length in bits</td>
<td>32</td>
<td>128</td>
</tr>
<tr>
<td>Address format</td>
<td>Dotted decimal</td>
<td>Hexadecimal Notation</td>
</tr>
<tr>
<td>Dynamic addressing</td>
<td>DHCP</td>
<td>SLAAC/DHCPv6</td>
</tr>
<tr>
<td>IPSec</td>
<td>Optional</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Header length</td>
<td>Variable</td>
<td>Fixed</td>
</tr>
<tr>
<td>Minimal packet size</td>
<td>576 bytes (fragmented)</td>
<td>1280 bytes</td>
</tr>
<tr>
<td>Header checksum</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Header options</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Flow</td>
<td>No</td>
<td>Packet flow label</td>
</tr>
</tbody>
</table>

There are two commonly used transport layer protocols, **UDP** (User Datagram Protocol) and **TCP** (Transmission Control Protocol). TCP is most commonly used protocol which offers flow control and guarantees that data will be transferred to its destination. UDP is used for faster data transfers which allow data loss like streaming audio or video, and it does not use flow control [24]. Comparison of TCP and UDP can be found in the Table 4.3.

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable – monitors message transmission to ensure receipt of all packets</td>
<td>Unreliable – no concept of ACK, retransmission or timeout</td>
<td></td>
</tr>
<tr>
<td>Heavyweight – dedicated connection</td>
<td>Lightweight – no dedicated end-to-end connection for multicast communication</td>
<td></td>
</tr>
<tr>
<td>Lower speed</td>
<td>Higher speed</td>
<td></td>
</tr>
<tr>
<td>Connection-oriented</td>
<td>Connectionless</td>
<td></td>
</tr>
<tr>
<td>Suitable for applications that require high reliability</td>
<td>Suitable for applications that need fast and efficient transmission</td>
<td></td>
</tr>
<tr>
<td>Used by: HTTP, HTTPS, FTP, SMTP, Telnet</td>
<td>Used by: DNS, DHCP, TFTP, SNMP, COAP</td>
<td></td>
</tr>
<tr>
<td>Flow control</td>
<td>No flow control</td>
<td></td>
</tr>
</tbody>
</table>
**MQTT** (the Message Queue Telemetry Transport) is a lightweight asynchronous M2M connectivity protocol designed for constrained devices and unreliable networks [P50] built on top of TCP/IP protocol, whose main purpose is remote monitoring [19]. Its goal is to transport all collected data, from many devices, to the IT infrastructure [21]. It targets small devices in large networks that need to be monitored or controlled from the cloud. It is designed to allow a big number of clients to be supported by only one server (broker) and uses publish-subscribe architecture. “Messages received from MQTT brokers are queued in message handler, and then switched to corresponding applications or the managing agent according to the message topic” [P50]. It can be well used in constrained environments. Another application of MQTT is in the networks with low bandwidth and limited processing capabilities and memory, and also high latency. MQTT brokers may require a password and username authentication, which should ensure security, handled by Secure Sockets Layer which is used to ensure privacy for HTTP transactions. It can be used for monitoring a large number of sensors and analysing their readings on only one location. In the Figure 4.21 MQTT-based IoT Architecture is shown. In the study [P49] prototypes are presented which were used for verification of publish-subscribe message-oriented paradigm in comparison with resource-oriented paradigm. It is concluded that first one better suits to “smart objects with restricted computing, storage, networking and energy resources” [P50]. A design for MQTT based IoT application and management framework is also proposed.

![Figure 4.21: MQTT-based IoT Architecture [30]](image)

**Extensible Messaging and Presence Protocol (XMPP)** is a client-server communication protocol based on XML designed for message exchanging. It is defined as open standard and uses an open systems approach of development and application. Flexibility, security, scalability and decentralization are XMPP’s most noticeable strengths and “an important feature of XMPP is smooth extensibility” [P51]. It uses name@domain.com addressing scheme which can be used in IoT as the easy way of addressing a service [21]. Since it is old protocol, it cannot provide required services for newer applications, but lately it regained attention as protocol suitable for IoT. It is built on TCP, and provides asynchronous publish-subscribe and synchronous request/response systems. One of its main weaknesses is that it does not support Quality of Service as the other mentioned protocols. It enables “real-time M2M messaging, presence management and dynamic topologies” [P51]. In the [P42], an approach which uses XMPP protocol for communication between device gateways and applications is presented. XMPP provides instant messaging functionality and because of
the open architecture is easily extendable. Communication approach proposed in [P42] combines advantages of this protocol and its extensions like publish-subscribe messaging pattern with the IPv6 QoS PI network.

The Constrained Application Protocol is an application layer protocol that runs over UDP, which helps keeping the overall implementation lightweight [19]. Architecture of the layers and position of the CoAP can be seen in Table 4.4. CoAP is a request/response protocol which supports synchronous and asynchronous communication and that is characterized by its simplicity [P50] [P28]. Since UDP is not characterized by high reliability, which means that a package can get lost on the way or there may be corruption while transferring it, CoAP needed to integrate mechanisms which help achieving reliability [19]. Reliability was achieved using transaction and request-response layers. First layer handles four types of messages between end points and they are Confirmable, Acknowledgment which acknowledges a Confirmable message, Non-confirmable, and Reset which indicates that a confirmable message has been received but its context is missing. Second layer is responsible for the transmission of requests and responses. Also, multicast and unicast transmission of data are supported, which means that information sharing is not only limited to one-to-one communication. This protocol can find its use in many IoT devices at “reduced cost while providing efficiency, interoperability and scalability at a lower level” [P41]. CoAP is created for use with low power sensors, switches and similar components that need to be remotely controlled through Internet networks. As noted in [P31] CoAP offers features that make it suitable for use in IoT devices and those are observable resources, group communication, application-layer fragmentation and it supports alternative transports.

The OpenMCT platform that provides a standard middleware for M2M oriented applications and services, recognized the benefits of CoAP protocol and is considering supporting CoAP for its future work [P35]. In the study [P31] the benefits of CoAP for IoT cloud services were researched and evaluated by using cloud-service-as-a-server scenario which enabled comparison with HTTP servers. In terms of scalability CoAP solutions performed much better than HTTP-based cloud services which are unsuitable for devices with small memory capacity. The possible usage of Web servers in constrained devices comes from the facts that those devices do not have to handle a great number of connections and requests and in the IoT client is more powerful than server [P49]. In the study [P49] basic Web application designs and protocols were presented and the analysis of new embedded Web server Smews which uses HTTP and TCP protocols for communication with attached results of performance and memory usage which enable comparison with performance of CoAP.
CoAP is a realization of Representational State Transfer (REST) architecture for constrained nodes and networks, but in the [P50] some defects of the resource oriented paradigm are pointed which are:

- “Requests, responses and observations are all loaded on smart objects, what power constrained devices can hardly afford,
- Duplicated requests and observations for the same single resource cost a waste of computing and networking,
- IP address of each device must be known so the service discovery mechanisms are introduced what complicates the structure,
- Network address translation issue faced for global accessibility in most common IPv4 subnet environment like Wi-Fi” [P50].

**Advanced Message Queuing Protocol** is an open standard application layer protocol. Features that are defining it are reliability, security and interoperability. It provides a wide range of features related to topic-based publish-and-subscribe messaging, flexible routing, transactions and security [25]. The reason behind designing this protocol was to establish reliability in large companies that depend on moving data around their organization. It provides asynchronous publish-subscribe communication with messaging. Companies like JP Morgan, American banking and financial services company, use it to process 1 billion messages a day. NASA uses it for Nebula Cloud Computing and Google uses it for complex event processing [26].

**Comparison of protocols**
In the Table 4.5 comparison of mentioned protocols is given, including their most noticeable features.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Transport</th>
<th>Messaging</th>
<th>Security</th>
<th>Arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoAP</td>
<td>UDP</td>
<td>Req/Resp</td>
<td>Medium</td>
<td>Tree</td>
</tr>
<tr>
<td>HTTP/REST</td>
<td>TCP</td>
<td>Req/Resp</td>
<td>Low</td>
<td>Client-server</td>
</tr>
<tr>
<td>MQTT</td>
<td>TCP</td>
<td>Pub/Sub</td>
<td>Medium</td>
<td>Tree</td>
</tr>
<tr>
<td>XMPP</td>
<td>TCP</td>
<td>Pub/Sub</td>
<td>High-mandatory</td>
<td>Client-server</td>
</tr>
<tr>
<td>AMQP</td>
<td>TCP</td>
<td>Req/Resp</td>
<td>High-mandatory</td>
<td>Client-server</td>
</tr>
</tbody>
</table>

4.4.3 **Network service**
At the application layer runs application that provides data storage, communication, manipulation or other capability implemented using client-server or peer-to-peer architecture based on protocols, and it is called network service. It is hosted on server components to provide functionalities for client components of the network. Network services include Directory services, e-mail, file sharing, instant messaging, wireless sensor network, World Wide Web and others. Network service Domain Name Network is a hierarchical naming system for computers, services and resources which give names for IP and MAC addresses.
4.4.4 Security

Security represents a key component for safe and reliable communication and operations between connected devices in the Internet of Things. Interference with the controls of any devices in use by humans can represent a possible threat to human life. Network security consists of policies adopted by network administrator to prevent unauthorised access and modifications of the data. “Machine-to-machine communications involve interactions between cyber and physical worlds, which indeed introduces a lot of new issues in security and privacy” [P43]. Also, what must be taken into consideration is that IoT device security requirements may differ a lot from each other, and that would lead to hierarchy of security and trust approaches [P36]. The key points of security are authentication of the data, collected data needs to reflect the real situation, transmitted data needs to be confidential, and the “entire system needs to be stable, reliable and effective” [P47].

The vulnerability of IoT devices occurs for several reasons. Components are most of the time unattended which can lead to physical attach, their low capabilities disable implementing strong security mechanisms, and used communication is wireless [P48] which can make the possibility of attacks even greater. Studies [P43] and [P36] point to two sorts of attacks on the network communication that can violate privacy. One is inference attack which can, for example, be made by investigating electricity consumption in a building during some period by listening a smart meter in smart grid [P43] and learning private life patterns. Another is the distributed denial-of-service which can be harmful by “jamming autonomous M2M communications to disable the operation of physical systems” [P43]. In the [P45] implementation of the scalable OSCAR Object security architecture for the IoT was demonstrated. Used scenario concerned communication in which all devices are resources constrained and it can be used in Smart City deployments.

Study [P38] gave insight into new proposed security architecture for end-to-end communication that takes into account inter-domain scenarios and their implications on security. For authenticating and ensuring message integrity, usage of identifiers and certain signature mechanisms to sign the whole message is proposed [P40]. Security model for the ETSI M2M platform, suggested in [P51] and [P28], relies on end-to-end data protection, mutual authentication and integrity protection. In the [P4], system in which devices are automatically and securely provisioned and connected is demonstrated, with given attention to authentication and security. “The safety technology includes the network monitoring and firewall technology, network access control technology, distributes intrusion detection technology, light encryption/decryption algorithm and anti-malicious software” [P47]. There are several mechanisms for maintaining security of network communication and they are end-to-end security mechanisms and data encryption, control of access and authorization, activity auditing and equal protection across multiple protocols [33].

For the end-to-end security mechanisms, mobile apps and connected devices should be authenticated separately. Security on the endpoints is a necessary requirement for secure communication. It depends on:

- Identity of all the entities involved in communication,
- Protocols which are used to dynamically negotiate session keys, and required security functions,
• Algorithms to protect data in transmit through encryption or integrity checks.

The identity of connected device is best maintained at the hardware level, so it is not exposed to anyone, and if somebody wanted to violate the security he should steal the device, mobile app, and password. End-to-end security and its application in network and transport layer were verified in [P45] and the idea of applying end-to-end security within application layer was explored.

End-to-end data encryption allows only the unique recipient of a message to decrypt it. Data is encrypted automatically without user configuration. Typical server based systems do not include end-to-end data encryption, and they can guarantee protection of communication with the server but not between other nodes. "In order to ensure data authenticity and integrity end-to-end, it is beneficial to have asymmetric public-key based digital signatures which can be used to sign data at its origin (sensors)" [P34]. Control of access and authorization includes controlling which user type has what level of data access and it is of great importance in keeping the system safe. The problem of authentication and data integrity in IoT devices is noted in [P48]. Authentication requires infrastructures and servers that achieve their goal by exchanging messages with other nodes, which is not feasible in IoT since a big number of messages cannot be exchanged between devices and servers [P48]. Another mechanism for maintaining security is activity auditing. Keeping log records can be found useful since it can enable tracing and identifying patterns that can lead to errors or problems before they happen. Equal protection across multiple protocols is also of great importance. Making communication equally secure over wired and wireless protocols, like Wi-Fi, Bluetooth, ZigBee and others, should be provided so that the integrity of data does not get affected depending on which type of communication is used [32].
5 System architecture and implementation

We have now looked into IoT and specifically into system collaboration and information sharing. With this knowledge in mind, we have implemented a prototype that realizes two systems, each consisting of two devices, which are able to communicate and collaborate with each other. The schema of the two systems is presented in Figure 5.1.

The basic concept is that each one of microcontrollers, in this case specifically Arduinos, has a sensor that transmits its data to the PC, which then transmits that data to the second system. PC in the second system receives the data, and based on the data received it initiates action on the second microcontroller. In the next section we will describe the scenario implemented in this thesis.

5.1 Event scenario

The implemented scenario is shown in Figure 5.2. So, the Arduino 1 reads the temperature, checks if it exceeds the border, if it does, it turns on the red LED. Meanwhile, it is also transmitting the temperature reading to NUC 1, which forwards the temperature to NUC 2. NUC 2 checks if the temperature exceeds the limit, if it does, it turns on the motor if it is not turned on already, and if the temperature does not exceed the border or the temperature fell down below the border value then it turns off the motor.
5.2 System architecture

In this section we give a brief overview of the components used for the implementation of the chosen scenario.

5.2.1 Hardware

The hardware components our system was composed of were two Intel NUCs D54250WYKH, two Arduino UNOs R3 with Ethernet Shield R3, and a temperature and a motor sensor for Arduino.

Intel NUC is a Mini PC with the following features and configuration:\textsuperscript{15}:

- Intel Core i5-4250U CPU @ 1.30 GHz
- Intel HD 5000 graphics
- Intel WLAN 6235
- 8GB DDR3L memory
- Intel 525 mSATA SSD 120GB
- Four USB 3.0 ports
- 10/100/1000 Mbps Ethernet port
- 1 x HDMI 1.4a One Mini DisplayPort 1.2 supporting ultra-high definition.

In Figure 5.3 you can see Intel NUC, both its interior and exterior look.

\textsuperscript{15} taken from Intel official site \url{http://www.intel.com/content/www/us/en/nuc/nuc-kit-d54250wykh.html?ga=1.66550701.2090249407.1424442294}
Arduino Uno is a microcontroller board based on the ATmega328\textsuperscript{16} consisting of\textsuperscript{17}:

- 14 digital input/output pins
- analog inputs
- 16 MHz ceramic resonator
- USB port
- 32 KB flash memory, of which 0.5 KB is used by the bootloader
- 2 KB of static RAM
- 1 KB of electrically erasable programmable ROM.

All of the tools required for implementation of our scenario came inside the Arduino starter kit that we use for this project, including, as shown in Figure 5.4: a convenient wooden base for the Arduino and the breadbord, solid core and stranded jumper wires, resistors, capacitors, photoresistor, potentiometer, pushbuttons, temperature sensor, tilt sensor, LEDs, small servo motor, etc.

\textsuperscript{16} http://www.atmel.com/Images/doc8161.pdf
\textsuperscript{17} Taken from Arduino official site http://arduino.cc/en/Main/ArduinoBoardUno
For the implementation of our scenario, we decided to reuse the red LED, temperature sensor and the servo motor from the starter kit. As you can see in Figure 5.5, an Arduino Uno Ethernet shield was also used for this project. Ethernet shield was used for the purpose of connecting Arduino to the Internet – you just need to plug the shield onto Arduino board, and connect it to the network with an ethernet cable. The ethernet shield does not come with a PoE module so the Arduino board still has to be powered through USB cable or battery, because the shield gets its power from the board. The pin layout of the Arduino stays intact after plugging in an ethernet shield which means that more shields can be added later on.
The solderless breadboard that came with the Starter kit, shown in Figure 5.5, has 400 connection points and lets us test circuits before they are permanently soldered together. To power the breadboard, we connect it to the 5V pin on the Arduino board/Arduino ethernet shield with jump wire.

5.2.2 Software
The implementation of the scenario was done on Linux Ubuntu 3.14.25 operating system patched with real-time kernel. Communication between the two systems was achieved using OpenSplice data distribution service, and the code for OpenSplice and Arduino programs was written in the SublimeText3 editor with the Stino plugin which allows us to compile and upload sketches to the Arduino board.

5.3 Steps of implementation
1. Patching Ubuntu kernel with RT
2. Set-up zero-configuration networking on NUCs
3. Set-up zero-configuration networking on Arduinos
4. Set-up OpenSplice DDS on NUCs
5. Program the Arduinos
6. Set up collaboration and data sharing between system one (S1) and system two (S2)

5.3.1 Patching Ubuntu kernel with RT
Since Ubuntu version installed on NUCs is Ubuntu 14.10, it comes with kernel version 3.16.0-23 which at the time has no fitting RT patch version, so the closest kernel version, vanilla kernel 3.14.25, was downloaded and patched with the same version RT patch. They were downloaded from the following websites:


After downloading, the kernel, and patch tarball were unpacked, kernel source directory was entered and the kernel was patched with the command:

```
zcat ../patch-3.14.25-rt22 | patch -p1
```

After patching the kernel it was time for the configuration process to start. In this part we decide which features will be included in the final kernel. To enter the configuration menu we ran:

```
make menuconfig
```

Configuring the kernel can take a lot of time because there are a lot of options to go through, but the most important options to have in mind are:

- Enable:
  - CONFIG_PREEMPT
  - CONFIG_PREEMPT_RT_BASE
  - CONFIG_PREEMPT_RT_FULL
- Activate High-Resolution-Timer option if possible
  - `CONFIG_HIGH_RES_TIMERS`

After configuring the kernel we needed to build it and we did that with the following command:

```
make -j2
```

Usually the command to run is just `make`, but since NUC processor has two cores it was faster to build the kernel using both cores. When the building was done, we have to make sure that the modules are built and installed, and install the kernel by running:

```
sudo make modules_install
```

The last step of patching the kernel was to update grub and create a new entry so that we could start using our new real-time kernel.

```
cd /boot
sudo update-grub
```

After rebooting the NUCs and checking the kernel version, we could see that our kernel was successfully patched, as shown in Figure 5.6.

![Figure 5.6: Patched kernel](image)

### 5.3.2 Zero-configuration networking set-up on NUCs

Before starting with the set-up of zero-configuration on Ubuntu Linux, it is important to have the network files configured properly. First file is `/etc/network/interfaces` which should have only the following lines:

```
auto lo
iface lo inet loopback
```

The hosts file located in `/etc/` directory needs to be modified and have only the following lines:

```
127.0.0.1 localhost
#The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost
fe00::0
ff00::0
ff02::1 ip6-mcastprefix
ff02::2 ip6-allrouters
```

```
If we wanted to set up the Ubuntu system as a local network server then we would also add other hosts in the hosts file, but since we are using zero-configuration networking it is not necessary and would be redundant.

Since in our interfaces file we also had the auto Ethernet connection enabled by default, but later removed that line, we have wanted our Network Manager to handle interfaces that are enabled, otherwise we would have problems with losing connection in Ubuntu Linux, so we have also changed the NetworkManager.conf file in /etc/NetworkManager/. In the end the NetworkManager.conf file consisted of the following lines:

```
[main]
plugins=ifupdown,keyfile,ofono
dns=dnsmasq
[ifupdown]
managed=true
```

After this change, it is necessary to restart the NetworkManager by running:

```
sudo service network-manager restart
```

Since NUCs operating system is Ubuntu Linux, there is no need for the actual installation of Avahi zero-configuration tool because it comes preinstalled, but if it had not, we would install it by running:

```
sudo apt-get install avahi-daemon avahi-discover libnss-mdns
```

Avahi-daemon enables zero-configuration networking, Avahi-discover is a graphic user interface for mDNS/DNS-SD network services running on the local LAN, and libnss-mdns package is a GNU Name Service Switch (NSS) that provides host-name resolution via mDNS allowing name resolution with “hostname.local”. Hostname by default is the device name that users usually choose while installing the operating system. However, if we wanted to change the hostname we could do it by running:

```
sudo hostname newhostname
```

So for example, if we wanted to see if there is a IntelNUC user, we could run:

```
ping IntelNUC.local
```

After running the ping command we should get a result similar to the one in Figure 5.7.

![Figure 5.7: Successful ping](image)
If the host is unreachable (e.g., is not connected, the name is entered incorrectly, etc.) we would get a message like the one in Figure 5.8.

![Figure 5.8: Unsuccessful ping](#)

After successfully reaching both NUCs we wanted to check if we could communicate and share files just by using zero-configuration networking. The steps for that were:

1. Install Pidgin IM client
2. Set-up network file sharing

Pidgin is an instant messaging client that, among others, supports the Bonjour protocol, as shown in Figure 5.9. After installing Pidgin and creating an account under Bonjour Protocol, it was determined that both NUCs are able to successfully communicate over the network, even when there is no connection to the Internet.

![Figure 5.9: Pidgin Bonjour account](#)

Network file sharing allows us to share directories located on one computer with other computers on the network. In that case we have a computer where the directory is located – server, and the computer that is connecting to that server – client. To set-up NFS server\(^{18}\) we ran:

```
sudo apt-get install nfs-kernel-server
```

For easier managing we bind all of the directories we want to share to our export directory. So lets say we want to share our Public folder, we would run:

```

\(^{18}\) [https://help.ubuntu.com/community/SettingUpNFSHowTo](https://help.ubuntu.com/community/SettingUpNFSHowTo)
And then, so we wouldn’t have to run this after every reboot, we modify the /etc/fstab file by adding the following line:

```
/home/nuc/Public /export/Public none bind 0 0
```

Also, we have to check that the /etc/idmapd.conf file has the following lines in order to have the ID names automatically mapped:

```
[Mapping]
Nobody-User = nobody
Nobody-Group = nogroup
```

The final step of configuring the NFS server is exporting our directories to a local network by modifying /etc/exports file:

```
/export @myclients(rw,sync,no_root_squash,no_subtree_check)
/export/Public @myclients(rw,sync,no_root_squash,no_subtree_check)
```

Then we have to export the shares by running:

```
sudo exportfs -ra
sudo service nfs-kernel-server restart
```

To configure the NFS client computer we simply run:

```
sudo apt-get install nfs-common
```

And then by choosing the Browse Network option in our file manager Nautilus we can see now the hostnames of the computers that are sharing their directories, and mount/enter those directories. In Figure 5.10 we have an example of configured NFS where both computers act like servers and clients.
After this step, considering that we needed 4 devices connected to the Internet, we connected all our devices to a HP 1410-16G switch with the layout illustrated in Figure 5.11.

5.3.3 Zero-configuration networking set-up on Arduinos

After finishing the set-up of zero-configuration networking on NUCs, the next step was to implement zero-configuration networking with Arduinos. The EthernetBonjour library was downloaded from the following link: https://github.com/TrippyLighting/EthernetBonjour, and required some modifications of the existing EthernetUDP library. For the EthernetBonjour library to work properly we needed to modify EthernetUDP.h file by adding these lines to the public members of the class:

```c
virtual uint8_t beginMulti(IPAddress, uint16_t);
friend class EthernetBonjourClass;
```
EthernetUDP.cpp was modified by adding the following routine:

```c
/* Start EthernetUDP socket, listening at local port PORT */
uint8_t EthernetUDP::beginMulti(IPAddress ip, uint16_t port) {
    for (int i = 0; i < MAX_SOCK_NUM; i++) {
        uint8_t s = W5100.readSnSR(i);
        if (s == SnSR::CLOSED || s == SnSR::FIN_WAIT) {
            _sock = i;
            break;
        }
    }

    if (_sock == MAX_SOCK_NUM)
        return 0;

    byte mac[] = { 0x01, 0x00, 0x5E, 0x00, 0x00, 0x00 };
    mac[5] = ip[3];

    W5100.writeSnDIPR(_sock, rawIPAddress(ip));
    W5100.writeSnDPORT(_sock, port);
    W5100.writeSnDHAR(_sock, mac);

    _remaining = 0;
    socket(_sock, SnMR::UDP, port, SnMR::MULTI);

    return 1;
}
```

This library also allows us to implement an example that can register web service, and examples of that are viewable in Figures 5.12 and 5.13.

*Figure 5.12: Registering web service with the Arduino EthernetBonjour library*
In Figure 5.13 we can see a small web interface for the control (on/off) of Arduino LED. Both of the computers on the network are able to access the website and control the Arduino LED. Also, we managed to control LED on Arduino2 from Arduino1 using its serial monitor which is shown in Figure 5.14.

5.3.4 Set-up OpenSplice DDS on NUCs

After establishing that zero-configuration networking is up and running, it was time to set-up OpenSpliceDDS on NUCs. The process of installing OpenSplice was pretty straightforward. The first step was downloading the appropriate version of OpenSplice. We chose the DDS Community Edition Version 6.x for Linux kernel 3 and up (64-bit) Host and Target, gcc 4.6 compiler, x86 chipset (Ubuntu 12) from the following website: http://www.prismtech.com/dds-community/software-downloads.

After downloading the file we needed to extract it, enter the directory, change the OSPL_HOME variable in the release.com file and source that file. Since entering the OpenSplice directory, sourcing the release.com file, changing the OSPL_URI and LD_LIBRARY_PATH, and entering the directory examples were steps that needed to be repeated every time we open a new terminal, a small bash script consisting of five commands was written for the purpose of speeding the process.

The third command directs OpenSplice to the configuration file to use. Since we are using the community edition of OpenSplice we can use only single process deployment, if we were using Enterprise edition we could you multi-process shared memory
deployment also. Single process deployment means that the domain service, database administration and other needed services are all started within the DDS application process, when the application invokes create_participant operation. For the modification of the .xml configuration files, it is advised to use osplconf tool, shown in Figure 5.15.

![OpenSplice configuration tool](image1)

**Figure 5.15: OpenSplice configuration tool**

For our scenario we used the default ospl_sp_ddsi.xml configuration file which is used for standalone single-process deployment and standard DDSI networking, but with the `<NetworkInterfaceAddress>` first set to eth0 and then to wlan0. The first communication and examples were executed over the ethernet connections. After establishing a successful connection a D-Link router was set up and the NUCs were connected to the IoT Wi-Fi network so the OpenSplice DDS communication was set to be over wlan0. In Figure 5.16 you can see the PingPong example executed over eth0 interface. This example measures the round trip time.

![PingPong OpenSplice example](image2)

**Figure 5.16: PingPong OpenSplice example**

In Figure 5.17 we can see our PingPong RTPS packets monitored by Wireshark\(^\text{19}\).

\(^{19}\) [https://www.wireshark.org/](https://www.wireshark.org/)
In Figure 5.18 we see the execution of an OpenSpliceDDS Tutorial example over wlan0 interface. The example is executed between two NUCs. One NUC is a server and is running the MessageBoard, which displays all available messages, the UserLoad which keeps track of the users, and two Chatters – Maja and Maja2, which send messages to the MessageBoard. The second NUC is running one Chatter executable – tina. The first terminal window, top left, in Figure 5.18 is the MessageBoard, top right is the UserLoad, and the two terminal windows on the bottom are Chatter executables.
5.3.5 Programming the Arduinos

In order to implement the scenario we first have to connect sensors to the Arduino board. Here we have two different circuits. One with the temperature sensor and a LED, and one with the servo motor that is supposed to imitate a ventilator.

![Figure 5.19: Arduino 1 with temperature sensor](image)

The Arduino that we can see in Figure 5.19 has connected temperature sensor and a red LED. The first step of building this circuit was to connect the red jumper wire to the 5V pin on the Arduino and the other end in the plus/first long bus line in the breadboard to provide power to it. Then we connected the black jumper wire to the GND (ground) pin and to the adjacent bus line. The next step was to connect the cathode (short leg) of the LED to the ground using a 220ohm resistor, and the anode (long leg) to pin 3. The last step was to place the temperature sensor onto the breadboard with the rounded part facing away from Arduino, connecting the top leg of the sensor to the power, middle leg to the analog input pin A0, and the bottom leg to the ground.

After connecting the circuit, we need to program the Arduino. The program checked if the temperature that the sensor is reading is higher than the baseline temperature, and if it was, it would turn on the red LED, and if not it would turn it off. The program also implemented Serial Peripheral Interface (SPI), Ethernet and EthernetBonjour libraries.
The second Arduino, shown in Figure 5.20, has a connected servo motor. The first step was the same as before – connecting the breadboard to power and ground pins on the Arduino board. The next step is connecting the servo motor. The servo has three wires – black (ground), red (power) and white (control line). To connect the servo motor onto the breadboard, we have to first plug in the three male header pins into the female ends of the servo wires and then connect the male header to our breadboard. Then we connect the red wire to the power, black wire to ground, and white wire to pin 9 on the Arduino board. To smooth out any voltage changes that could occur, we also connect a 100uf capacitor by connecting its cathode to the ground and anode to the power, adjacent to the servo male headers. The program of this Arduino is reading characters input from the serial monitor. If the received character is ‘s’ the servo motor turns on and starts spinning, and if the received character is ‘x’ the servo motor turns off.

5.3.6 Collaboration between System1 and System2

Let’s say we have the two Arduinos close to each other. One acts like a thermometer and the other one like a ventilator. The first Arduino can let the second one know that the temperature in the room is too high, then the second Arduino can pitch in and lower down the temperature, all while communicating among themselves over the two NUCs DDS connection. That is the collaboration we wanted to achieve between the two systems, where each system consists of one Arduino and one NUC. Since the Arduino functionalities are set up at this point, there are just a few tasks left to achieve this collaboration:

1. Set-up the serial communication between Arduino and NUC
2. Set-up the DDS communication between two NUCs

Our Arduinos are powered by USB cable connected to NUC, so we could directly set-up a serial communication between the Arduino and the NUC. For the serial communication we use the original C I/O library <fcntl.h>. Since it is a low level library it can be
optimized for specific network topologies and hardware, and is supported on every major platform. With the use of that library, we solved the communication between our DDS applications and our Arduinos.

To send the Arduino temperature readings from NUC1 to NUC2, a publisher file was created and it transmits the temperature after fetching it from the Arduino1. The subscriber file implements a data reader using WaitSet that would wait for data to be available, read the data, and check if the received data, that is temperature, exceeds the borderline value. If the temperature exceeds the borderline value, the subscriber would send a ‘s’ character to the Arduino2 over already established serial communication, with the purpose of starting the servo motor. As soon as the NUC2 reads a value lower than the border value it sends an ‘x’ character to the Arduino2, that way stopping the servo motor. When the publisher stops transmitting the subscriber also ends.
6 Future work

This work touched upon IoT by demonstrating the collaboration between the two systems, but there are many opportunities for extending the scope of this thesis. As already mention in Chapter 1 of this thesis, this work was the first step towards future possibility of replacing the traditional centralized control of field devices with distributed intelligent devices that are connected to the cloud.

One of the many possibilities for future work is replacing the NUC to Arduino serial communication with an Ethernet communication that can be achieved by replacing the Arduino USB power supply with a battery power supply and connecting all of the machines to a hub. Second option is to remove the wires and put a Wi-Fi shield on the Arduino, that way enabling wireless communication and information sharing between the two systems. By achieving a wireless collaboration the next logical step would be to investigate possible applications of the architecture within ABB business units. The work presented in this thesis could be extended to the automation of alarm management systems, plant-floor monitoring for dispersed product sites, automation of windmills, robots, and so on. The work done so far is device-to-device collaboration, so the next step could also be the extension of the existing prototype by integrating cloud services and big data. Since most of the IoT devices generate some sort of data, usually too large and complex for storing into traditional relational database, by integrating these services we could send data into the cloud, analyse it, and provide feedback in real-time. The possible applications of IoT collaboration in industry are numerous and can help make factories safer, boost productivity and optimize operations.
7 Conclusion
This thesis was set to investigate the concept of Internet of Things and its applicability in industrial context. The systematic literature review part of the thesis sought to answer the following questions:

1. How to realize discovery and system collaboration between networking devices?
2. How to realize information sharing between connected devices?
3. What are the existing IoT solutions from other domains and what is their applicability in industrial context?

In this thesis we have shown how to successfully achieve system collaboration and information sharing between two devices using Internet of Things. System collaboration and information sharing between the two systems were achieved with the use of the OpenSplice DDS protocol. A systematic literature review answered the question about the existing IoT solutions from other domains, along with analysis of thirteen different IoT platforms. The literature review gave us a total of 51 publications which were analysed in order to answer the main research questions. 26 of those 51 publications were related to system collaboration, while 25 were related to information sharing. In the last part of the thesis work, the architecture was defined, and a prototype of the system collaboration architecture was implemented using OpenSplice DDS protocol and Arduino microprocessors.

From this thesis we can conclude that the Internet of Things has a wide span of uses, taking us closer and closer to the automated future. It is not something that we should look as a replacement for human beings; it is something that could allow us to focus on larger projects by reducing the time we spend on everyday simple tasks.
Bibliography


http://www.prismtech.com/vortex/industry-solutions/industrial-automation


http://www.prismtech.com/vortex/vortex-opensplice


[56] IoT SyS – Internet of Things Integration Middleware. Retrieved December 4, 2014, from:

[57] IoT SyS - Internet of Things integration middleware, Intro. Retrieved December 4, 2014, from:
https://code.google.com/p/iotsys/


[59] Ayla's IoT Cloud Platform, Enterprise-Class Application Enablement Powering the Internet of Things. Retrieved December 4, 2014, from:
https://www.aylanetworks.com/platform/key-capabilities

[60] Ayla's IoT Cloud Platform, Enterprise-Class Application Enablement Powering the Internet of Things. Retrieved December 4, 2014, from:
https://www.aylanetworks.com/platform/platform-components

http://www.kombridge.com/products/kombridge-things/

http://www.kombridge.com/products/kombridge-things/#teknisk-specifikation

Appendix 1 – List of Publications


Conference on Communication System Software and Middleware, Article No. 6, doi: 10.1145/2016551.2016557


[P31] Kovatsch, M., Lanter, M., Shieh, Z. Californium: Scalable Cloud Services for the Internet of Things with CoAP


[P37] Li, F., Vogler, M., Claesens, M., Dustdar, S. (2013). Efficient and scalable IoT service delivery on Cloud, 740-747, doi: http://dx.doi.org/10.1109/CLOUD.2013.64


[P50] Zhou, C., Zhang, X. Toward the Internet of Things Application and Management: A Practical Approach, doi: http://dx.doi.org/10.1109/WoWMoM.2014.6918928

<table>
<thead>
<tr>
<th>Features/Products</th>
<th>mBed</th>
<th>Oracle</th>
<th>Predix</th>
<th>Vortex+Open Splice</th>
<th>Intel</th>
<th>IoTYSYS</th>
<th>SeeControl</th>
<th>Ayla</th>
<th>Axeda</th>
<th>SiteWhere</th>
<th>Kombridge</th>
<th>IzoT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End-to-end security mechanisms</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>End-to-end data encryption</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Flexible &amp; configurable access and authorization control</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Activity logging</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Equal protection across multiple communication protocols</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>ARM IoT devices</td>
<td>“Real-time responsiveness for millions of device endpoints”</td>
<td>“Widest possible range of industrial devices”</td>
<td>different types of HW/Os, Prima nily X86, with some Arm builds, android install too, Intel Galileo and Edison boards, shields, sensors, actuators</td>
<td>Embe dded LoWPAN devices</td>
<td><a href="http://www.se">http://www.se</a> econtrol.com/ eex/es/</td>
<td>“ne aifi any mi co-con troler”</td>
<td>“connect to any product using any communication channel”</td>
<td>“provides interfaces for integrating with many existing device platforms”</td>
<td>“Support for devices equipped with communication modules from Cinterion, Telit etc.”</td>
<td>“Supports devices that communicate on Ethernet, LonTalk/IP-EN, and Wi-Fi (LonTalk/IP, Wi-Fi) media”</td>
<td></td>
</tr>
<tr>
<td>Protocols</td>
<td>CoAP/HTTP, MQTT, TLS/TCP, DTLS/UDP, CoA, XMPP</td>
<td>Ora de Java Embe dded</td>
<td>TCP/IP, UDP/IP, HTTP, MQ, AARQ, XMPP</td>
<td>MQTT, TCP/IP, CoAP, XMPP, SSH, Zigbee, etc.</td>
<td>CoAP, IPv6, LoWPAN, EX, HTTP</td>
<td>MQTT, UDP, HTTPS, TCP, GCoAP</td>
<td>“just about any as long as there is an IP backhaul”</td>
<td>“cloud agnostic”</td>
<td>“connect to any product using any experience over any communication channel”</td>
<td>“provides interfaces for integrating with many existing device platforms”</td>
<td>“Support for devices equipped with communication modules from Cinterion, Telit etc.”</td>
<td>“Supports devices that communicate on Ethernet, LonTalk/IP-EN, and Wi-Fi (LonTalk/IP, Wi-Fi) media”</td>
</tr>
<tr>
<td>Cloud agnostic</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Languages</td>
<td>C, Java SE and Java ME Clients</td>
<td>Java Embe dded</td>
<td>Ada, C, C++, C#, Java, JavaScript, Go, Python, Groovy, etc.</td>
<td>QCA, JavaScipt, Go, FreeScipt, Scala, Lua, Ruby</td>
<td>QCA, JavaScipt, Intel XDK IoT Edition</td>
<td>Arduino &amp; Wi-Fi ready</td>
<td>CoAP, X, CPAL 1.0 licence, code available on GitHub</td>
<td>“cloud agnostic”</td>
<td>“connect to any product using any experience over any communication channel”</td>
<td>“provides interfaces for integrating with many existing device platforms”</td>
<td>“Support for devices equipped with communication modules from Cinterion, Telit etc.”</td>
<td>“Supports devices that communicate on Ethernet, LonTalk/IP-EN, and Wi-Fi (LonTalk/IP, Wi-Fi) media”</td>
</tr>
<tr>
<td>Data requirements</td>
<td>NoSQL database</td>
<td>mBedOs and Device Server</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Data processing and analytics services</td>
<td>mBedOs and Device Server</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Open source</td>
<td>No, proprietary (except for mBed SDK)</td>
<td>Proprietary</td>
<td>Free evaluation (30day trial)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>