Lappeenranta University of Technology
School of Energy Systems
Master's Degree Programme in Electrical Engineering

Evgenii Kosenko

INDUSTRIAL IoT DEVELOPMENT FOR CONDITION MONITORING PURPOSES

Examiners:	Professor Olli Pyrhönen
	Associate Professor Tuomo Lindh
Supervisor:	M.Sc. Pekko Jaatinen

ABSTRACT

Lappeenranta University of Technology

LUT School of Energy Systems

Master’s Degree Programme in Electrical Engineering

Evgenii Kosenko

INDUSTRIAL IoT DEVELOPMENT FOR CONDITION MONITORING PURPOSES

Master’s Thesis

78 pages, 21 figures, 4 tables, 11 appendices

Examiners: Professor Olli Pyrhönen

Associate Professor Tuomo Lindh

Keywords: industrial IoT, IoT, Internet of Things, security, CPS, Beckhoff, mosquitto, MQTT, protocols, AMB, PLC

The research concentrates on internet of things (IoT) system design for industrial application of condition monitoring. The main goal is to study recent practices in the IoT field and implement them to a design of secured cloud-based cyber-physical system. Tools were chosen accordingly to literature review and task requirements. Thus, MQTT protocol communication protocol, mosquitto - open-sourced broker and TLS encryption protocol are used. Thingspeak cloud - stores, visualizes and analyzes data. Developed system was applied to real prototype, namely an AMB system and tested on temperature measurement example.

ACKNOWLEDGEMENTS

The thesis was carried out in frame of Energy Systems School, Lappeenranta University of technology. I would like to express all of my gratitude to my supervisor Pekko Jaatinen for his guidance and patience, to associate professor Tuomo Lindh for his advises, to professor Olli Pyrhönen for providing opportunity to work with great team in an exiting field.

I want to thank professor Victor Vtorov from St. Petersburg Electrotechnical University for his help. I wish to express my thanks to my college and teachers from St. Petersburg Electrotechnical University.

I am indebted to my dearest friends and colleges – Rostislav, Iurii, Maksim, Dmitrii for having countless discussions of ideas, advises. I am truly grateful to my parents, grandparents and dearest brother for their love and support. I wish to express my thanks for constantly supporting me to my dearest Valeriya.

TABLE OF CONTENTS

1. INTRODUCTION 7

1.1 Background 7

1.2 Aim, goals and delimitation 10

1.3 Research methodology 11

1.4 Organization of the study 12

2. COMMUNICATION PROTOCOLS AND CLOUDS 13

2.1 Communication Protocols 13

2.1.2 MQTT Protocol 16

2.1.3 AMQP Protocol 17

2.1.4 CoAP Protocol 18

2.1.5 OPC UA Protocol 19

2.1.6 Protocol Choice 19

3. CLOUD SOLUTIONS 22

4. IOT SECURITY PRACTICES 27

5. IoT DESIGN 32

5.1 General Structure of The Design 32

5.2 Communication Layer 33

5.3 Broker 36

5.4 Security Features Implementation 38

5.5 Data Access Interfaces 40

5.6 Testing Prototype Results 42

6. CONCLUSIONS 46

REFERENCES 47

LIST OF SYBMOLS AND ABBREVIATIONS

AMB - Active Magnetic Bearing

AMQP - Advanced Message Queuing Protocol

API - Application Programming Interface

CoAP - Constrained Application Protocol

CPS - cyber-physical systems

CSV -Comma-Separated Values

DDoS - Denial-of-Service attack

HTTPS - Hypertext Transferred Protocol Secure (HTTPS)

IDE - Integrated Development Environment

IoT - Internet of Things

IP - Internet Protocol

IPv4 – Internet Protocol version 4

IPv6 - Internet Protocol version 6

JSON -JavaScript Object Notation

LE - Low Energy

LWM2M - Lightweight M2M

M2M – Machine to Machine

MITM – Man in the Middle

MQTT - Message Queuing Telemetry Transport

OPC UA - OPC Unified Architecture

OWASP - Open Web Application Security Project

PLC - Programmable Logic Controller

QoS – Quality of Service

REST - Representational state transfer

SCADA - Supervisory Control and Data Acquisition

ST - Structured Text

STOMP - The Simple Text Oriented Messaging Protocol

TCP - Transmission Control Protocol

TLS - Transport Layer Security

UDP - User Datagram Protocol

XMPP - Extensible Messaging and Presence Protocol

1. INTRODUCTION

In this chapter background of the topic, general concept and examples of IoT are presented. Main aim, goals and delimitations are formulated. Methodology of the research is described.

1.1 Background

The idea of automotive industries has been in the air since the very begging of industrialization process. It is been driven by concept of providing products of as high goods quality as possible with respect to as minimal prime cost and production time as possible. Nowadays, four stages are presenting the whole evolution flow through timeline of almost three hundred years; the stages, of so-called industrial revolutions, are presented in Figure 1.

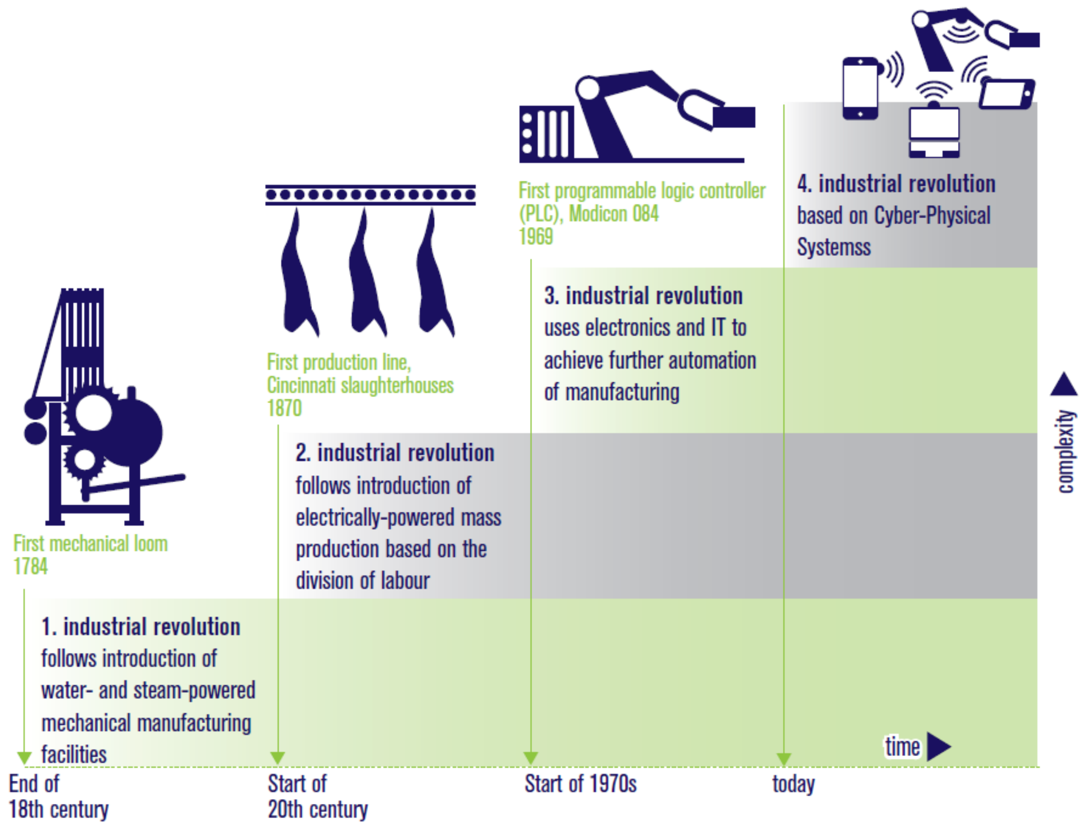

First mechanical loom
1784
1. industrial revolution
follows introduction of water- and steam-powered mechanical manufacturing facilities
First production line, Cincinnati slaughterhouses
1870
2. industrial revolution
follows introduction of electrically-powered mass production based on the division of labour
First programmable logic controller (PLC), Modicon 084
1969
3. industrial revolution
uses electronics and IT to achieve further automation of manufacturing
4. industrial revolution
based on Cyber-Physical Systemss
complexity
time
End of 18th century
Start of 20th century
Start of 1970s
today

Figure 1. Four stages of industrial revolutions [1].

Manual labor in factories was at the very start of industrialization, it was followed by mechanization of some parts of workflow since the advent of waterpower and steam power machines. The next turning point was the mass production, which was promoted by Henry Ford’s assembly line in the field of vehicles production [2]. It was a game changing idea with huge impact, it is still wildly used in the industrial field. The third turning point – is automated process with the help of computer and industrial robots.

Consequently, fourth revolution is the current stage of automation in manufacturing. That stage is called - Industry 4.0, the name was proposed by government of Germany as a part of stimulation actions of local manufactories at 2011, which, apparently, became world’s trend. In two years from introducing name of the revolution stage, the Research Union Economy – Science of the BMBF, introduced a description of the main ideas and concepts [3]. The stage is characterized by introduction to production chain cyber-physical systems (CPS).

Internet of Things (IoT) is an approach, which is aimed to help create a CPS. Originally, the term was proposed at 1999 by Kevin Ashton, engineer working at Procter & Gamble at the time [4]. Mr. Ashton revealed general idea that IoT is a new branch of global network, where data will be generated not by humans (filling forms, publishing text, photos, graphs and figures), but by machines, things and devices (publishing states, data from sensors). Alternatively, more formal definition has been given by Internet of Things Global Standards Initiative. IoT- a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies [5]. Therefore, IoT concept is interconnection of machines, real world sensors, devices with virtual cloud services, applications through which one machine can reach another. Application for IoT counts a wide range of fields and includes not only original industrial field, but also such as: travel, health care and retail.

For example, ABB company, a supplier of power and automation equipment is applying the IoT concept in production and in products for the customers. One example of predictive maintenance in heavy industry can be ABB’s gearless mill drive. Grinding process is an aggressive regime and can produce a lot of stress on the drive. Therefore, real-time monitoring system allows to avoid delays in flour producing process, which can cause substantial losses in production. The system able to alert customer about problems with the machine and prevent outages [6].

Another illustration of Industry 4.0 in work is company Fingrid - finish energy transferring company, they transmit energy from the producers to distribution companies and heavy industries via high voltage grids. The IoT approach allowed Fingrid to unite different blocks of their production made by different companies into the one ecosystem. This eases the overall system management [7]. In particular, by collecting the data from sensors Fingrid is able to check condition of instrument transformers and switchgear based on acoustic, discharge measurements, and predict fault scenarios using methods of Big Data analysis [8].

Rolls-Royce company is the famous luxury car producers but also a manufacturer of airplane engines. IoT concept can also use in the jet engines to prevent flight delays. Embedded sensors in engines send information to supervisory control and data acquisition (SCADA) program and display required information to a pilot of the plane. The analyzation system helps to calculate required fuel for the flight, maintenance of the engines and other parameters in real-time, alerting pilots and engineers on the ground. Rolls-Royce is using the Microsoft Azure platform as a core for their IoT application [9].

Description of experiences and applications of IoT in industrial field by companies can be summarized and main concept of IoT design presented on Figure 2.

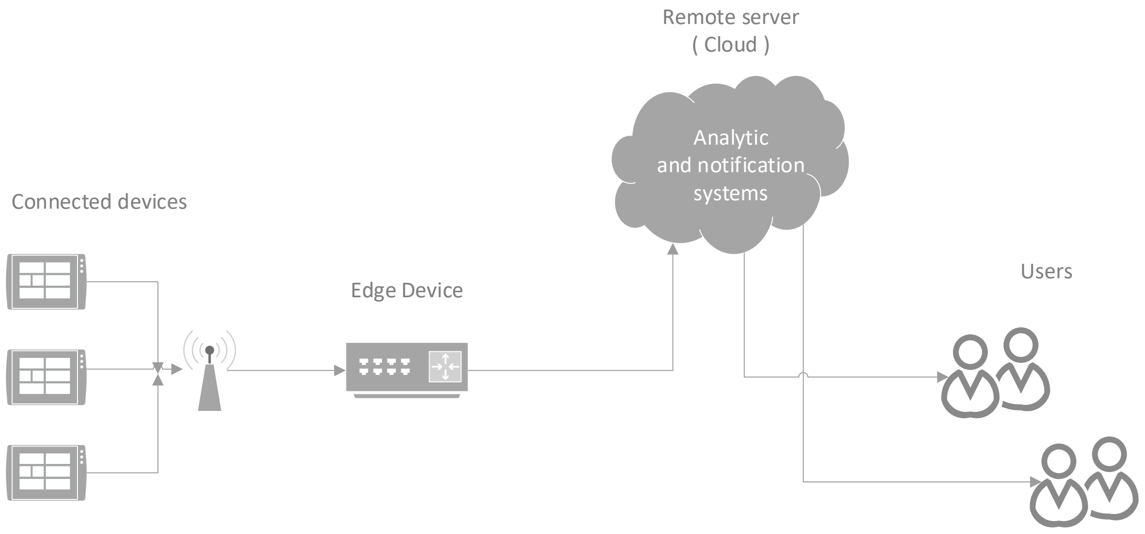


Figure 2. General IoT Structure

The basic structure of IoT system consists of a device or in other words - sensor, which is connected to a net. The sensor can send and, sometimes, receive data to/from local or remote server environment. Usually, data goes to a local server (edge device) where data reduction step take place. Embedded algorithms process the data and then required packets of data are sent to remote server via communication channel. The next chain of that structure is data processing, usually meaning storing collected data, analysis, visualization, which take place usually on remote server, cloud, and notification systems, meaning access point to the data for users.

1.2 Aim, goals and delimitation

The main aim of this research work is to create secure cloud-based CPS with respect to recent IoT practices. In accordance with the aim, following goals has to be achieved:

1. Data exchange protocol has to be chosen to establish communication channel
2. Cloud services options has to be reviewed
3. Security features applied to secure data and system
4. User notification system has to be proposed

This work was carried out in frame of the Lappeenranta University of Technology environment, thus, a hardware decisions were predefined by laboratory's equipment. Namely, industrial PC by Beckhoff is used as running platform due to build in Simulink interfaces, which enables fast prototyping. However, for low-cost-embedded IoT solution Beckhoff PC is too expensive and has excessively high computational power. For this reason, there are no limitations on the protocol selection. Nevertheless, all guidelines and decisions can be applied with other hardware solution. Besides, data receiving is not shown in final system due to local network limitations of the university.

Additionally, we do not take into account commercial features of cloud based solutions due to limited financial and work resources, free account of Thingspeak cloud is used instead. Hence, detailed descriptions and comparison of cloud servers are presented in Chapter 3.

1.3 Research methodology

A brief structure view of the report methodology is shown on following Figure 3.The research base is literature review, the IoT design, general structure and features were obtained. Common tools and practices in the field will be discussed and will be applied to developed IoT design. Detailed creating process of an IoT system prototype will be provided, with example of real application to temperature measurement of an active magnetic bearing (AMB) system. Several scenarios with use of different tools will be proposed to give respectfully full description of possible ways to design the system.

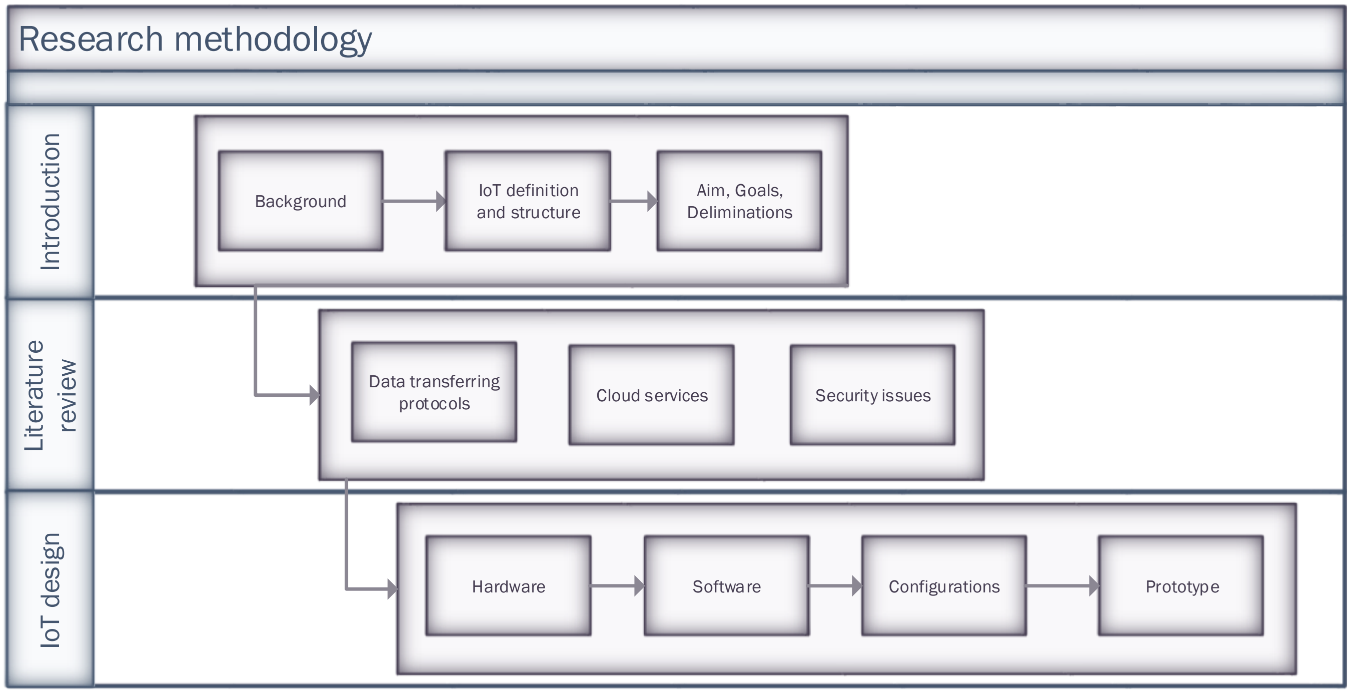


Figure 3. Research flow diagram

1.4 Organization of the study

The thesis report consists of five chapters, conclusions and summary.

Chapter 1 is introduction, where briefly described background of Industry 4.0, IoT concept. Main aim, goals and delimitations revealed.

Chapter 2 contains a description of data transferring protocols.

Chapter 3 describes cloud solutions in the research field.

Chapter 4 covers security issue in the IoT system design. Description of the main vulnerabilities and common attacks are reviewed as well as preventing actions.

Chapter 5 provides detailed information about developed IoT design. Test and results are presented.

2. COMMUNICATION PROTOCOLS AND CLOUDS

In this chapter overview of communication protocols in IoT field is presented. Detailed description of top four protocols is covered and choice for further IoT design is made.

2.1 Communication Protocols

In order to design a system for Industry 4.0, one of biggest challenge appears – establishing communication channel between devices. Coverage of the Internet of Things is wide starting from single device system up to massive deployments of real-time embedded systems. Consequently, it leads to broad options of different protocols. However, full communication stack consists of four different levels – “layers” that is shown in Figure 4.

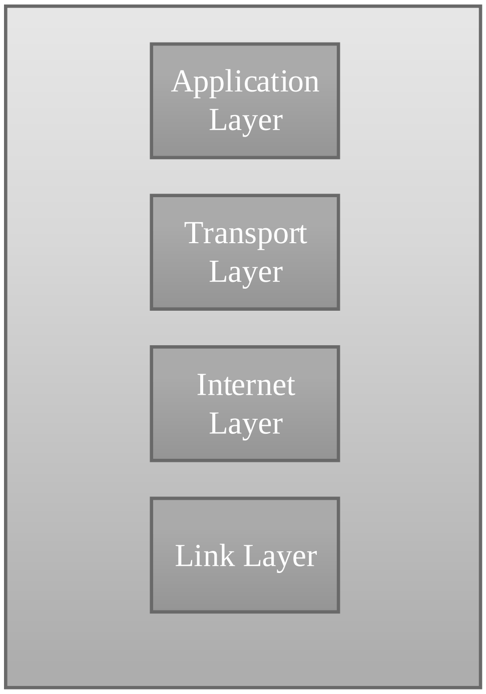


Figure 4. IoT communication stack

Namely, application layer is top layer of the Internet protocol suite, where payload is stored. Transport layer – standards and protocols, which is managing to provide end-to-end communication for application layer. There are two main protocols in the layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Followed by Internet layer – Internet Protocol (IP), layer, which includes protocols that define how data has to be transferred between hosts. Nowadays, two version of IP is used: version 4 (IPv4) and version 6 (IPv6). Finally, link layer – consist of protocols that manage a connection of a host to direct-connected network, for instance, laptop to router via Wi-Fi protocol.

Differences from the Internet when it comes to IoT concluded mainly in the application layer, where there are variety of different protocols, which try to cover requirements of the field.

According to number of studies [10, 11], the commonly used application layer protocols are listed below:

- MQTT (Message Queuing Telemetry Transport)
- CoAP (Constrained Application Protocol)
- STOMP (The Simple Text Oriented Messaging Protocol)
- XMPP (Extensible Messaging and Presence Protocol)
- Mihini/M3DA
- AMQP (Advanced Message Queuing Protocol)
- REST (Representational state transfer) - RESTful HTTP
- LWM2M (Lightweight M2M)
- OPC Unified Architecture (OPC UA).

However, research by Pertel concentrates on data protocols used in the field [12]. The research compares three sources of information: academic systematic review, experience of consulting companies, practices of industrial manufactures. They try to understand which protocol is more useful and appropriate in terms of Industry 4.0 concept with the help of bibliography survey

[12]. As a result, they came up with top four IoT data protocols: MQTT, AMQP, CoAP, OPC UA.

In the link layer researchers [11, 13] are marking the following protocols and standards, which are relevant for IoT field:

- IEEE 802.15.4
- ZWave
- 802.11 Wi-Fi
- Bluetooth Low Energy (LE)
- Zigbee
- NFC
- GPRS/2G/3G/4G/5G
- Ethernet
- RFID
- Sigfox

Final view of the communication model is shown in details in Figure 5.

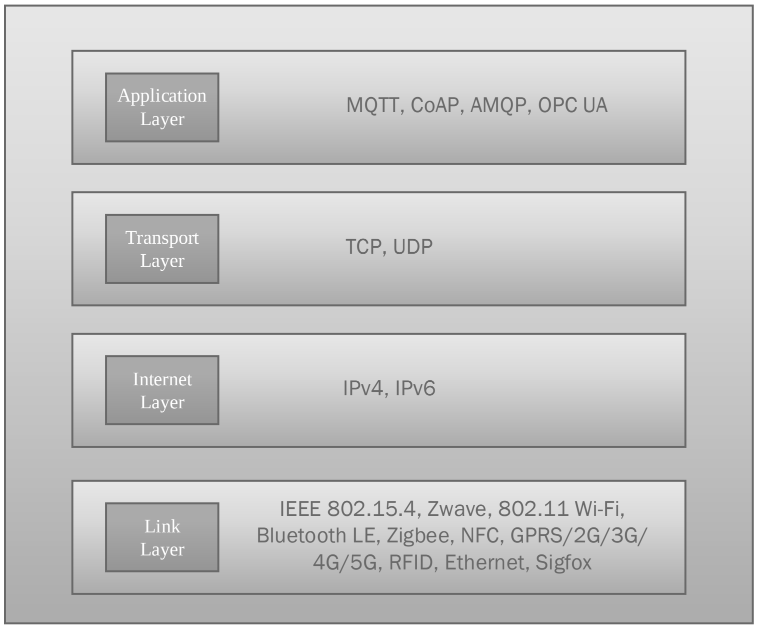

Application Layer
MQTT, CoAP, AMQP, OPC UA
Transport Layer
TCP, UDP
Internet Layer
IPv4, IPv6
Link Layer
IEEE 802.15.4, Zwave, 802.11 Wi-Fi, Bluetooth LE, Zigbee, NFC, GPRS/2G/3G/4G/5G, RFID, Ethernet, Sigfox

Figure 5. IoT communication suit.

Therefore, communication channel can be established by choosing appropriate standards and protocols for data transportation. To make the decision it is enough to choose protocol in application layer, because that protocol supports particular protocols in transport and Internet layers, so these layers will be predefined by application layer protocol. Link layer is usually specified by the hardware solutions. Thereby, below attention will be concentrated on review of data protocols particularly in the application layer.

2.1.2 MQTT Protocol

MQTT is a simple publish/subscribe based protocol. The main idea is that the clients subscribe on particular topics, playing role of addresses, in which they are interested in. The connection is maintained by TCP/IP protocol to a broker, which is playing role of a server. Therefore, publishers send messages to the broker, which spread them on topics to which clients (subscribers) are listening to. It was originally developed by IBM Company in 1999 and released as open source protocol standard in 2010 [14]. General structure can be seen in Figure 6.

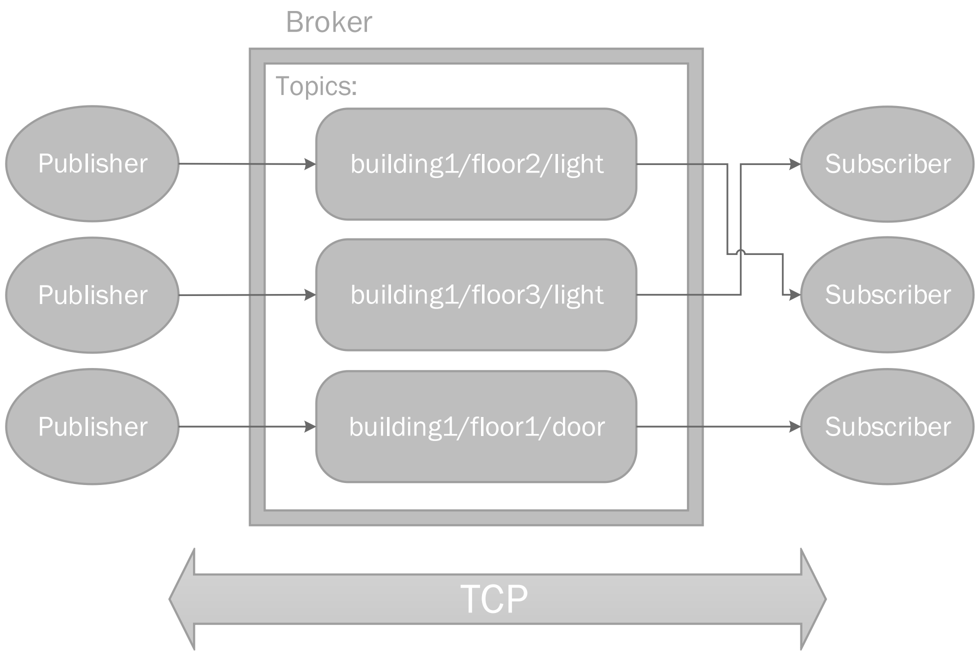


Figure 6. General structure of the MQTT protocol.

MQTT protocol works by exchanging control packets in specific form. Usually, there are three parts in a MQTT control packet: fixed header, variable header and payload [15]. Detailed information on control packets can be found in official standardized documentation [16].

Three quality of service (QoS) levels are available in MQTT, feature to control how messages are delivered. QoS 0 – at most once delivery, meaning that message is delivered without any feedback response, message can be lost if subscriber unexpectedly disconnects or broker fails. QoS 1 – at least once delivery, a broker will deliver message to subscriber at least once, however duplicate of the message is possible. QoS 2 – exactly once delivery, message is delivered only once and a broker ensure that there are no duplicates [15].

Topics in the protocol has hierarchical structure. For subscribers there is option to use wildcards to receive all messages on particular topic and all subtopics of it, using sharp (#) symbol, for instance to receive all messages from first building subscriber has to specify the following topic building1/#, consequently all messages from topics as building1/floor1/temperature, building1/floor2/temperature, building1/floor1/doors/lock1 will be received. Additionally, it is possible to subscribe on one level of subtopics beneath topic, using plus (+) symbol. For example, client subscribes on building1/+, hence user will receive all messages on that level like building1/floor1, building1/floor2 and not building1/floor1/temperature or building1/floor1/doors [15].

2.1.3 AMQP Protocol

AMQP is an open source protocol that was designed to provide a message passing system for business applications and organizations [17]. It provides publish/subscribe scheme. Devices in the net are listening to messages on each queue, which is quite similar with previously described MQTT. Therefore, the structure looks just like a previous one except the inner part of broker system. To define to whom a message is addressed to the publisher has to provide information for which queue the payload has to be delivered and, consequently, subscriber is listening to that particular queue. There is also distributing mechanism in the broker so called "exchange" block, which delivers message to desired queue. Example of the protocol scheme is shown on Figure 7.

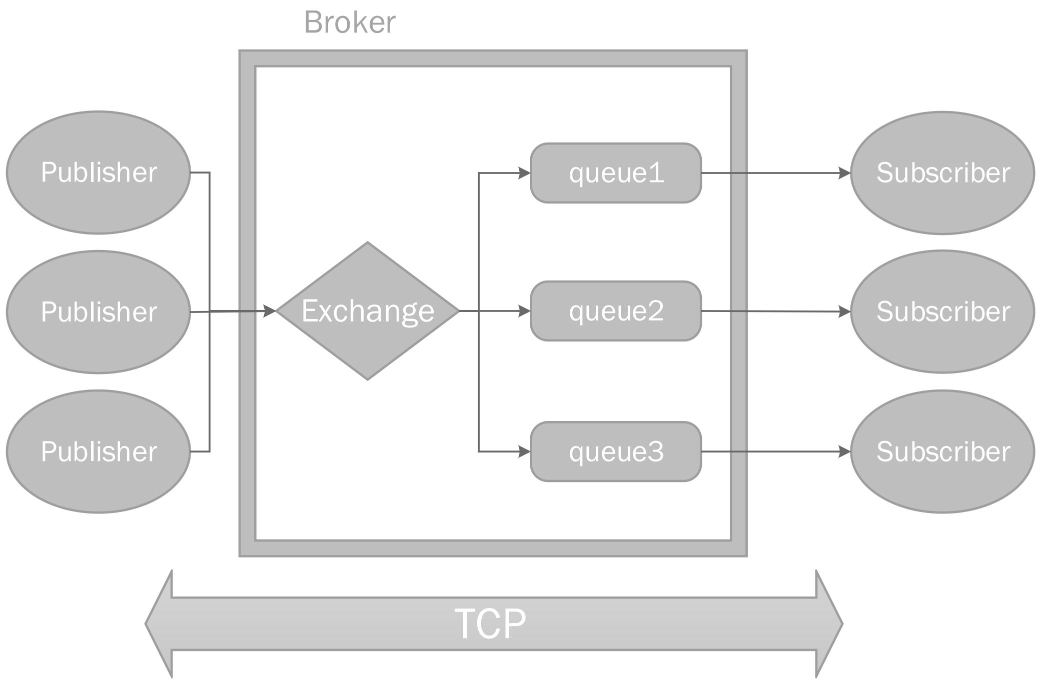


Figure 7. General AMQP structure.

2.1.4 COAP Protocol

Constrained application (CoAP) protocol is client-server based messaging protocol, which is working through User Datagram Protocol (UDP). It was developed for resource-constrained devices, which are mainly applied in IoT field. CoAP has similarity with HTTP and it uses identical commands: GET, POST, PUT, DELETE. Moreover, both of them can be easily mapped to each other [16]. General scheme of CoAP is shown in Figure 8.

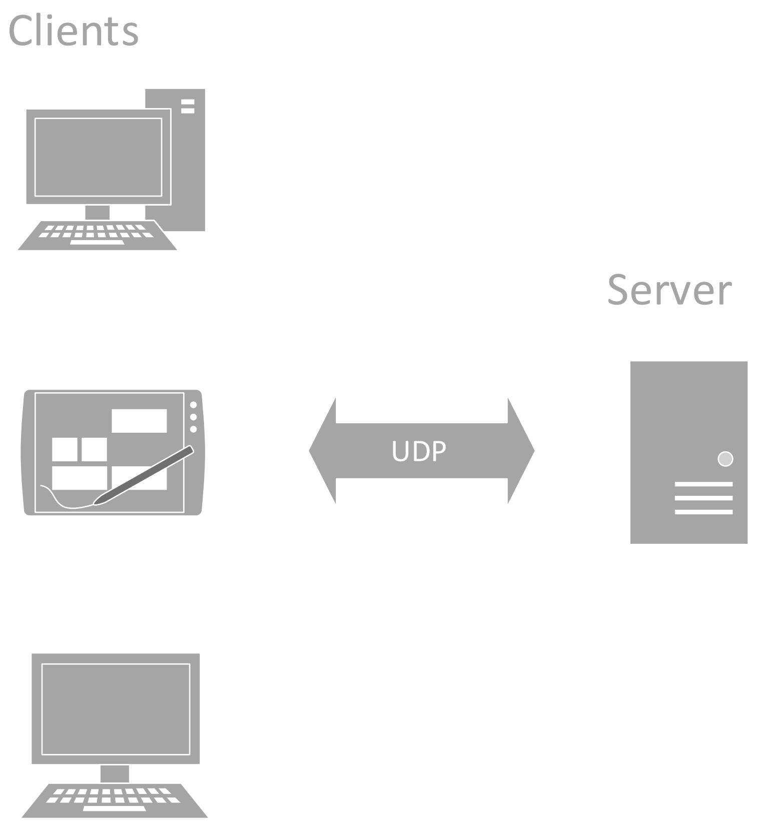


Figure 8. Client-server architecture.

2.1.5 OPC UA Protocol

Unified Architecture (OPC UA) is the next generation of the OPC standard with client-server architecture like CoAP. It is data exchange protocol with high demands for standardization aiming field of industrial automation [18].

2.1.6 Protocol Choice

Selection of suitable protocol is set mostly by tasks, which system has to solve. Common requirements for IoT system are: secured design and data, low power consumption due to either limited energy source, for instance work in remote areas such as a drilling station, either restricted price policy. Overall comparison of the main features of described protocols is listed in Table 1.

Table 1. Comparison of IoT data protocols [19, 20].

Protocols	Architecture	Transport Layer	Security feature
MQTT	Publisher/Subscriber	TCP	+
AMQP	Publisher/Subscriber	TCP	+
CoAP	Client/Server	UDP	+
OPC UA	Client/Server	TCP	+

MQTT, CoAP and AMQP have a light footprint, hence, can be easily used in the embedded systems with low power and low computational performance. OPC UA is heavier to run, with many additional features and configurations. Security feature column in Table 1 shows support of common security options such as data encryption, user credentials by protocol. Marks "+" stand for providing the support and "-" for opposite situation. Consequently, all protocols have pluses ("+") meaning that all of these protocols support configurable security features [15-19]. Detailed information on security topic will be presented in Chapter 4.

The focus of the study is to determine design of cloud based system, thus protocols support by cloud services has to be taken into account. Comparison between cloud service providers and supported protocols is shown in Table 2. Detailed discussion about cloud solutions will be provided in Chapter 3. It is notable that common cloud providers do not support OPC UA and AMQP.

Table 2. Supported protocols

Cloud service name	MQTT	AMQP	HTTP/HTTPS
IoT Hub (Azure)	+	+	+
AWS IoT (Amazon)	+		+

Continuation of Table 2

Watson IoT (IBM)	+		+
Thingspeak (Matworks)	+		+

Therefore, after brief review of protocol properties and support by cloud services it is possible to conclude that the most suitable protocol is MQTT. In further work this protocol will be used, however, choice of protocol is defined by requirements of a system and task itself and there is no exact uniform solution.

3. CLOUD SOLUTIONS

Cloud can generally be described as a server where user can store and manipulate data. Physical location of the server is not restricted. To clarify scenarios of use review of four cloud services for Internet of Things by largest companies in the IT and engineering fields will be discussed: Microsoft, Amazon, IBM and Mathworks. Microsoft, IBM and Amazon are providing rich list of web services, each service stands for particular task such as data storage, analytics, and virtual machines. Combining the items, user can create his own system to solve required tasks. Mathworks provides a ready-made cloud platform for IoT applications – ThingSpeak. It is free web service where you can collect sensor data and with the tools provided by the application visualization and data analysis can be done [21]. Detailed information about services can be found in references [22-24].

Considering an IoT application, services are proposing quite similar functionality. Generally, an IoT solution consists of different units on cloud side. First unit is receiver and transceiver for the data from and to a thing-device. Each of listed companies has their own name for that item: IoT Hub for Microsoft Azure, AWS IoT for Amazon, Watson IoT for IBM and ThingSpeak for Mathworks. Second unit is responsible for generating a reaction on incoming data, like send to database. Further items of the system are task dependent, for instance, it can be analysis, visualization, notification units. On Figure 9 and Figure 10 examples of typical IoT application are provided by Amazon and Microsoft.

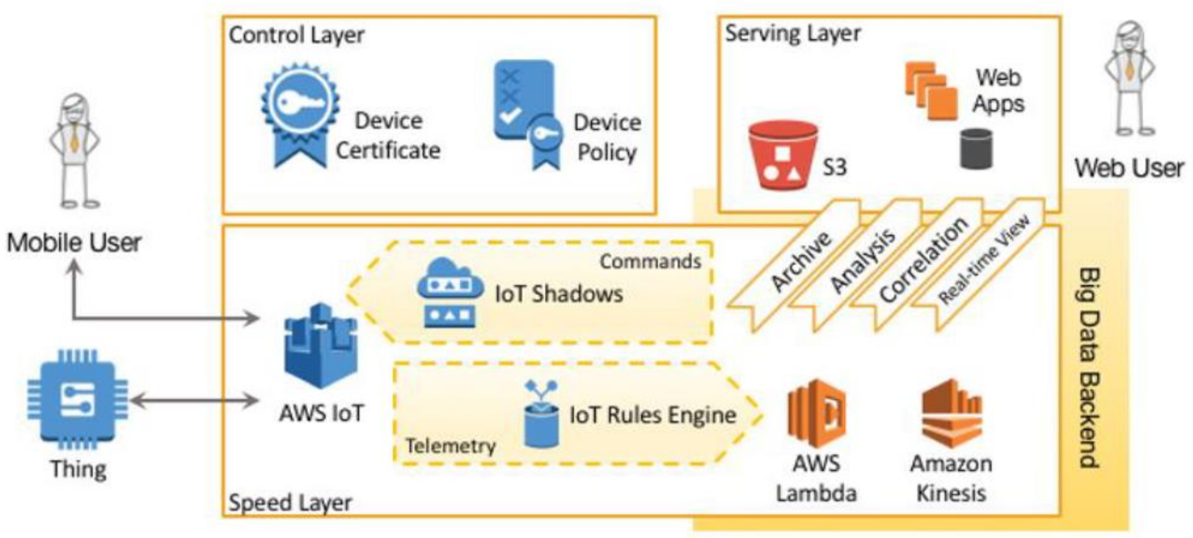


Figure 9. Structure of AWS IoT solution [25].

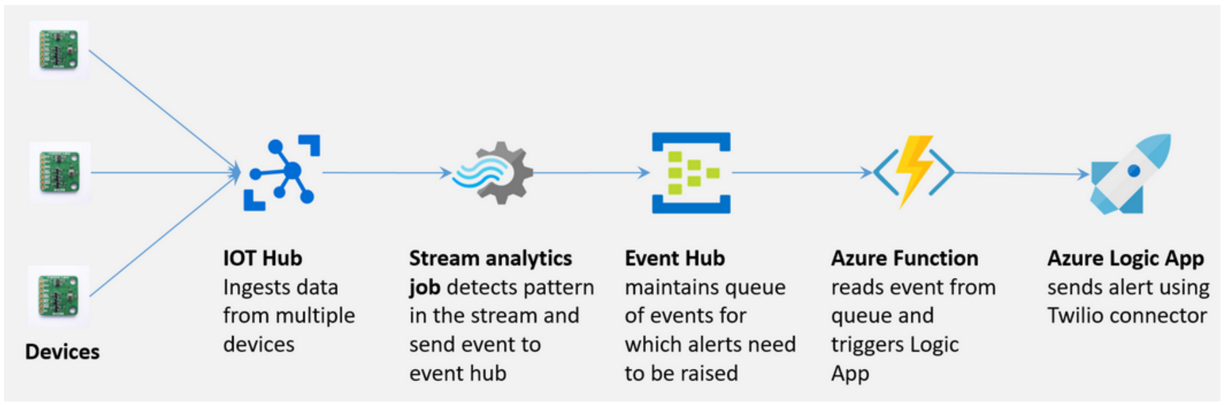

Devices
IOT Hub Ingests data from multiple devices
Stream analytics job detects pattern in the stream and send event to event hub
Event Hub maintains queue of events for which alerts need to be raised
Azure Function reads event from queue and triggers Logic App
Azure Logic App sends alert using Twilio connector

Figure 10. IoT system structure with use of Microsoft's services [26].

Price is calculated with respect to used items in system design. In Table 3 and Table 4 comparison of tariffs and limits of the services are shown. However, comparison is done only with respect to simple system with one node - device connectivity item due to prototyping focus of the study, time and financial limitations. Detailed information on pricing, quotas and limitations of cloud services can be found in official documentations [27-32]. In cases of IBM and Amazon solutions price is calculated individually with respect to client's task.

Table 3. Clouds Non- commercial plans comparison.

Plan limitations	**ThingSpeak by Mathworks**	**Azure IoT Hub by Microsoft**	**Watson IoT by IBM**	**AWS IoT by Amazon**
Number of messages	max 3 million msg/year (~8.200 msg/day)	max 8.000 msg/day	max 200 MB/month (~12.796 msg/day)	500,000 msg (trial for 37500 hours)
Message update interval limit	15 sec	12 msg/sec	5 msg/sec	100 msg/sec
Number of simultaneous MQTT subscriptions	3	-	500	500 subs/sec/account
Data storage capability	10 million messages for 3 years	Required additional unit	Required additional unit	Required additional unit
Message size restriction (device to cloud)	max 3 KB	max 256 KB	max 128 KB	max 512 KB

Table 4. Clouds commercial plans comparison.

Plan limitations	**ThingSpeak by Mathworks**	**Azure IoT Hub by Microsoft**			**Watson IoT by IBM**	**AWS IoT by Amazon**
Plan name	Standard	S1	S2	S3	Standard	-
Number of messages (msg/day)	$9 \cdot 10^4$	$8 \cdot 10^3$	$6 \cdot 10^6$	$3 \cdot 10^8$	unlimited	-

Continuation of Table 4

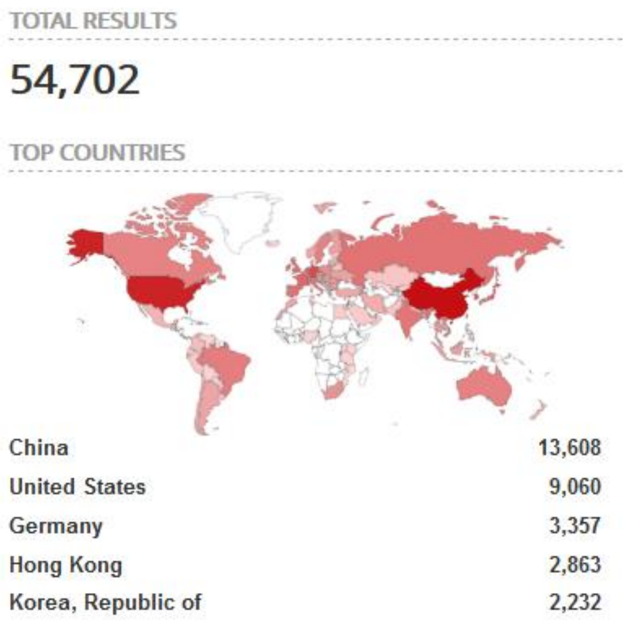
Message update interval limit	Every second	12 msg/sec	120 msg/sec	6000 msg/sec	10 msg/sec	20,000 msg/sec/account
Number of simultaneous MQTT subscriptions	50	-			500K	500 subs/sec/account
Data storage capability	100 million messages per unit for 3 years	-			-	-
Message size restriction (device to cloud)	max 3 KB	max 256 KB			max 128 KB	max 512 KB
Price (per unit per months)	48 €	21.09 €	210.83 €	2,108.25 €	calculated individually	calculated individually
Analytic tools	Included (MATLAB)	Additional (Stream Analytics 0.102 €/h)			Additional	Additional

From table above, Thingspeak with such features as included possibility to proceed data with MATLAB’s toolboxes on side of server, even with non-commercial license, web-service with visualization and cloud storage space are made the service the most suitable for prototyping and requirements of the study. However, for commercial use Thingspeak has quite limited features compared with other services.

Additionally, IoT field is a challenging area, it is still young and rapidly-grows. Probably, one of the biggest issue is to decide whereas go with third party services, either build your own solutions based on open source tools, for instance, PostgreSQL to store data, mosquitto broker to receive and transceive the data and libraries of python to react and analyze it. It has to be taken into account that cloud services are aiming to decrease lunch time of an application. Meaning, that to start any solution company can spend more time on product instead of spending time on back-end hardware decisions, developing scalable system for numbers of users and make it robust. Hence, by purchasing plan from third party company it is possible to start in number of hours in contrast of several weeks or months. Moreover, it is quite cheap if enterprise does not have a large data flow. Therefore, third party solutions are highly suitable for quick start and prototyping. However, it should be always in mind of the company that it is temporary decision and switching to their own solution in future will decrease product cost eventually. Nevertheless, it seems from companies practices that major part of them are balancing between those two edges, finding their golden mean.

4. IOT SECURITY PRACTICES

Security is the one of the most important aspect to have reliable IoT network. Designing net of real, physical devices and aiming to collect data on a remote server leads to an idea that while data is transferred it can be intercept. Moreover, bigger threat appeared when the concept of remote control of a device is under consideration, interception and taking control over channel of that kind can have crucial effect, especially when it comes to industrial cases. In the research by Syaiful following information was highlighted: 24998 MQTT broker worldwide devices were not using any encryption to protect data, meaning anybody can access it easily [33]. The same experiment with help of Shodan search engine was performed in August 2018, the results are presented on Figure 11 [33].


TOTAL RESULTS

54,702

TOP COUNTRIES

China	13,608
United States	9,060
Germany	3,357
Hong Kong	2,863
Korea, Republic of	2,232

Figure 11. MQTT brokers without encryption worldwide [33].

Therefore, number of devices without any encryption and default settings increases. The most recent security reports and issues have been reviewed as well as application of preventing actions have to be taken into account when concerned about IoT design.

Open Web Application Security Project (OWASP) is developing a project to define IoT vulnerabilities and help to create secure IoT [34]. OWASP's team came up with following list of top ten exposures:

- Insecure web interface
- Insufficient authentication/authorization
- Insecure network services
- Lack of transport encryption
- Privacy concerns
- Insecure cloud interface
- Insecure mobile interface
- Insufficient security configurability
- Insecure software/firmware
- Poor physical security

First point of list is insecure web interface that is covering issues linked with administrative interfaces, web pages of a device. Two main obstacles are specified: use of default credentials and absence of account lock out – security feature that locks any account that has failed to login more than a defined number of tries in a row.

Second point covers problems with the device authentication and the authorization to interfaces and services. Weak passwords, insecure password recovery mechanisms and absence of two-factor authentication feature. Two-factor approach confirms the user in a system by requesting not only credentials but also some personal information, for instance, identification number. These are the main factors of that list point. These obstacles are the same for "Insecure Cloud and Mobile interfaces" points.

Third point of the list describes insecurity of network services. Issues of network services of IoT system including device itself, cloud, web and mobile. Opened unnecessary ports in the network environment gives opportunity to attacks, OWASP also noticed that utilization of UPnP for ports, which exposed to the global internet leads to buffer overflow attacks.

Fourth point covers the same services as a previous one. The main obstacles are linked with the encryption issues in the transport layer of an IoT system. Thus, transferred sensitive data is not encrypted, encryption protocols poorly configured or not used at all.

Researchers of OWASP highlighted insufficient security configurability for IoT device as vulnerabilities list with following obstacles: credentials with password, encryption option is not available. Insecure software and firmware can be defined with quite similar obstacles, where update servers are not secured or encryption is not applied for updates.

The last but not least item in the list is poor physical security of an IoT device. Three obstacles presented in OWASP report – existence of unnecessary external ports in design of a device, open access to operating system through remove media and inability to limit administrative rights.

The top ten list was carried out in 2014, however, the research by Tankard has shown quite similar trends and issues [35]. Moreover, in the report following statistic showed us that 70 % of all IoT devices do not encrypt data transmission and 80 % have opened backdoor. With simple preventing actions can significantly improve security [35].

Common attack practices to an IoT system according to Russel are [13]:

- Wired and wireless scanning and mapping attacks
- Protocol attacks
- Eavesdropping attacks (loss of confidentiality)
- Cryptographic algorithm and key management attacks
- Spoofing and masquerading (authentication attacks)

- Operating system and application integrity attacks
- Denial of service and jamming
- Physical security attacks (for example, tampering, interface exposures)
- Access control attacks (privilege escalation)

Recent security issues have been analyzed by Cisco in 2018 [36]. Positive trends in the main vulnerability categories stay the same. Nevertheless, Cisco research group have found out a new type of exposures – botnets [36]. IoT botnets is a distributed network of infected IoT devices that are used to work as parts of one computer system. The aim of the net is to initiate massive distributed denial-of-service attack (DDoS), consequently, partly or fully control under IoT device is taken by ill-wishers. Two main characteristics of the network are rapid exponential grow and low detection rate [37]. One of largest botnets Mirai in 2016, in peak had 380 thousands of invaded devices [38].

Therefore, general rules should be taken into account to provide high level of security in IoT design:

- Use strong passwords in all system's segments.
- Implement multi – factor authentication where it is possible.
- Minimize open ports to global network.
- Do not utilize UPnP.
- Encrypt communication between components of system.
- Implement encryption protocols such as transport layer security (TLS) protocol.
- Do not use proprietary encryption solution, open backdoor issue possible.

- Anonymize collected data from devices.
- Use lockout feature where possible.
- Sign updates.
- Secure update servers.
- Minimize unauthorized physical access to IoT device or servers.
- Minimize external port such as USB.
- Limit administrative rights.

To sum up, high interest to the IoT topic from manufactures, developers and consumers together with fast growing and competitive market’s demands leads to necessity quickly respond sometimes in disregard for security side of the issue. However, by following reviewed set of general rules and awareness of possible threats will improve security of a device design greatly. Nevertheless, it should be realized that basic principles cannot fully satisfy all aspects in the topic and cannot be counted as a universal solution. These principles can be used as a starting point with decent degree of coverage the main ideas. For this reason, the individual needs and project demands will form treating way to security question. In addition, project OWASP Internet of Things Project including IoT Attack Surface Areas Project and guidelines for manufacturers, developers and even for consumers can be used to find out detailed information and recent practices in the topic of IoT security [39-42].

5. IOT DESIGN

In this section, detailed description of developing process of cloud-based CPS system is described. Designing process starts with general structure of the system and followed by security feature implementation, description of data access interfaces. The chapter is summarized by presenting results of prototype test.

5.1 General Structure of The Design

In the research hardware solutions is predefined. It is an industrial PC by Beckhoff, due to described delimitations in introduction part. IoT based cloud measurement approach is applied to an active magnetic bearing (AMB) system where different quantities can be measured such as temperature, current, voltage, speed and rotor position. These quantities can be used for the system condition monitoring. General structure of the system is presented on Figure 12.

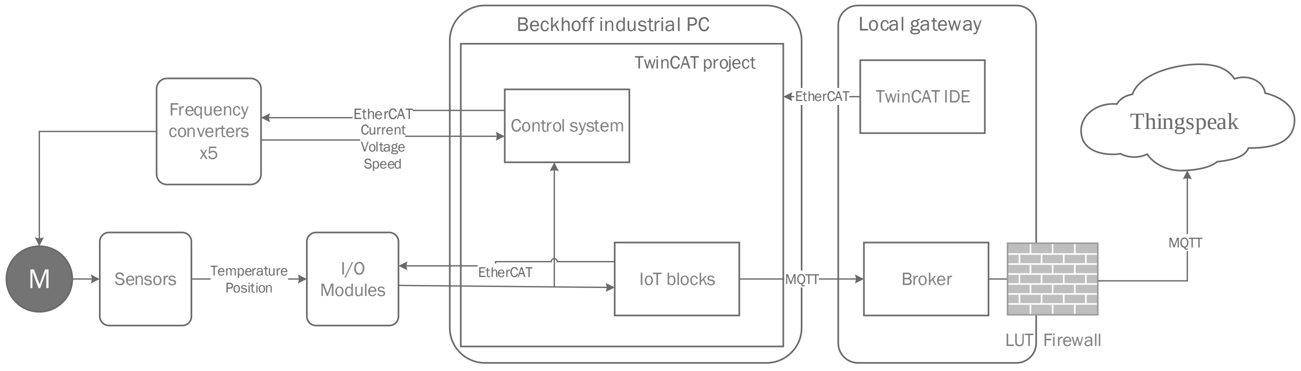


Figure 12. Industrial IoT structure.

Software aspect of the project has several parts. First, it is control system of the AMB, which is generated from MATLAB Simulink block and transferred to TwinCAT integrated development environment (IDE) where programming Bechoff's products is made. Second, it is inputs and outputs of the hardware, where data from sensors is coming. Third is part for programmable logic controller (PLC), using structured text (ST) and IoT library by Beckhoff, group of IoT blocks were developed. These blocks can read data from Simulink control system part, where feedback sensors are used, then form required packets and send it to requested cloud, in our case for Thingspeak.

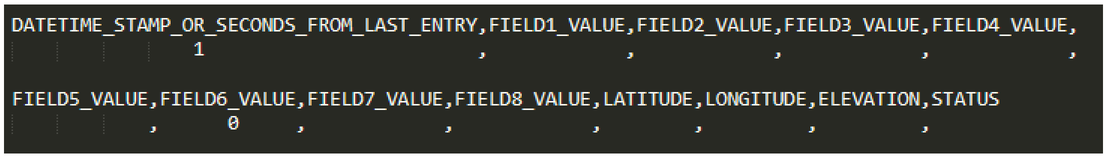
5.2 Communication Layer

MQTT protocol is in charge of data transporting from Beckhoff industrial PC to the broker. To publish data from TwinCAT project IoTpublisher functional block was developed, with two inputs – a payload to publish and configurations of the publisher. To configure the publisher FB_IoTPublisherConfigs functional block was created. Specifying address of a broker host in hostName parameter, through which port data will be transferred in portNumber and to which topic the data has to be published in sTopicPub parameter, publisher configuration is done. Next step is to pass configurations and desired payload to IoTpublisher functional block.

MQTT client is configured according to defined settings from FB_IoTPublisherConfigs with help of FB_IoTMqttClient functional block from Beckhoff's Tc3_IoTBase library. Right after connection is established with broker. If an error then appeared functional block stops and show error code in hrErrorOccurred variable. All codes of errors can be found in Beckhoffs documentation [43]. In case of successfully established connection payload is sent to the broker by Publish method of FB_IoTMqttClient functional block. Full design of developed system and all functions is shown on Figure 18.

Raw data cannot be send without preparation, usually cloud services required particular view of data to be sent. In case of Thingspeak and MQTT protocol, it is required to provide specific topic format. It has to be in the following format to update particular field in a channel: *channels/<channelID>/publish/fields/field<fieldnumber>/<apikey>*, where channelID and apikey can be found in Thingspeak’s profile settings and fieldnumber is a number of desired field to publish data to. There is an option to publish simultaneously to multiple fields in a channel, following topic format is required: *channels/<channelID>/publish/<apikey>*. In that case additional formatting has to be done with payload. Firstly, fields number and value to publish has to be provided as follows: field1=100&field2=50&…&fieldx=value. It is also possible to provide some information like timezone, location of channel, status and date of creation the message. Detailed information can be found in Thingspeak documentation [22].

Non-commercial Thingspeak’s plan allows to publish data every 15 seconds, Table 3, this suits not for very dynamic measurements. Moreover, commercial plan allows posting message every second, Table 3, and it is not suitable for such task as driver position measurements. For that reason, Thingspeak’s bulk-write feature was applied to the system. Bulk-write option allows to publish up to 960 messages for non-commercial plan and 14400 for commercial one with one upload [22]. However, bulk update is not supported by MQTT protocol and uses HTTPS instead. To use that publishing method new subscriber was introduced to the broker system, the client is subscribed on *bulkWrite* topic. When data is published to the topic the subscriber, namely python script MQTTtoHTTPSbridge, receive data from broker using MQTT and wrap incoming data in HTTPS POST method. The bulk-write feature supports two data formats for data to publish: JavaScript Object Notation (JSON) and Comma-Separated Values (CVS). In the system second one is used due to simple formatting and parsing. Therefore, on side of TwinCAT project thingspeakPack function was created to wrap a payload in CVS format, each message is separated with a pipe character. A message has to contain following update parameters separated with comma: timestamp, field number, latitude, longitude, elevation, status. If parameter is not used, it has to be still separated with comma. It can be shown on one message example as follows, first string represents name of parameters and second is the format of payload:



```
DATETIME_STAMP_OR_SECONDS_FROM_LAST_ENTRY,FIELD1_VALUE,FIELD2_VALUE,FIELD3_VALUE,FIELD4_VALUE,
                1                        ,            ,            ,            ,            ,

FIELD5_VALUE,FIELD6_VALUE,FIELD7_VALUE,FIELD8_VALUE,LATITUDE,LONGITUDE,ELEVATION,STATUS
            ,      0     ,            ,            ,        ,         ,         ,
```

Figure 13. Bulk-write update parameters package

In other words to publish, for instance, three messages to field 6 without any additional parameters of payload have to have as following look:

```
1,,,,,,0,,,,,,|2,,,,,,1.5388,,,,,,|3,,,,,,-1.5388,,,,,,|
```

Functional thingspeakPack has two input arguments: data and field number. Calling that function cyclically it will add new values to existing buffer with respect to described pipe-comma format. Consequently, MQTTtoHTTPSbridge will pack payload in HTTPS POST method with defined parameters: url has to be in the following format *https://api.thingspeak.com/channels/<channel_id>/bulk_update.csv* , contect-type of POST method has to be specified as *application/x-www-form-urlencoded* and body parameters have to contain *write_api_key*, *time_format*, *updates*. Hence, general format of bulk-write request has the following format:

write_api_key=WRITE_API_KEY&timeFormat=TIME_FORMAT&updates=payload

Detailed installation guide, code of IoT blocks can be found in appendices: installation guide of the system – Appendix 1, configurations of broker – Appendix 2 – 4, example of the main program to send data points to Thingspeak cloud – Appendix 5, thingspeakPack – Appendix 6, IoTpublisher – Appendix 7, FB_IoTPublisherConfigs – Appendix 8, MQTTtoHTTPSbridge – Appendix 9. Detailed structure of described system on Figure 14.

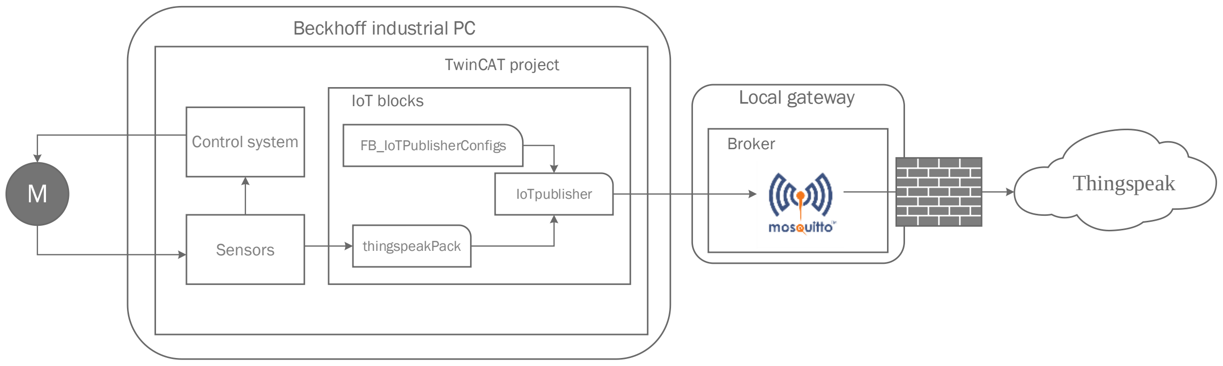


Figure 14. Detailed system structure

Therefore, proposed system is able to send data to Thingspeak cloud in two different modes. First point-by-point mode allows to send one message every 15 seconds, one message can carry eight field updates, status flag, channel location data. In that mode Tingspeak cloud can receive up to 8200 messages daily, maximum 3 million messages per year. It is worth noting that during the research daily peak point does not exceed 3760 messages. Second mode is bulk-write update mode, where 960 messages can be sent in single bulk-update every 15 seconds. However, limitation of total messages, which can be published to Thingspeak cloud remains the same, 3 million messages per year for non-commercial plan. Hence, it is possible to estimate data rates of the communication chain with accordance to upper limits. Maximum receiving speed of data from convertors and sensors is 30 MB/s, since control system block, Beckhoff PC, are working with sample rate equal to 50 μs and maximum payload for EtherCAT protocol is 1.5 KB. Comparatively to data flow through IoT node of the chain can handle speed rate of 0.192 MB/s, using bulk-update option.

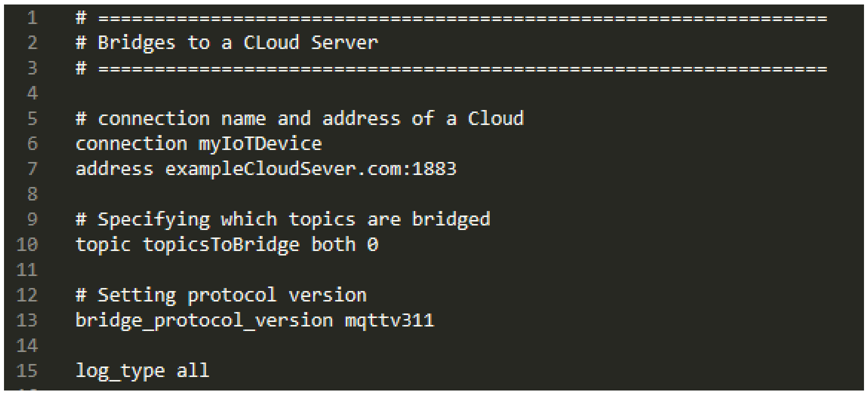
Therefore, the limitations give understanding that not all data has to be and can be transferred to a cloud. Moreover, time delays, to send data, analyze it and send reaction back, are crucial in such cases as bypassing obstacles by autonomous car, calling for help from wearable device or taking decision to stop workflow on a factory due to emergency signal. Therefore, partly data has to be analyzed locally, on an edge device (local server) with decent computational power, decision has to be considered with accordance to task priority.

5.3 Broker

As it shown on Figure 14, the data goes not directly to the cloud, the caused by architecture inside our university network environment, that fact has two sides. On the one hand, it has positive impact, it means that IT department has their own security walls, for instances firewall and identification of all devices, which leads to additional step of our design security. On the other hand, additional node in our system has to be created, which automatically leads to increasing system complexity, consequently, more force has to be applied to create secured system and maintain it. It is worthwhile noting, that quite commonly in the industrial field there is no direct connection to global network due to security issues, thus we are concerning respectively real case in the research. As a result, instead of running our broker on the side of cloud it will be deployed on a local (inside plant, in our case university) server, which has access to global net, it will be our local gateway (edge device) with function of a “bridge”. The bridge will retransfer all the data to defined subscribers. Therefore, our broker has publisher – PLC part of TwinCAT project, and subscriber is the cloud server. The publisher sends data to local broker, right after that the broker bridges data to the subscriber.

Considering broker systems, choice turns on Mosquitto broker developed by Eclipse Foundation [44]. Eclipse Mosquitto is an open source message broker that implement latest versions of MQTT protocol, namely 3.1 and 3.11 versions. Moreover, it is lightweight and suitable for low power cases, it is reachable by configuring it with help of wide range of instructions [44]. Additionally, the project provides implementation of MQTT protocol as a client libraries written on C, C ++ and for Python, in that particular case it is called Paho [44]. The Mosquitto broker was compiled from source code with additional editing due to a bug, which exist when running on Windows operational system and bridging with encryption. Detailed information on how to fix the problem can be found on Github’s page of the project [45].

Therefore, proposed structure can collect data and bridge it to a cloud server. Mosquitto bridge configured in, generally, three steps. Firstly, all configurations are done in file with “.conf” extension. Secondly, name of connection to establish and address of a cloud server, where data has to be retransferred, have to be specified. Thirdly, what topics has to be bridged, then which version of protocols, finally, turning logging on in order to make debugging process easier. Example of code can be found on Figure 15.



```
# ==================================================================
# Bridges to a CLoud Server
# ==================================================================

# connection name and address of a Cloud
connection myIoTDevice
address exampleCloudSever.com:1883

# Specifying which topics are bridged
topic topicsToBridge both 0

# Setting protocol version
bridge_protocol_version mqttv311

log_type all
```

Figure 15. General case of Mosquitto bridge configurations.

Hence, described system can handle almost all requirements of an industrial IoT device. Moreover, direct implementation of the prototype is possible but not recommended, because none of security features were applied and the design is totally unsecured. However, the developer can rely on high level of net security handled by IT department. Clearly, it is not the best practices, and in the IoT world one of the key trend is to make security of that kind of devices out of a box feature, meaning by default and embedded in the design itself [41].

5.4 Security Features Implementation

As it was discussed above all transferred data between nodes of the system has to be encrypted, thus, latest version of TLS, version 1.3, is in use. TLS is cryptographic protocol that allows to provide secured communication over network. One of the notable applications of it is protection clients on webpages while browsing with known from daily life - hypertext transferred protocol secure (HTTPS), which is using TLS.

Underneath a cryptographic protocol lies an idea to establish closed channel by encrypting data with help of special keys – sets of rules known only to those who are willing to exchange any data, hence, we are getting end-to-end communication and avoiding “Man In The Middle” attacks (MITM). MITM attack is when hacker are getting accesses to communication channel and receive all transferred data, thus, even if somebody will listen encrypted channel it will not be possible to understand anything about payload.

The private channel is organized by performing so called handshake process. The process is done in several steps: exchange of greeting messages, key exchange, and finally we will get a channel with symmetric encryption. Implementation of the latest version of the encryption protocol is done with help of OpenSSL toolkit, which includes full-featured TLS protocol, functions of general-purpose cryptography library. Furthermore, OpenSSL project is distributed under an Apache-style license, meaning that is possible to use it in commercial and non-commercial projects freely, moreover source code is opened, and anybody can assure themselves that there are no backdoors in it, that is one of major security feature, discussed in second chapter. Included cryptography library allows to generate public and private keys as a part of certificates, which is needed in handshake step.

Additional configurations are done in broker settings to embed encryption feature as well as settings for user credential with respect to security rules from previous chapter. Full code is listed in appendix section. As a testing example the following program is used, the Beckhoff PC is generating sinusoidal wave and send it to cloud, general structure of the system and result are shown on Figure 16.

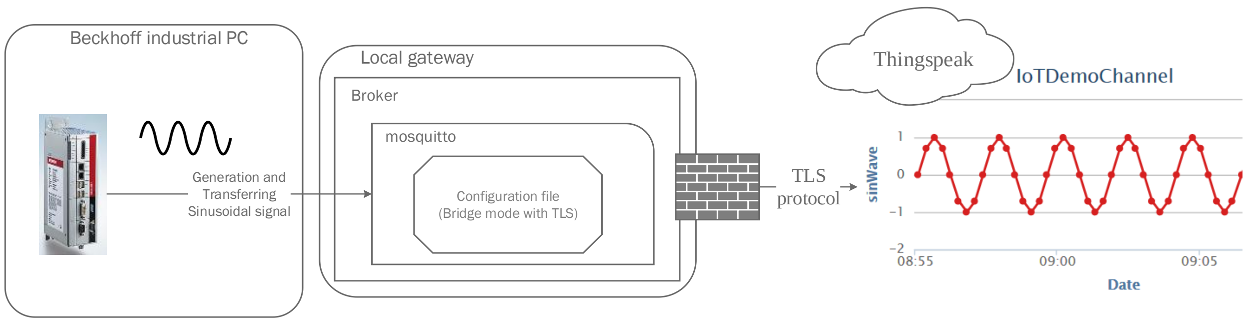


Figure 16. Sinusoidal waveform to cloud.

5.5 Data Access Interfaces

There are several ways to access data and interact with it, through Thingspeak web interface, desktop MATLAB or TwinCAT IoT communicator application by Beckhoff for smartphones. However, all of them provide possibility to work with data, they are all closed-source platforms, which are not secured due to possible backdoors. Hence, Telegram messenger's bot was created as an option to notify users about critical states of system and provide status of the system. The messenger is cloud-based mobile and desktop messaging application, focus of the company is security, speed and openness [46].

Web-interface of Thingspeak service consist of channels with fields, maximum number of fields is eight. There are two types of channel: private and public. Fields represent data storage unit. Therefore, data published to channel to particular field and automatically is plotted with respect to field configuration like: name of axis, format, timescale, type and color of line. Hence, data can be in non-plottable format, for instance text. In that case field will not plot data automatically, but will just store it. However, it is possible to create custom rule to visualize received data in the cloud with help of MATLAB Visualizations functions. Another option is to use desktop MATLAB to access data. Standard library is providing functions to read, write and plot in cloud data [22].

Next option is notification systems. They are playing roles of subscribers in our design. Thus, the broker has to be accessed from outside LUT network, which is not possible due to strict policy of IT department's firewall system. Therefore, to resolve the issue it is necessary to run broker on a remote server with direct access to global net. Eclipse Foundation has open server with running mosquitto broker on it, which is used for testing purposes [47]. Hence, notification applications are subscribed on eclipse broker and the local broker will bridge data to it, broker is working in two bridges mode.

One solution of notification system is IoT Communicator application for mobile devices by Beckhoff [48]. It is an app supported by iOS and Android operation systems. The application provides possibility to subscribe to a topic structure and receive real-time data, plot graphs, receive warning notifications and send commands back. To make it works: application has to be installed on mobile devise, address of broker, in global net, topic, port number has to be specified in settings tab and in TwinCAT project iotCommunicatorAppPublisher block has to be added. General structure of IoT communicator is shown on Figure 17. Additionally, such features as authentication, encryption, auto-reconnect and keep screen on features can be configured in settings tab to tune the application for a task purposes, detailed information can be found in official documentation [48].

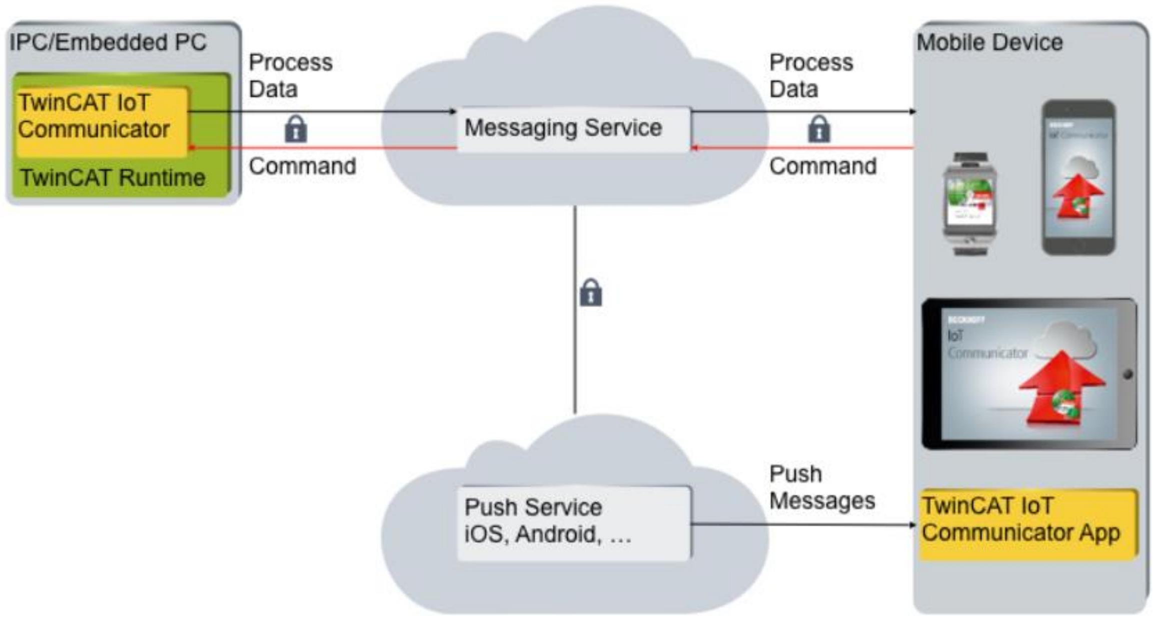


Figure 17. Example of IoT Communicator App application [44].

Another proposed solution is Telegram chat bot, which is account of messenger operated by software. Open-sourced application programming interface (API) allows to create application with similar features to previous one [49]. The developed software act as subscriber to our broker, to particular topic structure, and can post warning messages to the chat. Plotting data feature was developed to allow user to check current state of a system. To make that notification system works Telegram client has to be installed on mobile, laptop or PC devices, chat bot has to be added to dialog list, developed software has to be run on a server. Considering TwinCAT project, one more instance of IoTPublisher has to be added to send data to particular topic, for example lut/AMBsystem/warnings/chatbot.

Therefore, final design of designed system can be shown in details as follows on Figure 18. Detailed code of notification system can be found in Appendix 10 for IoT Communicator application and in Appendix 11 for Telegram chat bot.

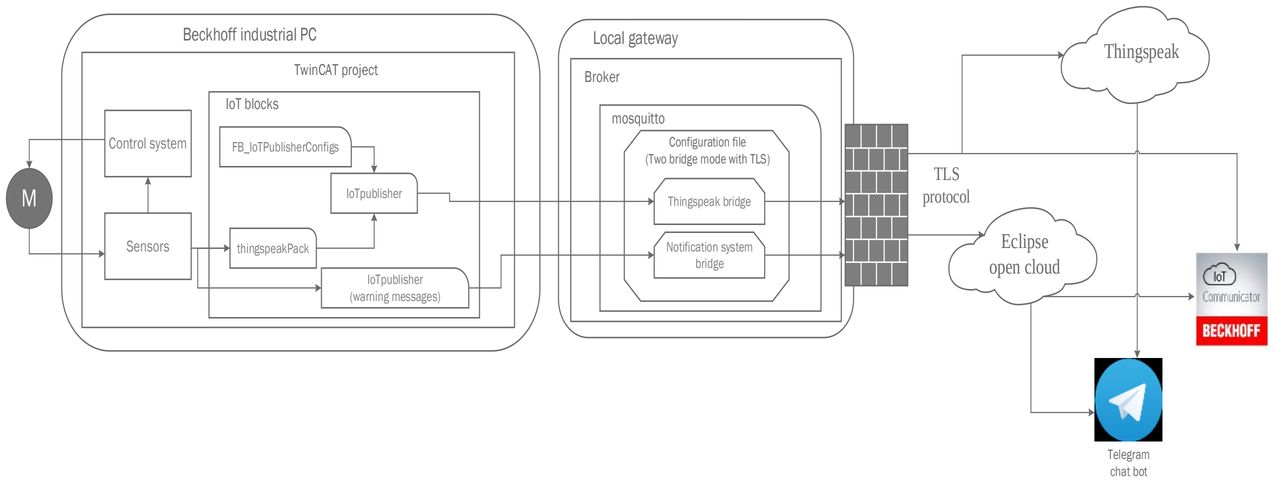


Figure 18. Design of industrial IoT system.

5.6 Testing Prototype Results

Performance of designed system was tested on temperature measurement task. The system was stably collecting information for several weeks. However, a bug was found during debugging process, the system went freezing from time to time. Additionally, in IoT Communicator app status of device was always offline, whereas it was functioning properly. It was found that instance of standard IoT library, which is used in developed IoTpublisher block, was causing that problem. After changing type of variable from input variable to input-output variable the freezing issue was gone as well as status issue in the Beckhoff Communicator application. Some minor fixes was applied during work process, most of them was caused by inaccurate information in documentations such as not clear or sufficient examples on informational Beckhoff portal, confusing typo issues in Thingspeak documentations, wrong methods to send data were described. However, during the research work Thingspeak was informed about issues and they fixed it. Nevertheless, task was solved and following charts are representing results from different interfaces of the system. On Figure 19, Thingspeak interface is presented with plot of temperature measurements for two days.

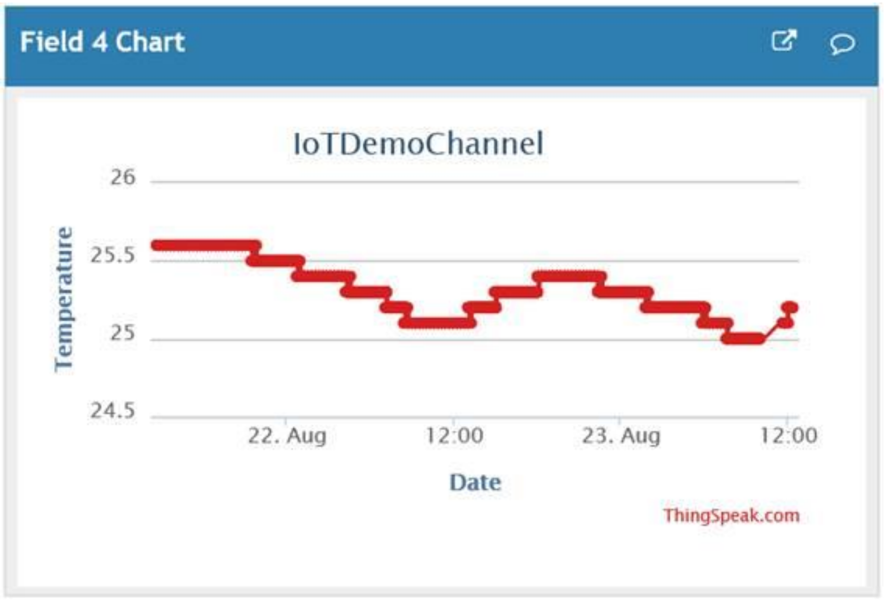


Figure 19. Temperature measurement visualization in Thingspeak web-interface with 15 seconds step

Figure 20 represents interface of Beckhoff Communicator application. There are three windows, on the first one list with devices is given, by clicking on basementLab user will reach second one, where real-time temperature measurements will appear. It is possible to plot a graph from measured points, as it was discussed previously, example on third window.

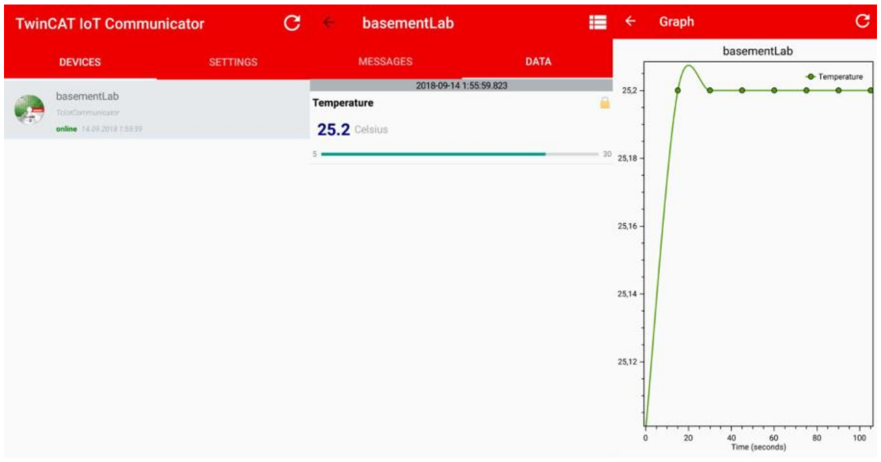
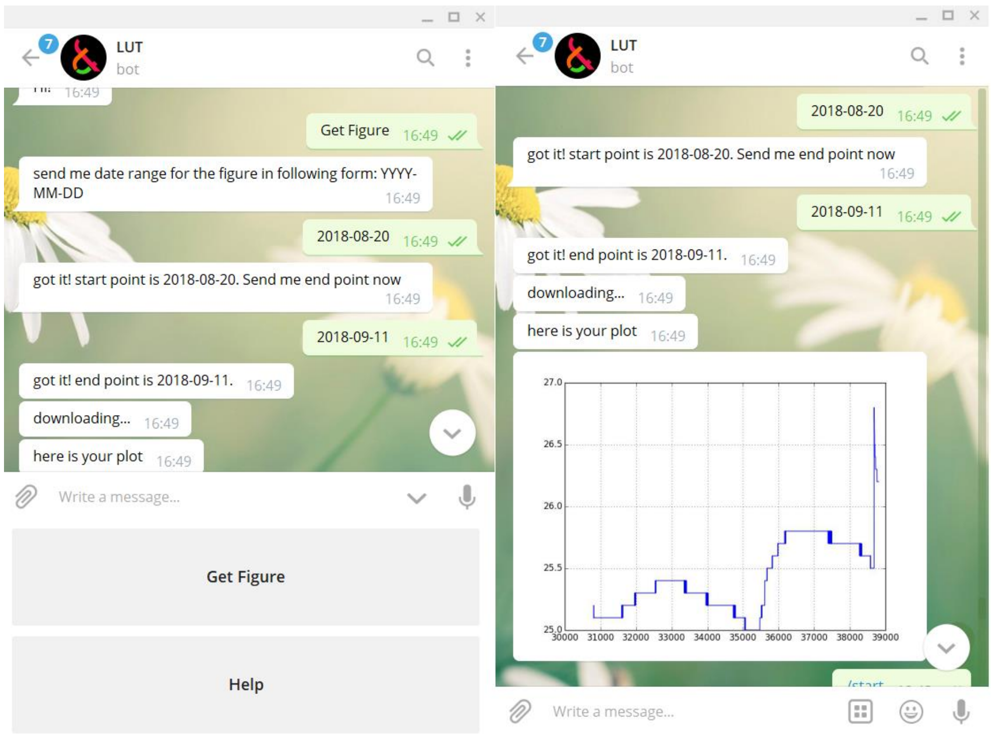


Figure 20. Temperature measurements in IoT Communicator app

On Figure 21 Telegram chat bot notification system is presented. By starting dialog user automatically will receive warning messages. Additionally, get figure feature was developed by clicking the button and providing desired date range, user will receive graph of measurements for the period from the cloud.


LUT
bot

Get Figure 16:49

send me date range for the figure in following form: YYYY-MM-DD 16:49

2018-08-20 16:49

got it! start point is 2018-08-20. Send me end point now 16:49

2018-09-11 16:49

got it! end point is 2018-09-11. 16:49

downloading... 16:49

here is your plot 16:49

Figure 21. Temperature measurements in Telegram chat bot

6. CONCLUSIONS

In conclusion, the main aim of the study was achieved, the secured cloud based IoT CPS were developed. The following goals were solved in order to achieve the aim: data exchange protocols were reviewed and MQTT was chosen to establish communication channel, cloud services reviewed and Thingspeak has been chosen as the most suitable for prototyping purposes, recent studies in security field reviewed and implemented in the system design. User notification system has been proposed. The system is applied and tested on condition monitoring task for AMB system. Number of experiments were done in frame of the research work. Firstly, simple example of wave generator with synchronization data to cloud and notification system. Secondly, long-term experiment of temperature measurements of a coil in AMB system, it lasted for a week. Thus, tests allowed to find bugs, errors of developed system and fix them.

Developers can use examples, created tools and applied technics as a starting guide to creating an IoT solution. Nevertheless, decisions in the research can not be taken as strict rules, choice of tools highly depended on requirements of the task. As well as implementing security features not guarantee hundred percent protection. Periodical maintenance and updates of a system with respect to recent studies in the field can support high security level. Functionality of developed system is scalable and can be rearranged to different tasks. Limitations of the system are generally connected with limitations of third-party products such as cloud service, which can be tuned with respect to needs of application.

REFERENCES

1. Kagermann H., Wahlster W., Helbig J., 2013, Recommendations for implementing the strategic initiative INDUSTRIE 4.0., Germany, Heilmeyer und Sernau
2. Henry Ford, Samuel Crowther, 1992, My life and work, Garden City, N.Y., Doubleday Page & Co.
3. Federal Ministry for Economic Affairs and Energy, 2018, Plattform Industie 4.0 [online document]. [Accessed 3 May 2018]. Available at https://www.plattform-i40.de/I40/Navigation/EN/ThePlatform/PlattformIndustrie40/plattform-industrie-40.html
4. Kevin Ashton, 2009, That 'Internet of Things' Thing, RFID Journal [online document]. [Accessed 4 June 2018]. Available at http://www.rfidjournal.com/articles/view?4986
5. SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS, Next Generation Networks – Frameworks and functional architecture models, 2013, Overview of the Internet Things, Switzerland Geneva
6. AIG company, IoT Case Studies: Companies Leading the Connected Economy, Part 2, 2017
7. Fingrid, 2015, Case study [online document]. [Accessed 3 May 2018]. Available at https://www.ibm.com/case-studies/fingrid
8. Fingrid, 2017, Main grid development plan 2017–2027, Finland, Helsinki
9. Rolls-Royce, 2016, Rolls-Royce and Microsoft collaborate to create new digital capabilities [online document]. [Accessed 4 May 2018]. Available at https://customers.microsoft.com/en-US/story/rollsroycestory
10. Postscapes, 2018, IoT Standarts and Protocols [online document]. [Accessed 23 August 2018]. Available at https://www.postscapes.com/internet-of-things-protocols/#protocols
11. IoTSense, 2017, Top Internet of Things (IoT) Data Protocols [online document]. [Accessed 11 June 2018]. Available at http://www.iotsense.io/blog/top-internet-of-things-iot-data-protocols/

12. Vinicius Moura Pertel, Maicon Saturno, Fernando Deschamps, Eduardo de Freitas RochaLoures, 2017, Analysis of IT Standards and Protocols for Industry 4.0, 24th International Conference on Production Research, July, Poznan, Poland
13. Brian Russel, Drew Van Duren, Siddhi Rane 2016. Practical Internet Of Things Security. Birmingham: Packt Publishjing Ltd.
14. OASIS, What is MQTT?, Who invented MQTT? [online document]. [Accessed 4 May 2018]. Available at http://mqtt.org/faq
15. OASIS, MQTT Version 3.1.1 Plus Errata 01 [online document]. [Accessed 5 May 2018]. Available at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180924
16. International Business Machines Corporation (IBM) Eurotech, MQTT V3.1 Protocol Specification [online document]. [Accessed 5 May 2018]. Available at http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#utf-8
17. AMQP, AMQP is the Internet Protocol for Business Messaging [online document]. [Accessed 10 May 2018]. Available at http://www.amqp.org/about/what
18. CoAP, CoAP RFC 7252 Constrained Application Protocol [online document]. [Accessed 10 May 2018]. Available at http://coap.technology/
19. OPC Foundation, Unified Architecture [online document]. [Accessed 10 May 2018]. Available at https://opcfoundation.org/about/opc-technologies/opc-ua/
20. Paul Duffy, Beyond MQTT: A Cisco View on IoT Protocols [online document]. [Accessed 11 June 2018]. Available at https://blogs.cisco.com/digital/beyond-mqtt-a-cisco-view-on-iot-protocols
21. Jahanzaib Imtiaz, Juergen Jasperneite 2013. Scalability of OPC-UA down to the chip level enables "Internet of Things" Industrial Informatics (INDIN) 11th IEEE International Conference July 2013, p. 500-505
22. The Mathworks, Thingspeak [online document]. [Accessed 16 May 2018]. Available at https://se.mathworks.com/help/thingspeak/index.html
23. AWS, AWS IoT Core Documentation [online document]. [Accessed 16 May 2018]. Available at https://aws.amazon.com/documentation/iot/

24. Microsoft, IoT Hub Documentation, [online document]. [Accessed 16 May 2018]. Available at https://docs.microsoft.com/en-us/azure/iot-hub/
25. IBM Cloud, Internet of Things Platform [online document]. [Accessed 17 May 2018]. Available at https://console.bluemix.net/docs/services/IoT/index.html#gettingstartedtemplate
26. Olawale Oladehin, Brett Frasncis 2017. Core Tenets of IoT [online document]. [Accessed 17 May 2018]. Available at https://d1.awsstatic.com/whitepapers/core-tenets-of-iot1.pdf
27. Gandhali Smant 2017. Create an End-to end IOT Scenario with Azure Services [online document]. [Accessed 18 May 2018]. Available at https://blogs.msdn.microsoft.com/gsamant/2017/02/24/create-an-end-to-end-iot-scenario-with-azure-services/
28. IBM Cloud, Internet of Things Platform 2018. Quotas [online document]. [Accessed 18 May 2018]. Available at https://console.bluemix.net/docs/services/IoT/reference/quotas.html#quotas
29. Microsoft Azure 2018. Reference - IoT Hub quotas and throttling [online document]. [Accessed 18 May 2018]. Available at https://docs.microsoft.com/en-gb/azure/iot-hub/iot-hub-devguide-quotas-throttling
30. Microsoft Azure 2018. Azure subscription and service limits, quotas, and constraints [online document]. [Accessed 18 May 2018]. Available at https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#iot-hub-limits
31. Amazon Web Services 2018. AWS Service Limits [online document]. [Accessed 18 May 2018]. Available at https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_iot
32. Amazon Web Services 2018. AWS IoT Core Pricing [online document]. [Accessed 18 May 2018]. Available at https://aws.amazon.com/iot-core/pricing/
33. Syaiful Andy, Budi Rahardjo, Bagus Hanindhito 2017. Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System. Proc. EECSI 2017, 19-21 September , Yogyakarta, Indonesia

34. OWASP, Top IoT Vulnerabilities Platform [online document]. [Accessed 17 May 2018]. Available at https://www.owasp.org/index.php/Top_IoT_Vulnerabilities
35. Colin Tankard, Digital Pathways, 2015, “The security issues of the Internet of Things”, Computer Fraud & Security
36. Cisco, Cisco 2018 Annual Cybersecurity Report [online document]. [Accessed 18 May 2018]. Available at https://www.cisco.com/c/en/us/products/security/security-reports.html
37. Cisco, “The IoT is only just emerging but the IoT botnets are already here,” p. 39, Cisco 2017 Midyear Cybersecurity Report Report [online document]. [Accessed 18 May 2018]. Available at cisco.com/c/m/en_au/products/security/offers/cybersecurity-reports.html
38. Brian Krebs, October 1 2016, “Source Code for IoT Botnet ‘Mirai’ Released” KrebsOnSecurity blog[online document]. [Accessed 23 May 2018]. Available at krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/
39. OWASP, OWASP Internet of Things Project [online document]. [Accessed 24 May 2018]. Available at https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=Main
40. OWASP, IoT Attack Surface Areas Project [online document]. [Accessed 24 May 2018]. Available at https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Attack_Surface_Areas
41. OWASP, IoT Security Guidance [online document]. [Accessed 24 May 2018]. Available at https://www.owasp.org/index.php/IoT_Security_Guidance
42. R. A. Light 2017. Mosquitto: server and client implementation of the MQTT protocol. *The Journal of Open Source Software*, vol. 2, no. 13
43. Beckhoff Information System, TF6701 TC3 IoT Communication (MQTT) [online document]. [Accessed 1 August]. Available at https://infosys.beckhoff.com/english.php?content=../content/1033/tf3600_tc3_condition_monitoring/27021598926818571.html&id=

44. Eclipse Foundation, Paho [online document]. [Accessed 24 May 2018]. Available at http://wiki.eclipse.org/Paho
45. GitHub, Mosquitto bridge problem due to non blocking socket connect #478 [online document]. [Accessed 1 August]. Available at https://github.com/eclipse/mosquitto/issues/478
46. Telegram, What is Telegram? [online document]. [Accessed 24 May 2018]. Available at https://telegram.org/faq#q-what-is-telegram-what-do-i-do-here
47. Eclipse Foundation, Sandboxes [online document]. [Accessed 24 May 2018]. Available at https://iot.eclipse.org/getting-started/#sandboxes
48. Beckhoff Information System, TF6730 TC3 IoT Communicator Overview [online document]. [Accessed 10 May]. Available at https://infosys.beckhoff.com/index_en.htm
49. Telegram, Telegram Bot Platform [online document]. [Accessed 26 July 2018]. Available at https://telegram.org/blog/bot-revolution

APPENDIX 1. Installation guide of the system

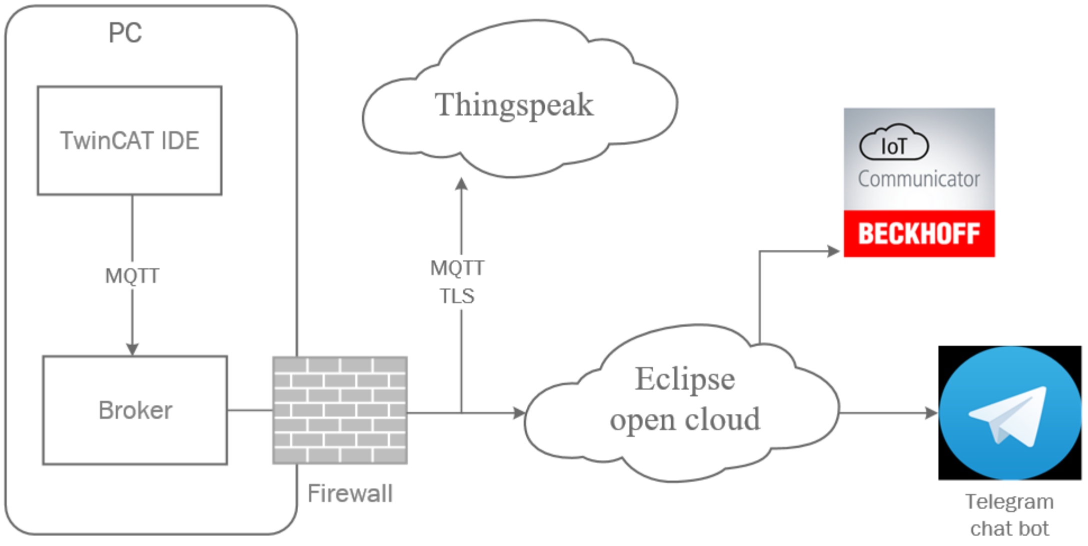
IoT installation guide

Contents

- Overview
- Install requirenments
- Mosquitto (MQTT broker)
- TwinCAT project
  - Quick start example
  - Tools
    - Desciption
    - Configuration
      - FB_IoTPublisherConfigs
      - IoTpublisher
      - iotCommunicatorAppPublisher
      - MQTTtoHTTPSbridge.py
      - thingspeakPack
- Debugging
- Notification systems
  - Telegram bot
  - IoT communicator application
- Troubleshooting
  - mosquitto

Overview

In that guide a step-by-step tutorial is described to provide help with configuring an IoT system on simple example.
The system consist of five nodes as shown on following figure

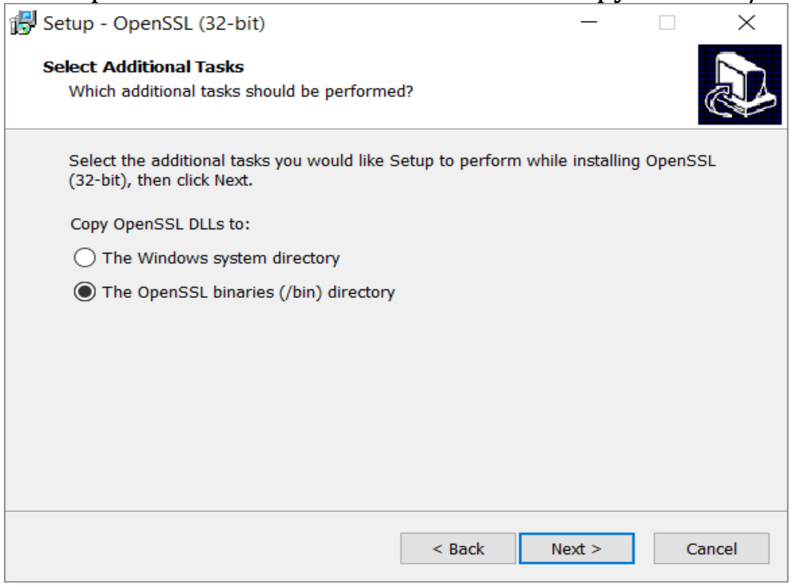


Program in TwinCAT IDE will generate data and transfer it to broker, which is runing on same machine.
The broker will recieve data and bridge it to Thingspeak cloud or to another broker, which can be acessed from global network. For example, on Eclipse open cloud. Option with simultanious brindging will be described too. Two notification system will be proposed as subscribers to global runing broker - IoT Communicator app and Telegram chat bot.
All necessary files and source codes can be found in IoTProjectSrc.zip

Install requirenments

1. Mosquitto (MQTT broker)

- Install OpenSSL (Win32OpenSSL-1_0_2n.exe) to root folder
- On step "Select Additional Tasks" choose copy DLLs to /bin directory



Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing OpenSSL (32-bit), then click Next.

Copy OpenSSL DLLs to:

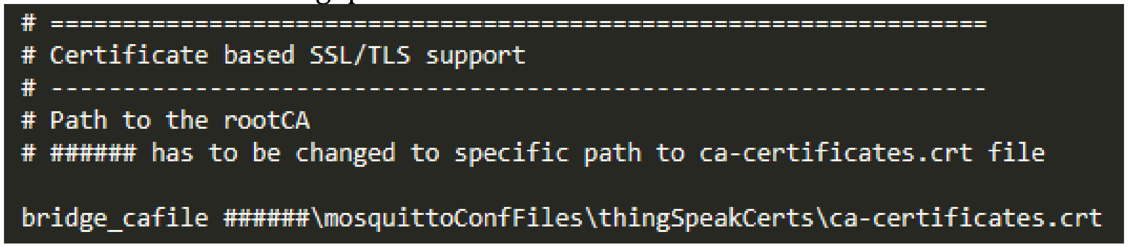
○ The Windows system directory
● The OpenSSL binaries (/bin) directory

- Installing mosquitto (mosquitto-1.4.15a-install-win32.exe)
- Put pthreadVC2.lib in mosquitto folder

2. Install TwinCAT
   `Requirements on all devices at least TwinCAT v.3.1.4022`
3. To get API keys for Thingspeak cloud create an account
4. To get API keys for AWS IoT follow instructions here

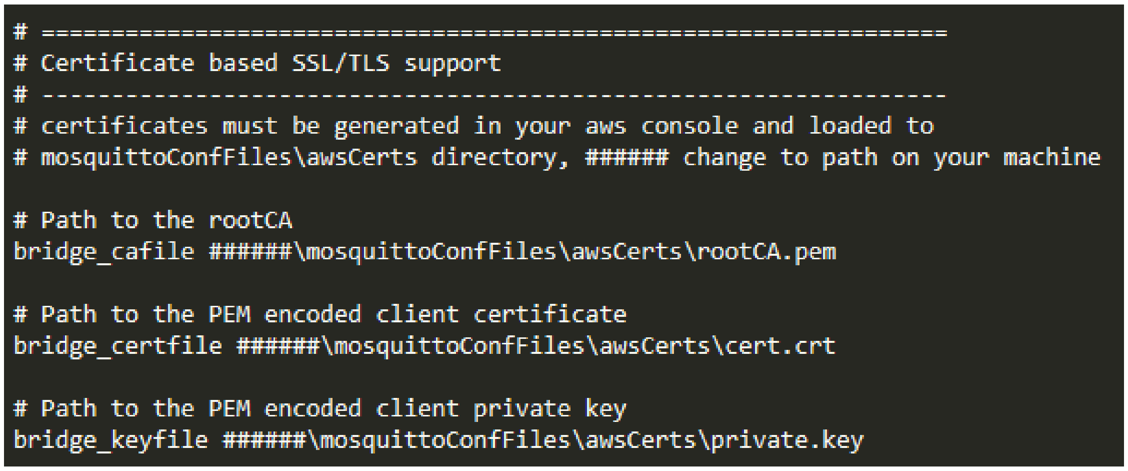
Mosquitto (MQTT broker)

- Execute mosquito.exe from CLI to start broker
  - Open cmd on Windows (press win + r, type cmd and press enter)
  - Navigate to folder where mosquito was installed Exmaple: `cd "Program Files (x86)\mosquitto"`
  - mosquitto.exe (the command will start MQTT broker with default configurations, more detailed information on configuration file could be found here)
  - To start the broker with custom configuration file execute mosquitto with –c attribute as follows: `mosquitto.exe -c C:\somePathToConfFile\confFile.conf` somePathToConfFile has to be changed to real path where configuration file is stored
- To start broker in bridge mode you have to choose one of provided .conf files
  - thingSpeakBridgeTLS.conf
    - bridging all topics which start with remote/thingspeak to ThingSpeak cloud
  - awsBridgeTLS.conf
    - bridging localgateway_to_awsiot topic to AWS IoT cloud
- In this .conf files some tuning has to be made
  - Path to certificates have to be specified, ###### has to be changed accordingly
    - For ThingSpeak:



```
# =================================================================
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# Path to the rootCA
# ###### has to be changed to specific path to ca-certificates.crt file

bridge_cafile ######\mosquittoConfFiles\thingSpeakCerts\ca-certificates.crt
```

- For AWS:



```
# =================================================================
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# certificates must be generated in your aws console and loaded to
# mosquittoConfFiles\awsCerts directory, ###### change to path on your machine

# Path to the rootCA
bridge_cafile ######\mosquittoConfFiles\awsCerts\rootCA.pem

# Path to the PEM encoded client certificate
bridge_certfile ######\mosquittoConfFiles\awsCerts\cert.crt

# Path to the PEM encoded client private key
bridge_keyfile ######\mosquittoConfFiles\awsCerts\private.key
```

- twoBridges.conf All logs will be displayed in CLI window. In case of problems go to troubleshooting section.

TwinCAT project

Quick start example

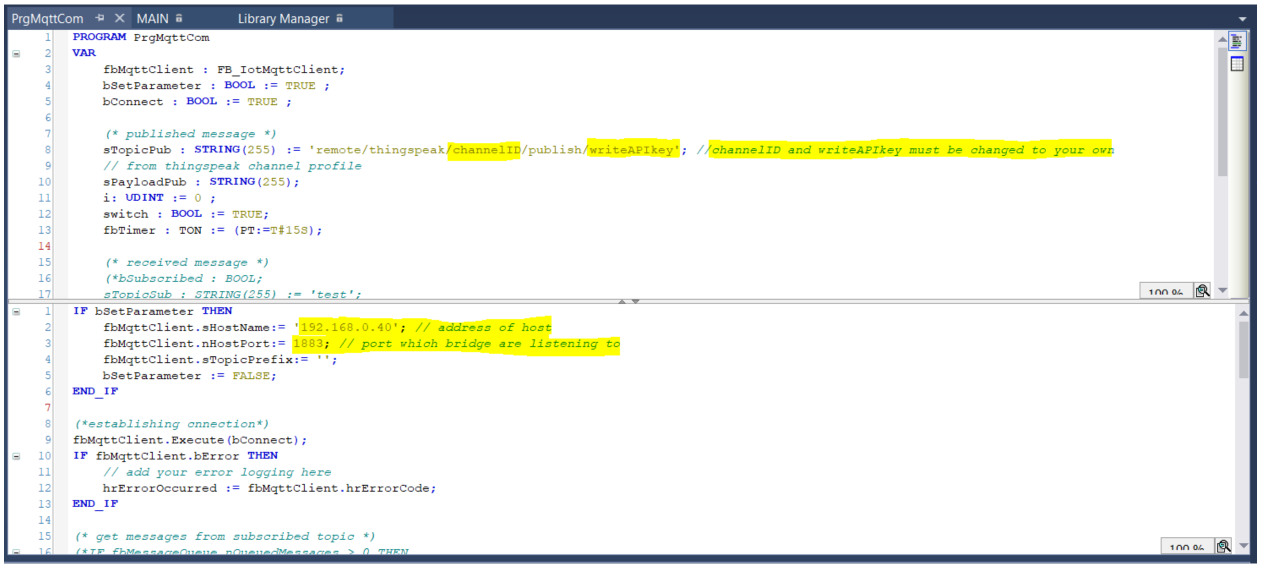
To send data to broker let's run Beckhoff's example, which works out of a box with several changes. That example also can be found in IoTProjectSrc directory. The example send value of counter, which goes up and down, each 15 seconds, to a broker. On Beckhoff webpage it is located under *Samples* tab, the name is *IotMqttSampleUsingQueue*.

If you are runing example downloaded from documantation's page change value of timer to 15 sec:

```
fbTimer : TON := (PT:=T#15S);
```

It is Thingspeak cloud's limitation.

Before running the project mosquitto broker has to be runing in bridge mode with thingSpeakBridgeTLS.conf file and several changes has to be done in PrgMqttCom function file: 1. Api key and channel id have to be changed to your own 2. Address of host (in our case broker's one) have to be changed 3. Topics should to be specified for your needs

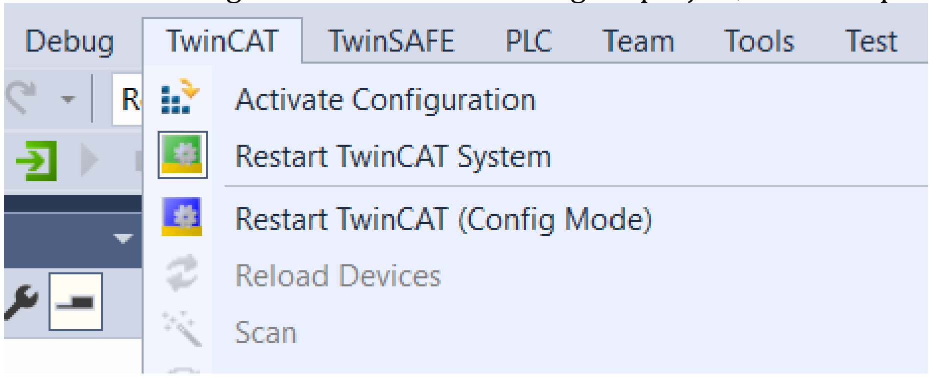


```
PROGRAM PrgMqttCom
VAR
    fbMqttClient : FB_IotMqttClient;
    bSetParameter : BOOL := TRUE ;
    bConnect : BOOL := TRUE ;

    (* published message *)
    sTopicPub : STRING(255) := 'remote/thingspeak/channelID/publish/writeAPIkey'; //channelID and writeAPIkey must be changed to your own
    // from thingspeak channel profile
    sPayloadPub : STRING(255);
    i: UDINT := 0 ;
    switch : BOOL := TRUE;
    fbTimer : TON := (PT:=T#15S);

    (* received message *)
    (*bSubscribed : BOOL;
    sTopicSub : STRING(255) := 'test';
```

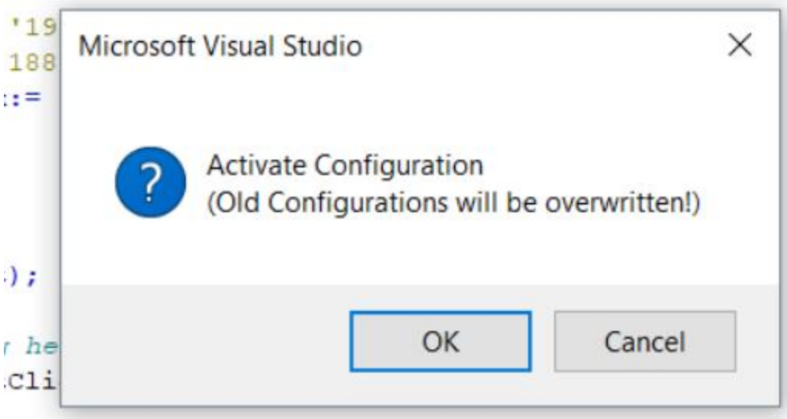
```
IF bSetParameter THEN
    fbMqttClient.sHostName:= '192.168.0.40'; // address of host
    fbMqttClient.nHostPort:= 1883; // port which bridge are listening to
    fbMqttClient.sTopicPrefix:= '';
    bSetParameter := FALSE;
END_IF

(*establishing onnection*)
fbMqttClient.Execute(bConnect);
IF fbMqttClient.bError THEN
    // add your error logging here
    hrErrorOccurred := fbMqttClient.hrErrorCode;
END_IF

(* get messages from subscribed topic *)
(*IF fbMessageQueue.nQueuedMessages > 0 THEN
```

To run project:

1. Activate Configuration in order to configure project, install required licenses

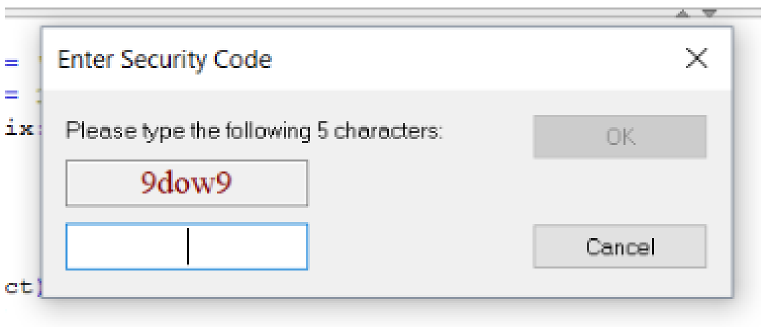


* Press "OK" to proceed configuration



Activate Configuration
(Old Configurations will be overwritten!)

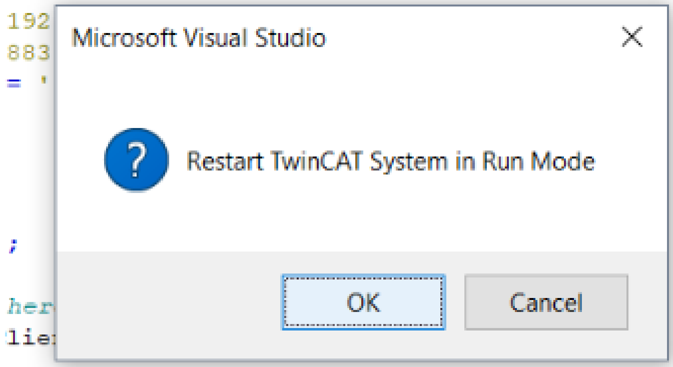
* It could be asked to type security code for updating required licenses


Enter Security Code

Please type the following 5 characters:

9dow9

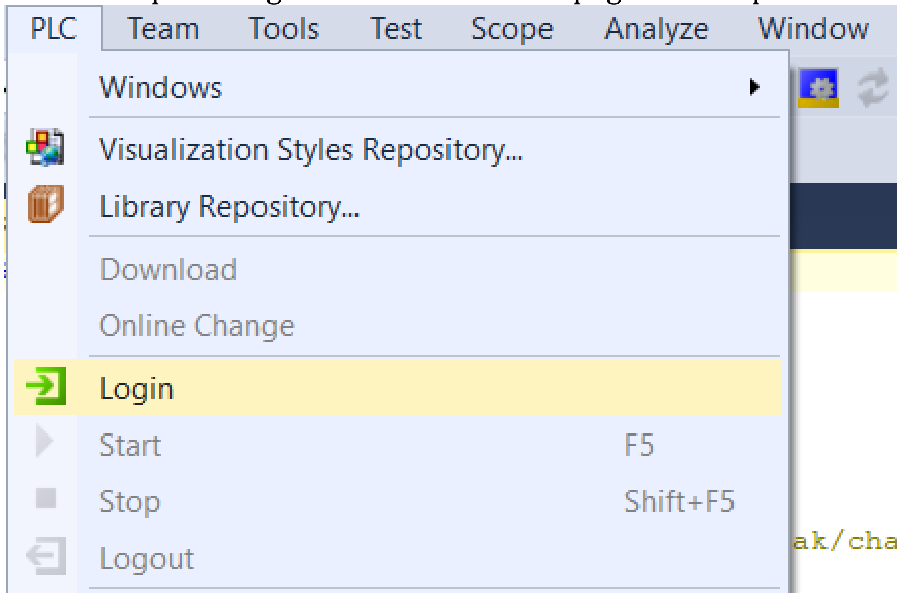
* Press “OK” to restart/start TwinCAT in run mode


Microsoft Visual Studio

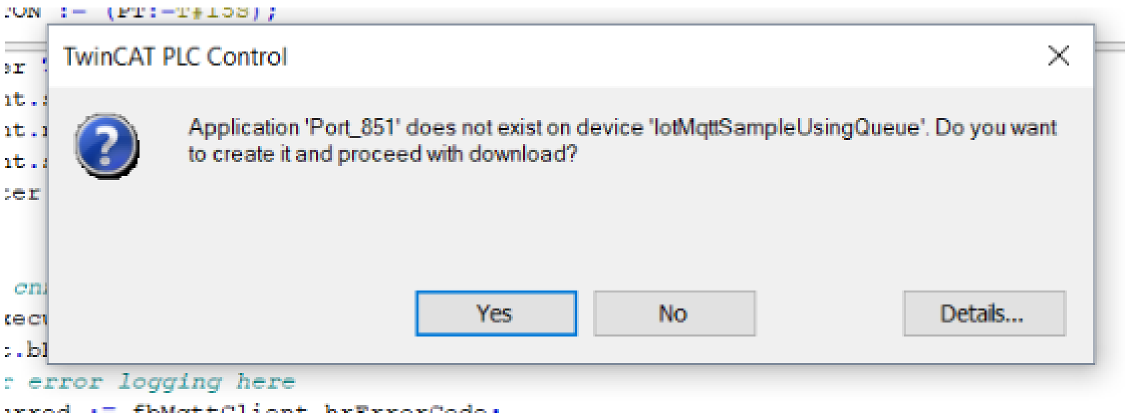
Restart TwinCAT System in Run Mode

2. Login and run compiled project *

In PLC tab press Login button or on front-page toolbar panel

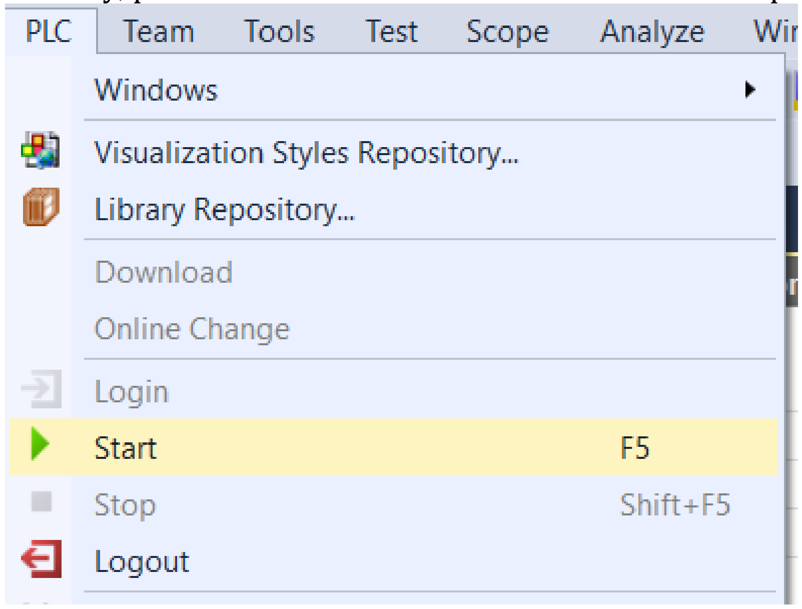


* Press *Yes*


TwinCAT PLC Control

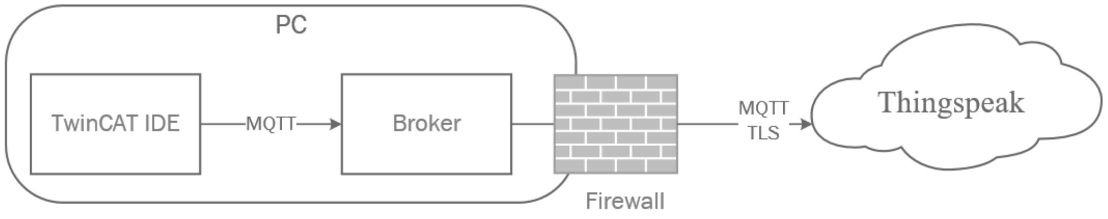
Application 'Port_851' does not exist on device 'IotMqttSampleUsingQueue'. Do you want to create it and proceed with download?

* Finally, press *Start* button in PLC tab or on the toolbar panel or F5



* To stop running program press *Stop* in PLC tab or *Shift + F5* and then *Logout*

Each 15 sec data packages has to appear in broker window and in your channel on Thingspeak cloud. Configured system has following structure:



###

Tools #### Desciption In quick start example standard functional blocks were used to publish data from TwinCAT project.
To make integration to a project easier, custom tools where developed:
1. FB_IoTPublisherConfigs functional block 2. IoTpublisher functional block
3. iotCommunicatorAppPublisher functional block

4. MQTTtoHTTPSbridge.py script
5. thingspeakPack function

Configuration

FB_IoTPublisherConfigs

FB_IoTPublisherConfigs was created to configure the publisher

```
FB_IoTPublisherConfigs := (  hostName := 'brokerAddress',
                             portNumber :=1883,
                             sTopicPub := 'remote/thingspeak/yourChannelNum
ber/publish/yourApiKey');
```

Specifying address of a broker host in hostName parameter, through which port data will be transferred in portNumber and to which topic the data has to be published in sTopicPub parameter, publisher configuration is done. Next step is to pass configurations and desired payload to IoTpublisher functional block. ##### IoTpublisher IoTpublisher it is functional block with two inputs – a payload to publish and configurations of the publisher and status output.
`IoTpublisher(pubConfigs := configuration, sPayloadPub := valueToPublish, sendStatus => status);` >pubConfigs - input for configurations fo publisher >sPayloadPub - desired value in string format >>sPayloadPub can get 2000 characters max >sendSatus - output status of publisher Bool type >>TRUE if message was sent successfully >>FALSE if error occured, inside block code of error can be found in *hrErrorOccurred* variable
>>>detailed error description can be found here
##### iotCommunicatorAppPublisher Functional block was created to publish data to IoT Communicator application by Beckhoff.
`iotCommunicatorAppPublisher(stData := appData, fbIot := appConfig, sendStatus => appPubStatus)` >stData - input data to publish in form of following structure

```
TYPE ST_ProcessData :
STRUCT
    {attribute 'iot.DisplayName' := 'Temperature'}
    {attribute 'iot.ReadOnly' := 'true'}
    {attribute 'iot.Unit' := 'Celsius'}
    {attribute 'iot.MinValue' := '5'}
    {attribute 'iot.MaxValue' := '30'}
    nTemp  : LREAL;
END_STRUCT
END_TYPE
```

detailed information can be found in documentation fbIot - configuration Configuration of this publisher is done with help of standard FB_IotCommunicator block from TC3_iotcommunicator library

```
FB_IotCommunicator := (
    sHostName := 'brokerAddress',     // MQTT Broker Adress
    nPort := 1883,                    // MQTT Port
    sMainTopic := 'topic',          // Main Topic
    sDeviceName := 'deviceName');  // Device Name
```

sendStatus - output status of publisher Bool type >TRUE if message was sent successfully >FALSE if error occured, inside block code of error can be found in *hrErrorOccurred* variable
>>detailed error description can be found here It is similar to IoTpublisher, with difference in specific data form, which is requested by the Communicator application
##### MQTTtoHTTPSbridge.py The script is representing subscriber to a broker. It recieves data and wraps it to HTTPS format and sends it to Thingspeak cloud using *bulkWrite*
All parameters marked with *####* has to be configured with respect to your own values
How to use:

```
python MQTTtoHTTPSbridge.py
```

To send demo values:

```
python MQTTtoHTTPSbridge.py -d
```

To clead channel's feed:

```
python MQTTtoHTTPSbridge.py -c
```

thingspeakPack

thingspeakPack function was created to wrap a payload to CVS format, each message is separated with a pipe character. A message has to contain following update parameters separated with comma: timestamp, field number, latitude, longitude, elevation, status. Hence payload has to look like:

```
TIMESTAMP,FIELD1_VALUE,FIELD2_VALUE,FIELD3_VALUE,FIELD4_VALUE,FIELD5_VALUE,

FIELD6_VALUE,FIELD7_VALUE,FIELD8_VALUE,LATITUDE,LONGITUDE,ELEVATION,STATUS
|
DATETIME_STAMP_OR_SECONDS_FROM_LAST_ENTRY,FIELD1_VALUE,FIELD2_VALUE,FIELD3_VALUE,
FIELD4_VALUE,FIELD5_VALUE,FIELD6_VALUE,FIELD7_VALUE,FIELD8_VALUE,LATITUDE,LONGITUDE,ELEVATION,STATUS
```

thingspeakPack has one input argument, it is payload in *STRING* format.
Function is used when *bulkWrite* option has to be performed. By calling the function cyclically it will add new values to existing, preparing them to be published

Notification systems

Two options proposed to realize data exchange between TwinCAT PLC and smart device via an MQTT message broker:
1. IoT communicator application by Beckhoff
2. Telegram chat bot

IoT communicator application

Description

Detailed information can be found in documentation.
Download IoT communicator application for iOS or Android
#### Configuration In *SETTINGS* tab simple configuration has to be made to start receive data: 1. Broker Address specified
`iot.eclipse.org` 2. Port number set 3. Topic provided
`eclipsecloud/test/temperature`
After go to *DEVICES* tab. Your device has to appear with *online* status

Telegram bot

Description

Telegram chat bot can recieve warning messages and plot data in desired time range
#### Configuration To turn on telegram bot plotGraph.py script has to be executed
`python.exe pathToScript/plotGraph.py` >Python has to be installed
>bot token provided in *botToken* variable >to get token follow instuction here

Debugging

To test settings of cloud side or debug work of broker and program, mosquitto provides two utilities
mosquitto_pub and mosquitto_sub.
By default folder with these utilities is the same as for mosquitto broker.

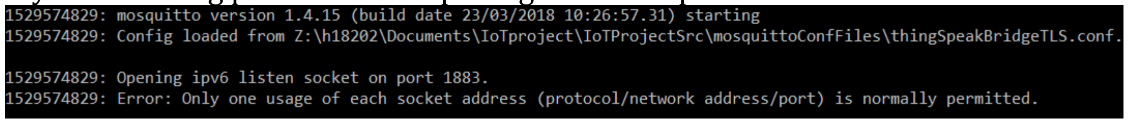
To publish a test message to ThingSpeak cloud:
```
mosquitto_pub.exe -h mqtt.thingspeak.com -p 1883 -t
channels/chatID/publish/writeAPIKey -m "field1=100"
```

To subscribe to our broker and listen to all topics, all data, which is going through:
```
mosquitto_sub.exe -p 1883 -t #
```

Troubleshooting

mosquitto

- If you are facing problem with opening socket on port



```
1529574829: mosquitto version 1.4.15 (build date 23/03/2018 10:26:57.31) starting
1529574829: Config loaded from Z:\h18202\Documents\IoTproject\IoTProjectSrc\mosquittoConfFiles\thingSpeakBridgeTLS.conf.

1529574829: Opening ipv6 listen socket on port 1883.
1529574829: Error: Only one usage of each socket address (protocol/network address/port) is normally permitted.
```

  It means that instance of mosquitto has been already launched and you will have to stop it manually by executing following:
  `sc stop mosquitto`
  Note: Command prompt has to be run as administrator

APPENDIX 2. Simple broker configuration

```
# =============================================================
# Bridges to thingSpeak
# =============================================================
# Establishing connection with the cloud
connection thingspeak
address mqtt.thingspeak.com:1883
# Setting protocol version
bridge_protocol_version mqttv311
notifications false
cleansession true
log_type all
```

APPENDIX 3. Broker bridge configuration

```
# =================================================================
# Bridges to thingSpeak
# =================================================================
# Establishing connection with the cloud
connection thingspeak1
address mqtt.thingspeak.com:8883
# Specifying which topics are bridged
# all topics which start with remote/thingspeak will be
# bridged with prefix channels/######, where ###### your
# write API key of your channel on thingSpeak.com
topic # out 0 remote/thingspeak/ channels/
# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false
notifications false
cleansession true
log_type all
# =================================================================
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# Path to the rootCA
# ###### have to be changed to specific path to ca-certificates.crt file
bridge_cafile ###### \ca-certificates.crt
```

APPENDIX 4. Two bridges configuration

```
# =================================================================
# Bridges to thingSpeak
# =================================================================
# Establishing connection with the cloud
connection thingspeak1
address mqtt.thingspeak.com:8883
# Specifying which topics are bridged
# all topics which start with remote/thingspeak will be
# bridged with prefix channels/######, where ###### your
# write API key of your channel on thingSpeak.com
topic # out 0 remote/thingspeak/ channels/
# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false
notifications false
cleansession true
log_type all
# =================================================================
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# Path to the rootCA
# ###### have to be changed to specific path to ca-certificates.crt file
bridge_cafile ######\ca-certificates.crt
# =================================================================
# Bridges to eclipse
# =================================================================
# Establishing connection with the cloud
connection eclipse-org
address iot.eclipse.org:8883
# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false
notifications false
cleansession true
topic # out 0
log_type all
# =================================================================
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# Path to the rootCA
# ###### have to be changed to specific path to ca-certificates.crt file
#bridge_cafile ######\ca-certificates.crt
```

APPENDIX 5. Main program of TwinCAT project

```
PROGRAM MAIN
VAR
	thingspeakPub : IoTpublisher;
	appPub : iotCommunicatorAppPublisher;
	pack : formPack;
	thingspeakConfig : FB_IoTPublisherConfigs := (
		hostName := '#####’,
		portNumber :=1883,
		sTopicPub := 'remote/thingspeak/#####');
	appConfig : FB_IotCommunicator := (
		sHostName := #####, // MQTT Broker Address
		nPort := 1883,              // MQTT Port
		sMainTopic := 'builds',     // Main Topic
		sDeviceName := 'basementLab');  // Device Name
	thingspeakPubStatus : BOOL;
	appPubStatus : BOOL;
	pubTimer : TON := (PT:=T#15S);
	dataV : ARRAY [1..30] OF LREAL;
	value : LREAL := 27.23;
	pubValue : STRING(500);
	appData : ST_ProcessData;
	i : INT;
	once : BOOL := TRUE;
END_VAR
```

```
pubTimer(IN:=TRUE);

IF pubTimer.Q THEN

        pubTimer(IN:=FALSE);

        pubValue := thingspeakPack(value,1);

        thingspeakPub(pubConfigs := thingspeakConfig, sPayloadPub := pubValue, sendStatus =>

                        thingspeakPubStatus);

        appData.nTemp := value;

        appPub(stData := appData, fbIot := appConfig, sendStatus => appPubStatus);

        GVL.dataToSend := '';

        GVL.delta_t := 1;

END_IF
```

APPENDIX 6. thingspeakPack functional block

```
FUNCTION thingspeakPack : STRING
VAR_INPUT
        data : LREAL;
        fieldNumber : UINT;
END_VAR
VAR
END_VAR
```

```
thingspeakPack := CONCAT(CONCAT(CONCAT('field', TO_STRING(fieldNumber)),'='),
LREAL_TO_STRING(data));
```

APPENDIX 7. IoTpublisher functional block

```
FUNCTION_BLOCK IoTpublisher
VAR_INPUT
        pubConfigs : FB_IoTPublisherConfigs;
        sPayloadPub : STRING (2000):= '';
END_VAR
VAR_OUTPUT
        sendStatus : BOOL := FALSE;
END_VAR
VAR
        fbMqttClient : FB_IotMqttClient;
        bSetParameter : BOOL := TRUE ;
        bConnect : BOOL := TRUE ;
        hrErrorOccurred : HRESULT; // contains the latest occurred error
        statusStr : STRING(255);
END_VAR
(*check if parameters is set*)
IF bSetParameter THEN
        fbMqttClient.sHostName:= pubConfigs.hostName;
        fbMqttClient.nHostPort:= pubConfigs.portNumber;
        fbMqttClient.sTopicPrefix:= '';
        bSetParameter := FALSE;
END_IF
(*establishing cnnection with the host*)
fbMqttClient.Execute(bConnect);
IF fbMqttClient.bError THEN
        hrErrorOccurred := fbMqttClient.hrErrorCode;
```

```
END_IF
IF fbMqttClient.bConnected THEN
            fbMqttClient.Publish(    sTopic:= pubConfigs.sTopicPub,
            pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub)),
            eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
            sPayloadPub := '';
            IF fbMqttClient.bError THEN
                  // add your error logging here
                  hrErrorOccurred := fbMqttClient.hrErrorCode;
            ELSE
                  sendStatus := TRUE;
            END_IF
END_IF
```

APPENDIX 8. FB_IoTPublisherConfigs

```
FUNCTION_BLOCK FB_IoTPublisherConfigs

VAR_INPUT

        hostName  : STRING(255);

        portNumber : UINT;

        sTopicPub : STRING (500);

END_VAR

VAR_OUTPUT

END_VAR

VAR

END_VAR
```

APPENDIX 9. MQTTtoHTTPSbridge

```
import paho.mqtt.client as mqtt
import requests
def sendPacket(payload):
    url = 'https://api.thingspeak.com/channels/###/bulk_update.csv'
    headers = {'content-type': 'application/x-www-form-urlencoded'}
    packet = {'write_api_key':###,'time_format':'relative'}
    packet['updates'] = payload
    r = requests.post(url, headers = headers, data = packet)
    print(r.text)
    print(r.status_code)
def on_message(mqttc, obj, msg):
    print("Message: " + "topic = " + msg.topic + " q = " + str(msg.qos) + " " + str(msg.payload))
     sendPacket(msg.payload)
def on_subscribe(mqttc, obj, mid, granted_qos):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))
def on_log(mqttc, obj, level, string):
    print(string)
    mqttc = mqtt.Client("pythonScriptClient")
    # Poke the methods into the class:
    mqttc.on_message = on_message
    mqttc.on_subscribe = on_subscribe
    # mqttc.on_log = on_log
    mqttc.connect("###", ###, ###)
    mqttc.subscribe("bulkWrite", 0)
    mqttc.loop_forever()
```

APPENDIX 10. iotCommunicatorAppPublisher

```
FUNCTION_BLOCK iotCommunicatorAppPublisher
VAR_INPUT
	stData: ST_ProcessData;		// Values to send
END_VAR
VAR_IN_OUT
	fbIoT : FB_IotCommunicator;
END_VAR
VAR_OUTPUT
	sendStatus : BOOL := FALSE;
END_VAR
VAR
	hrErrorOccurred : HRESULT; // contains the latest occurred error
	timer : TON;
END_VAR
```

```
fbIoT.Execute(TRUE);
timer(IN := NOT timer.Q, PT := T#1S);
IF fbIoT.bConnected AND timer.Q THEN
	fbIoT.SendData(ADR(stData), SIZEOF(stData));
END_IF
IF fbIoT.bError THEN
	// add your error logging here
	hrErrorOccurred := fbIoT.hrErrorCode;
ELSE
	sendStatus := TRUE;
END_IF
```

APPENDIX 11. Telegram chat bot

```
from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters,
RegexHandler, ConversationHandler
import paho.mqtt.client as mqtt
import matplotlib.pyplot as plt
import numpy
import requests
import io
import pandas as pd
import logging
url = "https://api.thingspeak.com/channels/###/fields/#.csv?api_key=###"
chatId = ###
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)
CHOOSING, SET_START_POINT, SET_END_POINT = range(3)
reply_keyboard = [['Get Figure'],
                  ['Help']]
markup = ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True)
start = ''
end = ''
def start(bot, update):
    global reply_keyboard
    update.message.reply_text(
        'Hi!',
        reply_markup=ReplyKeyboardMarkup(reply_keyboard, one_time_keyboard=True))
    return CHOOSING
def getFigure(bot, update):
    update.message.reply_text('send me date range for the figure in following form: YYYY-
MM-DD')
    return SET_START_POINT
def setStart(bot, update):
    global start
    print(start)
    start = update.message.text
    print(start)
    bot.send_message(chat_id=chatId, text=('got it! start point is ' + start +'. Send me end point
now'))
    return SET_END_POINT
def setStop(bot, update):
    global end
    end = update.message.text
    bot.send_message(chat_id=chatId, text=('got it! end point is ' + end + '.'))
    print(end)
    plot(bot, update)
    return CHOOSING
def plot(bot, update):
    global start, end
    # start = '2018-08-20%2000:00:00'
    # end = '2018-08-21%2000:00:00'
    start = str(start) + '%2000:00:00'
    end = str(end) + '%2000:00:00'
    bot.send_message(chat_id=chatId, text=str('downloading...'))
    data = requests.get(url+'&start='+start+'&end='+end)
    df = pd.read_csv(io.StringIO(data.text))
    plt.plot(df['entry_id'], df['field4'], '-')
    plt.grid(True)
    plt.savefig ( "./my_img.png" )
    bot.send_message(chat_id=chatId, text=str('here is your plot'))
    bot.send_photo(chatId, photo=open('my_img.png', 'rb'))
def MQTTsubscribe():
     mqttc = mqtt.Client("pythonScriptClient")
     mqttc.connect("test.mosquitto.org", 1883, 60)
     mqttc.subscribe("plants/Building6/TcIotCommunicator/Messages/#", 0)
     mqttc.loop_start()
     return mqttc
def on_message(mqttc, obj, msg):
    global chatId, dp
    print("Message: " + "topic = " + msg.topic + " q = " + str(msg.qos) + " " + str(msg.payload))
    data = str(msg.payload)
    dp.bot.send_message(chat_id=chatId, text=data[2:(len(data)-1)])
def on_subscribe(mqttc, obj, mid, granted_qos):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))
def helpFunc(bot, update):
   print('help')
   return CHOOSING
def done(bot, update, user_data):
    user_data.clear()
    return ConversationHandler.END
def error(bot, update, error):
    logger.warning('Update "%s" caused error "%s"', update, error)
```

```
updater = Updater("botToken")
dp = updater.dispatcher
def main():
    global dp
    conv_handler = ConversationHandler( entry_points=[CommandHandler('start', start)],
        states={
            CHOOSING: [RegexHandler('^Get Figure$',
                                    getFigure),
                       RegexHandler('^Help$',
                                    helpFunc),
                      ],
            SET_START_POINT: [MessageHandler(Filters.text,
                                           setStart),
                            ],
            SET_END_POINT: [MessageHandler(Filters.text,
                                          setStop),
                           ],
        },
        fallbacks=[RegexHandler('^Done$', done)] )
    dp.add_handler(conv_handler)
    mqttc = MQTTsubscribe()
    mqttc.on_subscribe = on_subscribe
    mqttc.on_message = on_message
    # log all errors
    dp.add_error_handler(error)
    # Start the Bot
    updater.start_polling()
    updater.idle()
if __name__ == '__main__':
    main()
```