

**KARADENİZ TECHNICAL UNIVRSITY  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE**

**COMPUTER ENGINEERING GRADUATE PROGRAM**

**QUALITY OF SERVICE FOR IETF 6TiSH PROTOCOL**



**MASTER THESIS**

**Diliara İBRAHİMOVA**

**MAY 2018  
TRABZON**



**KARADENİZ TECHNICAL UNIVERSITY**  
**THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**



**This thesis is accepted to give the degree of**

**By**  
**The Graduate School of Natural and Applied Sciences at**  
**Karadeniz Technical University**

**The Date of Submission** : / /

**The Date of Examination** : / /

**Supervisor** :

**Trabzon**

KARADENİZ TECHNICAL UNIVERSITY  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING GRADUATE PROGRAM  
Dillara IBRAHIMOVA

QUALITY OF SERVICE FOR IETF 6TISH PROTOCOL

Has been accepted as a thesis of

MASTER OF SCIENCE

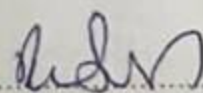
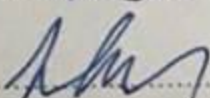
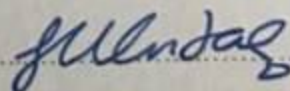
after the Examination by the Jury Assigned by the Administrative Board of  
the Graduate School of Natural and Applied Sciences with the Decision Number 1752 dated  
08 / 05 / 2018

Approved By

Chairman : Prof. Dr. Rifat YAZICI

Member : Assoc. Prof. SEDAT GÖRMÜŞ

Member : Assoc. Prof. GÜZİN ULUTAŞ

  
.....  
  
.....  
  
.....

Prof. Dr. Sadettin KORKMAZ  
Director of Graduate School

## **FOREWORD**

This thesis is written as the completion of the master of Computer engineering, at Karadeniz Technical University. The subject of this thesis is focused on implementing three different qualities of service mechanisms in 6TiSCH.

I would like to express my gratitude to my master thesis Asst. Prof. Sedat GÖRMÜŞ, without his help, support, patience this work would not be completed. I am also grateful for my classmate Hakan AYDIN for his valuable help.

I would also like to thank my family and friends for being there for me in every time that I face challenges during the preparation of my thesis. I could not do it without them and I express my deepest gratitudes to them.

Diliara IBRAHIMOVA

Trabzon 2018

## **THESIS STATEMENT**

I declare that this Master Thesis, I have submitted with the title “Quality of Service for IETF 6TiSCH Protocol” has been completed under the guidance of my Master supervisor Asst. Prof. Sedat GÖRMÜŞ. All data used in this master thesis are obtained by simulation and experimental work done as part of this work in our research labs. All referred information used in the thesis has been indicated in the text and cited in a reference list. I have obeyed all research and rules during my research, and I accept all responsibilities if proven otherwise. 25/05/2018



## TABLE OF CONTENTS

	<u>Page No</u>
FOREWORD.....	III
THESIS STATEMENT.....	IV
TABLE OF CONTENTS .....	V
ÖZET .....	IX
LIST OF FIGURES .....	X
LIST OF TABLES .....	XII
LIST OF ABBREVEATIONS .....	XIII
1. INTRODUCTION .....	1
2. OVERVIEW .....	4
2.1. WSN .....	5
2.2. Application Areas of WSN.....	6
2.3. IEEE 802.15.4 Standard .....	7
2.3.1. The Network Model.....	8
2.3.1.1. Node Types.....	8
2.3.1.2. Network Topologies .....	8
2.3.2. The IEEE 802.15.4 Architecture .....	9
2.3.2.1. IEEE 802.15.4 PHY Layer .....	10
2.3.2.2. IEEE 802.15.4 MAC Layer .....	12
2.3.2.2.1. CSMA/CA .....	13
2.3.2.2.2. Acces Points.....	14
2.3.2.2.3. The Network Establishing .....	15
2.3.2.2.4. Message Transmit.....	17
2.4. New Developments for MAC Layer Design .....	17

2.4.1.	IEEE 802.15.4 TSCH .....	17
2.5.	Data Link Layer .....	19
2.5.1.	6LoWPAN Protocol.....	19
2.5.2.	6top .....	21
2.5.2.1.	Centralized Scheduling.....	22
2.5.2.2.	Distributed Scheduling .....	23
2.5.2.2.1.	Flows .....	24
2.5.2.2.2.	Cell Types .....	25
2.5.2.2.2.1.	A Hard Cell.....	25
2.5.2.2.2.2.	A Soft Cell .....	25
2.6.	Network Layer .....	26
2.6.1.	6TiSCH.....	26
2.6.1.1.	The 6tisch Stack.....	27
2.6.2.	RPL.....	27
2.6.2.1.	Messages in the RPL. ....	28
2.6.2.2.	The Process of Constructing the DODAG Graph for RPL.....	29
2.6.2.3.	RPL Security.....	30
2.7.	Application Layer .....	31
2.7.1.	IETF CoAP .....	31
2.7.2.	CoAP Features .....	31
2.8.	Related Work .....	35
3.	QUALITY OF SERVICE.....	37
4.	IMPLEMENTING QoS ALGORITHMS TO 6TISCH.....	40
4.1.	Development Platform.....	40
4.1.1.	Contiki OS .....	40
4.1.1.1.	The Contiki's Kernel Architecture .....	46
4.1.1.2.	The Power Saving in Contiki.....	47

4.1.1.3.	Services and Libraries in Contiki OS .....	48
4.1.1.4.	Communication in Contiki .....	48
4.1.1.5.	The Loosely Coupled Communication Stack in Contiki.....	49
4.1.1.6.	Protothread.....	49
4.1.2.	Cooja Simulator .....	51
4.2.	System Model .....	53
4.2.1.	Greedy Priority Queuing.....	55
2.1.2.	WFQ.....	57
4.2.2.	Fairness .....	59
4.2.3.	Jain's Index .....	60
5.	EXPEREMIMENTAL RESULTS .....	61
6.	CONCLUSION AND FUTURE WORK .....	65
7.	REFERENCES .....	66
CURRICULUM VIATE		



Master Thesis

SUMMARY

QUALITY OF SERVICE FOR IETF 6TiSCH PROTOCOL

Diliara Ibrahimova

Karadeniz Technical University  
The Graduate School of Natural and Applied Sciences  
Computer Engineering Graduate Program  
Supervisor: Assoc. Prof. Sedat GÖRMÜŞ  
2018, 86 Pages

Internet of Things (IoT) has become a hot research topic recently. There are many research efforts aiming to tackle the challenges posed by IoT ranging from new protocol developments to Fog Computing for reducing the impact of IoT traffic on the Internet backbone. 6TiSCH is a new protocol being standardized by Internet Engineering Task Force (IETF) that aims to address many challenges that are inherited by IoT due to use of low power wireless devices. In this paper, analyze and report preliminary performance results of several well-known Quality of Service (QoS) routines for the 6TiSCH protocol. The QoS mechanisms are realized using the available 6TiSCH protocol stack within Contiki Operating System.

**Key Words:** Wireless Sensor Networks, Internet Of Things, IEEE, IETF, 6LoWPAN, 6TiSCH, QoS, Fairness, WFQ, GPQ.

Yüksek Lisans Tezi

ÖZET

IETF 6TiSCH PROTOKOLÜNDE HİZMET KALİTESİ

Diliara Ibrahimova

Karadeniz Teknik Üniversitesi  
Fen Bilimleri Enstitüsü  
Bilgisayar Mühendisliği Anabilim Dalı  
Danışman: Dr.Öğr.Üyesi Sedat GÖRMÜŞ  
2018,86 Sayfa

Nesnelerin İnterneti günümüzün popüler araştırma konularından biridir. Bu teknolojinin internet omurgası üzerine bindireceği ekstra veri trafiğinin oluşturacağı zorlukları ortadan kaldırmaya yönelik olarak yeni protokol geliştirilmesinden kenar hesaplama (Fog-Edge Computing) kadar birçok alanda araştırmalar yürütülmektedir. Internet Engineering Task Force (IETF) 6TiSCH protokolü, düşük güçlü kablosuz aygıtların Nesnelerin İnternetinde kullanılması ile birlikte ortaya çıkan problemlerin bir çoğunu ortadan kaldırmayı hedefleyen bir protokoldür. Bu tezde, literatürde daha önce uygulanmış üç servis kalitesi (Quality of Service - QoS) mekanizmasını 6TiSCH protokolü için uygulayarak bu mekanizmaların performansı irdelenmiştir. Bu çalışmadaki servis kalitesi mekanizmaları tarafımızdan Contiki işletim sistemi için geliştirilen 6TiSCH protokolü yığına entegre edilerek test edilmiştir.

**Anahtar Kelimeler:** Kablosuz Duyarga Ağlar, Nesnelerin İnterneti, Hizmet kalitesi, IEEE, IETF, 6LoWPAN, 6TiSCH, WFQ, GPQ.

## LIST OF FIGURES

	<u>Page No</u>
Figure 1. The proposed protocol model for WSN .....	4
Figure 2. WSN Architecture .....	5
Figure 3. The seven-layer ISO-OSI and IEEE 802 standards models.....	7
Figure 4. IEEE 802.15.4 Topologies.....	9
Figure 5. LR-WPAN device architecture .....	10
Figure 6. Physical Layer Operating Frequency Bands .....	11
Figure 7. PHY Layer Packet Structure .....	12
Figure 8. CSMA/CA mechanism.....	14
Figure 9. Network establishing.....	16
Figure 10. Messages transmitting.....	17
Figure 11. Diagram of a standard TSCH timeslot and example slotframe. ....	18
Figure 12. IPv6 network with a 6LoWPAN mesh network .....	20
Figure 13. 6top functionalities.....	22
Figure 14. The centralized scheduling.....	23
Figure 15. The distributed scheduling .....	24
Figure 16. IETF 6tisch: Routing over 802.15.4e MAC.....	25
Figure 17. 6TiSCH IPv6-enabled protocol stack for LLNs .....	27
Figure 18. RPL .....	28
Figure 19. Abstract Layering of CoAP.....	32
Figure 20. The CoAP message Header (4 bytes),.....	32
Figure 21. DTLS in Protocol Stack .....	34
Figure 22. Parametrs of QoS .....	38
Figure 23. Contiki OS network stack .....	41
Figure 24. Contiki network stack .....	43
Figure 25. The Contiki OS: the system programs are partitioned into core services and loaded programs.....	44
Figure 26. A structure of a process in Contiki.....	45
Figure 27. Example of process. “Hello world” .....	46
Figure 28. Example of sealed code of protothread.....	50

Figure 29. The interface of Cooja simulator.....	52
Figure 30. Comparison of simulators for wireless sensor networks.....	53
Figure 31. Simulation Network .....	54
Figure 32. Buffer Priorities.....	55
Figure 33. Priorities based on TCP/UDP port .....	56
Figure 34. Scheme of using GPQ QoS mechanism.....	57
Figure 35. WFQ priority queuing. ....	58
Figure 36. Scheme of using WFQ QoS mechanism.....	59



## LIST OF TABLES

	<b><u>Page No</u></b>
Table 1. The comparison of the basic characteristics of IEEE 802.15.4.....	7
Table 2. Characteristics of PHY layer .....	10
Table 3. TSCH primitives.....	16
Table 4 Parameters used in the study. ....	55
Table 5. GPQ average amount packets.....	62
Table 6. FIFO average amount packets .....	62
Table 7. WFQ average amount packets.....	63
Table 8. Jain's index .....	63

## LIST OF ABBREVEATIONS

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
6TiSCH	IPv6 over the TSCH mode of IEEE 802.15.4e
AES	Advanced Encryption Standard
BBR	Backbone router
CoAP	Constrained Application Protocol
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
DAO	Destination Advertisement Object
DIO	DODAG Information Object
DODAG	Destination Oriented Directed Acyclic Graph
EB	Enhanced Beacon
IoT	Internet of Things
IEEE	Institute of Electrical & Electronic Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPHC	IP Header Compression
IPv6	Internet Protocol Version 6
KB	Kilobyte
LLN	Low-Power and Lossy Networks
LowPAN	Low-Power Wireless Personal Area Network
MAC	Media Access Control
OF	Objective Function
PAN	Personal Area Network
PHY	Physical Layer
PKC	Public Key Cryptography
PSK	Pre-Shared Key
RAM	Random Access Memory
ROM	Read Only Memory
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
RFD	Reduced Function Device
Rx	Received

TCP	Transmission Control Protocol
TSCH	Time Slot Channel Hopping
Tx	Transmit
TLS	Transport Layer Security
UDP	User Datagram Protocol
WSN	Wireless Sensor Network



## 1. INTRODUCTION

One of the main components of the modern world is information and computer networks. Power consumption of a network is defined by the network size and the ability of protocols to effectively use the inherent capabilities. Network capabilities are used in many fields of activity, for example, in Wireless Sensor Network (WSN) monitoring and management of objects, collection, transfer and primary processing of data. In this scenario, network nodes can have significant differences in computing, communication and memory resources. One of the main challenges faced by WSN is to integrate different devices with different capabilities and resources to form a functioning network. A decisive role in this is played by network protocols i.e. - the protocol stack. A striking example of this is the TCP/IP protocol stack, which is the basis of the vast majority of modern networks of different levels, scale, and purpose. The largest and most used of them is the Internet, which provides global communications and services. The most important factor in TCP/IP's success is that standard is simple to use and provides a flexible medium for different application scenarios. The Internet already includes several billions of nodes and is on the verge of moving to a new version of the IP namely IPv6 protocol, which provides a more flexible addressing scheme and a decent amount of address space. The widespread introduction of Internet enabled automation systems has shown how effective is IPv6 networking. This type of applications is based on branched networks of sensors, controllable nodes, and mechanisms. Even for a small automated system, the number of IPv6 enabled small objects can exceed several hundreds. In such a scenario, nodes generally are wireless devices connecting to each other using narrow band radios creating a WSN. With the introduction of new application areas, WSNs are increasingly employed to automate various aspects of daily lives. Applications such as Smart Grid, Smart City and Smart Health require low power operations with limited energy resources. Often such applications require operating using batteries. Furthermore, nodes used in WSN applications connect to each other and to the Internet creating a connected Internet of Things (IoT) [1]. IoT devices are expected to meet stringent power budgets with high reliability and need to operate autonomously without user interaction.

The working cycle of any WSN can be categorized into 3 phases [2]. The first phase is the birth which includes initial organization, optimization and configuration. The second



phase is life which includes detection, sensing, data transmission and reporting. And the final phase is death when the failure occurs within the WSN in more than one device lowering the Quality of Service (QoS).

One of the most popular physical layer standards for WSNs is the IEEE 802.15.4 standard [3]. It is considered as the de-facto physical layer standard for the IoT stack. The IEEE 802.15.4 integration with the upper layers (such as 6LowPAN [4], RPL [5] and CoAP [6]) has not been inspected the umpteen issues, and they are not solved yet. The 802.15.4 has been extended with a Time Slotted Channel Hopping (TSCH) Medium Access Control named as 802.15.4-e TSCH [7] to support scenarios such as Industrial Internet that requires high reliability and deterministic delay. While, deterministic delay and reliability are the primary design goals of TSCH MAC, the 802.15.4e standard does not explicitly indicate how QoS mechanisms are to be implemented within TSCH context [8].

The basic principle of IEEE802.15.4e TSCH is that the nodes in the Low-power Lossy Network (LLN) communicate by following a schedule. A Time Division Multiple Access (TDMA) [9] schedule describes each node's behavior in each time slot: within the time-slot the node can be active and transmitting, or the node can be listening for receive, or it can be in inactive mode turning off its radio. The way this schedule is constructed dictates the amount of traffic that the LLN can produce and hence the latency of the network. The schedule also indirectly controls the amount of energy each node consumes. Recently, 6TiSCH is being developed at the IETF to enable IPv6 routing over the TSCH mode of the IEEE802.15.4e standard [10]. 6TiSCH [11] presents an architecture in which low-power wireless devices form a multi-hop LLN. This LLN connects to the traditional Internet through one or more LLN Border Routers(LBRs).

The main aim of 6TiSCH is to evolve a standard approach to manage the TSCH schedule and match it in opposition to the traffic needs in the network [12]. 6TiSCH protocol dynamically assume bandwidth resources to the nodes in the network according to the application requirements [12]. 6TiSCH does not specify explicit algorithms for QoS. Quality-of-Service is a set of requirements to provide better networking services over current technologies. They are used by the network while moving a packet stream from the source to the destination.

In this study, we implement several QoS algorithms for 6TiSCH. The proposed mechanisms aim to create fair queuing for a finite capacity queue with time-stamp index

and service priority where the smart devices performance with varying traffic conditions are analyzed. Based on the analytical model and taking into account such important parameters of performance measures such as mean queue length, packet loss, and service delay, we conduct various simulation experiments using the Cooja emulator of Contiki OS [14].

This dissertation is organized as follows. In Section 2, each layer of WSN protocol stack is briefly discussed and also the section contains a review of new developments for MAC layer. QoS is briefly discussed in Section 3. Section 4 discusses development platform and system model, and implementing QoS algorithms in 6tisch. In Section 5, the conclusion is presented and finally, in Section 6, conclusion and future research are presented.

## 2. OVERVIEW

At the end of the twentieth century, Kristofer Pister, a professor of IT science from the University of CA, USA, formulated the concept of "Smart Dust" [13] - a system of an arbitrary finite set of electromechanical motes capable of exchanging information in an independent spatial configuration. The implementation of this concept in practice led to the emergence of wireless sensor networks (WSN) [14].

The architecture for WSN is created with the following protocol stack model. The protocol model is composed of 5 layers as shown in Figure 1.

The main aim of the power management, mobility management and task management planes is to optimize the network performance by collecting data across different layers.

A step-by-step description of each layer of the WSN protocol stack is briefly discussed in this section. Furthermore, the section includes a review of new developments in WSN research where the main aim is to bring Internet connectivity to small devices for applications such as Smart Grid and Industrial automation.

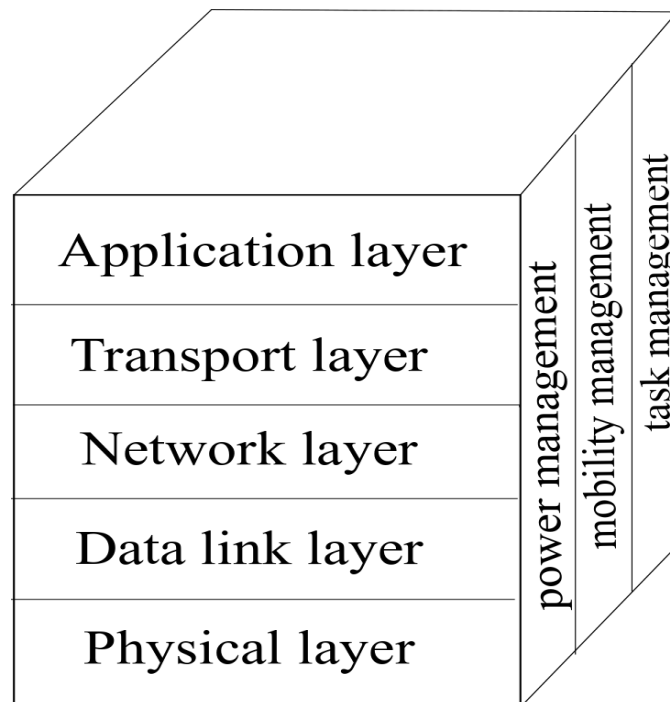


Figure 1. The proposed protocol model for WSN

## 2.1. WSN

Wireless sensor networks are broadly used for automation and control systems in modern computing. Interacting with control devices, the sensors create a distributed, self-organizing network for collecting, processing, and transmitting information. The architecture of WSN is given in Figure 2. The concept of a "self-organizing network" is defined as a system in which devices "know how" to find each other and create a network in the event of a failure of any of the nodes. The technology of sensor networks does not require expensive cables for the construction of the network with auxiliary equipment. Since the sensor network, supports the main interfaces and protocols that are currently in use, it is possible to integrate it into an existing network without a large-scale reconstruction. Small low-power sensors are used in hard-to-reach places covering large deployments areas. Development and introduction of sensory networks in all spheres of life will provide a huge number of benefits to humanity. There are several of unresolved challenges in WSNs, but the deployment and ease of use advantages of WSNs make them attractive to companies requiring sensors in their applications.

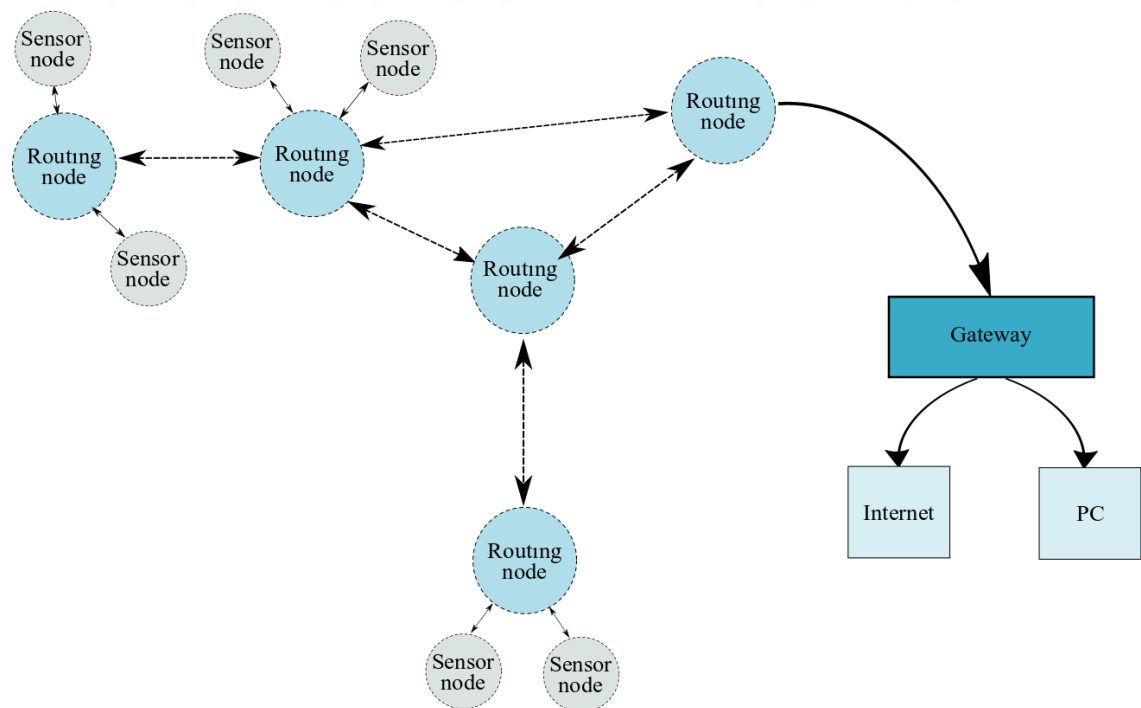


Figure 2. WSN Architecture

There are several standardization efforts in the field of WSNs tackling challenges at each layer of the protocol stack. Institute of Electrical and Electronics Engineers (IEEE), the International Telecommunication Union (ITU), the Internet Engineering Task Force (IETF) and the International Organization for Standardization (ISO) are just a few of such organizations that aim to bring standard solutions to WSNs.

## **2.2. Application Areas of WSN**

One of the first areas of WSN application began in the military sphere for surveillance in a combat situation. Nowadays using wireless sensor networks became popular in many areas of human life, such as industrial monitoring, health, defense, monitoring environment, control of the movement of objects, etc. Such flexible wireless networks can be deployed in challenging environments. The steady downward trend in the cost of the organization WSNs and improving their operational parameters make it possible to position the WSN as highly effective and promising solutions for telemetry data collection systems, remote diagnostics, process monitoring and information exchange. The WSN technology is implemented in different tasks and applications, such as:

- Monitoring and detection of smoke, fires of forests;
- Seismic monitoring and identification of potential tension in tectonic plates;
- Status monitoring and remote control of the perimeter of objects in security systems;
- Environmental monitoring (identification and forecasting of natural disasters);
- Automatic remote control settings with radioactive, gas - storage of oil and other potentially dangerous industrial facilities;
- Monitoring of traffic and transport infrastructure (bridges, crossings, overpasses, etc.);
- Monitoring of buildings and structures;
- Site monitoring, notification, and organization of reliable communication during rescue operations;
- Monitoring of industrial facilities and characteristics technological processes;
- Monitoring of medical and biological parameters of living organisms.

### 2.3. IEEE 802.15.4 Standard

IEEE 802.15.4 [15] is the low-rate and low-duty cycle standard that determines the physical layer and medium access control within the network 7-level OSI model (Figure 3).

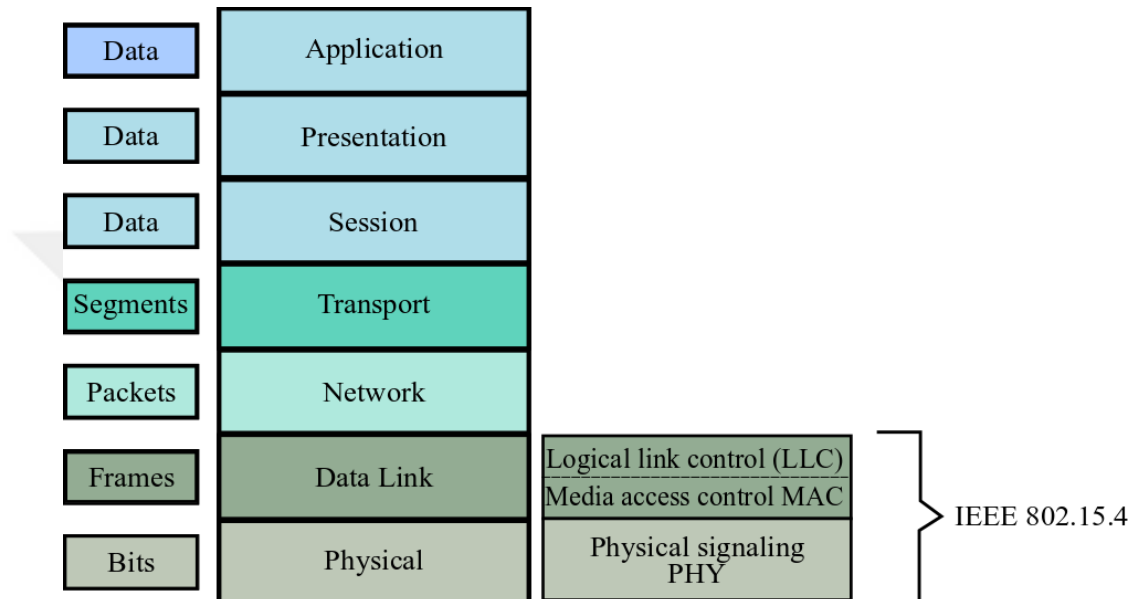


Figure 3. The seven-layer ISO-OSI and IEEE 802 standards models.

The network and application layers are left undefined. The IEEE 802.15.4 standard is aimed at creating networks for managing and monitoring autonomous devices with low power consumption. It can be used to build a wide variety of networks of various topologies with a packet or streaming information, as well as various levels and parameters of security. The standard is supported by the IEEE 802.15 workgroup.

Table 1. The comparison of the basic characteristics of IEEE 802.15.4

	802.11g (Wi-Fi) WLAN	802.15.1 (Bluetooth) WPAN	802.15.4 LR-WPAN
Range	~100 m	~10-100 m	~10 m
Raw Data Rate	11 Mbps	1 Mbps	<=0,25 Mbps

Table 1. (continued)

Power Consumption	Medium	Low	Ultra low
Cost/Complexity	Up	Low	Ultra low

Table 1 shows a comparison of the basic characteristics of low-rate wireless personal networks IEEE 802.15.4 (LR-WPAN), with the WLAN standard and the WPAN standard. As can be seen from the table, IEEE 802.15.4 low-rate networks are designed for applications where using Wi-Fi is too expensive and there is no need for the performance that Bluetooth has.

### **2.3.1. The Network Model**

#### **2.3.1.1. Node Types**

The wireless network is built on the foundations of the IEEE 802.15.4 standard and includes two types of physical devices as shown in Figure 4: The Full Function Devices (FFDs) and devices with light functions (RFD-Reduced-Function Device) [16]. A FFD acts as a Personal Area Network (PAN) coordinator [17] or a coordinator, or just as end node (device). An RFD is a physical device, which isn't able of serving as a PAN coordinator or a coordinator. An RFD is supposed to work for applications that perform simple tasks, for instance, a passive infrared sensor or a light switch. An RFD does not have the necessity to transmit a large number of data packets and only associates with a one FFD at a time. For implementing RFD minimal resources and memory capacity is needed.

#### **2.3.1.2. Network Topologies**

The IEEE 802.15.4 standard provides guidance on possible network types, some of them are the star network topology and the peer-to-peer network topology. Figure 4 shows a network diagram with both of them. Any network, regardless of topology, should have one coordinator. Each device in the network uses a unique 64-bit identifier, which is

determined by the network coordinator. Also, in some cases, a 16-bit identifier within a restricted network can be used.

Networks with a star topology are great for covering small areas where the communication is set between devices and a single central controller, the referred the PAN coordinator. In such a network, all devices interact only with the PAN coordinator. All devices in a network have unique addresses, called as extended addresses. During the association process, a short address also can be assigned to devices. Depending on the target network a device will use the extended address inside PAN use the short address. RFD devices do not need the sending of the extended and short address fields. The PAN is often mains powered, while the remaining devices are generally battery powered.

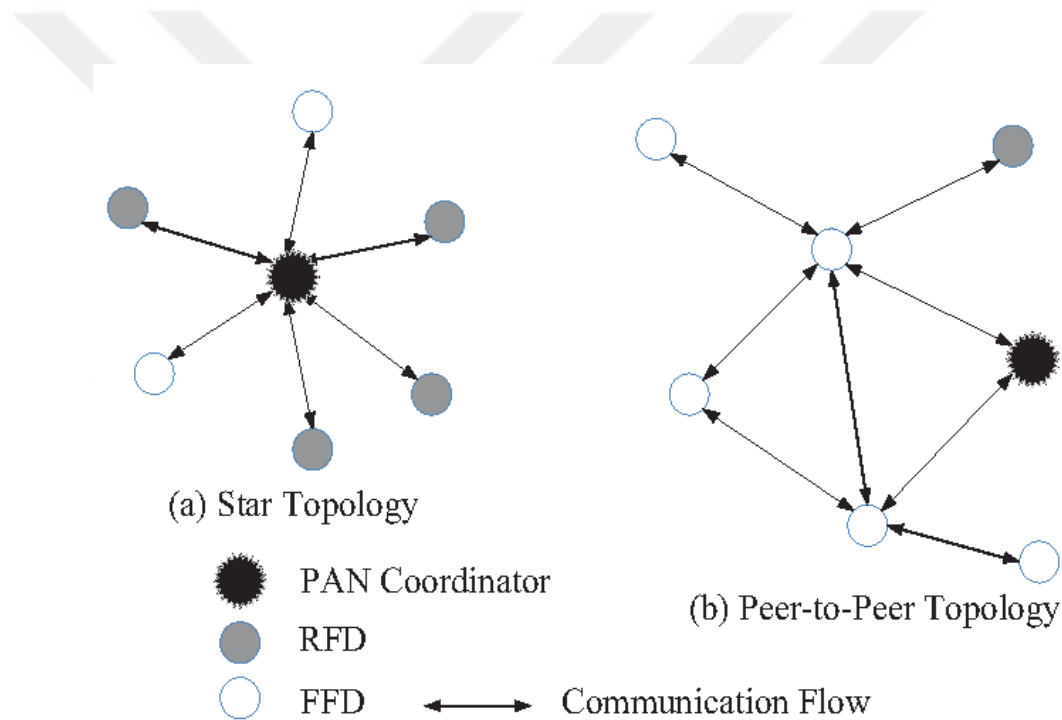


Figure 4. IEEE 802.15.4 Topologies.

### 2.3.2. The IEEE 802.15.4 Architecture

The IEEE 802.15.4 architecture is designed with a number of layers in order to simplify the standard. In Figure 5 these layers are shown as blocks in a graphical representation. Every layer is in control of the strict set of standard's functions and



proposes services to the higher layers. Use of interfaces between layers is necessary to determine the logical links which are represented in IEEE 802.15.4 standard.

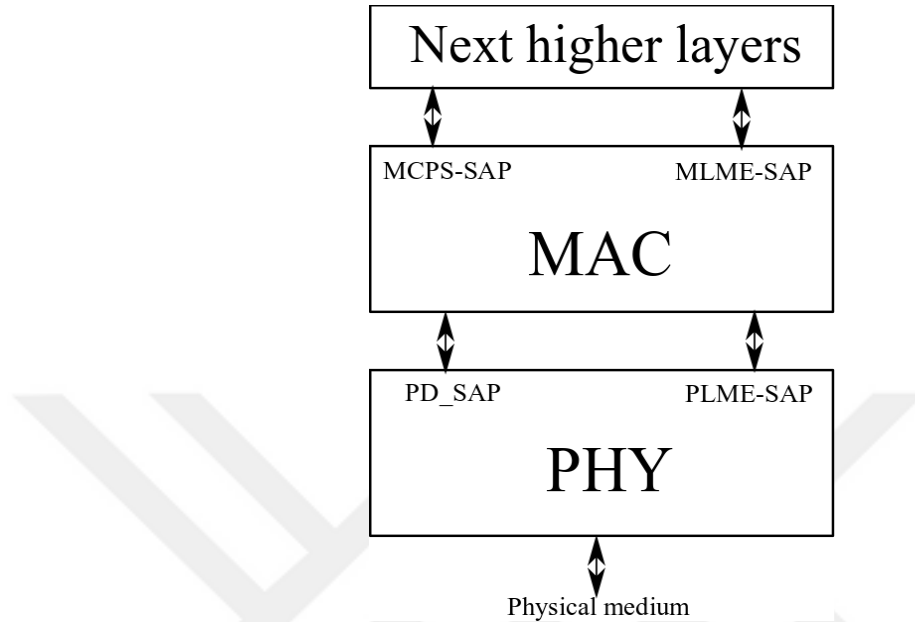


Figure 5. LR-WPAN device architecture

### 2.3.2.1. IEEE 802.15.4 PHY Layer

PHY layer includes specifications which control the radio channel and packet data flow. Table 2 shows basic characteristics of PHY layer. The Carrier Sense Multiple Access (CSMA) [20] with Collision Avoidance (CA) is used by the PHY layer for having access to the radio channel.

Table 2. Characteristics of PHY layer

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip (kchip/s)	Modulation	Bite rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868-868.6	300	BPSK	20	20	Binary
	902-928	600	BPSK	40	40	Binary

Table 2. (continued)

2450	2400- 2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal
------	-----------------	------	--------	-----	------	----------------------

There are some features of the IEEE 802.15.4 PHY layer:

- Activation and Deactivation of radio transceiver,
- Energy Detection (ED),
- Link Quality Indication (LQI),
- Channel Selection,
- Clear Channel Assessment (CCA),
- Transmission or Reception of packets over physical medium [18].

The IEEE 802.15.4 standard identifies three different frequency bands which consist 27 channels are spread across them, as shown in Figure 6 [19]. The 868.3MHz frequency band with 300kHz bandwidth and 0,6MHz channel space is used in Europe; the 902-928MHz frequency band 600kHz bandwidth and with 2MHz channel space is used in America; the 2,4-2,4835GHz frequency band with 2000kHz bandwidth and 5MHz channel space is used in worldwide [20].

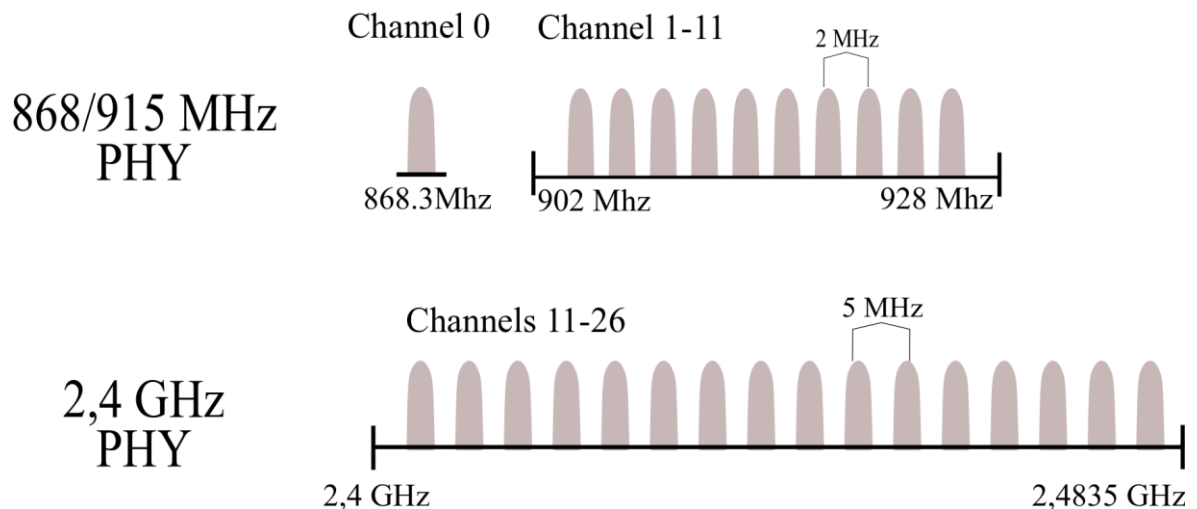


Figure 6. Physical Layer Operating Frequency Bands

Four different frames (Data, Acknowledgement, Beacon and MAC Command) are defined by PHY layer. Each of them has a unique function.

PHY Packet Fields is shown in Figure 7 and contains

- Synchronization header. (SHR) is a preamble sequence (4 octets) – The function of this part is synchronizing. The receiver is synchronizing the incoming signal and a start of the frame delimiter, which signals the end of the preamble
- Start of Packet Delimiter (1 octet)
- PHY – Header (1 octet) – It carries the frame length byte, that indicates the PSDU length
- PSDU (0 to 1016 bits) – Data field

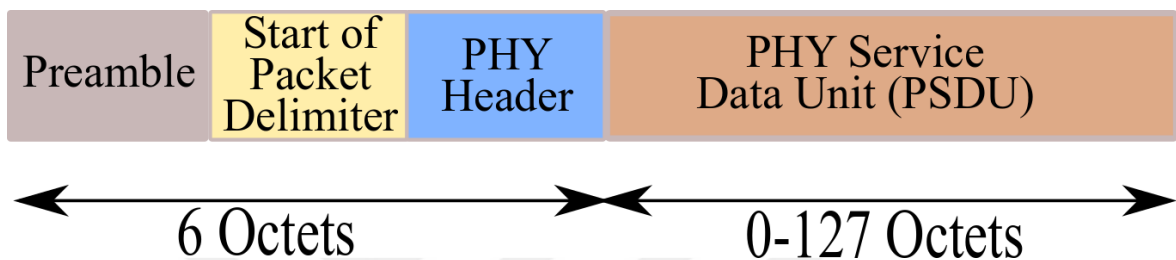


Figure 7. PHY Layer Packet Structure

### 2.3.2.2. IEEE 802.15.4 MAC Layer

The data service and the management service constitute the Medium Access Control sublayer. The MAC sublayer provides a beacon management and channel access. Providing hooks for implementing application-appropriate security mechanisms, GTS management, frame validation, association/disassociation, acknowledged frame delivery, are also features of the MAC sublayer. The CSMA-CA algorithm is used in the IEEE 802.15.4 standard. The CSMA-CA algorithm requires checking the channel before starting transmission in order to avoid collisions with the current transmission from another network device.

#### Features of IEEE 802.15.4 MAC Layer

- The MAC Layer is a simple and flexible protocol
- Cost is extremely low
- The implementation of MAC layer is ease
- Short range operation and reliable data transfer
- The MAC layer has very low power consumption

### 2.3.2.2.1. CSMA/CA

The probability of hidden terminal collision becomes higher when the transmitted packet size is large. This problem is addressed by a mechanism called as Collision Avoidance (CA). Carrier Sense Multiple Access With Collision Avoidance (CSMA/CA) [21] is a MAC protocol in which:

- The virtual carrier sense mechanics is used
- A station that is about to start transmitting sends a jam signal
- After a long wait for all stations that can send a jam signal, the station starts transmitting the frame
- If during transmission the station detects a jam signal from another station, it stops the transmission for a random period of time and then retries

The slotted CSMA/CA algorithm uses a Back-off Period (BP). BP is basic time unit and it is equal to 80 bits that is 0,32ms. The operations which can occur are channel access, back-off count and CCA. Each of them can occur at the boundary of a BP (that must be aligned with the super frame time slot boundaries) [22].

The difference between CSMA and CSMA/CA is the transmission of two reservation packets namely RTS and CTS. A node, ready to send a frame, listens to the communication medium. In the absence of a carrier, it sends a short RTS request signal and a certain time waits for a CTS from the destination. In the absence of an answer (the possibility of collision is implied), the transfer attempt is postponed, when a response is received, a frame is sent to the medium. When a broadcast request is sent (RTS contains address 255), the CTS is not expected. The method does not completely avoid collisions, but it considerably reduces the probability of collisions. The method is characterized by the simplicity and low cost of access chains. The CSMA/CA mechanism is illustrated in Figure 8 [23]. Generally, the operation of exchanging four messages in CSMA/CA is named as four-way handshaking.

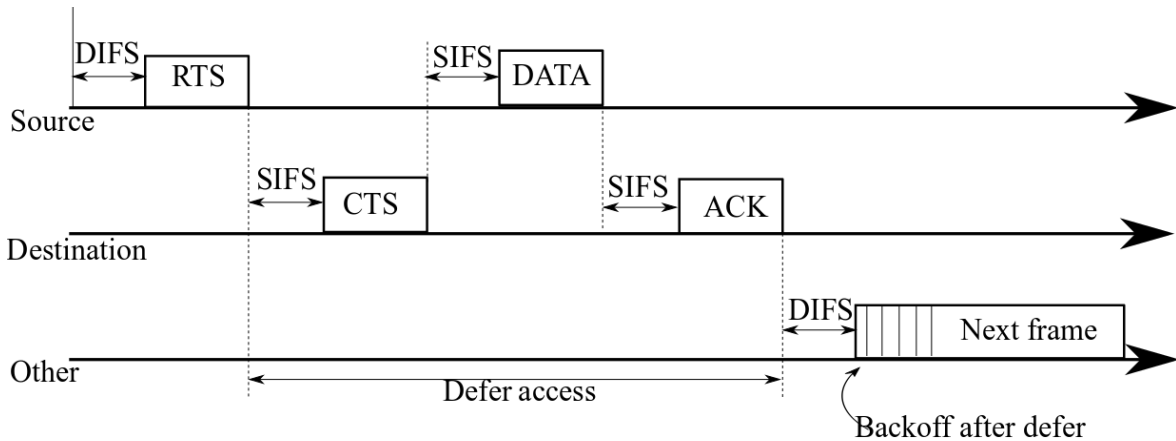


Figure 8. CSMA/CA mechanism

Avoidance of collisions is used to make the performance of CSMA higher by giving the network to a single transmitter. This function is assigned to the jam signal in CSMA/CA [22]. Improving the efficiency is achieved by reducing the likelihood of collisions and repeated attempts to transfer. But waiting for the jam signal creates additional delays, so with other methods more easy to achieve better results. Avoidance of collisions is useful in practice in situations where timely detection of a collision is impossible - for example, when using radio transmitters.

In the best case, when all stations in the system can hear each other, CSMA-CA can guarantee a 36% of channel utilization.

#### 2.3.2.2.2. Acces Points

Communication between network protocols is exchanged via Service Access Point (SAP). The MAC layer provides two services for communication with the upper layers:

- MLME - MAC Layer Management Entity
- MCPS - MAC common part sublayer

There are 4 types of messages:

- request - request from the top layer to the MAC layer;
- confirm - the MAC layer response to the top layer request;
- indication - the message from the MAC layer to the top layer, which indicates the internal event of the layer;

➤ response - is sent from the top layer to the MAC layer to complete the procedure previously caused by the primitive indication.

### **2.3.2.2.3. The Network Establishing**

Through the MAC layer functions, each physical device has an ability to find a network and connect to it. When a device is on, the transceiver receives a command from an upper layer entity to start a scan. In a case when a device cannot find the network and if device based upon FFD, it may try to create its own network. If the network is found, the node will attempt to connect to this network. After the device has received the permission from the network to associate, the message is sent through the MLME-SAP to the upper layer, and in this part the device's 64-bit IEEE address (each network device is given a unique serial number at the time of manufacture, which is then used for unambiguous detection on the network among other networks in the world) changing to a right short address according to the PAN Coordinator's requirements. When a device wants to separate itself from the network, it will get a Disassociate command from the PAN Coordinator.

In order to establish the network, a PAN coordinator is needed to be selected. Each network must have one coordinator. Figure 9 shows the network establishing, which begins when the top layer sends the MLME-SCAN.request primitive to the MLME-MAC Management Service or MAC Layer Management Entity, requesting an active channel scan. For example in Table 3 TSCH primitives is shown [24].

After the channel scan is completed, the results are sent back through the MLME-SCAN.confirm primitive. If the results are acceptable, then the top layer selects the identifier of the personal network and sends the MLME-START.request primitive to the MAC layer control object.

The MLME-START.request primitive requires the MAC layer to put the identifier of the personal network in the PIB. After this, the MAC layer sends the MLME-START.confirm primitive to the top layer. This device becomes the coordinator of the personal network and the network is formed.

Table 3. TSCH primitives

Name	Request	Indication	Response	Confirm
MLME-SET-SLOTFRAME	8.2.19.1	-	-	8.2.19.2
MLME-SET-LINK	8.2.19.3	-	-	8.2.19.4
MLME-TSCH-MODE	8.2.19.5	-	-	8.2.19.6
MLME-KEEP-ALIVE	8.2.19.7	-	-	8.2.19.8

Attributes of the network are stored in the PIB-PAN information base. PIB, an MLME-GET request is needed to be sent to the MAC layer with its index. To change a meaningful MLME-SET request.

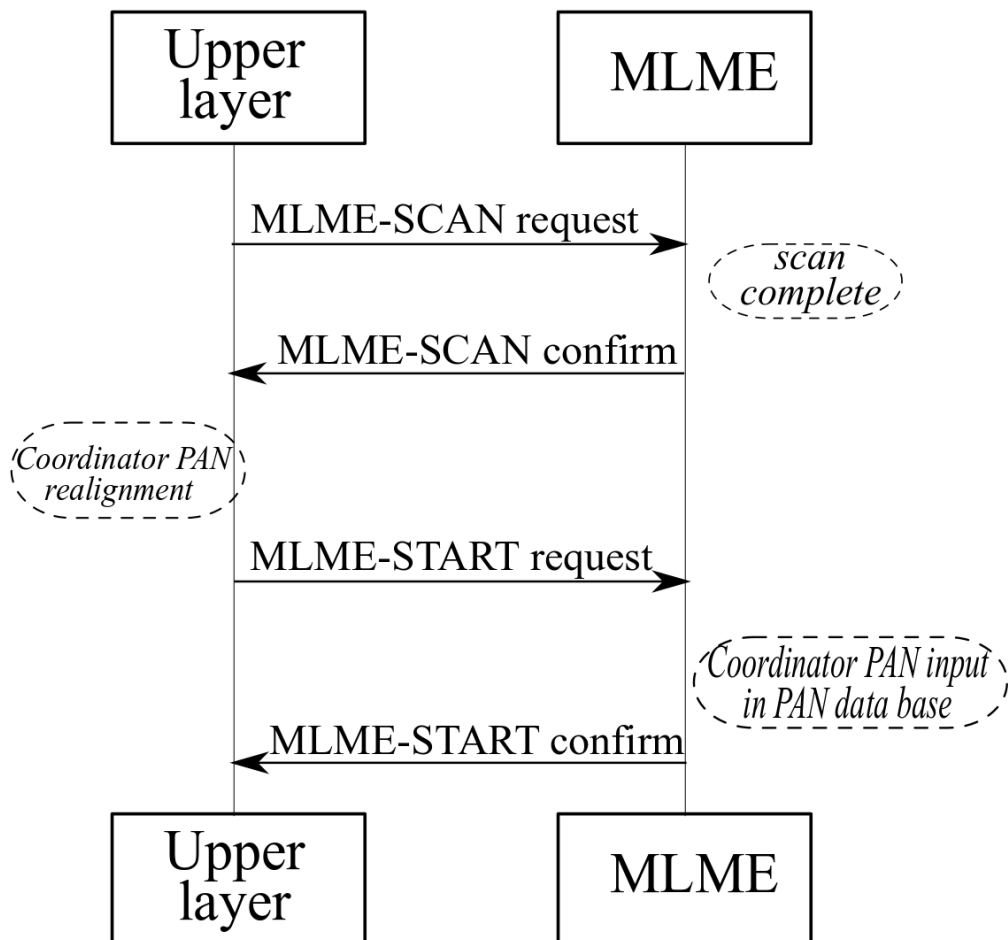


Figure 9. Network establishing

#### 2.3.2.2.4. Message Transmit

To send a message to another network device, the sender application sends the MCPS-DATA.request request to the MAC layer. The MCPS-DATA.request request contains a message, the recipient's address, the length of messages and other parameters. The MAC layer of the sender transmits the message over the radio channel to the recipient MAC layer, which in response sends a delivery confirmation (Acknowledgement-ACK), if it was requested. The MAC layer of the receiver sends a message to the top layer of MCPS-DATA.indication, reporting the received message. Figure 10 shows the process of messages transmitting from one network device to another.

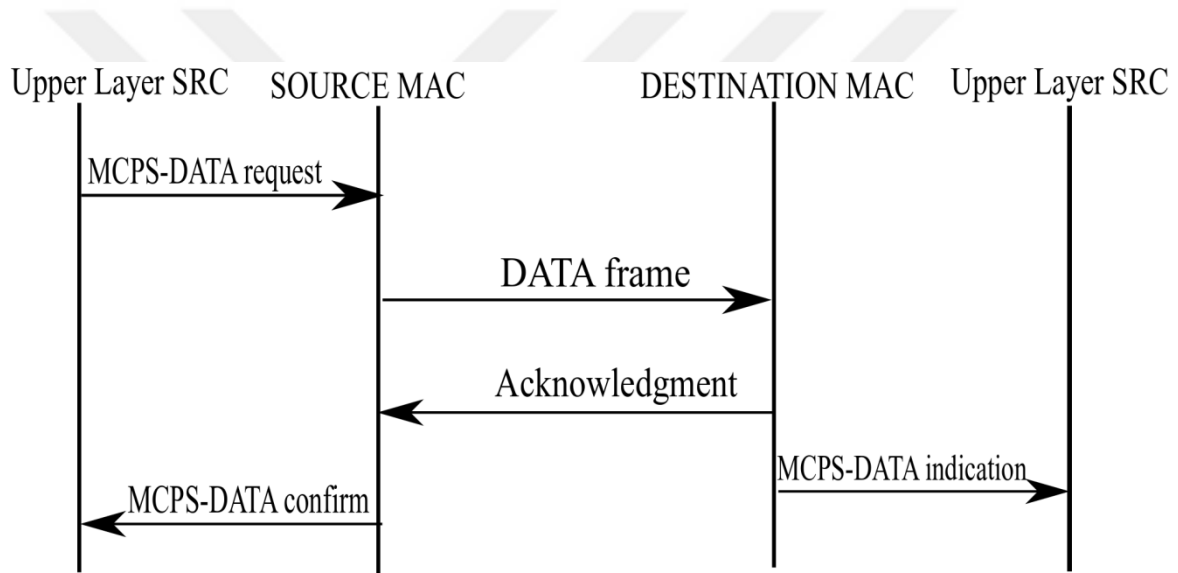


Figure 10. Messages transmitting

## 2.4. New Developments for MAC Layer Design

### 2.4.1. IEEE 802.15.4 TSCH

In 2012 the TSCH (Time-Slotted Channel Hopping) [25] was introduced. It was designed as an amendment to the MAC layer of IEEE 802.15.4 standard. TSCH is designed to increase the efficiency of motes synchronization. TSCH can be divided into Time Slot and Channel Hopping parts. Time Slot is a part where time is slicing up into slots (as shown in Fig 11).



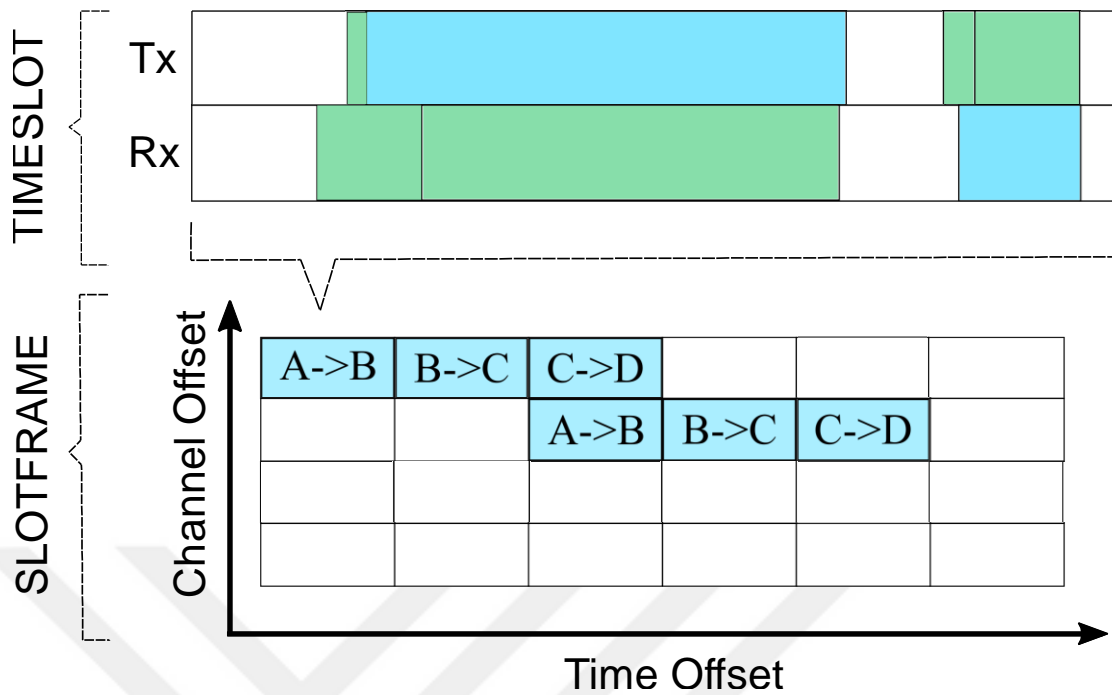


Figure 11. Diagram of a standard TSCH timeslot and example slotframe [26].

A slot is a time period is intended for transmitting. The time slot's duration is not determined by the standard. For example, sending a maximum-length frame (for IEEE 802.15.4) takes about 4 ms, but sending ACK takes about 1 Ms. When time is sliced into time slots the next step is coming, where time slots grouped into slot frames. Slot frames continuously repeat time slots over time. TSCH creates a schedule, using this schedule each node knows when it's time to transmit, time to receive and time to sleep. For each scheduled cell, the schedule specifies a slotOffset and a channelOffset. The channelOffset is using for Channel Hopping. Nodes use Eq. 1 for selecting a frequency.

$$\text{frequency} = F \{(\text{ASN} + \text{channelOffset}) \bmod \text{nFreq}\} \quad (1)$$

where:

- F is the function which consists of a lookup table containing the set of available channels;
- nFreq is the number of available frequencies;
- ASN is Absolute Slot Number

The channelOffset for both is the same and it is written in their schedule, the ASN counter is also the same, therefore they compute the same frequency. Consequently, even with a static schedule pair of nodes hop between channels. This is called channel hopping.

TSCH creates a synchronized mesh network. Each node joins the network after it hears a beacon from another node. TSCH network needs to have strict synchronization. Generally, to keep track of time, each node is equipped with a real-time clock. However, time in a node can drift in relation to the other neighboring nodes. Therefore periodical re-synchronization with neighbor node's clock is needed. Each node sends its own network time to its neighbors. The synchronization is updated when a node gets a data or ACK frame from its time source. In IEEE802.15.4 standard two mechanisms for synchronization in TSCH network are defined [27]. These are:

- AB - Acknowledgment-Based synchronization. This method assumes transmitting information about the difference in time between the supposed time of frame arrival and its factual arrival time by the receiver to the sender node in acknowledgment. The sender node is a node which synchronizes with the clock of the receiver.

- FB - Frame-Based synchronization. The principle of this method is exactly the opposite. The computed delta is used by the receiver node for mounting its own clock. So, the receiver synchronizes with the senders' clock.

## **2.5. Data Link Layer**

### **2.5.1. 6LoWPAN Protocol**

The Internet Engineering Task Force (IETF) released the 6LoWPAN standard (IPv6 over Low-Power Wireless Personal Area Networks) in 2007 which is an open standard, specific to the RFC 6282 (IETF) [28]. Protocol acts as an additional layer for making the IPv6 suitable for the lower-power and lossy networks. A remarkable feature of 6LoWPAN is that it was originally created to support low-power 2.4 GHz wireless networks built on IEEE 802.15.4, but now this standard is realized and used in many other networks, including wireless networks in the bands below 1 GHz, Smart Bluetooth, data transmission over power lines (PLC) and low-power Wi-Fi networks [29].

In Figure 12 an example of an IPv6 network, including a mesh 6LoWPAN network is showed. The uplink to the Internet is provided by an access point (AP) acting as an IPv6

router. In a typical configuration, several different devices are connected to the access point, such as PCs, servers, and so on. The 6LoWPAN network connects to the IPv6 network using the edge router. A boundary router provides three actions: data exchange between 6LoWPAN devices and the Internet (or other IPv6 networks), local data exchange between devices in the 6LoWPAN-network and the formation and service of the radio network (6LoWPAN-network).

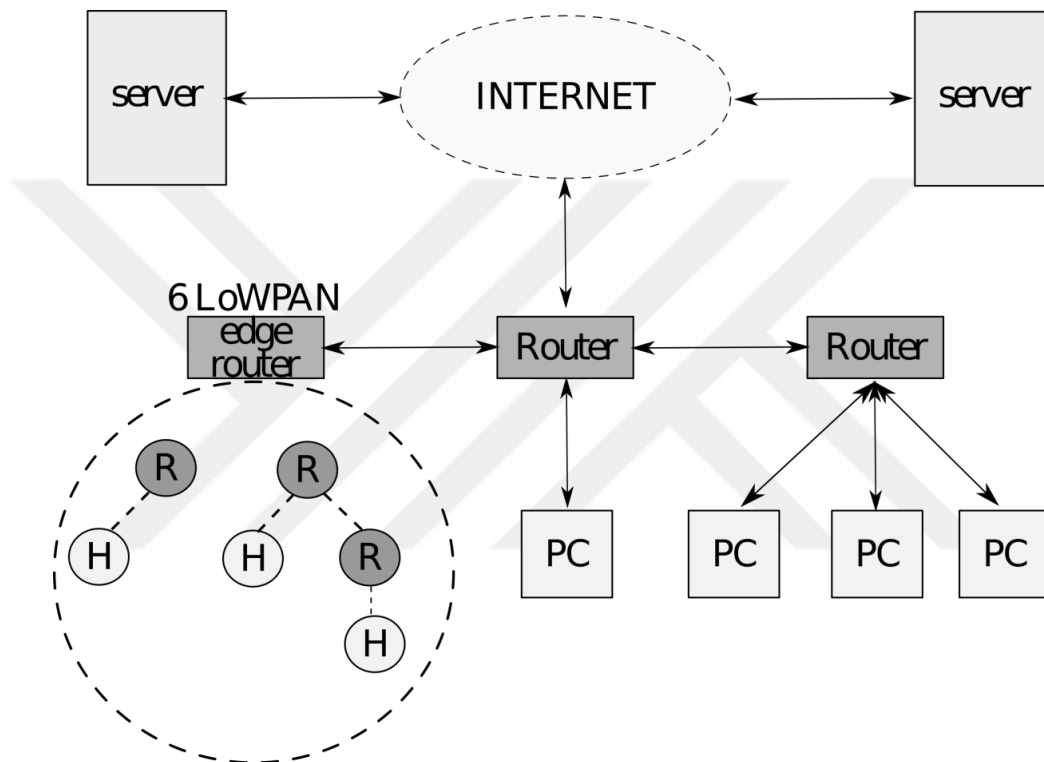


Figure 12. IPv6 network with a 6LoWPAN mesh network

The IETF 6LoWPAN working group was created to address the problem of transmitting IP data packets over IEEE 802.15.4 channels in a manner that satisfies open standards and provides interaction with other IP channels and devices in the same way as with IEEE802.15.4 devices.

This solution has many advantages. Each sensor in the 6LoWPAN network has a individual IPv6 address. This allows many companies to produce LR-WPAN devices that can work together on the same network, allows them to communicate with these devices. Each node of the sensor network is accessible from external networks by IP address. This eliminates the need for complex gateways for each local IEEE 802.15.4 protocol, the many

adapters used by existing applications to communicate through these gateways, simplifies gateway-specific authentication and security procedures.

Many well-established IP-based software tools such as ping, traceroute, SNMP can be immediately used to network and service LR-WPAN devices. Also on the IP-based NAT functions (address substitution), load balancing, caching can be easily implemented. Existing models of data transmission at the program level and services based on HTTP / XML / SOAP make it possible to simplify the development of applications for LR-WPAN networks and unify the integration of devices into the existing corporate network using 6LoWPAN. The Routing Protocol for Low Energy and Lossy Networks (RPL) [5] supports multiple graphs and there is the possibility of sending packets over connections with different parameters. RPL is a protocol for low power and lossy networks. It is based on destination oriented directed acyclic graphs (DODAG) [30]. This protocol provides paths from the router to the receiving node, while the routers require the storage of a small amount of overhead and routing tables containing information about the parent nodes in DODAG. A more detail description of RPL will be given in Section 2.6.2.

### **2.5.2. 6top**

6TiSCH Operation Sublayer (6top) is a logical link layer in the 6TiSCH architecture. 6TiSCH is discussed in more details in Section 2.6.1. As shown in Figure 13 the 6top layer provides to the upper protocol layers management and data interfaces, it also simplifies observing and statistics collection. The main goal of 6top is to let the bandwidth negotiation with one-hop neighbors. Based on the bandwidth need a scheduling function [31] in the 6 top layer will start the slots to be deleted or added.

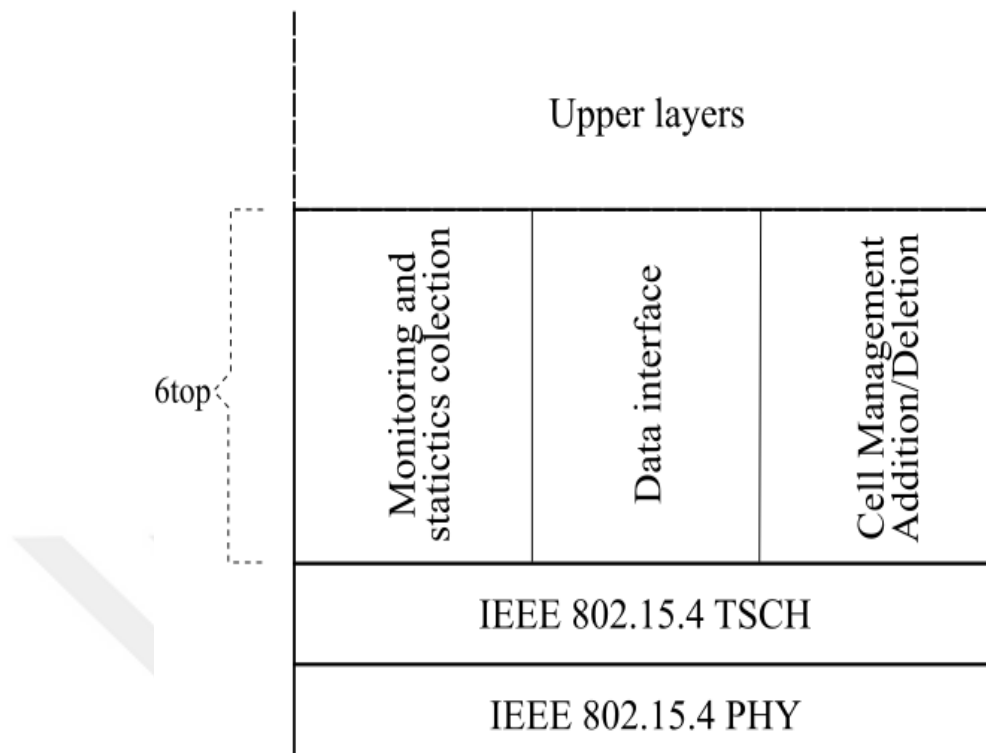


Figure 13. 6top functionalities [18].

6top can be used with centralized and decentralized scheduling approaches. The centralized approach is suggesting collecting topology and traffic requirements by a central PCE (Path Computation Element), which create a communication schedule and through the network transmit it to other nodes. The decentralized approach is suggesting computing node's schedule according to local information strict by nodes.

### 2.5.2.1. Centralized Scheduling

The main role in centralized planning belongs to PCE-node protocol (e.g. CoAp). Centralized scheduling is shown in Figure 14. It collects the network state information and also the traffic requirement of the nodes. Having this information help PCE to have global view of the network and create a schedule for each node in the network. The created schedule is then transmitted to each device within the network. Each node in the network periodically transmits information to the PCE. The PCE after analyzing information of all nodes as a Management Entity (ME) creates a TSCH schedule for all network and send it

back to nodes [32]. While building the schedule PCE checks the QoS requirements of the nodes in the network and assign resources to each node according to its QoS need.

For installing a track in the network PCE can use two various approaches. There are

- PCE can commune to every node on the track individually. This approach is easier to maintain and it can better ensure the correct track installation.
- PCE can commune just to the source node. In addition to installing the resources along the track, the source node uses a separate protocol. This approach minimizes the number of control traffic between the PCE and the LLN in the network.

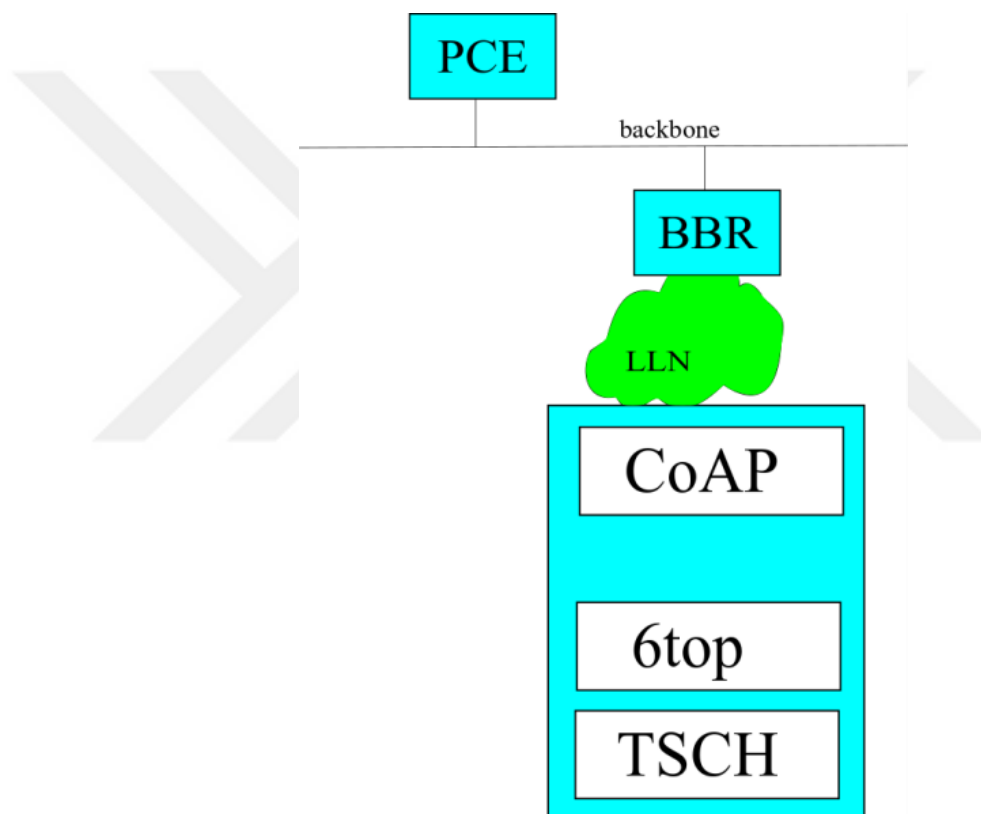


Figure 14. The centralized scheduling

#### 2.5.2.2. Distributed Scheduling

The distributed scheduling is akin to creating a set of routes between nodes in the network. Each node determines their schedule themselves by performing local processing. Every node usually needs information from neighbor nodes to be able to determine their position. Neighbors schedule bandwidth with each other by scheduling unassigned cells

i.e., soft cells. The 6top monitoring process checks performance of the scheduled cells and reschedules the ones that perform badly.

A protocol is required to reserve MAC-layer resources for the multi-hop route created by RPL, to meet certain QoS constraints.

In this study a distributed scheduler is used. Accordingly, a protocol is needed to reserve MAC-layers resources along the multi-hop path identified by RPL routing protocol, to satisfy certain QoS requirements. After reception of application QoS request, the 6top layer configures the appropriate MAC layer resources.

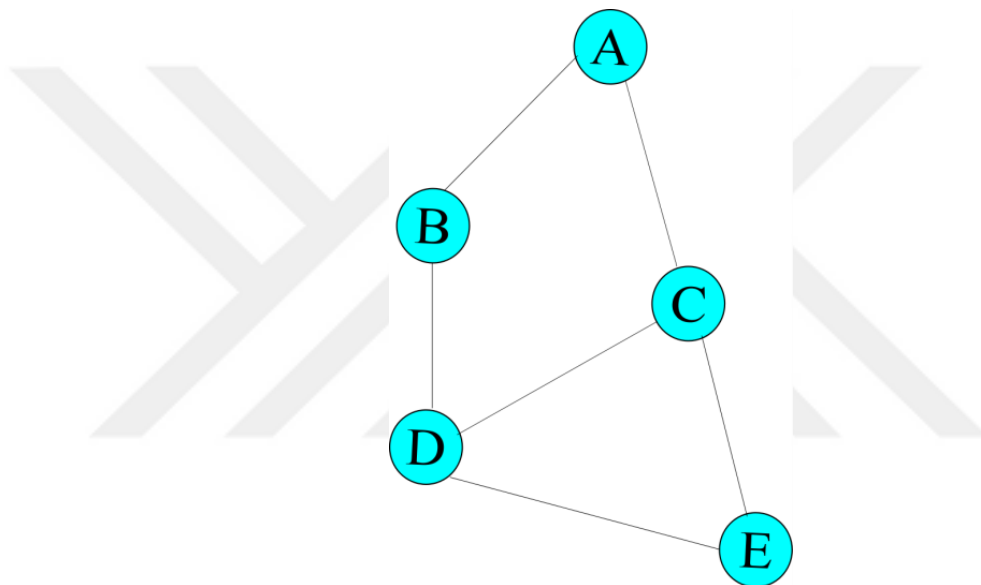


Figure15 The distributed scheduling

#### 2.5.2.2.1. Flows

- Label Switching: the task of 6top is to map input cells and output cells in the same route in a specific node.
- 3 routing can be keep away from by doing this layer (similar to MPLS).
- Each flow can have multiple input and output cells associated with it.
- The 6top layer is responsible for forwarding the incoming frames belonging to a “Track” to the destination (incoming cells are mapped to outgoing cells).

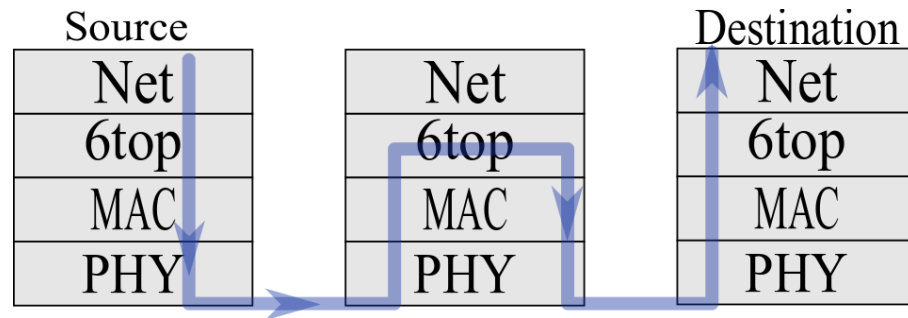


Figure 16. IETF 6tisch: Routing over 802.15.4e MAC

#### 2.5.2.2.2. Cell Types

Four types of cells condition are defined in IEEE 802.15.4e standard: Transmit (TX), Receive (RX), Shared and Timekeeping [32]. Each cell can be configured to be either a TX or RX or Shared cell.

Upper layers can manage a part of a slotframe, this part is called chunk. Generally in centralized scheduling chunk is delegated by the PCE to a node, but when a decentralized approach is used chunk is claimed automatically by any participated node. Given this mechanism, cell is qualified as soft or hard cell.

##### 2.5.2.2.2.1. A Hard Cell

A cell which has been requested specifically and it cannot be dynamically reallocated by 6top is called a hard cell. Generally, hard cells are used in centralized scheduling, because there are scheduled by a PCE (after installing only PCE can move node inside and delete it from the schedule). The cell is installed by PCE given exact slotOffset, slotframe ID, and channelOffset.

##### 2.5.2.2.2.2. A Soft Cell

A cell which can be relocated dynamically by 6top is called a soft cell. Generally, soft cells are using in the upper layer of 6top by decentralized (distributed) scheduling. In contrast to hard cells soft cells is installed by a certain bandwidth requirement. Soft cells are installed through the negotiation procedure, the information about how many cells



must be scheduled to a given neighbors is indicated by scheduling entity The performance of every cell to the same neighbor in the same network is monitoring by the 6top keeps tracking. If 6top monitoring noticed that a cell performs significantly worse than others scheduled cells toward the same neighbor, it relocates this cell at different slotOffset and channelOffset inside the TSCH schedule.

## **2.6. Network Layer**

### **2.6.1. 6TiSCH**

In October 2013, the IETF 6TiSCH Working Group was created to combine TSCH and IPv6. The main idea of 6TiSCH is to link IEEE802.15.4e TSCH (“industrial” performance) with IPv6 through the IETF 6LoWPAN and ROLL standardization efforts and recommendations. 6TiSCH creates a network whose resources are allocated using a mix of centralized and distributed scheduling. The basic principle of 6TiSCH is putting together an architecture binding existing standards (RPL, 6LoWPAN, COMAN, ForCES/OpenFlow, NSIS/RSVP, and Diffserv) over 802.15.4e TSCH. 6Tisch Work Group works on different aspects of routing over 802.15.4e MAC:

- 6top (6TiSCH Operation Sublayer): Responsible for scheduling of resources in a distributed manner.
- PCE (Path computation element): Scheduling of resources that will enable forming of the network.
- Security: responsible for letting nodes to securely join the sensor network and communicate.
- Distributing of configuration information using light-weight protocols.

6TiSCH is supported not just by open-source implementations (OpenWSN, Contiki, RIOT, and TinyOS) but also by several companies, that are building commercial product lines with it. The last version of Contiki OS maintains a basic 6TiSCH configuration including 6Top layer with RPL as the default routing protocol.

### 2.6.1.1. The 6tisch Stack

The 6TiSCH stack is a reference stack that implemented as middle layer between lower (MAC) and upper levels (6LowPAN) of the protocol stack. The main purpose of which is to help implement an IPv6-based IOT stack on top of a TSCH MAC easier. There are a few functions proposed by the IETF that related to transport, routing, or security.

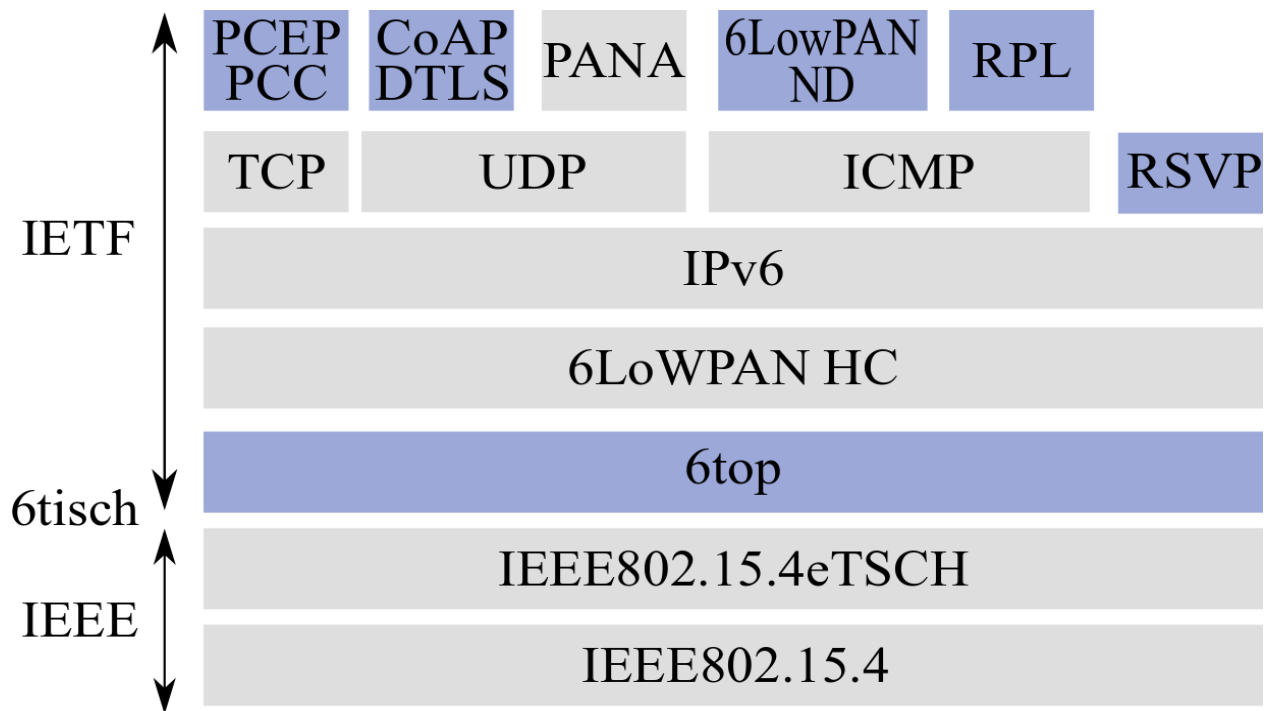


Figure 17. 6TiSCH IPv6-enabled protocol stack for LLNs

### 2.6.2. RPL

Each connection in the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) network is represented by a set of indicators, such as speed, power consumption, encryption support, etc., which are the basis for building the acyclic DODAG graphs. An example of a graph in RPL is shown in

Figure 18, the nodes are arranged in the form of a tree, each of them has a rank (from 1 to 3 in the figure), on the basis of which they select the route of sending messages to the root node. In a converged sensor network, each RPL router defines a set of parent nodes,

each of which is a potential target for the next hop on the path to the "root nodes" in DODAG, and the preferred parent node [33].

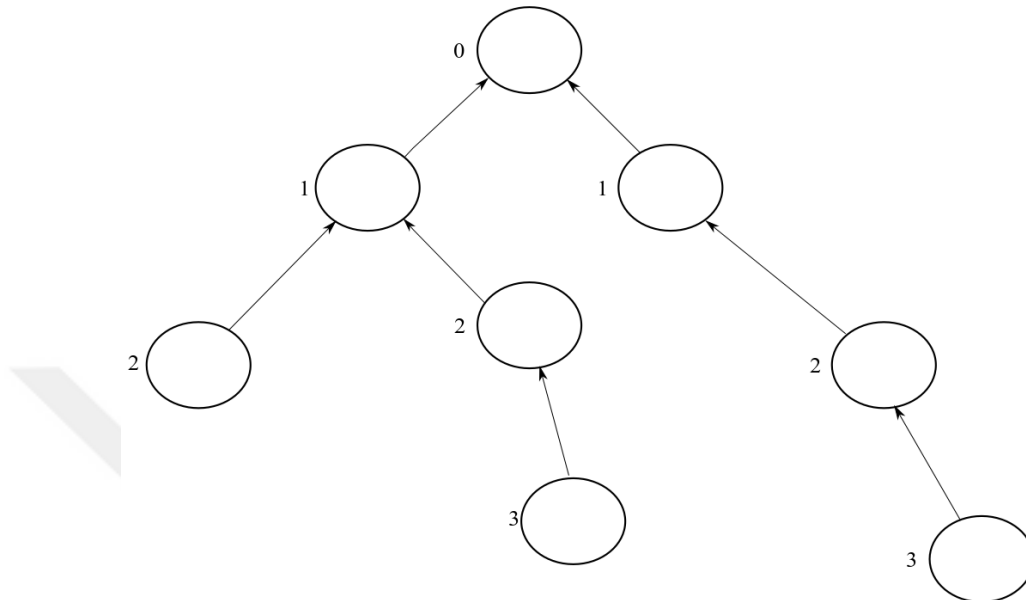


Figure 18. RPL

The advantage of RPL is that it supports several types of messages, there are point-to-point, point-to-multi-point, multi-point-to-point, and multiple graphs are also supported. In the network, multiple graphs are created and the node selects a graph that is suitable for data transfer, depending on the type of data or application for which they are required.

#### 2.6.2.1. Messages in the RPL.

There are three types of control messages to create and maintain the RPL topology and the routing table are used in RPL. There are

- DIO - DODAG Information Object,
- DIS - DODAG Information Solicitation,
- DAO - DODAG Destination Advertisement Object.

RPL uses DIO messages to form, support, and detect DODAG. When the RPL network starts, nodes begin to exchange information about the DODAG using DIO

messages that contain information about the DODAG configuration. These messages help nodes get information about DODAG and select their parent nodes.

DIS Messages - any node uses DIS messages to request DIO messages from neighboring nodes. The latter is required for a node in the case where it cannot get the DIO through a predetermined time interval.

DAO Messages - The RPL protocol uses DAO messages to distribute the node prefix to the predecessor nodes in support of downstream traffic. The DAO message can also be used to distribute availability information and to record routes for areas that include unsaved nodes.

#### **2.6.2.2. The Process of Constructing the DODAG Graph for RPL.**

Each node is assigned a certain rank in such a way that it grows with the node's hop distance from the edge router. Forwarding of the packet to the border router is done by sending it to the parent node with the lowest cost toward the root node.

The formation of DODAG is regulated by several components such as DODAG rank, the objective function (OF), the path metrics and the node settings. A node can be part of several graphs, and a DODAG instance can have multiple root DODAGs with a different set of nodes.

When the flow timer expires the DIO message is sent. The main idea is to transmit DIO messages when there are discrepancies in the DODAG, for example, when a node receives a DIO message with new DODAG parameters such as an OF, a new serial DODAG number or a change in the rank of the parent node, etc . Another example about when to send a DIO message could be loops; when a node gets a data packet from a child node that is intended to be sent down along the same child node according to its routing table, this creates a discrepancy resulting in sending a DIO with an infinite rank. Sending a DIO with an infinite rank within the network is done to initiate a local repair. If a discrepancy is detected, the node resets its DIO timer to increase the frequency of the DIO message announcement. After stabilizing the graph, the frequency of sending DIO messages is reduced, in order to reduce the amount of signaling traffic.

When a node starts the initialization process on the network, it can suspend the process of sending messages until it hears the DIO declaration from the existing graph. In

addition, a node can distribute a DIS message to interrogate its neighbors and speed up the reception of a DIO message from them.

Another choice is to initialize your own floating DODAG graph and send multicast DIO messages to the new DODAG graph (note that this may be preferable if the graph is needed to establish and maintain an internal connection between the set of nodes in the absence of the target DODAG). Uni-directional DIO communications are transmitted in response to unidirectional DIS messages and includes a complete set of DODAG configuration options.

After receiving the DIO message, the node must determine whether it needs to process the received message. If the DIO message has errors in the format, the node discards it without notifications. If not, the node must determine whether the DIO message was sent from the candidate node for the connection. The concept of a candidate in the neighborhood is closely related to the notion of local trust, depends on the concrete implementation and is used to determine if the node is suitable for the selected parent node. At the time a node learns a neighbor, it takes a period of time to control if that the link is dependable.

The node then determines whether the DIO message is associated with the DODAG, to which it already is a member. If the rank of the source node of the DIO message is less than the rank of the recipient node and some value configured by the RPL protocol called DAGMaxRankIncrease, the DIO is processed. This rule is referred the maximum depth rule.

### **2.6.2.3. RPL Security.**

RPL security is available as additional extensions. There are three security modes, in which the RPL nodes can operate. When the first mode, named "unsecured" control of the RPL message, is used the RPL messages are transmitted without any extra security measures. Unsecured mode means that RPL networks can use other simple security features (for example, a data link layer) to meet the security requirements of the application. In the preinstalled mode the nodes join RPL network using pre-shared keys. For example, there are predefined keys that allow them to process and create protected RPL messages. When the authenticated mode is used, nodes can go into network as end nodes using a predefined key to a pre-set mode, or join as a node redirection to obtain an

authentication key. Each message has a secure version of the RPL. Security level (32-bit and 64-bit MAC and ENC-MAC modes are supported), and algorithms (CCM and AES-128) are supported using the messages specified in the protocol. Secure options ensure the integrity and reproduction of protection, as well as confidentiality and protection.

## **2.7. Application Layer**

### **2.7.1. IETF CoAP**

The Constrained Application Protocol (CoAP) is a RESTful web transfer protocol for resource-constrained networks and nodes [34]. The protocol is designed taking the strict band-width requirements of LLNs into account. CoAP easily interfaces HTTP for integration with the web while meeting the mentioned requirements. But the main goal of protocol is not to blindly compress HTTP, CoAP is not a replacement for HTTP. Messages in CoAP work well for small payloads whereas HTTP does not work well with small data payloads.

### **2.7.2. CoAP Features**

There are 2 basic CoAP features:

- Embedded web transfer protocol
- Asynchronous message exchanges [35].

The UDP transport protocol helps send and receive messages in out-of-order. Requests and responses are processed asynchronously (independently of each other), this function makes the task of client and server software implementation with high message throughput easy. A CoAP messaging layer is used to deal with the request/response interactions using Method and Response Codes as shown in Figure 19 [36].

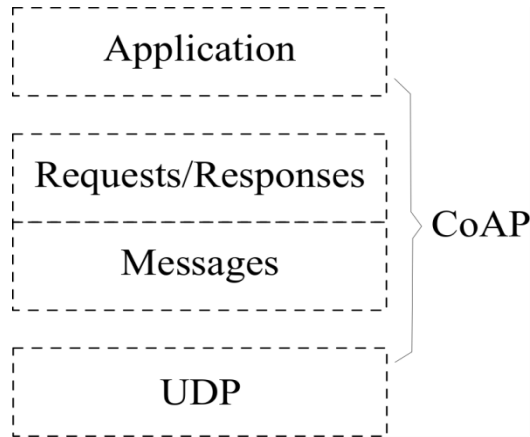


Figure 19. Abstract Layering of CoAP

- UDP binding with optional reliability.

- Group communication and multicast:

Sending a message to an IP multicast group is supported by CoAP. One message can be sent to a group of clients and also enables an easy-to-implement group communication. The detail about group communication capabilities of CoAP is described in [37].

- Content-type and URI support

The "coap" and "coaps" URI schemes are used for identifying CoAP resources and for supplying a way of discovering the resources [34].

- Low header overhead

Short fixed-length binary headers (4 bytes) in messages are used by CoAP. Example of CoAP message header is shown in Figure 20 [34],

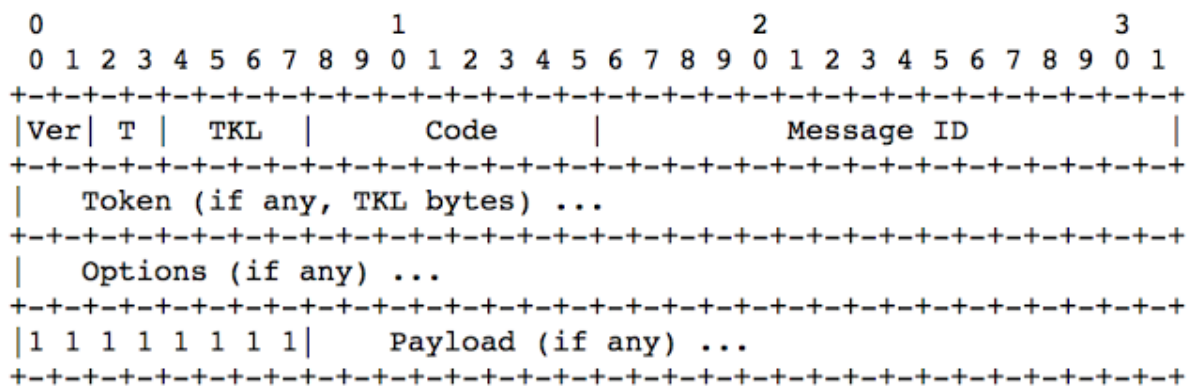


Figure 20. The CoAP message Header (4 bytes),

where:

Ver – Version

T – Message Type (Four different message types are used in CoAP transactions, there are Confirmable - if no packets are lost, then each Confirmable message calls one return message of Acknowledgment or Reset type.; or Non-Confirmable, Acknowledgement, Reset), with its help, the application developer can control the reliability of packet delivery.

TKL – Token Length, if any, the number of Token bytes after this header,

Code – request method or Response Code,

Message ID – 16-bit identifier for matching responses,

Token – Optional response matching token [38].

➤ GET, POST, PUT, DELETE requests

The server provides its resources to URL addresses, and clients access them through these standard methods. The GET method is protected; therefore it must take on a resource just a recover action. The GET and DELETE methods are idempotent (no matter how many times method is used, the result will be the same), unlike PUT, POST is not idempotent [39].

➤ DTLS based PSK, RPK and Certificate security.

The security theme for the "Internet of things" is especially relevant. CoAP supports encryption, but without TCP standard TLS (Transport Layer Security) cannot be used to ensure communication security. Therefore, CoAP uses DTLS (Datagram Transport Layer Security). Figure 21 shows where DTLS is in the protocol stack. The following mechanism is implemented in DTLS to address security issues:

- a. packet retransmission
- b. assigning sequence number within the handshake
- c. replay detection



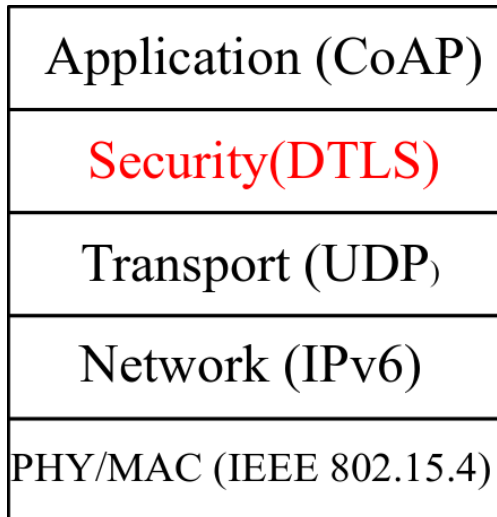


Figure 21. DTLS in Protocol Stack

- Subset of MIME types and response codes

There are 3 classes of response codes in CoAP:

- ✓ Success is the response code represents the request was successfully accepted, conceived, and received,
- ✓ Client Error is the response represents the request include or bad syntax or cannot be fulfilled,
- ✓ Server Error is response code represents the server failed to fulfill an evidently valid request.

- Built-in discovery [35]

A built-in resource discovery function is available in the CoAP protocol. With this function, clients can automatically detect all the resources of the CoAP server. This mechanism is used especially for M2M scenarios.

- Optional observation and block transfer

There is a CoAP extension that allows customers to monitor resources. Once the Observe request is sent and the client receives the corresponding grant, all updates are automatically pushed by the server to the client. In the case where resources are infrequently updated the client doesn't need to poll for updates. When resources are updated infrequently the observable resources are an efficient mechanism for retrieving the updated information from the target device.

## 2.8. Related Work

As identified in [40], the IEEE.802.15.4 specification does not determine the use of queues to handle upper-layer data. For improving 6TiSCH QoS authors of [40] propose the following rules: 1. the node is presented to keep in the queues a configurable number of upper-layer data packets per link for a configurable time which needs to cover the high priority data. 2. Frames are presented by the 802.15.4 layer are added to a queue with a priority that is higher than a priority of frames coming from the upper layers. A frame type Beacon is added to a queue with a higher priority as compared to data frame types.

In [41], Quality of Service for Wireless Body Area Sensor Network is investigated. The Quality of Service is increases with a suitable time slot allocation scheme. The scheme is based on improving the energy efficiency by reducing the number of packet drops. Another aspect of the research addresses the optimization to reduce the node's duty cycle. The proposed protocol assumes the calculation of the time slot duration based on payload size and the slot's schedule that depends on the packet's priority and its lifetime.

In [42], the authors propose another method for improving the QoS by processing information in real time. Adaptive QoS approach involves the use of open-source Contiki OS, a flexible and lightweight OS for tiny networked sensors. The system's main environment involve a database for keeping linked historical data, a mathematical modeling of environment for hosting QoS valuation models, and network simulator tools for modeling the virtual sensor clouds and utilizing real-life history data of sensor clouds.

Authors of [43] implement a mechanism which improving the QoS by dividing all the data into three distinct queues with different priorities. The proposed solution has a better performance as compared to a First In First Out (FIFO) queuing method in terms of end-to-end delay. Furthermore, the proposed approach also outperforms multi-level queue scheduler scheme. But, implementing a QoS mechanism with three separate queues requires a large amount of data memory which is a scarce resource in a low power device. The memory where the queue buffers are implemented deemed to be one of the important criteria affecting the QoS of a WSN. Due to lack of buffer size, some information may have to be dropped. On the other hand, implementing large buffers increases the cost of the WSN node and also the energy consumption. In [44], experiments were conducted for different MAC layer queue sizes where the aim is to analyze the impact of varying size queue buffers on the QoS of the network. In the area of improving QoS, there exists a large

body of literature. However, the fairness of the proposed QoS schemes for WSNs is not analyzed as comprehensively. In WSNs, we need to implement different QoS routines targeting different applications. Each application depending on its QoS requirement may need a different QoS approach. In [45], a mechanism for prioritizing a selected set of nodes in the network, which was originally proposed for 802.11 devices in [46], is implemented for 802.15.4. The mechanism can be used by a certain set of nodes to jam the network for a pre-defined interval. This enables the selected set of nodes to access the network with a higher priority. But, such QoS approaches prioritizing nodes may starve the rest of the network where the nodes have a lower priority. Furthermore, the extra messaging to enable channel access can have a negative impact on the network throughput. To the best of our knowledge, this work presents the first QoS study for IETF 6TiSCH protocol. The study will make use of the TSCH MAC's [47] ability to allocate new resources to the high priority data on demand via the 6Top protocol. The main aim of this study is to improve the fairness among the traffic flows having different priorities.

### 3. QUALITY OF SERVICE

With the introduction of Internet enabled low power devices, WSNs solutions need to deal with the challenges of the Internet leading to creation of complex networks of small devices. Many applications with different quality requirements are emerging, which makes the QoS as one of the key area of research for low power wireless IoT networks. QoS is discussed as a measure of service quality that the network offers to the end user through the application.

The QoS is introduced in networks to determine the network's ability to meet the requirements and characteristics. The quality of service determines whether the network can provide a data transmission service under the specified conditions. In other words, the quality of service is an integral characteristic and is described by a set of parameters. In classic networks, such parameters are used bandwidth, packet delay, jitter or delay spread, as well as the probability of packet delivery. Unlike classic networks, wireless sensor networks are not only a service for data transmission, but also for the collection and processing of data. Accordingly, the quality of service in wireless sensor networks and its provider will also differ from the classical view [48]. In Figure 22 Parameters of QoS are shown [49]. These are:

- Bandwidth (BW) - This indicator shows how many data packets can be transmitted per unit of time.
- Delay – This indicator shows sending data packet delay
- Jitter - This indicator shows the variation of the delay with respect to neighboring data packets
- Packet Loss - This indicator determines the number of packets lost on the network during the transmitting.

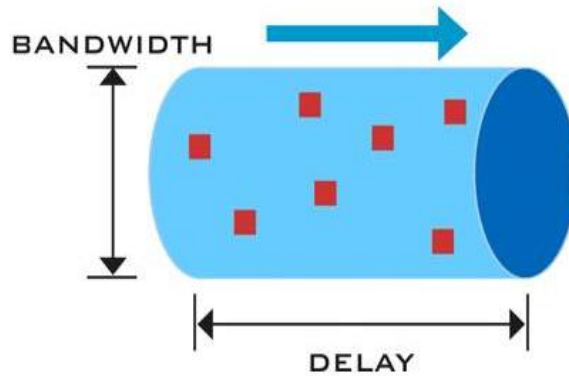


Figure 22. Parametrs of QoS

Quality of service parameters are often interrelated. In [48] two main approaches for QoS provisioning in WSNs are surveyed. There are:

- The Layered approach. This approach considers QoS for each level (as MAC, network, and transport layers) separately.
- The Cross-layer approach. This approach considers QoS through all layers of the communication protocol stack. That means that it can dispose of information from the entire network.

The QoS is necessary in the case when there traffic arrival pattern in the network is variable and the instantaneous capacity of the network is not adequate for this instantaneous traffic leading to a temporary or permanent bottle-neck. The temporary congestions in the communication networks generally handled via queuing mechanisms.

Basic QoS queuing mechanisms are:

- FIFO - The FIFO is the simplest line queuing. The data packet that comes first is processed first, regardless of its priority.
- Priority Queuing: the Priority Queuing is a queue that arranges the packets in the queue according to their priorities. A data packet with the highest priority will be processed first.
- Fair Queuing: The Fair Queuing is a queue mechanism, which allocated each traffic class equal access opportunities to the wireless channel. For every data packet, an appropriate fraction of the bandwidth is allocated. FQ guarantees an equitable distribution of resources for all packets. That means that a larger packet gets a larger share of the bandwidth of the line. However, this mechanism does not take into account the priority of the data packages.

➤ **Weighted Fair Queuing:** The technology that provides different classes of traffic with different rights (the "weight" for different queues is different), but simultaneously serves all queues.

This study presents performance results of FIFO, Priority Queuing and WFQ mechanisms that are implemented for IETF 6TiSCH protocol.



## **4. IMPLEMENTING QoS ALGORITHMS TO 6TISCH**

### **4.1. Development Platform**

There are quite a few open-source operating systems for microprocessor systems of nodes, such as Contiki OS [50], TinyOS and FreeTos. Nowadays Contiki is one of the most popular open source embedded operating systems. Contiki OS is expected to be one of the main operating systems that will be used to connect billions of the Internet of Things devices. Various OS functions cover program/process management, resource management, memory management and communication control.

#### **4.1.1. Contiki OS**

Contiki OS is designed for embedded network systems, in particular, for smart devices. The first version of Contiki OS was released by a team of developers in the field of industry and scientific centers, in 2003 [51].

Contiki OS supports a complete IP stack with standard protocols such as UDP, TCP, and HTTP, as well as standards for low-power networks such as 6lowpan, RPL and CoAP. The stack of Contiki IPv6 software was developed for Contiki by Cisco, fully certified within the framework of the ReadyLogoIPv6 program. Contiki is the first operating system for sensor networks that provides TCP / IP communication (using the uIP stack (microIP) - an open TCP / IP stack/module that is designed for microcontrollers with 8- and 16-bit architectures). The Contiki network stack is approximate in Figure 23.

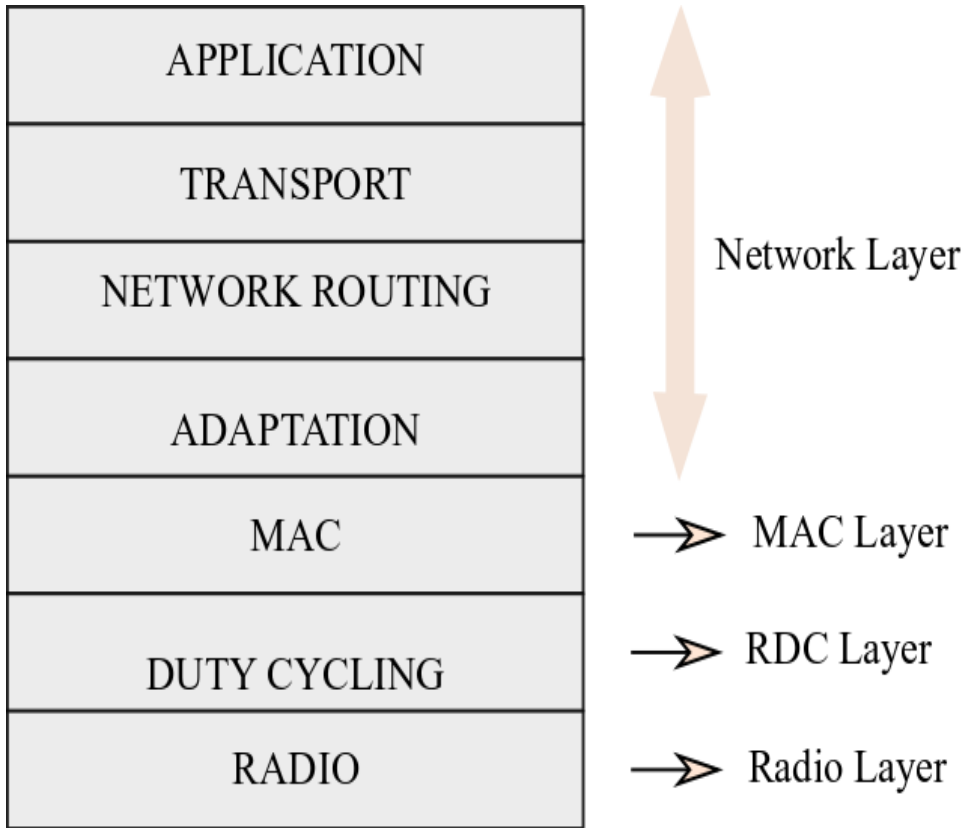


Figure 23. Contiki OS network stack

The Contiki OS is a hybrid operating system. Contiki can be discussed as an example of a modular kernel design, combining features from monolithic and microkernel. This allows it to inherit some advantages of both systems. Contiki implements a hybrid model through an event-driven kernel. Contiki implements multithreading as an application library. It is not necessary to link with the program if the program requires no multithreading features. There are two key benefits when building a system like this. First, the per-thread stacks and the locking mechanisms are not needed for concurrency. This benefit is evident by looking at why per-thread stacks are not suitable for embedded devices. Per-thread stack approach allocates a stack for each thread. When a thread has created the memory for stack must be assigned or allocated. This creates a problem because memory assigned to a specific task cannot be used by another concurrent thread. The second benefit is that event-driven model with the state-driven programming is difficult for programmers to manage. In addition, not all programs can be expressed as state machines. A purely event-driven OS is providing a task could consume the full CPU time completely. When the task is complex and requires a long CPU time, the system cannot respond to external



events sometime. A preemptive multi-threaded system overcomes the problem in such situation. Hence, a hybrid system can be considered as the most suitable for such kind of problems.

Similar to microkernel design, Contiki also uses messaging mechanism for communication between services and the kernel. However, instead of using the peer messages discussed in microkernel lesson, Contiki uses posting events. The only difference between pure messages and event posting is formatting. There is no hardware abstraction layer or HAL [52] in Contiki. Therefore, applications and drivers can directly communicate with the hardware. The hardware abstraction layer is a layer between application, component code, and hardware. This is mean that device drivers and applications communicate straight with the hardware.

The Contiki OS executable system has two parts - the core and the user programs. These services can be updated or replaced dynamically independently of each other at runtime, which, according to the developers, leads to a flexible system structure. Based on a modular kernel approach, in Contiki, loading and unloading models at runtime abilities are realized. This allows Contiki to load only the necessary services, consequently allowing it to have a reduced kernel size. Almost all abstractions are implemented as libraries and services, except for CPU multiplexing, which is the only obstruction provided by the core system. The implementation is almost similar to a schedule of a microkernel. Looking at this implementation, Contiki is developed in C language, and it's applications are also written in C. Therefore, it is easy to develop an application, reprogram, and replace services. Contiki is easily portable. This implies that it possible to run it on various microcontroller architectures such as the CC2538 [53], CC2650 [54], and the MSP430 [55], and so on. It is easy to add support for new microcontrollers using the built-in libraries and existing Contiki platform. For example, Contiki can be easily ported to their own board by creating configuration files for the specific microcontroller and edit existing general configuration files of Contiki.

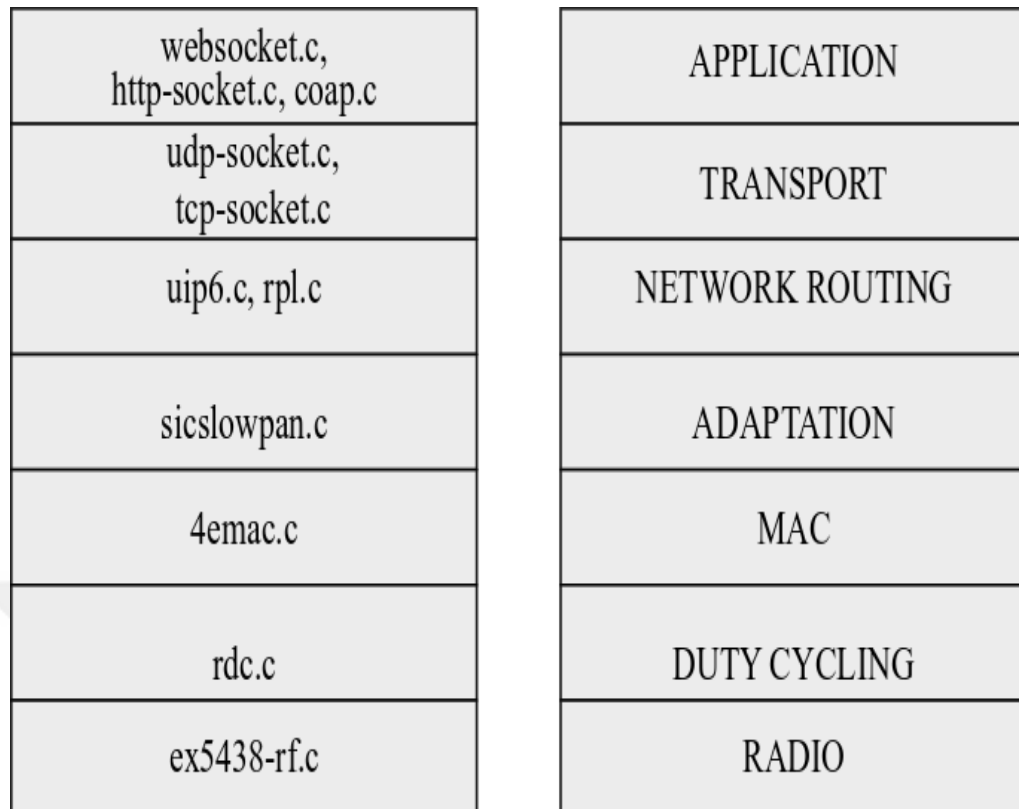


Figure 24. Contiki network stack

Contiki OS supports all popular microcontrollers and embedded devices. It provides IPv4 and IPv6 connectivity through the uIP and uIPv6 protocol stacks. In Figure 24, the IPv6 protocol stack is exchanged.

The IPv6 stack is for smart devices that have been certified with IPv6 Ready. Thus, Contiki offers modules for solving various tasks at the required network level. With a default configuration, Contiki uses 2 kb of RAM and 40 kb of ROM in Figure 25 [56].

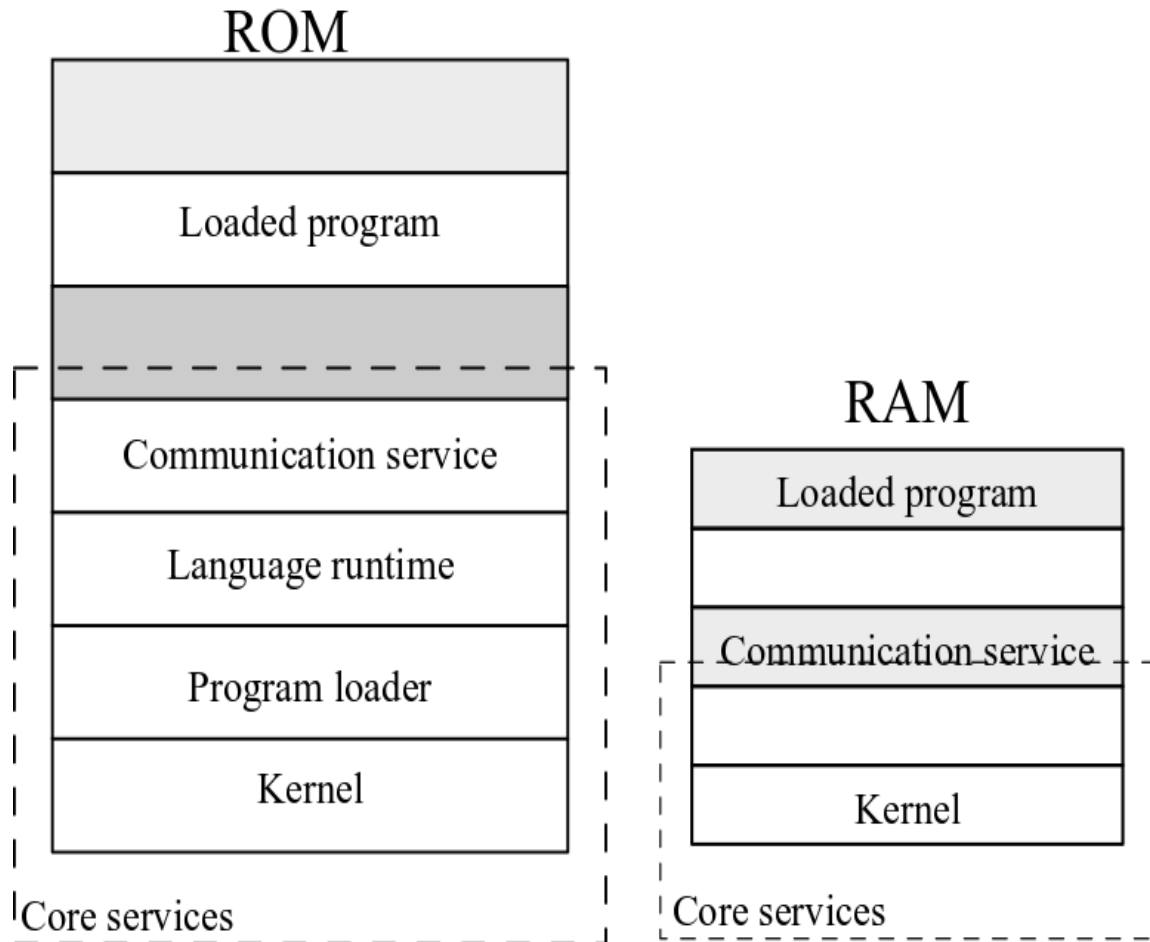


Figure 25. The Contiki OS: the system programs are partitioned into core services and loaded programs

Contiki consists of a kernel that manages events; the program is loaded and unloaded dynamically at runtime. Protothreads, which provide a linear streaming style of kernel initialization are used processes.

One address space is shared by all processes and they execute in the same protection domain. This enables Contiki to run in memory constrained devices. A process in Contiki is a piece of program code which is executed by Contiki's system. A process is started in two ways. The first is when Contiki starts, and the second when the module loaded into the memory. A process runs when an event related to the process occurs, such as a timer or an external event.

There are two types of execution modes in Contiki. These are cooperative and preemptive modes. Cooperative process code runs sequentially in a queue. This means that the first process has the right to use the CPU, while other processors wait for their turn to access the CPU. After the process occupying the CPU finishes, then the next waiting

process in the queue gets the right to use the CPU. In contrast, preemptive mode handles process differently. In the preemptive context, a running process can be stopped by an interrupt, and the higher priority test immediately takes over the right to use the CPU. After finishing its job, it returns the right to access the CPU resources to the interrupted process. The Contiki process is determined by control block and the process thread. The process control block is composed of information about each process such as the state of the process, the pointer to the next process, name of the process, a pointer to a process thread, the state of the protothread and internal flags. Figure 26 shows a structure of a process in Contiki.

```

struct process {
    next;
    const char *name;
    int (*thread)(struct pt*,
        process_event_t, process_data_t);
    struct pt pt;
    unsigned char state, needspoil;
}

```

Figure 26. A structure of a process in Contiki

The process Control Block is only used by the kernel. Therefore, users do not have any right to access the Control Block directly. This helps the system protect the process and avoid unexpected mistakes or bugs caused by programmers. Actually, the kernel only keeps a pointer to the process of state which is health in the process' private memory. This means that every process has its own state, and the state cannot be accessed by other processes. The Process Control Block's structure is simple, and it does not contain complex information. Therefore, it is lightweight, and it just occupies a few bytes of memory.

In Contiki, a Process Control Block cannot be announced or defined directly. The block is defined via the process macro. As shown in Figure 27 the process macro has two parameters, the name of the Process Control Block and the textual description.

```
PROCESS(hello_world_process, "Hello world process");
```

Figure 27. Example of process. "Hello world"

The variable name of the Process Control Block is used for accessing the process, while the description text of the process is used by programmers for debugging. Contiki uses posting events for inter-process communication. This mechanism is also similar to the messaging mechanism in Microkernel described before. Contiki's system is partitioned in two parts as show in Figure 25. The first part is the core, and the second is the loaded program section. The partitioning is specific to the deployment in which Contiki is used and it is made at the compile time.

Multithreaded mode with priorities in the Contiki OS is implemented using an application library that runs on a kernel, managed by events. Applications that provide multithreaded processing are linked to the running application as needed, i.e. if it explicitly requires a multithreaded model of computation.

#### 4.1.1.1. The Contiki's Kernel Architecture

In the heart of the Contiki OS is a modular kernel architecture. Therefore, the kernel is minimal. Due to this reason, the kernel comprises only a few lines of code that perform vital functions. For example, the kernel consists of an event scheduler, which is light in terms of code size. In Contiki, all program execution is triggered either through the polling mechanism or by events which are sent by the kernel. In Contiki, the kernel supports two types of events, synchronous and asynchronous. Asynchronous events are not delivered to the receiving process immediately after being posted. Instead, they are stored in an event queue. The events in the queue are delivered to the receiving process by the kernel. The kernel delivers an event from the event queue by looping through it. The role of the asynchronous event receiver can perform on a specific process or all running processes. In case if the receiver is a specific process, then this process is invoked by the kernel to deliver the event. If the receiver of the event is established to be processed in the system, then step-by-step the same event is delivered by the kernel to all processes. The process post function is used for posting an asynchronous event. First, the size of the event queue is

evaluated and if there is room for the new event, then the event is added to the queue. Otherwise, the function will return zero.

Synchronous events work differently from asynchronous events. In the sense that when a synchronous event is dispatched, it is delivered to the receiving process immediately. Another difference between asynchronous and synchronous events is the type of receiver. While asynchronous events can be received either by a specific process or all running processes, a synchronous event can only be received by a specific process. The mechanism for invoking a synchronous receiver is similar to calling a function. A synchronous receiver is called, so it performs its task, after which it returns control to the posting process. Besides events, the kernel supports a polling mechanism. The polling mechanism behaves like a high priority event which is scheduled between asynchronous events. Typically, processes operating close to the hardware level use polling for evaluating hardware status. In a case, if a poll is scheduled, each process which implements a poll handler is named sequentially according to priority. In a similar way to event posting functions, Contiki also provides a process poll function for posting a poll. Posting a poll has a similar effect as an interrupt. It causes the receiving process to be scheduled as soon as possible. A single shared stack is used in Contiki for each process execution. A stack's space requirements are reduced by uses of asynchronous events. An event identifier is an 8-bit number used as a unique identifier for an event. When a process receives an event It uses the event identifier to determine the actions to perform. There are several event identifiers reserved by the Contiki kernel to handle process control, inter-process communication, and peripheral access.

Contiki schedules all events using a single level hierarchy, and events cannot be preempted by other events. The only way to preempt an event is using interrupts. It is important to notice the interrupt must be supported by an underlying real-time executive. Correspondingly, interrupt handlers cannot post any event. For a poll event's requesting a polling flag is used. The interrupt handlers in a manner to request instant polling are provided by this flag.

#### **4.1.1.2. The Power Saving in Contiki.**

In general, power can be saved by putting inactive nodes to sleep. In some cases, choosing appropriate level protocols may also reduce power consumption. Contiki also

provides a mechanism for reducing power consumption even though there is no explicit abstraction for it. The mechanism works by checking the size of the systems event queue. If the event queue is empty, the micro-controller goes into a sleep mode until it's woken up by an interrupt.

#### **4.1.1.3. Services and Libraries in Contiki OS**

The services in Contiki are implemented as modules. When a particular service is required the corresponding module is loaded thus making Contiki run efficiently. From a certain point of view, a service is different from other application programs since a service can be used by several application programs. A service is a type of shared library. One of the benefits of Contiki Services is that they can be dynamically replaced at random. With this strategy Contiki minimizes the number of modules to be loaded during booting time. As a result, memory researches can be used more efficiently.

In every operating system, each service should have a unique identifier, defined by the system or users. The format of the identifier depends on the particular operating system. In Contiki, a service identifier is a textual string which presents the functionality of the service. Identifiers are used when application programs invoke a service. When a service is requested, ordinary string matching is used by the service layer for request installed services.

#### **4.1.1.4. Communication in Contiki**

Communication plays an essential role in any wireless sensor network system. Basically, communication in wireless sensor network architecture includes three parts. Sensor nodes, a gateway, and the backend system. Sensors collect data and send it to the gateway wirelessly. Depending on the network characteristics, a particular communication protocol is used. For example, Wi-Fi or Bluetooth can be used for sending data from sensor nodes to a gateway. Then the data is forward via the internet and reach the cloud. Finally end-users can access data or use services offered by the cloud. Contiki, implementers communication as a service. Therefore it has all the features of Contiki services, such as runtime replacement. As a result, it allows to simultaneously load

multiple communication stacks. This helps to reduce the latency of loading modules one by one. In addition, this feature can be used for comparing different communication protocols.

#### **4.1.1.5. The Loosely Coupled Communication Stack in Contiki.**

Communication in Contiki may be divided into different services. The communication stack uses synchronous events to communicate with a program application. Synchronous events is used instead of asynchronous events or polling because that for all communication processing, a just single buffer can be used since it is necessary that synchronous event handlers run to completion. This approach ensures that there is no need to copy data to intermediate buffers. The bias drivers operate between the communication stack and the hardware. When the communication service is awoken, it inspects the packet in the buffer. Based on the content of the packet header, the service searches for the application which is the destination of the packet. When it finds the exact application, it put a synchronous event up to that application program. After receiving the event generated by the communication stack, the application program proceeds to process the packet. In some cases, it is necessary for the application program to replay by posting some data to the communication stack before returning the control to it. Then, when the communication stack takes control, it places the reply in the communication buffer and appends its header to the outgoing packet. Finally, it gives control to the device driver, so that the packet can be transmitted.

#### **4.1.1.6. Protothread**

The most important features of Contiki is Contiki protothreads. In order to provide sequential flow of control, Contiki provides protothreads. Protothreads are lightweight and stand as stackless threads with only two bytes of memory per thread. One of the benefits of using Contiki's protothreads is that systems overhead can be small compared to other kernels. This comes as a result of sharing the stack by old protothreads. This has the additional benefit that context switching can be done easily by stack rewinding. It is more efficient to implement Contiki protothreads than applying traditional ways of multithreading on wireless sensor network applications. Contiki's protothreads are



extremely lightweight. Similar to other modules and processes in Contiki, protothreads are completely written in standard C. This avoids any extra requirement for using and compiling protothreads. A protothread basically works within a single function and cannot span over other functions. It is possible for a protothread to call another function; however it cannot plug the flow inside the called function.

Protothreads are implemented using local continuations, which are used to present the current execution state of a program at the specific places. A local continuation can be implemented using or the machine specific assembler code, or the standard C constructs, or the compiler extensions. The machine specific assembler code can be done when the processor state is saved or restored. As a result, each proto-thread uses around 16 to 32 bytes of memory. The second way of using a standard C requires only 2 bytes of state information per each protothread. Although this approach has slower memory overhead it still has some challenges and restrictions. The last method is to use certain C compilers which have a specific C extensions for implementing protothreads. For example, a GCC version supporting label pointers can be a good candidate to perform this. As a result, only four bytes of RAM per protothread are required. Figure 28 shows an example of sealed code of protothread.

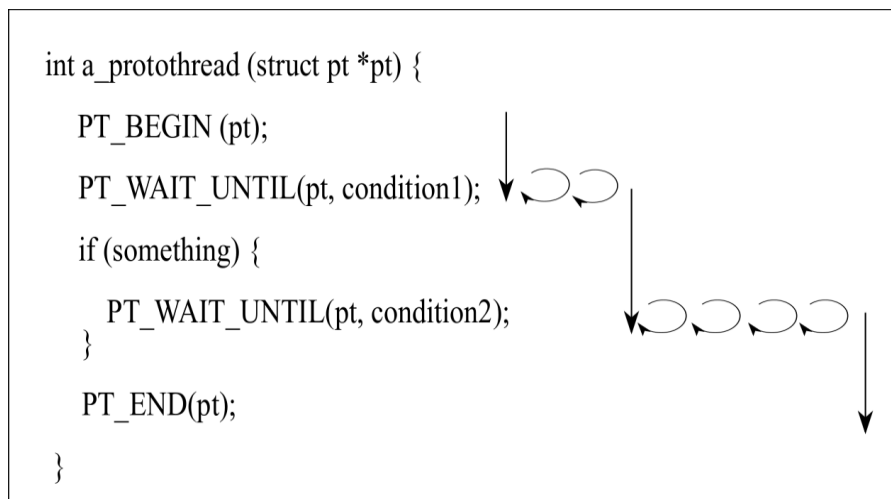


Figure 28. Example of sealed code of protothread

There are some strict requirements for an operating system of embedded wireless sensor network devices. These requirements arise due to resource constraints. Memory, which is one of the most precious resources of embedded devices, must be considered

when choosing a suitable OS. The minimum RAM and ROM required to run a simple wireless sensor network application. Note that RAM and ROM usage varies depending on the actual application. For instance, a simple wireless sensor network application takes less than 2 kilobytes and 30 kilobytes respectively.

#### 4.1.2. Cooja Simulator

Devices with Contiki often operate in accordance with basic wireless networks. The development and debugging of software for such networks is a time-consuming task. Since 2.0 version Contiki application includes COOJA simulator. COOJA is the simulator that specifically designed for Wireless Sensor Networks. Cooja provides functions for modeling devices and networks [57].

The simulated node in COOJA has three main properties: data in memory, node type, and its components. The node type defines the general properties for the nodes. For example, the nodes of the same type run the same program code on the same simulated hardware architecture.

tools/cooja/ - Cooja Simulator source code is located in this folder.

The interface of Cooja simulator shown in Figure 29. The window of Cooja simulator is filled with the main simulating tools:

- The Network window - the window is populated by sensors node, the functionality of this tool is the showing the location of each node in the network and the visualization of the status of each node.

- Script editor - this panel shows a code of test script.

- The Simulation Control window - with this panel the simulation can be started, paused, reloaded or executed. The time of execution and the speed of simulation are shown in this panel.

- The Mote Output window - all output of serial interface of the nodes are shown in this window.

- The Timeline window - the simulation timeline that show messages and events (channel change, log outputs, LEDs change).

Also Cooja simulator has Notes window for temporary notes in simulation.

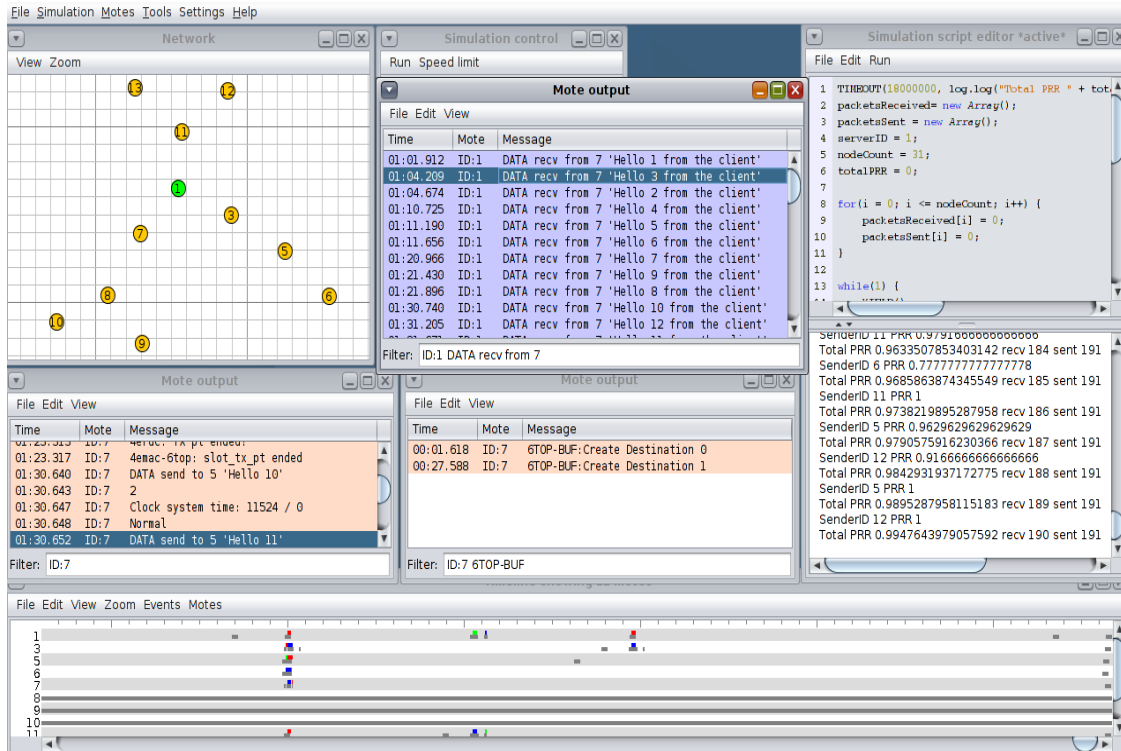


Figure 29. The interface of Cooja simulator

The Cooja simulator's authors in [58] claim that the simulator can work on three layers: the network layer, layer of OS (operating system) and the layer of set machine code instruction. The compare of simulators for wireless sensor networks is given in

Figure 30. Therefore Cooja is the cross level simulations and it can perform Contiki programs and other options: either as a compiled code right on the host of the CPU, or launching on the emulator of the TIMSP430 microprocessor.

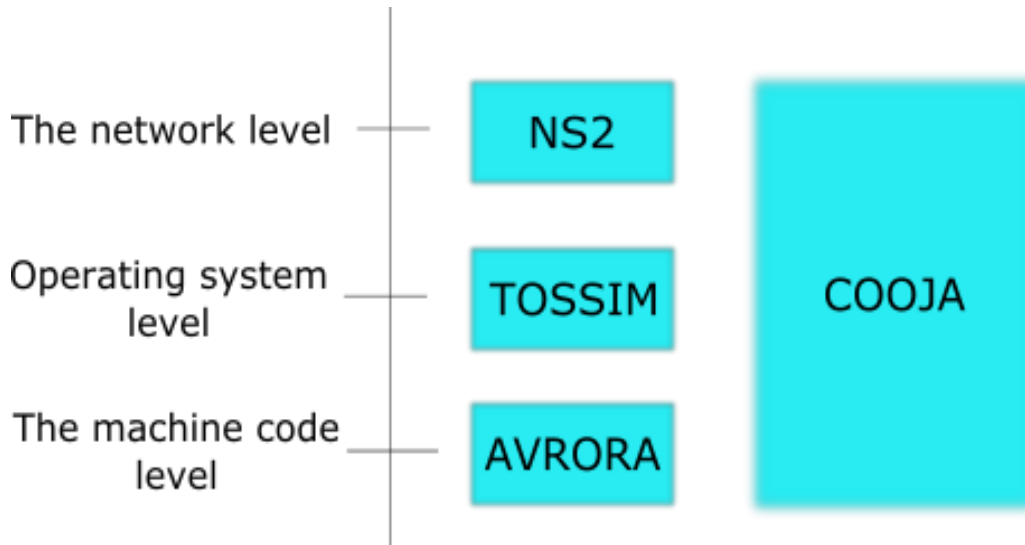


Figure 30. Comparison of simulators for wireless sensor networks

➤ The network layer is used mainly by protocol developers, where hardware reporting is omitted (the proper simulation of the behavior of some specific hardware is not such an important issue). The layer controls the radio devices and propagation of the sensor. Also in this layer, Heterogeneous network written in Java can be implemented in Cooja.

➤ In the operating system level, the main aim is to simulate the execution native operating system code. Especially for Contiki developers is important to have ability to testing and assessment of changes in libraries of Contiki.

➤ The machine code instruction set level can be used as an alternative method. Using Java-based microcontroller emulator nodes having various structures may be simulated instead of a compiled Contiki OS.

Cooja supports everything necessary to support IoT protocols such as 6LoWPAN, 6TiSCH and RPL.

## 4.2. System Model

The goal of this work is to implement different QoS mechanisms for 6TiSCH protocols. QoS components were designed and integrated into the 6TiSCH protocol stack seamlessly to enable the performance comparison of several QoS approaches. All experiments are conducted using Contiki OS. Network consists of 20 TI exp5438 [59]

notes. They are used to simulate the implemented QoS mechanisms. The nodes are uniformly distributed in a 300m by 250m rectangular area as indicated in Figure 31.

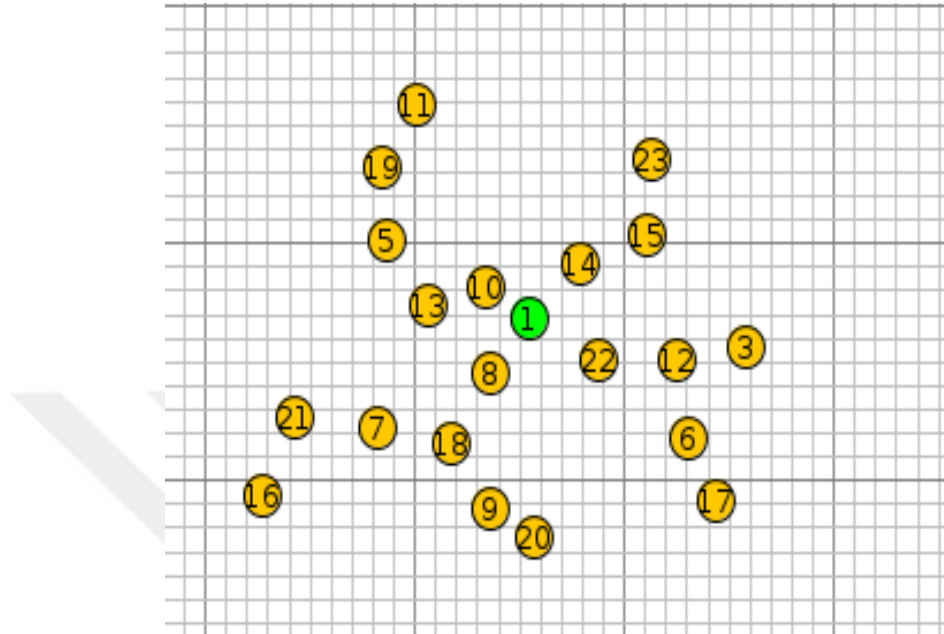


Figure 31. Simulation Network

The testbed includes the scripting editor which provided by Cooja simulator. The scripting editor is designed and used as a tool for the collection of simulation results.

Each mote at the same time sent 3 messages with different priorities into the 6TiSCH buffer simultaneously. As shown in Figure 33 three different ports are used. The priority class of the packet depends on the port the data originates from. Therefore three priority classes namely HIGH, NORMAL and LOW are defined for three different applications represented by three different ports. Weights of priority are given as 4, 2 and 1 respectively as shown in Figure 32. The performance results are averaged over 5 runs. Each experiment is run for 35 minutes with a message interval of 5 seconds. During this 35 minutes period, every mote inserts around 300 messages into the TSCH buffer.

Table 4. Parameters used in the study.

Parameters	Value
Number of nodes	20
Startup delay(minute)	35
Packet amount of each node	300
number of experiments	5
Propagation mode	Cooja UDGM [14]

#### 4.2.1. Greedy Priority Queuing

Greedy Priority Queuing (GPQ) was studied in 1954 in [60] study and also was known as Head-of-Line priority. GPQ is an abstract data structure like a stack or a queue, where each element has priority. GPQ mechanism schedules traffic such that A higher-priority data gets serviced before low-priority data. If the elements have the same priorities, they are located depending on their position in the queue. The main problem of GPQ mechanism is that this kind of traffic scheduling can cause the data of other lower-priority to not send. In our study GPQ mechanism uses 3 different data priority - High, Normal and Low as shown in Figure 32.

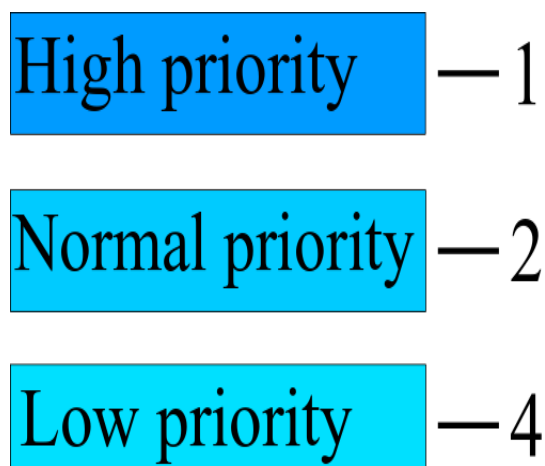


Figure 32. Buffer Priorities

The mechanism of priority traffic processing enables the separation of all traffic into a small number of classes, assigning to each class a certain numerical priority. Classification can be done in different ways and represents a separate task. Packets can be prioritized according to the type of network protocol. For example in our study, priorities are assigned according to the TCP/UDP port number as shown in Figure 33.

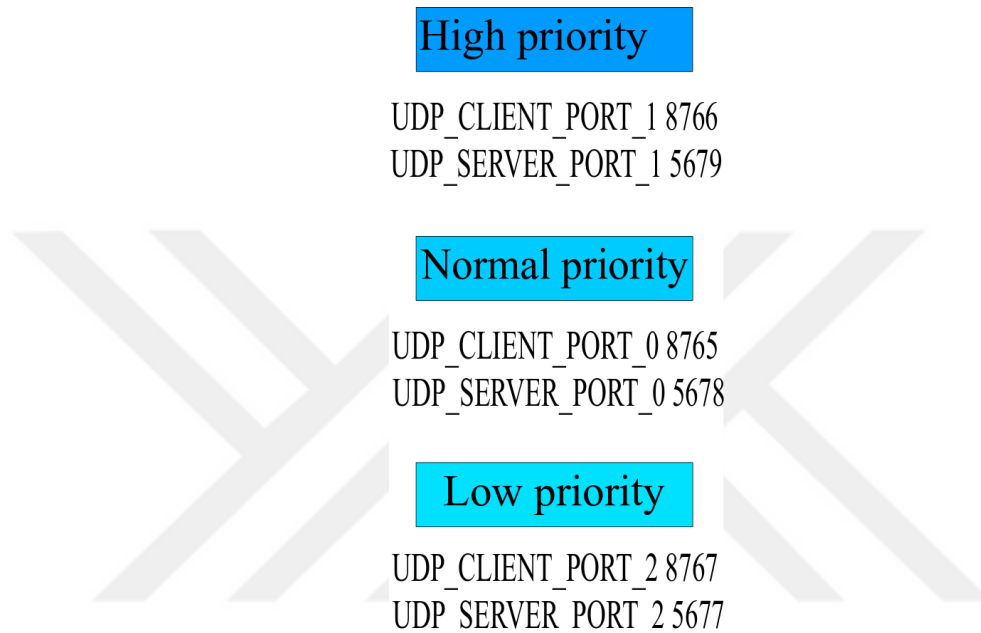


Figure 33. Priorities based on TCP/UDP port

In data packets, a special field is supposed where the specified priority value can be recorded, for example, HIGH, NORMAL and LOW priority. Using the priority field within the packet, high priority data is inserted into the beginning of the queue, followed by data with medium priority, and then with low priority as shown in Figure 34. GPQ provides a high quality service for the packets with the highest priority. As for the other priority classes, the quality of their service is lower than that of the high priority packets.

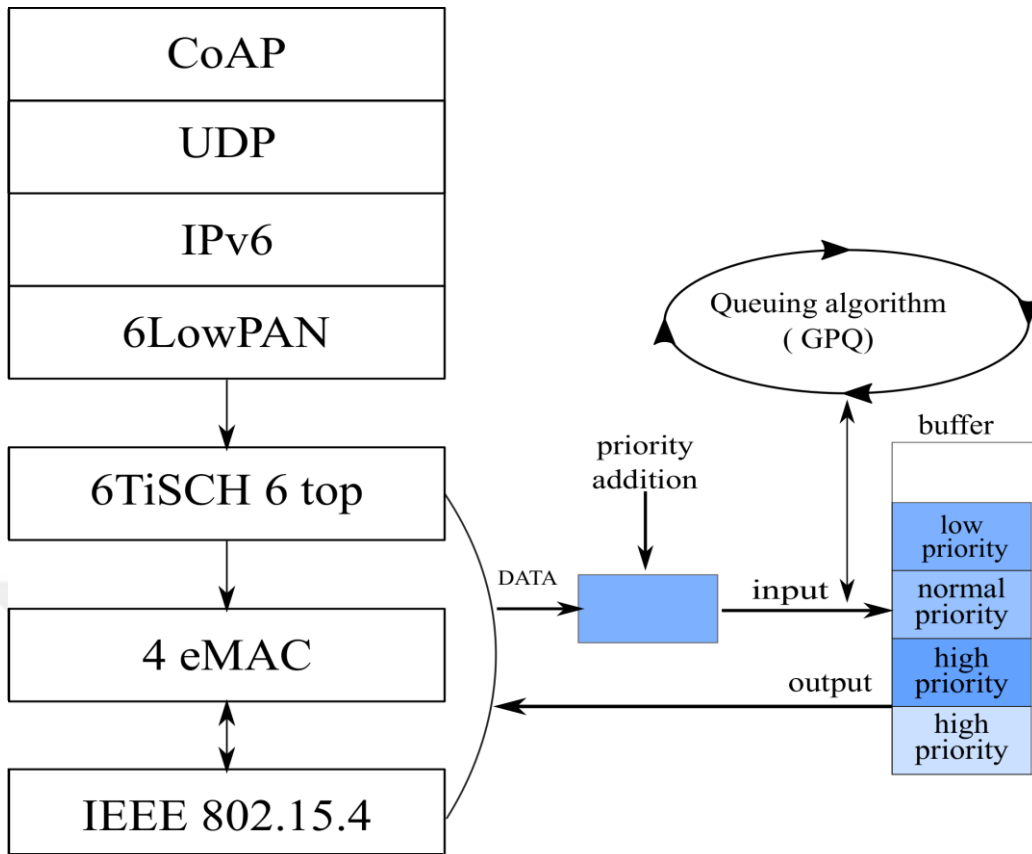


Figure 34. Scheme of using GPQ QoS mechanism

### 2.1.2. WFQ

Weighted Fair Queuing (WFQ) [61] was presented by Lixia Zhang, Alan Demers, Srinivas Keshav, and Scott Schenke in 1989. WFQ provides fair data sending according to assigned weights. WFQ provides different traffic classes with different rights, but simultaneously serves all data. In our study when implementing this algorithm in the Contiki OS, time-stamp and priority are used as input parameters. The weight of each packet is determined by the formula (2.1.2.2) and shown in Figure 35.



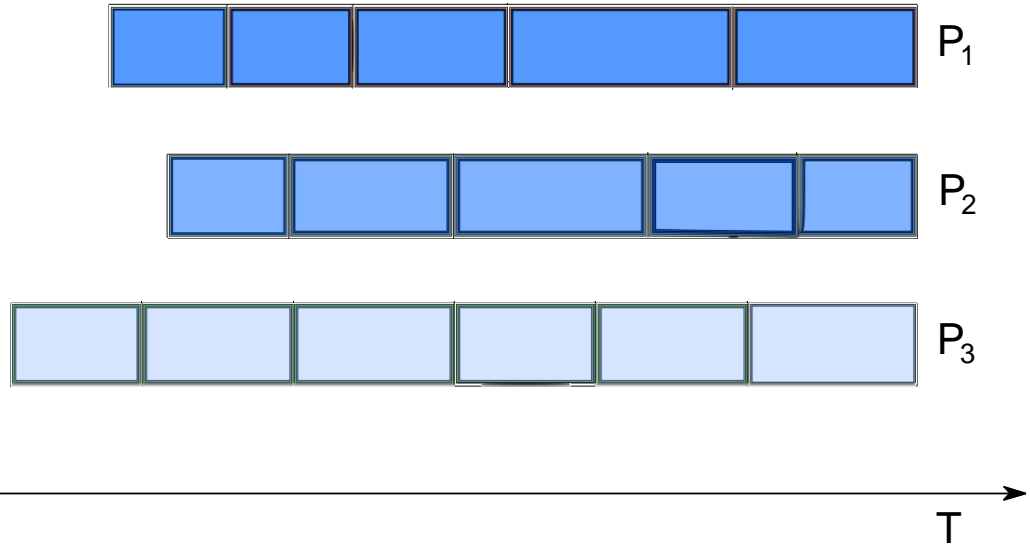


Figure 35. WFQ priority queuing.

$$DW = t / (1 + P) \quad (2.1.2.2)$$

DW - data weight;

T - the different between the current time and the timestamp of the instance when the packet is inserted to the TSCH buffer;

P - the priority of the packet.

The packet with the smallest finish weight is chosen for transmitting. WFQ mechanism guaranties that each data regardless of priority gets access to the transmitting and sends proportional to the assigned data weights. The algorithm of the WFQ mechanism work is shown in Figure 36.

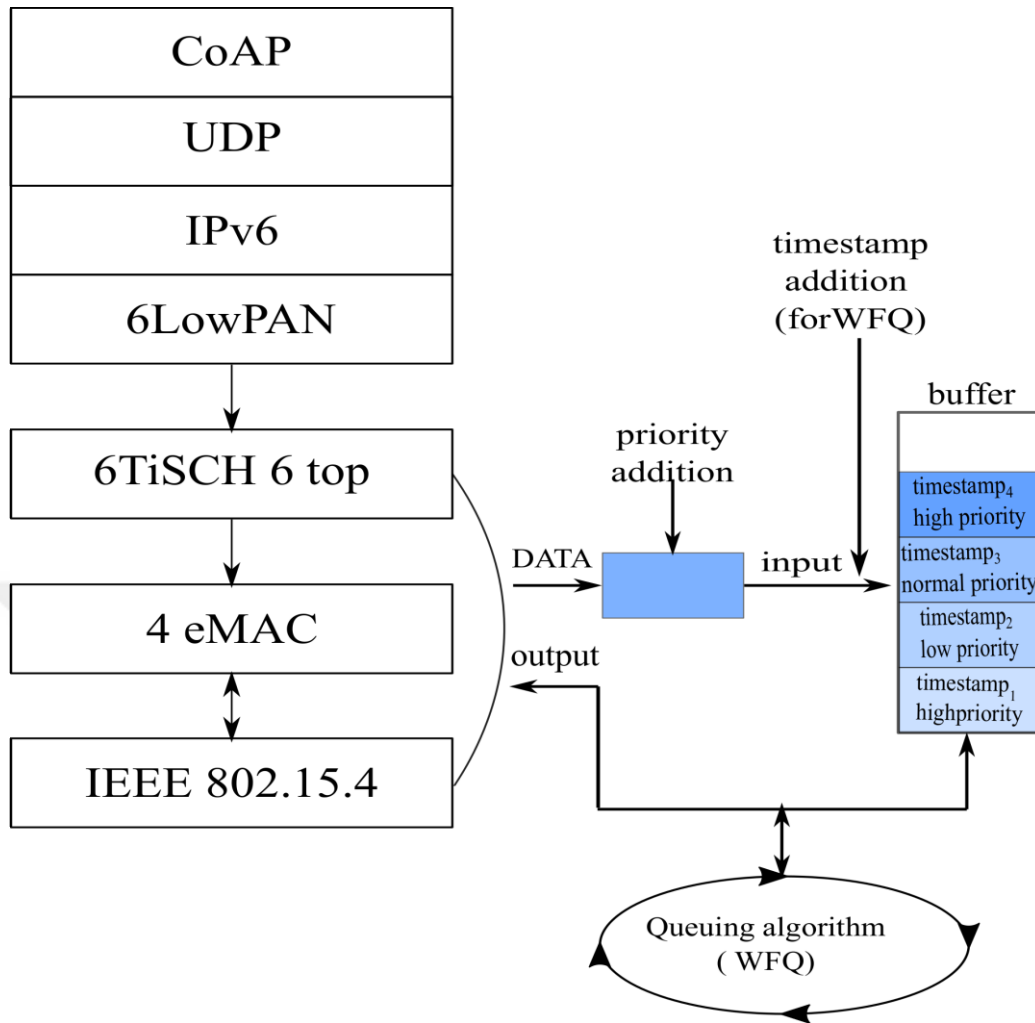


Figure 36. Scheme of using WFQ QoS mechanism

#### 4.2.2. Fairness

Fairness is one of the most important challenges of wireless sensor networks. Each node and each application need to be assigned with the right bandwidth, as well as QoS. The QoS requirements of the nodes need to be fairly satisfied. When the traffic loads of the nodes are similar, energy consumption also might turn out to be similar. Also, the link quality, cost, throughput and other performance aspects might be taken into account when the fairness of the node is considered. These issues show the significance and diversity of fairness issues in wireless networks.

In Wireless Networking area, the consequence of an unfair resource allocation among different node may cause resource starvation, resource wastage or redundant allocation, so we can overcome this problem to attribute resource sharing or allocation

fairly. It is not simple to make a common definition of fairness since it is subjective. Most of the fairness research is around assigning a value to the shared resource. The definition of fairness is influenced by the value assigned to the resources by the designer of the system or by the nodes of the system.

### 4.2.3. Jain's Index

One of the fairness indexes is Jain's index. It was first proposed by Rajendra K. Jain in [62]. In his study he proposed four desired properties of the fairness index:

- Independent of population size is. It is mean that no matter how population size is, it needs to be independent and the index should be scalable with the number of users.
- Independent of scale and metric: It is meant that no matter which measures or metrics are used the index has not changed and it shows that the variance also can play the role of a fairness index too.
- Boundedness: The index can be between 0 and 1 and should be finite.
- Continuity: It is mean that the index function must be continuous on allotments and must have the capability to measure different allotments.

The Jain's Index is given as;

$$f(x) = \frac{[\sum_{i=1}^n x_i]^2}{n \sum_{i=1}^n x_i^2} \quad (3)$$

where  $0 \leq f(x) \leq 1$  is called as Jain's index and is one of the widely studied fairness measures in the literature [63]. In the equation,  $x_i$  represents the data points and  $n$  represents the population size. For having 100% fairness, all users should get the same amount. Jain's index always lies between 0 and 1. If the system is 100% fair the fairness index is 1.

Specifically, the value of Jain's index can be interpreted as the fraction of received data. Therefore, a bigger Jain's index matches to a fairer data transmission. Especially, the fairest case shows up when all the data with different priority get the same weight, which makes the Jain's index equal to one another, so we utilize here Jain's index to evaluate fairness in the network.

## 5. EXPERIMENTAL RESULTS

The implemented queuing mechanisms are analyzed using the network setup described in System Model section. Each of the queuing mechanisms performance is tested via 5 experiments with different random seed values. This section summarizes the averaged performance results of the experiments for each queuing mechanisms with their respective standard deviation values. Standard deviation values are calculated using the equation given in (4) and given in Table 5, Table 6 and Table 7.

$$STD = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{cp})^2}{n-1}} \quad (4)$$

STD - Standard deviation value,

n - number of observations

x - observed values of the sample items.

As it can be seen from the result tables, the average number of sent and dropped packets for all three algorithms is approximately the same. This is expected as the 6TiSCH protocol was configured to service the incoming traffic with only 3 TSCH slots which is barely adequate for the generated traffic. Hence, some of the generated data cannot be serviced by the radio. The relative receive performances of all priority classes amount to around 72% for all of the evaluated QoS queuing mechanisms. However, if the results for each queuing mechanism are analyzed, it can be seen that different queuing mechanisms transmit different amount of data packets belonging to different priority classes.

When GPQ mechanism is used as the QoS mechanism in 6TiSCH node, it is observed that highest priority data packets are given precedence as expected. The GPQ average amount packets are shown in Table 5. In this case, 98% of the data packets belonging to high priority class are successfully transmitted over the network. But, the low priority data packets mostly dropped from the transmit queue resulting in a poor quality of service for this type of traffic.

Table 5. GPQ average amount packets

GPQ				
	Send	Receive	%	STD
High priority	1928	1894	98%	10.1
Normal priority	1910	1639	86%	4.47
Low priority	2142	769	36%	4.9
Total	5980	4302	72%	-

On the other hand, the FIFO queuing represents a fair treatment of all the data packets without any distinction. The FIFO average amount packets are shown in Table 6. In this case, all the data packets are treated equally and there is almost no distinction between the priority classes. Since the packets belonging to each priority class are generated randomly with equal probabilities, the results show a similar delivery ratio for all the priority classes. Of course, this represents almost 100% fairness as given in Table 8. Although, a high fairness value is desired for a queuing algorithm, it is desirable to give precedence to high priority class traffic as compared to other priority classes in 6TiSCH protocol. This observation is especially correct for high priority traffic class which generally is made up of protocol messages for maintaining the network.

Table 6. FIFO average amount packets

FIFO				
	Send	Receive	%	STD
High priority	1962	1290	66%	4
Normal priority	1937	1424	74%	2,77
Low priority	2067	1591	76 %	4,35
Total	5966	4305	72%	-

WFQ algorithm treats each traffic class according to the weights calculated using the equation given in Eq. (2.1.2.2). The WFQ average amount packets are shown in Table 7. In

this case, although the highest priority traffic is given precedence as compared to the other priority classes, the weights are used to make sure that the high priority traffic does not throttle the traffic with low priority. As it can be seen from Table 7, the highest priority class traffic achieves a delivery ratio of 88% followed by Normal priority and low priority traffic at 70% and 60% delivery ratios respectively.

Table 7. WFQ average amount packets

WFQ				
	Send	Receive	%	STD
High priority	1954	1710	88%	1.64
Normal priority	1915	1342	70%	1.8
Low priority	2108	1258	60%	2.3
Total	5977	4310	72%	-

This represents a fairer result as compared GPQ approach at the same time maintaining precedence for the high priority traffic. Furthermore, its results show a fairness of 97.5% according to Table 8 and quite close to FIFO queuing.

Table 8. Jain's index

	FIFO	GPQ	WFQ
High p.	31%	45%	40%
Normal p.	34%	39%	32%
Low p.	35%	16%	28%
Jain's Fairness Index	0.99	0.881	0.975

These results on investigating QoS routines for 6TiSCH protocols show that it is possible to achieve fairness while maintaining a higher delivery ratio for high priority traffic as compared to other traffic priority classes. It is shown through simulations that using WFQ mechanism can deliver the important protocol data by giving precedence to

high priority for maintaining network stability. At the same time, the other priority classes are also serviced according to the chosen weights in the queuing mechanism.



## **6. CONCLUSION AND FUTURE WORK**

In conclusion, we have implemented three different QoS algorithms for IETF 6TiSCH protocol. Our goal is to show the preliminary performance results for such well known QoS mechanisms for 6TiSCH protocol. As indicated by the experimental results, it is possible to achieve fairness among different traffic priority classes while maintaining the precedence between them using WFQ mechanism. All these results represent a constant flow of randomly generated data traffic in 6TiSCH network. It is of interest to analyze bursty traffic and its QoS as future work. Furthermore, it is also of interest to analyze the impact of having deeper queues in TSCH MAC to maintain the QoS of the network in such bursty traffic patterns. As a follow up study, we are aiming to implement additional QoS mechanisms and deploy them to real life networks to analyze their impact on 6TiSCH based IoT networks.



## 7. REFERENCES

1. Xia, F., Yang, L. T., Wang, L., and Vinel, A., Internet of things, International Journal of Communication Systems, 25, 9 (2012), 11-101.
2. Dohler, M., Wireless sensor networks: the biggest cross-community design exercise to-date, Recent Patents on Computer Science, 1, 1, 1,12 (2008), 9–25.
3. Montenegro, G., Kushalnagar, N., Hui, J., Culler, D. and Corp Arch Rock, IETF Tools: IPv6 over IEEE 802.15.4.  
<https://tools.ietf.org/html/rfc4944> 1 September, 2007
4. Shelby, Z. and Bormann, C., 6LoWPAN: The wireless embedded Internet, 43. John Wiley & Sons, 2011.
5. Accettura, N., Grieco, L. A., Boggia, G. and Camarda, P., Performance analysis of the rpl routing protocol in Mechatronics, 2011 IEEE International Conference, May 2011, Washington, IEEE proceedings book, 767–772.
6. Kovatsch, M. , Duquennoy, S. , and Dunkels, A. , A low-power coap for contiki, in Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference, May 2011, Valencia, IEEE Press, 855–860.
7. Internet Engineering Task Force (IETF), Using ieee 802.15. 4e time-slotted channel hopping in the internet of things (iot): Problem statement, Luxembourg, 2015.
8. De Guglielmo, D., Brienza, S., and Anastasi, G., Ieee 802.15. 4e: A survey, Computer Communications, 88, 15,8 (2016), 1–24.
9. Arisha, K., Youssef, M., and Younis, M., Energy-aware tdma-based mac for sensor networks, System-level Power Optimization for Wireless Multimedia Communication Conference, April 2002, Baltimore, Proceedings book, 21–40.
10. Thubert, P., Watteyne, T., Palattella, M. R. , Vilajosana, X. , and Wang, Q. , Ietf 6tsch: Combining ipv6 connectivity with industrial performance, 2013 Seventh International Conference, July 2013, Massachusetts, AAAI Press, 541–546.
11. Dujovne, D., Watteyne, T., Vilajosana, X. , and Thubert, P. , 6tisch: deterministic ip-enabled industrial internet (of things), IEEE Communications Magazine, 52, 12 (2014), 36–41.
12. Görmüs, S. and Yavuz, A. F., A protocol for internet of things: Ietf 6tisch, in Signal Processing and Communications Applications Conference (SIU) - 2017, IEEE Press, May 2017, 1–4

13. Hsu, V., Kahn, J. M. , and Pister, K. S., Wireless communications for smart dust. Electronics Research Laboratory, College of Engineering, University of California, <https://www2.eecs.berkeley.edu/Pubs/TechRpts.html> 1 January 1998.
14. Lewis, F. L. et al., Wireless sensor networks, Smart environments: technologies, protocols, and applications, January 2004, New York, Wiley Online Library Press, 11–46.
15. IEEE Standart for Local and metropolitan area network: Part 15.4: Low-rate wireless personal area networks (lr-wpans), Networking and Internet Architecture, New York, 2011.
16. Callaway, E., Gorday, P., Hester, L., Gutierrez, J. A., Naeve, M., Heile, B., and Bahl, V., Home networking with ieee 802.15. 4: a developing standard for low-rate wireless personal area networks, IEEE Communications magazine, 40, 8 (2002), 70–77.
17. Howitt, I. and Gutierrez, J. A., Ieee 802.15. 4 low rate-wireless personal area network coexistence issues, Wireless Communications and Networking Conference, May 2003, New York, IEEE Press, 1481–1486.
18. Adams, J. T., An introduction to ieee std 802.15. 4, Aerospace Conference, March 2006, IEEE Press, 8–16.
19. Wang, Q., Liu, X., Chen, W., Sha, L. , and Caccamo, M. , Building robust wireless lan for industrial control with the dsss-cdma cell phone network paradigm, IEEE Transactions on Mobile Computing Conference, June 2007, New York, IEEE Press 706–719.
20. Park, T. R., Kim, T. H. , Choi, J. Y., Choi, S. , and Kwon, W. H. , Throughput and energy consumption analysis of ieee 802.15. 4 slotted csma/ca, Institotion of Engineering and Technology Conference, September 2005, Vancouver, IET Press 1017–1019.
21. Koubaa, A , Alves, M. , and Tovar, E., A comprehensive simulation study of slotted csma/ca for ieee 802.15. 4 wireless sensor networks, 5th IEEE International Workshop on Factory Communication Systems Conference, 2006, New York, IEEE Press 183–192.
22. Akyildiz, I. F. and Vuran, M. C., Wireless sensor networks, 4. John Wiley & Sons, San Diego, 2010.
23. T. I. of Electrical and I. Electronics Engineers, Ieee standard for low-rate wireless networks, IEEE, New York, 2015.
24. Watteyne, T., Handziski, V., Vilajosana, X. , Duquennoy, S., Hahm, O., Baccelli, E. and Wolisz, A., Industrial wireless ip-based cyber-physical systems, Proceedings of the IEEE, 104, May 2016, New York, IEEE Press, 1025–1038.

25. Duquennoy, S., Elsts, A., Nahas, B., and Oikonomou G., TSCH and 6tisch for contiki: Challenges, design and evaluation, HAL archives-ouvertes, September 2017, HAL Press, 1-9.
26. Watteyne, T., Palattella, M., and Grieco, L., Using ieee802.15.4 e tsch in an iot context: Overview, problem statement and goals draftietf-6tisch-tsch-05 <http://tools.ietf.org/html/draft-ietf-6tisch-tsch.html> 5 Jan. 2015
27. CISCO, Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, New York, Sept. 2011.
28. Olsson, J., 6lowpan demystified, Texas Instruments Magazine, (2014), 1-13.
29. Vasseur, J.-P., Kim, M., Pister, K., Dejean, N., and Barthel, D., Routing metrics used for path calculation in low-power and lossy networks. <https://tools.ietf.org/html/rfc6551> 1 March 2012.
30. Dujovne, M. P. D., Grieco, LA. and Accettura, N., 6tisch 6top scheduling function zero (sf0), Internet Engineering Task Force, vo 8. (2016), 1-22.
31. Thomas, L., Shalu, R., Daniel, J. J., Anand, S., and Hegde, M., 6tisch operation sublayer (6top) implementation on contiki os, Communication Systems and Networks (COMSNETS), 2017 9th International Conference, January 2017, Bangalore, IEEE Press, 423-424.
32. Mukhopadhyay, S. C. and Suryadevara, N., Internet of things: Challenges and opportunities, Internet of Things, (2014), 1–17.
33. Muthanna, A., Prokopyev, A., Paramonov, A., and Koucheryavy, A., Comparison of protocols for ubiquitous wireless sensor network, in Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2014 6th International Congress, May 2014, Stockholm, GlobalEvents Press, 334–337.
34. Shelby, Z., Hartke, K., and Bormann, C., The constrained application protocol (coap). <https://tools.ietf.org/html/draft-ietf-core-coap-13> 9 June 2013.
35. Adryan, B., Obermaier, D., and Fremantle, P., The Technical Foundations of IoT. Artech House, Boston, 2017.
36. Shelby, Z., Hartke, K., Bormann, C., and Frank, B., Rfc 7252, Constrained Application Protocol (CoAP). <http://tools.ietf.org/html/rfc7252> 6 August 2014.
37. Rahman, A. and Dijk, E., Rfc 7390 Group communication for coap, Internet Engineering Task Force (IETF). <https://tools.ietf.org/html/rfc7390> 1 October 2014

38. Karagiannis, V. , Chatzimisios, P. , Vazquez-Gallego, F. , and Alonso-Zarate, J. , A survey on application layer protocols for the internet of things, Transaction on IoT and Cloud Computing, 3, 1 (2015), 11–17.
39. Shelby, Z., Constrained application protocol (coap) draft-ietf-core-coap-03. <https://tools.ietf.org/html/draft-ietf-core-coap-13> 6 December 2012.
40. Vilajosana, X., Pister, K., and Watteyne, T., Minimal ipv6 over the tsch mode of ieee 802.15. 4e (6tisch) configuration. <https://tools.ietf.org/html/rfc8180> 1 May 2017
41. Puri, T., Challa, R. K., and Sehgal, N. K., Energy efficient qos aware mac layer time slot allocation scheme for wbasn, in Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference, June 2015, London, ICACCI Press 966–972.
42. Ezdiani, S. , Acharyya, I. S., Sivakumar, S. , and Al-Anbuky, A. , An iot environment for wsn adaptive qos, in Data Science and Data Intensive Systems (DSDIS), 2015 IEEE International Conference, July 2015, Sydney, IEEE Press,586–593.
43. Karim, L., Nasser, N. , Taleb, T. , and Alqallaf, A. , An efficient priority packet scheduling algorithm for wireless sensor network, in Communications (ICC), 2012 IEEE International Conference, June 2012, Ottawa, IEEE Press , 334–338.
44. Al-Anbagi, I. , Erol-Kantarci, M. , and Mouftah, H. T. , An adaptive qos scheme for wsn-based smart grid monitoring, in Communications Workshops (ICC), 2013 IEEE International Conference, June 2013, Budapest, IEEE Press, 1046–1051.
45. Boughanmi, N., Song, Y.-Q. , and Rondeau, E., Priority and adaptive qos mechanism for wireless networked control systems using ieee 802.15. 4 in IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society, November 2010, Glendale, IEEE Press, 2134–2141.
46. Sobrinho, J. L. and Krishnakumar, A. S., Quality-of-service in ad hoc carrier sense multiple access wireless networks, IEEE Journal on selected areas in communications, 17, 8, August 1999, New York, IEEE Press, 1353–1368.
47. <https://datatracker.ietf.org/wg/6tisch/about/>, Ipv6 over the tsch mode of ieee 802.15.4e (6tisch). 20 August 2017.
48. Balen, J., Zagar, D., and Martinovic, G., Quality of service in wireless sensor networks: a survey and related patents, Recent Patents on Computer Science, 4, 3 (2011), 188–202.
49. Wang, Y., Liu, X., and Yin, J. , Requirements of quality of service in wireless sensor network, in Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies,

2006. ICN/ICONS/MCL 2006. International Conference, May 2006, Morne, IEEE Press, 116–116.
50. Vasseur, J.-P. and Dunkels, A. , Interconnecting smart objects with ip: The next internet, Burlington, 2010.
  51. Yu, C.-H., Doppler, K. , Ribeiro, C. B. , and Tirkkonen O. , Resource sharing optimization for device-to-device communication underlying cellular networks, IEEE Transactions on Wireless communications, December 2011, New York , IEEE Press, 2752–2763.
  52. Siddha, V. , Ishiguro, K. , and Hernandez, G. A., Hardware abstraction layer, US Patent 8, New York, 2012.
  53. Cc2538 powerful wireless microcontroller system-on-chip for 2.4-ghz iee 802.15. 4, 6lowpan, and zigbee applications, CC2538 datasheet.  
<http://www.ti.com/lit/ds/symlink/cc2538.html> 1 April 2015.
  54. Cc2650 simplelink multistandard wireless mcu, CC2650,  
<http://www.ti.com/lit/ds/symlink/cc2650.html> 1 July 2016.
  55. Eriksson, J. , Dunkels, A. , Finne, N. , Osterlind, F. , and Voigt, T. , Mpsim—an extensible simulator for msp430-equipped sensor boards, Poster/Demo session, 118, January 2007, The Netherlands, EWSN Press, 1-22.
  56. Dunkels, A., Gronvall, B., and Voigt, T., Contiki-a lightweight and flexible operating system for tiny networked sensors, in Local Computer Networks, 2004. 29th Annual IEEE International Conference, December 2004, Tampa, IEEE Press, 455–462.
  57. Stehlik, M., Comparison of simulators for wireless sensor networks. PhD thesis, Masarykova univerzita, Fakulta informatiky, Masaryk, 2011.
  58. Osterlind, F., Dunkels, A. , Eriksson, J. , Finne, N. , and Voigt, T. , Cross-level sensor network simulation with cooja, in Local computer networks, proceedings 2006 31st IEEE Conference, November 2006, Tampa, IEEE Press, 641–648.
  59. Gaspar, P. D., Santo, A. E., and Ribeiro, B., Msp430 microcontrollers essentials-a new approach for the embedded systems courses: Part 1-overview and tools, in Education and Research Conference (EDERC), 2010 4th European, December 2010, Nice, IEEE Press, 66–70.
  60. Cobham, A., Priority assignment in waiting line problems, Journal of the Operations Research Society of America, 2, 1 (1954), 70–76.
  61. Semeria, C., Supporting differentiated service classes: queue scheduling disciplines, Juniper networks, (2001), 11–14.
  62. Jain, R., Chiu, D.-M. , and Hawe, W. R., A quantitative measure of fairness and discrimination for resource allocation in shared computer system, 38. Eastern

Research Laboratory, Digital Equipment Corporation Hudson, September 1984, Hudson, DECH Press, 1-38

63. Shi, H. , Prasad, R. V., Onur, E. , and Niemegeers I. , Fairness in wireless networks: Issues, measures and challenges, IEEE Communications Surveys & Tutorials, 16, 1 (2014), 5–24.



## **CURRICULUM VIATE**

Diliara Ibrahimova was born in 14<sup>th</sup> of March, 1993 in Crimea, Ukraine. In 2010 graduated from the school №6 in Simferopol. She has a bachelor degree in Computer Science from Crimea, in 2014, Ukraine. Diliara Ibrahimova has been studying at Karadeniz Technical University since 2014 for a master degree in Computer Engineering.

Ibrahimova speaks English, Turkish, Russian and Ukrainian languages.

### **Publication:**

Ibrahimova, D., Görmüş S., “QoS for IETF 6TiSCH”, SIU (Signal Processing and Communication Applications Conference), Izmir, Turkey, May 2018.