ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Rıdvan SÖYÜ

PERFORMANCE ANALYSIS OF CONGESTION CONTROL PROTOCOLS IN INTERNET OF THINGS NETWORKS

DEPARTMENT OF COMPUTER ENGINEERING

ADANA-2020

ABSTRACT

MSc THESIS

PERFORMANCE ANALYSIS OF CONGESTION CONTROL PROTOCOLS IN INTERNET OF THINGS NETWORKS

Rıdvan SÖYÜ

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Prof. Dr. Selma Ayşe ÖZEL
Co-Supervisor : Asst. Prof. Dr. Alper Kamil DEMİR
Year: 2020, Pages: 77
Juries : Prof. Dr. Selma Ayşe ÖZEL
: Assoc. Prof. Dr. Umut ORHAN
: Asst. Prof. Dr. Onur ÜLGEN

The extensive flow of information in the Internet of Things (IoT) networks greatly increases the importance of congestion control. Therefore, in the academic literature, there are many studies about how to solve congestion control and on which layer to handle it. In this thesis, it is examined how the protocols written for congestion control on the application layer (CoAP, CoCoA, etc.) perform experimentally by running various combinations of several sub-layers with different protocols in IoT networks. In addition, the relationship between the application layer protocols and the objective functions of the network layer protocol RPL is examined. As a result, it has been proved that OF0 performs better than MRHOF, and CoCoA Strong was observed as the best performing congestion control mechanism. The most efficient results were obtained in scenarios where OF0 was used with ContikiMAC.

Keywords: Congestion Control, Congestion Control in IoT, RPL Objective Functions, Network performance, IoT Network Stack

ÖZ

YÜKSEK LİSANS TEZİ

NESNELERİN İNTERNETİ AĞLARINDA TIKANIKLIK KONTROL YÖNTEMLERİNİN PERFORMANS ANALİZİ

Rıdvan SÖYÜ

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Prof. Dr. Selma Ayşe ÖZEL
Eş Danışman : Dr. Öğr. Üyesi Alper Kamil DEMİR
Year: 2020, Sayfa: 77
Jüri : Prof. Dr. Selma Ayşe ÖZEL
: Doç. Dr. Umut ORHAN
: Dr. Öğr. Üyesi Onur ÜLGEN

Nesnelerin interneti ağlarındaki yoğun bilgi akışı, tıkanıklık kontrolünün önemini oldukça artırmaktadır. Bu sebeple akademik literatürde tıkanıklık kontrolünün hangi ağ katmanında ve nasıl çözüleceğine dair birçok çalışma mevcuttur. Bu tez çalışmasında uygulama katmanında tıkanıklık kontrolü için yazılan protokollerin (CoAP, Default CC vs.), farklı katmanlardaki farklı protokollerle olan çok sayıdaki kombinasyonunun performans analizi yapılmıştır. Bunun yanı sıra uygulama katmanı protokollerinin, ağ katmanı protokolü olan RPL in farklı amaç fonksiyonları ile olan ilişkisi incelenmiştir. Tüm deneyler Cooja simülatörü üzerinde, Contiki işletim sisteminin kurulu olduğu sanal düğümler kullanılarak yapılmıştır. Alınan sonuçlar neticesinde OF0, MRHOF'a göre daha iyi performans göstermiş, tıkanıklık kontrol mekanizmalarından da CoCoA Strong en iyi performansı veren algoritma olarak gözlenmiştir. OF0'ın ContikiMAC ile beraber kullanıldığı senaryolarda ise en verimli sonuçlar elde edilmiştir.

Anahtar Kelimeler: Tıkanıklık Kontrolü, Nesnelerin İnterneti Ağlarında Tıkanıklık Kontrolü, RPL Amaç Fonksiyonları, Ağ Performansı, Nesnelerin İnterneti Ağ Yığını

EXTENDED ABSTRACT

After the development of the computer and the Internet, one recent important technology that can be considered as the third wave innovation is deemed to be the “Internet of Things”. In IoT networks, “thing” can be anything (e.g., human, device, etc.) that can be connected to the Internet, and the IoT is the system that allows these objects to be connected to each other at any place and time even though they are not identical. In the IoT networks, sensors are the devices that have the most important role; and since they are continuously sharing information, these networks require huge capacities of storage and computational power.

Devices used in the IoT networks are often called as “restricted devices”. In order to maximize optimization while minimizing energy consumption, the features such as data storage area, processor speed, and memory are quite low compared to devices used in daily life. Considering the continuity of data sharing and the size of the network, there are still many problems for the Internet of Things that have not been solved and need to be investigated yet. One of the most important issues is congestion because it is almost inevitable to have congestion in networks where data roaming is very high.

Congestion in networks is one of the main problems that has always been in question and researched since the existence of computer networks. Protocols such as RESTful HTTP, XMPP (Extensible Messaging and Presence Protocol) and MQTT (Message Queuing Telemetry Transport), which run on the application layer in the Internet of Things networks, use the transport layer protocol TCP for reliability and congestion control. TCP must meet the requirements for data transfer in such networks and is, therefore, a very important protocol for the IoT. However, it is not enough to use existing solutions and protocols for IoT networks. Unlike other networks, these kinds of networks are set up from devices that have all kinds of technical features and requirements. Therefore, when considering

congestion control in IoT, there must be a mechanism that will be effective for all kinds of devices.

Congestion is directly proportional to the number of devices in the networks. The IoT is a network where different devices can communicate with each other. Therefore, it is not possible to meet the requirements of these networks with the existing TCP protocol, a modification or a different protocol should be developed in this protocol for issues such as initiation of connections in the networks, congestion control, transmission rate adjustment, and connection security.

In this thesis, the effects of the protocols which are used in all layers in the Internet of Things network on congestion are investigated, and the performance analysis of each combination of these protocols are done. In these combinations, 4 application layer protocols, 2 OF in the network layer and related to the RPL protocol, 1 transport layer protocol, 2 RDC protocols, and 1 MAC layer protocols are used. In the physical layer, PDR values are chosen from 3 different values and experiments are designed accordingly. In addition, 3 and 9 clients are used as the number of clients. The number of scenarios obtained using these numbers is determined as 98. Each scenario is repeated at least 3 times to test its accuracy.

All simulations are done using the Contiki operating system and the Cooja network simulation environment. The fact that the Contiki operating system has an application such as Cooja and that it has been developed specifically for the Internet of Things affected the factor in this choice. Cooja simulation environment, on the other hand, is a program used in most network simulations and researches due to its virtual devices and flexibility, which played an important role in choosing the Cooja simulation program. Californium is preferred because it offers different application layer protocols on the client-side. Since the simulation environment is in a virtual environment, the Tunslip program that comes with the Contiki operating system is used to communicate in real-time with Californium.

While examining the results obtained from the applied scenarios, the congestion control mechanisms used in the application layer, and the objective functions in the network layer are taken into consideration. The effects of the combinations of application layer with these objective functions on congestion have been investigated. At the end of these examinations, it has been observed that the objective functions affect the congestion together with congestion control mechanisms. In fact, in scenarios where OF0 is used, more positive results are obtained compared to scenarios with MRHOF. When OF0 is used, it has been observed that both the number of packets received per unit time and the average delay times are lower. On the other hand, CoCoA Strong, one of the congestion control mechanisms, performs better than all other mechanisms. Another result is that the ContikiMAC protocol achieves better results than the NullRDC protocol.

According to the results obtained from this study, not only the protocol used in the network stack and the topology of the network but also the objective functions in the network layer have an important effect on the congestion of IoT networks. This shows the importance of each protocol and algorithm in the network performance of an IoT. These results will guide the researches that will be done in the future.

GENİŞLETİLMİŞ ÖZET

Nesnelerin interneti, bilgisayar ve internetin geliştirilmesinden sonra artık üçüncü dalga yenilik olarak kabul edilmektedir. Nesnelerin interneti ağlarında "nesne", internete bağlanabilecek herhangi bir şey (insan, cihaz vs.) olarak nitelendirilir ve nesnelerin interneti de bu nesnelerin birbirleriyle birebir aynı özelliklerde olmasa bile herhangi bir yerde ve zamanda birbirlerine bağlı olmasını sağlayan sistemdir. Nesnelerin interneti ağlarında sensörler en önemli rolü alan cihazlardır ve sürekli bir bilgi paylaşımı olduğundan bu ağlar çok büyük kapasitelerde depolama alanına ve hesaplama gücüne ihtiyaç duyarlar.

Nesnelerin interneti ağlarında kullanılan cihazlar "kısıtlı cihazlar" olarak adlandırılırlar. Enerji tüketimini minimuma indirirken optimizasyonu da maksimuma getirmek için bu cihazların veri depolama kapasitesi, işlemci hızları ve bellek gibi özellikleri günlük hayatta kullanılan cihazlara göre oldukça düşüktür . Veri paylaşımının sürekliliği ve ağın büyüklüğü de göz önüne alındığında nesnelerin interneti için henüz çözülmemiş ve hala araştırılması gereken birçok problem mevcuttur. Bu problemlerden en önemlilerinden biri ise tıkanıklıktır çünkü veri dolaşımının çok yüksek olduğu ağlarda tıkanıklığın olması nerdeyse kaçınılmazdır.

Ağlardaki tıkanıklık, bilgisayar ağları var olduğundan bu yana her zaman söz konusu olan ve üzerine araştırma yapılan temel problemlerden biridir. Nesnelerin interneti ağlarında uygulama katmanında çalışan RESTful HTTP, XMPP ve MQTT gibi protokoller, güvenilirlik ve tıkanıklık kontrolü için taşıma katmanı olan TCP'yi kullanırlar. TCP'nin bu tip ağlarda veri aktarımındaki gereksinimleri karşılaması gerekir ve bu yüzden de nesnelerin interneti için oldukça önemli bir protokoldür. Ancak nesnelerin interneti ağlarında var olan çözümlerin ve protokollerin kullanılması yeterli değildir. Çünkü bu ağlar diğer ağlardan farklı olarak her türlü teknik özelliğe ve gereksinime sahip cihazlardan

kurulmaktadır. Bu yüzden nesnelerin interneti ağlarında tıkanıklık kontrolü göz önüne alınırken her türlü cihaz için etkili olacak bir mekanizmanın olması gerekir.

Ağlardaki cihaz sayısı ile tıkanıklık doğru orantılıdır. Nesnelerin interneti ise farklı cihazların birbirleri ile haberleşebildiği ağlardır. Bu yüzden bu ağların gereksinimlerini karşılamak mevcut TCP protokolü ile mümkün olmayıp, ağlardaki bağlantıların başlatılması, tıkanıklık kontrolü, iletimin oranının ayarlanması ve bağlantı güvenliği gibi konular için bu protokolde bir modifikasyon yapılması ya da farklı bir protokolün geliştirilmesi gerekmektedir.

Bu tez çalışmasında nesnelerin interneti ağlarında tüm katmanlarda kullanılan protokollerin tıkanıklığa etkileri ve bu protokollerin her bir kombinasyonunun performans analizi yapılmıştır. Bu kombinasyonlarda 4 adet uygulama katmanı protokolü, ağ katmanında bulunan ve RPL protokolü ile ilişkili 2 adet amaç fonksiyonu, 1 adet taşıma katmanı protokolü, 2 adet RDC protokolü ve 1 adet MAC katmanı protokolü kullanılmıştır. Fiziksel katmanda ise PDR değerleri 3 farklı değer üzerinden seçilmiş ve deneyler buna göre tasarlanmıştır. Ayrıca istemci sayısı olarak da 3 ve 9 belirlenmiştir. Bu sayılar doğrultusunda elde edilen senaryo sayısı 98 olarak belirlenmiştir. Doğruluğunu test etmek açısından her senaryo en az 3 kez tekrarlanmıştır.

Tüm simülasyonlar Contiki işletim sistemi ve Cooja ağ simülasyon ortamı kullanılarak yapılmıştır. Contiki işletim sisteminin Cooja gibi bir uygulamaya sahip olması ve nesnelerin interneti için özel olarak geliştirilmiş olması bu seçimde etken olmuştur. Cooja simülasyon ortamı ise sahip olduğu sanal cihazlardan ve sunduğu esneklikten dolayı çoğu ağ simülasyonunda ve araştırmalarında kullanılan bir programdır ve bu da Cooja simülasyon programının seçilmesinde önemli rol oynamıştır. İstemci tarafında farklı uygulama katmanı protokolleri sunması nedeniyle Californium tercih edilmiştir. Simülasyon ortamı sanal bir ortamda bulunduğu için Californium ile gerçek zamanlı haberleşebilmesi için Contiki işletim sistemi ile beraber gelen Tunslip programı kullanılmıştır.

Uygulanan senaryolardan elde edilen sonuçlar incelenirken uygulama katmanında bulunan tıkanıklık kontrol mekanizmaları ve ağ katmanında kullanılan amaç fonksiyonları dikkate alınmıştır. Uygulama katmanının bu amaç fonksiyonları ile kombinasyonlarının tıkanıklığa etkileri incelenmiştir. Bu incelemeler sonunda amaç fonksiyonlarının gerçekten de tıkanıklık kontrol mekanizmaları ile beraber tıkanıklığa etki ettiği görülmüştür. Öyle ki OF0 kullanılan senaryolarda MRHOF kullanılan senaryolara göre daha olumlu sonuçlar alınmıştır. OF0 kullanıldığı zaman hem birim zamanda alınan paket sayısının daha fazla olduğu hem de ortalama gecikme sürelerinin daha düşük olduğu gözlenmiştir. Diğer yandan tıkanıklık kontrol mekanizmalarından CoCoA Strong ise diğer tüm mekanizmalara göre daha iyi performans sergilemiştir. Bir diğer elde edilen sonuç ise ContikiMAC protokolünün NullRDC protokolünden daha iyi sonuçlar elde etmiş olmasıdır.

Bu çalışmadan elde edilen bilgilere göre bir nesnelerin interneti ağında tıkanıklığa sadece katmanlarda kullanılan protokoller ve ağın topolojisi değil, aynı zamanda ağ katmanındaki amaç fonksiyonları da önemli derecede etki etmektedir. Bu da bir nesnelerin interneti ağında her protokol ve algoritmanın ağın performansında ne derece önemli olduğunu göstermektedir. Alınan bu sonuçlar, bu çalışmadan sonra yapılacak olan araştırmalara yön verecektir.

ACKNOWLEDGEMENT

I would like to present my gratitude to my supervisors Prof. Dr. Selma Ayşe ÖZEL and Asst. Prof. Dr. Alper Kamil DEMİR for sharing their respectable knowledge, guidance and patience for the research and writing of this thesis.

I would like to thank members of the MSc thesis jury, Assoc. Prof. Dr. Umut ORHAN and Asst. Prof. Dr. Onur ÜLGEN, for their suggestions and corrections.

My special thanks for my family, and my friend Aslı Nazlı ŞİRE and R. A. Ömür SALTIK for their endless support and encouragements for my life and academic career.

CONTENTS **PAGE**

LIST OF FIGURES **PAGE**

LIST OF ABBREVIATIONS

6LoWPAN	: IPv6 over Low-Power Wireless Personal Area Networks
6TiSCH	: IPv6 over the TSCH mode of IEEE 802.15.4e
ACK	: Acknowledgement
AIMD	: Additive Increase Multiplicative Decrease
BLE	: Bluetooth Low Energy
CADC	: Congestion Aware Duty Cycle MAC
CSMA	: Carrier Sense Multiple Access
CTS	: Clear to Send
CoAP	: Constrained Application Protocol
CoCoA	: CoAP Simple Congestion Control/Advanced
CODA	: Congestion Detection and Avoidance
DAG	: Directed Acyclic Graph
DODAG	: Destination-Oriented Directed Acyclic Graph
DCCC6	: Duty Cycle-Aware Congestion Control for 6LoWPAN Networks
ETX	: Expected Transmission Count
ESRT	: Event-To-Sink Reliable Transport
GCC	: GNU Compiler Collection
GTCC	: Game Theory-Based Congestion Control Protocol
GTCCF	: Game Theory-Based Congestion Control Framework
HTTP	: Hyper-Text Transfer Protocol
IEEE	: Institute of Electrical and Electronics Engineers
IETF	: Internet Engineering Task Force
IP	: Internet Protocol
IPv4	: Internet Protocol v4
IPv6	: Internet Protocol v6
IoT	: Internet of Things

LACAS : Learning Automata-Based Congestion Avoidance Algorithm in Sensor Networks
MAC : Medium Access Control
MTU : Maxiumum Transmission Unit
MQTT : Message Queuing Telemetry Transport
NAT : Network Address Translation
Obs : Observing Resource
OF : Objective Function
OSI : Open Systems Interconnection
PDR : Packet Delivery Ratio
PSFQ : Pump-Slowly, Fetch Quickly
RAM : Random Access Memory
RDC : Radio Duty Cycle
REST : Representational State Transfer
ROM : Read-Only Memory
RPL : Routing Protocol for Low-Power and Lossy Network
RTO : Retransmission Timeout
RTS : Request to Send
RX : Receive / Receiver
TCP : Transmission Control Protocol
TLS : Transport Layer Security
TSCH : Time-Slotted Channel Hopping
TX : Transmit / Transmitter
UDP : User Datagram Protocol
VBF : Variable Backoff Factor
WFCC : Weighted Fairness Congestion Control
WPAN : Wireless Personal Area Network
WSN : Wireless Sensor Networks
XMPP : Extensible Messaging and Presence Protocol

1. INTRODUCTION

1.1. Internet of Things and Constrained Devices

With the development of technology, application areas of electronic devices and networking are expanding. Consequently, the hardware features of these devices are developing rapidly, and also the memory sizes increase. Therefore, this growth turns into a huge cost for the end-user. In addition, the energy consumption of the devices increases due to continuous workload, and then the need for a perpetual charge problem arises. The solution to such problems is very important for IoT networks which will be designed for projects such as smart-home, health monitoring, and so on. Because these systems are always communicating with one another, they continuously transfer and gather data with each other. Furthermore, since the number of devices required for the systems to be installed will be many, the installation of such systems will result in very high costs. Therefore, the dimensions, memory sizes, and energy usage of the devices used in IoT networks should be quite optimal. It is, therefore, necessary to use devices that are relatively small in memory, consume much less energy than normal devices, and are as small as possible in IoT networks.

Figure 1.1. IoT and Application Areas (Candoğan, 2020)

An IoT network contains constrained devices that are connected to each other wirelessly and sense the environment to collect data about humidity/ temperature, accelerometer, light, etc. And then, these nodes transmit and forward these data between each other to perform special tasks for their intended presence in that environment (Buratti, Martalo, Verdone, & Ferrari, 2011). Because of this kind of behavior, these devices are also called as smart objects. Smart objects contain embedded tiny operating system which is special for this kind of devices. Also, these devices include specialized microprocessors, pre-built sensors, and actuators, a power source (especially a battery), built-in or pluggable radio device, USB, and UART connections. The power supply constantly supplies power to the device and ensures that it has the ability of always on-state. The sensors and actuators give the device the ability to interact and gather data. The microprocessor is the core of the device and specialized for some kind of operations to gathered data with limited complexity. UART and USB connections are generally used for embedding an operating system or adding additional components such as sensors (Dunkels & Vasseur, Interconnecting Smart Objects with IP, 2010).

Figure 1.2. Different types of constrained devices (Development tools Archives, n.d.; Vilajosana, Tuset, Watteyne, & Pister, 2015; Insense Examples, n.d.)

Constrained devices are the key element for IoT networks and they are special devices in almost every aspect. These kinds of devices are constrained about power consumption, size, available memory, bandwidth, etc. For example, the device called “OpenMote” has 32KB of RAM, 512KB of flash capacity and in low power mode, it use 0.4µA. (Texas Instruments, 2015). Due to these features,

constrained devices must be flashed with special operating systems (e.g., RIOT, Contiki, TinyOS, etc.) which have an objective of providing a generic, open-source software system that can handle a small number of hardware resources efficiently (Bormann, Ersue, & Keranen, 2014).

RFC7228 defines three classes of constrained nodes:

- Class 0: << 10KiB data size (e.g., RAM), << 100 KiB code size (e.g., Flash)
- Class 1: 10 KiB data size (e.g., RAM), 100 KiB code size (e.g., Flash)
- Class 2: 50 KiB data size (e.g., RAM), 250 KiB code size (e.g., Flash)

Class 0 devices are the most restricted sensor devices. These devices are so limited in terms of memory and processing capacity that they cannot even communicate directly with the Internet. These devices can participate in the Internet with the help of other devices that can directly communicate with the Internet and operate as gateways, servers, and proxies. Class 0 devices cannot provide security in the traditional sense and are usually preconfigured with very small datasets. They respond with keepalive signals for administrative purposes and are used as on/off switch or as a simple health indicator (Bormann, Ersue, & Keranen, 2014).

Class 1 devices also have very limited capabilities in terms of code space and processor capacity. They do not use a fully implemented protocol stack, such as HTTP and Transport Layer Security (TLS) to communicate with the Internet. However, it is possible for them to communicate with the Internet without using a gateway when using customized protocols such as CoAP together with UDP. With all this in mind, devices of this class can be fully integrated into an IP network, but they should use specialized stingy protocols for criteria such as memory, code space, and power consumption (Bormann, Ersue, & Keranen, 2014).

Class 2 devices are the least restricted and basically capable of supporting most protocols stacks. Despite these features, they still benefit from lightweight protocols that use less bandwidth and consume fewer energy. Since they consume less resources for the network, they use more resources for application thus it is an advantage of these kinds of devices. Therefore, the use of Class 2 devices can reduce the cost and increase the interoperability (Bormann, Ersue, & Keranen, 2014).

Also, most of the network protocols that we use in other devices in daily life cannot work efficiently with a network that is built with constrained devices due to not specialized about limited memory and low bandwidth. That's why sensor networks need customized network protocols and topologies. For example; Ethernet and WiFi are protocols in conventional networks, but in constrained networks Zigbee, BLE, DASH7, Z-Wave are used instead in the data link layer.

In an IoT network, constrained devices are generally close to each other. This kind of networks is named as "edge network". Edge network is a distributed information technology architecture where client data is processed around the network as close to its source as possible. In our simulations, sensor nodes are placed at a distance of 30 meters between them and it is kind of an edge network.

Transmitting huge amounts of raw data over a network puts a huge load on network resources. In some cases, it is much more efficient to process data near its source and only sends data of value over the network to a remote data center. Especially for the Internet of Things (IoT) systems and embedded devices, the data comes from the physical world with various sensors and acts to change the physical state with various outputs and actuators. For example, instead of constantly publishing data about the oil level in a car's engine, a sensor can periodically send summary data to a remote server, or a smart thermostat can transmit data if the temperature goes outside the specified limits. Alternatively, an intelligent Wi-Fi security camera placed on the elevator door sends data when a certain pixel

percentage varies significantly between the two consecutive images. This shows that there is a movement.

1.2. Congestion Control and Congestion Control Approaches

Trying to access to a shared resource by multiple distributed clients simultaneously causes congestion. Congestion occurs in many different situations, and has a wide range of studies about it. It usually occurs on networks with high message traffic, and is understood by reduced network response time and increased retransmissions. Congestion control algorithms, together with congestion avoidance algorithms are trying to avoid this problem. These algorithms are examined in detail in Section 2 (Subir, 2015).

As the number of sensors increases, transmissions between devices also increase which leads to more traffic in a network, hence more congestion problems may occur. There are a lot of proposals about the congestion control mechanism in IoT networks but there are still many issues to investigate.

The congestion problem can be discussed under two categories considering the where and how packets are lost (Kang, Zhang, & Nath, 2007).

A. Reasons for Packet Loss

i. **Packet Collisions in the Medium:** In a wireless sensor network, two nodes may want to transmit a packet at the same time, and this situation can cause packet loss.

ii. **Buffer Overflow:** In a node, buffer or queue overflow can cause packet drops. This definition is also used as a congestion definition in most of the research papers. The main reason for overflow is the ratio of transmitting/receiving packet is higher than expected.

B. Source of Packet Loss

a. **Source to Source Congestion:** If a network topology is created densely when the sensors are generating data packets at a crucial event, the hotspot will show up near the sources.

b. **Sink to Sink Congestion:** Unlike the previous reason, sparsely created topologies can also cause hotspots to burst even if they are generating data at a low rate.

c. **Forwarder Congestion:** In a wireless sensor network that includes more than one source-sink pair, their flows can intersect and the area around this intersection causes the creation of hotspot.

On the web, HTTP (Hyper-Text Transfer Protocol) protocol is used for fetching files such as images, texts, videos, etc. But HTTP is not designed with the consideration of energy consumption and memory constraints. To deal with this situation in IoT networks, the CoAP (Constrained Application Protocol) protocol is specifically created for the application layer instead of HTTP. Generally, the CoAP interaction model works similarly to HTTP but there are slightly different methods in this protocol.

CoAP is a UDP based protocol and it is used for especially constrained IP networks. Because of its simplicity, there is no standard congestion control mechanism developed for CoAP. To make congestion control in CoAP, the default mechanism used is simply retransmit the lost messages. But there are experimental works on different approaches for congestion control in CoAP protocol.

In the previous researches, two objective functions which are OF0 and MRHOF, or different congestion control mechanisms are examined with different metrics and algorithms were measured and evaluated among themselves. In the literature review section, these researches are given. Also, in the literature, there is no study investigating the relationship between objective functions and congestion control mechanisms under a single study. But in this thesis study, we investigate

the performance of congestion control mechanisms (i.e., Default CC, CoCoA Strong, CoCoA, and Linux RTO) with different networks that implement different protocols by making several congestion control mechanism and network combinations. Besides that, the effects of the objective functions of RPL (Routing Protocol for Low-Power and Lossy Networks) on congestion by applying different congestion control mechanisms are especially studied in this thesis. After investigating all of that, performance analysis is done for each network setup and the results are presented.

2. RELATED WORKS

In this section, the congestion control in wireless sensor networks and the internet of things networks are summarized. Although this thesis covers the congestion control mechanisms for the internet of things network, to study the congestion effectively, congestion in other networks should be examined.

2.1. Congestion Control in Wireless Sensor Networks

There are different attempts to overcome the congestion problem in the Wireless Sensor Networks - (WSN). Despite the successful results of these studies still, some issues persist. Additive Increase Multiplicative Decrease (AIMD) is developed for controlling the transmission rate of sensor nodes adaptively and it uses loss as the signal of collision. The control of the transmission rate depends on NACK messages because the pieces of information about the congestion are added to the NACK packet. If the NACK is not received the mechanism increases the rate progressively, and if the NACK is received, the rate will be decreased. Comparison to TCP, this method uses adaptive rate control at every single node cooperatively at the application layer (Woo & Culler, 2001).

CODA (COngestion Detection and Avoidance) is one of the protocols that is written for WSN and it aims to solve the congestion problem in an energy-efficient way. This protocol can handle the congestion with closed-loop regulation, open-loop hop-by-hop backpressure, and receiver-based congestion detection mechanisms. Even though the energy efficiency provided, the reliability problem continues (Wan, Eisenman, & Campbell, CODA: Congestion Detection and Avoidance in Sensor Networks, 2003).

Event-to-Sink Reliable Transport (ESRT) protocol is implemented for reliable transport and it includes a congestion control component. This component has a mechanism for energy conversation and reliable detection, but the way of

solving the congestion problem may cause additional energy consumption (Akan & Akyildiz, 2005).

The technique called “Pump-Slowly, Fetch Quickly” (PSFQ) is focused on rate-limiting, and it is open-source. Just like the other techniques, PSFQ asserts that the algorithm is energy-saver and scalable. The main concern is distributing the data slowly (pump slowly), and fetching the data loss to neighbors quickly (fetch quickly). For comparison, ESRT is not focused on data flow but PSFQ is (Wan, Campbell, & Krishnamurthy, 2005).

In LACAS (Learning Automata-Based Congestion Avoidance Algorithm in Sensor Networks) the main focus is equalizing the rate of transmission and processing rate and it focuses on the healthcare WSNs. It is important that the algorithm learns from the past and applies the learned knowledge to future scenarios. This also affects performance because of learning capability. But the assumption of equalizing the rates may not be applicable because when the high number of packets is under circulation, the packet drop possibility is increasing (Misra, Tiwari, & Obaidat, 2009).

The other approach for congestion control is Weighted Fairness Congestion Control (WFCC) and it is designed to propose a decentralized and fairness guaranteed method. It uses weighted nodes to be understood the level of significance and combining this with the data rate of sensor nodes to control congestion. But the data rate must be regulated at certain intervals so it is very hard to implement regulation because the data frequency in the WSN is very high (Li, Li, & Yu, 2012).

DCCC6 (Duty Cycle-Aware Congestion Control for 6LoWPAN Networks) is one of the congestion control schemes for WSNs. In this protocol, the dynamic buffer system is used and the focus is on duty cycling. The notification of child nodes that are involved in congestion problem and rates are regulating by DCCC6 according to duty cycling mechanism. Considering the simulation results, it is

claimed that this protocol provides higher goodput and the packet loss is decreased compared to previous works (Michopoulos, Guan, Oikonomou, & Phillips, 2012).

The congestion control protocol named CADC (Congestion Aware Duty Cycle MAC) also uses a mechanism like DCCC6 for RDC adaptation for solving the problems that include changing the pattern and traffic load of the network quickly. It doesn't need any time synchronization, extra packets for the information exchange and this protocol is not depended on any topology or protocol. Because of this specialty of the protocol, it is very easy to transfer to other WSNs (Michopoulos, Oikonomou, Phillips, & Guan, 2014).

The RPL-based protocol GTCC (Game Theory-Based Congestion Control Protocol) is proposed for congestion issues originated from child and parent nodes. Using the game theory, this protocol changes the parent node when the congestion is detected by using the packet flow rate. GTCC regulates this parent change with a random timer and the change is limited to 1 in a certain time. Rate adaptation is used in this protocol when the congestion is not reduced enough and the rate of packet sending is lowered by this mechanism. According to the authors, GTCC is more powerful in comparison to ContikiRPL when the OF0 and OF-ETX are used (Sheu, Hsu, & Ma, 2015). Also, there are some studies about RPL-based networks using the game theory as the base reference. For example, GTCCF (Game Theory-Based Congestion Control Framework) is one of these kinds of approach and it assumes that the network is a noncooperative game problem (Al-Kashoash, Hafeez, & Kemp, 2017).

2.2. Congestion Control in Internet of Things Networks

TCP (Transmission Control Protocol) is commonly used by many Internet services. Even if the network substructure changes, TCP and its applications will likely continue to be used in the future. Because of these reasons, the congestion problem is generally solved by using TCP and most of the literature is about that.

In TCP Reno, the window size is changed as a loop in a normal situation. The window size keeps being raised until packet loss occurs. The throughput of the connection would be reduced when the window size is entrapped because of packet loss. It cannot be prevented because of its basic characteristic of the congestion control mechanism embraced in TCP Reno; it can determine network congestion only by packet loss. However, reducing the window size is not sufficient when the TCP connection itself causes congestion because of its too large window size. If the window size is favorably controlled such that the packet loss does not occur in the network, the throughput decline to the reduced window can be fudged (Jacobson, 1988).

TCP Vegas inspects its window size by analyzing RTTs (Round Trip Time) of packets that have sent before. If observed RTTs increase, TCP Vegas notices that the network begins to be congested, and throttles the window size. If RTTs decreases, the sender host of TCP Vegas notices that the network is unwound from the congestion, and augment the window size again (Brakmo & Peterson, 1995).

As TCP Vegas, FAST TCP is also a delay-based TCP algorithm that uses queuing delay as the indication of congestion. This algorithm has asynchronous and independent components that are window control, data control, burstiness control and estimation in terms of stability, scalability, and RTT. But this algorithm has also disadvantages about minimal RTT (Jin, et al., 2005).

TCP SACK is an improved version of TCP Reno. Selective acknowledgement together with selective retransmission can raise performance when multiple segments are dropped within one window. Selective acknowledgement is reached by adding to the acknowledgement a list of the discrete blocks of data that have been received (Floyd, Henderson, & Gurtov, The NewReno modification to TCP's fast recovery algorithm, 2004). Reno TCP has performance issues when multiple packets are lost from one window of data without SACK. TCP SACK allows receivers to report any unrequited data they

have received. When coupled with a selective retransmission policy implemented in TCP senders, considerable savings can be achieved.

TCP FACK is the improvement of TCP SACK with Forward Acknowledgement. It has some extra properties over TCP SACK, to predict the amount of data transmitted, it uses SACK options fertile and it also provides a better way to the window when congestion occurred. The goal of FACK is to perform delicate congestion control and increase connection throughput during the recovery phase (Mathis & Mahdavi, 1996).

HS-TCP, The HighSpeed version of TCP, is a lately proposed rectification to the TCP congestion control mechanism. It is dedicatedly designed for high bandwidth-delay networks (Floyd, 2003).

Scalable TCP is a change to the TCP congestion window update algorithm on the server-side. It propounds a strong mechanism to improve performance in highspeed wide area networks using traditional TCP receivers. Scalable TCP is intended to be increasingly deployable and behaves identically to traditional TCP stacks when small windows are convenient (Kelly, 2003). Like traditional TCP, HSTCP and STPC detect congestion through packet losses.

CUBIC protocol changes the linear window growth function of present TCP standards to be a cubic function for enhancing the scalability of TCP over fast and long-distance networks. It also ensures that window growth is independent of RTT, resulting in fairer bandwidth allocations between streams with different RTTs (round trip times) - so these streams enlarge congestion windows by the same rate. While in the steady-state, CUBIC aggressively increases the window size when the window is far from saturation point, and slowly when it is close to the saturation point. This feature makes CUBIC very scalable when the network's bandwidth and latency product is large, and also extremely stable and fair to standard TCP streams (Ha, Rhee, & L., 2008).

TCP-FIT introduces the idea of delay-based TCP into the TCP CUBIC algorithm framework. TCP-FIT can improve throughput performance over wireless

networks more than CUBIC. TCP-FIT algorithm was inspired by parallel TCP, but with the significant differences that only one TCP connection with one congestion window is constituted for each TCP session, and that no changes to other layers of the end-to-end system need to be made (Wang, Wen, Zhang, & Han, 2011).

2.3. Performance Evaluations About Congestion Control Mechanisms and Objective Functions

There are a lot of studies about evaluating the performance of congestion control mechanisms or objective functions. In each of them, different algorithms, test scenarios, and performance metrics are used. In this thesis, performance metrics, test scenarios, simulation parameters, and chosen algorithms are determined according to these studies.

Mardini, Ebrahim, and Al-Rudaini (2017) used Contiki 2.7 to analyze the performance of RPL objective functions. They run the simulations on the Cooja simulator and used the metrics such as average received packets, average lost packets, average power consumption, and average listen duty cycle. According to their work, OF0 is better than MRHOF when the main concern is energy, static-grid topology, and convergence time (Mardini, Ebrahim, & Al-Rudaini, 2017).

Pradeska, Widyawan, Najib, and Kusumawardani (2016) made tests to evaluating the reliability, energy, mobility, and quality of objective functions. They used the PDR, hop count, latency, packet delivery ratio, power consumption, and convergence time as performance metrics. According to their results, when the network quality is consi dered, MRHOF gives better performance than OF0 while OF0 is suitable for fast formation networks and low power consumption (Pradeska, Widyawan, Najib, & Kusumawardani, 2016).

In other work that performed by Lamaazi, Benamar, and Jara (2017), using different scenarios and metrics, the performance analysis of MRHOF and OF0 is investigated using metrics such as hop count, ETX, packet loss, energy, and control

traffic overhead. This study shows that OF0 is better when the nodes are mobile and MRHOF is better when the nodes are static (Lamaazi, Benamar, & Jara, 2017).

Another research about comparing the objective functions is studied by Qasem, Altawssi, Yassien, and Al-Dubai (2015). They used various parameters for using as metrics such as PDR, power consumption and RX. In their research, results show that these parameters are important for network and MRHOF has a better RPL working mechanism than OF0 when the network is dense (Qasem, Altawssi, Yassien, & Al-Dubai, 2015).

Abuein, et al. (2016), tested PDR and energy consumption in non-mobile networks with fixed and random topologies. According to their study, in low-density networks, OF0 gave better PDR results. However, MRHOF gave better energy consumption values with high-density networks (Abuein, et al., 2016).

Betzler, Gomez, Demirkol, and Paradells (2015) developed a new congestion control mechanism named CoCoA+ to improve the CoCoA algorithm and compare the performance of this algorithm with Default CoAP and CoCoA. According to them, their results show that CoCoA+ is better than Default CoAP and CoCoA when considered the overall PDR, end-to-end delay, and Settling Time metrics (Betzler, Gomez, Demirkol, & Paradells, 2015).

Betzler, Gomez, Demirkol, and Paradells (2016) made another study about comparative analysis with Default CoAP, CoCoA, CoCoA Strong, Basic RTO, Linux RTO, Peak-Hopper-RTO congestion control mechanisms. All studies are implemented using real testbeds with different performance evaluation metrics such as overall throughput, Settling Time, PDR, and merging delay. In this work, the results show that the CoCoA is better than the default CoAP and other variations outperform in terms of providing flexible and adaptive RTO calculation (Betzler, Gomez, Demirkol, & Paradells, 2016).

Betzler, Gomez, and Demirkol (2015) used three different congestion control mechanisms which are default CoAP, CoCoA, and observe mechanism. They used PDR as the main performance metric for the tests that are emulated for

GPRS and UMTS link. According to this study, CoCoA is better in terms of fairness, maintaining, and improving the performance of stable and secure CoAP communications (Betzler, Gomez, & Demirkol, 2015).In their studies, Demir and Abut (2018) investigated the performance of Default CC and CoCoA algorithms using different grid topologies by sending continuous messages from clients to different servers. According to results, CoCoA is not always work better than Default CC in terms of throughput (Demir & Abut, 2018).

3. MATERIALS AND METHODS

3.1. Materials

In this thesis, several software, hardware, and network protocols are used according to simulation requirements. The software are used to create and prepare the test environment, the hardware are used to embed and run these software, and the network protocols are used to measure the performance of different network stacks. Under this section, how and for what purpose these materials are used explained.

3.1.1. Software Environments

A lot of software and software development environments are used during the studies. A virtual machine to operate the operating system, an operating system specially developed for IoT networks, a network simulator designed for network tests, client and server software that implemented using different congestion control mechanisms, and software that bridging and communicating clients and servers are used. In the next sub-section, installation and usage of these software will be given with details.

3.1.1.1. VMware Workstation

VMware Workstation is a virtualization software that can allow using multiple operating systems simultaneously on particular operating systems (Supported host operating systems for Workstation Pro 12.x and 14.x (2129859), n.d.). These operating systems are called virtual machine and each one operates its own system.

VMware Workstation allows a user to set the requirements of the new virtual machine. It supports bridging which allows to the virtual machine to use the root computer’s hard drive, USB, sound card, network adapter, etc. In addition,

data sharing between the root machine and a virtual machine is also possible with drag and drop features.

VMware Workstation can suspend the operating system and save the state of a virtual machine that is called snapshot. These snapshots restored when the user wants to return to work and then, the virtual machine returns its own saved state. These snapshots work even if the root computer shuts down. This feature provides data integrity and flexibility (Warren, 2012).

The Ubuntu operating system is installed on VMware Workstation using Cooja Simulator to perform experiments in this study.

3.1.1.2. Contiki-NG Operating System

Contiki-NG is a protothread based operating system which is open source and implemented with C programming language. It is designed and developed for IoT applications and includes several network tools and network architecture with a lot of network protocols. Contiki-NG provides a useful platform for implementing IoT applications with memory-constrained wireless microcontrollers such as ARM Cortex-M3/M4 and the Texas Instruments MSP430 (Contiki-NG Home, n.d.) which are designed to consume less power than traditional microcontrollers do. In addition, Contiki-NG has a lot of drivers and sample codes for IoT based microcontrollers. 2 KB of RAM and 40 KB of ROM are enough for simple Contiki-NG configuration (Dunkels A. , Contiki: The Open Source OS for the Internet of Things, 2015).

The Contiki-NG operating system comes with a lot of pre-implemented protocols and services such as IPv4, IPv6, TCP, UDP, 6TiSCH, Routing Protocol for Low-Power and Lossy Network (RPL) and Constrained Application Protocol (CoAP), and a lot of internet-based services. It has also a layered communication stack for low-power radio networking which is called Rime. It is a lightweight protocol that reduces the implementation of sensor network protocols (Dunkels A. , 2007).

The Contiki-NG build system is specially designed for embedding the full operating system image and the binaries include applications to different microprocessors (The Contiki-NG build system, n.d.).

3.1.1.3. Cooja Network Simulator

Cooja Simulator is a multi-featured and handy network simulator that is implemented with Java programming language. It is specifically designed for building and tracking the IoT network simulations. Cooja can execute binaries of different types of motes. In Cooja, users can create different nodes using programs that are implemented with the C programming language. These nodes are called the Contiki mote and the Cooja can simulate the actually compiled motes that are executing the Contiki system. The mote type determines the type of sensor hardware and which Contiki applications are to be simulated (Cooja Simulator, 2016).

Because of using Java Native Interfaces, the Cooja simulator depends on compilers and linkers of each node. Cooja Simulator compiles the Contiki operating system as a shared library for the native platform and loading library into Java using Java Native Interfaces, then simulator can control and analyze each node on the system. This is performed by informing the Contiki operating system to handle an event or fetching the memory. Cooja can create heterogeneous networks by compiling and executing several different nodes simultaneously (Dunkels, Eriksson, Finne, & Voigt, 2006). In the Figure 3.1, the appearance of a running Cooja simulator and its tools can be seen.

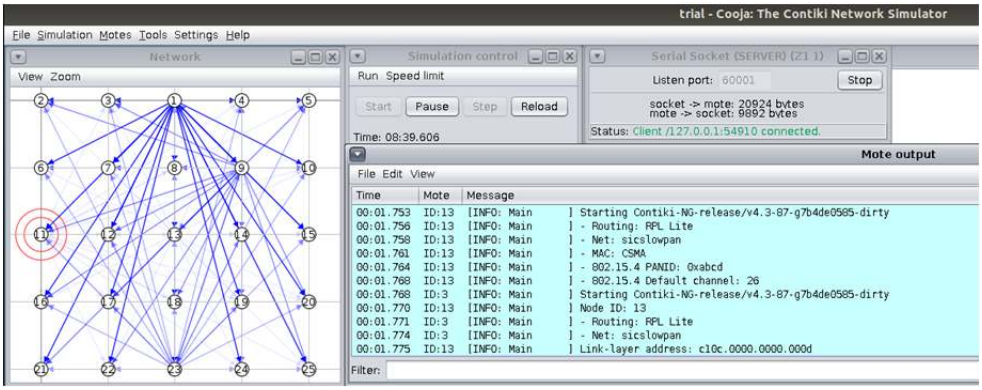

Figure 3.1. Cooja Network Simulator

Cooja Simulator needs Java version 1.6 and the build tool “ant” to work, but all of these requirements are already provided by Contiki operating system. To use Cooja, there is no need for any other installation.

Cooja simulation configurations can be saved for later use. This configuration includes motes and their types, plugins, and medium configuration, etc. However, the simulation state is not saved. This means when a saved simulation is loaded, the start time is zero (Tutorial: Running Contiki-NG in Cooja, n.d.).

3.1.1.4. Tunslip

Tunslip is a little program written with C programming language and it helps to connect between real and virtual networks. This tool comes with the Contiki-NG operating system and because of Tunslip is a C program, it must be compiled before using (Build Tunslip6, n.d.). This tool is used for bridging the virtual network traffic (which is created and simulated in Cooja Simulator) between a border router and a host. After using the Tunslip, the border router will take a second IP address which is a global address and when communicating the other networks, border router uses this IP address instead of its local IP address.

3.1.1.5. Eclipse

Eclipse is an open-source integrated development environment that is generally used for developing Java applications. Furthermore, Eclipse can be used for developing applications with several programming languages if the corresponding plugins are installed. With these third-party plugins, Eclipse becomes more extensible even used for developing network or database systems.

In this study, Eclipse is used because of its simplicity of installation. The latest version of this program can be installed from Ubuntu Software Center. Also, the Californium framework and its examples which are used for CoAP client is implemented with Java programming language can be configured very easily with Eclipse. CoAP client examples will be configured for setting the arguments and arranging the codes that are used for performance analysis.

3.1.1.6. Californium

To analyze congestion in sensor networks, the nodes must be used CoAP with CoCoA (Congestion Control/Advanced). In order to use CoAP, a pre-implemented CoAP framework is required. At the back-end or receiver side, this framework is Californium which is implemented with Java programming language considered from RFC7252. It is a special CoAP framework that is specially written for IoT applications and can be embedded in constrained devices with compatible operating systems (CoAP in Java, n.d.).

The Californium is an open-source project and accessible from it's GitHub repository. Also, Californium comes with a lot of handy examples that can be used for IoT applications and experimental works.

3.1.1.7. Erbium

Like Californium, Erbium is a CoAP implementation that is written using a C programming language. It is implemented based on the rules of RFC7252 and supports block-wise transfers. The Contiki operating system uses Erbium as a low-power REST Engine (A Quick Introduction to the Erbium (Er) REST Engine, n.d.).

In Contiki, there are three examples of Erbium implementations under the examples folder: er-example-server.c, er-example-client.c and er-plugtest-server.c. In this thesis, the Californium is used as a client, therefore Erbium is employed as a server-side application in all motes except border router in the Cooja simulator.

3.1.2. Hardware Environments

In this thesis, since experiments are done virtually in the simulation environment, no physical hardware is required. Although virtual hardware are used in simulations, in the below sub-sections these hardware and their features are given to explain better the experimental study performed.

3.1.2.1. Zolertia Z1 Mote

The Z1 platform is a general-purpose development platform for wireless sensor networks. This platform is often used by developers, researchers, and hobbyists. This platform, which is compatible with the Tmote family, offers twice the performance but also includes some improvements (The Z1 Mote, 2018). Figure 3.2 depicts a Zolertia WSN platform Z1.

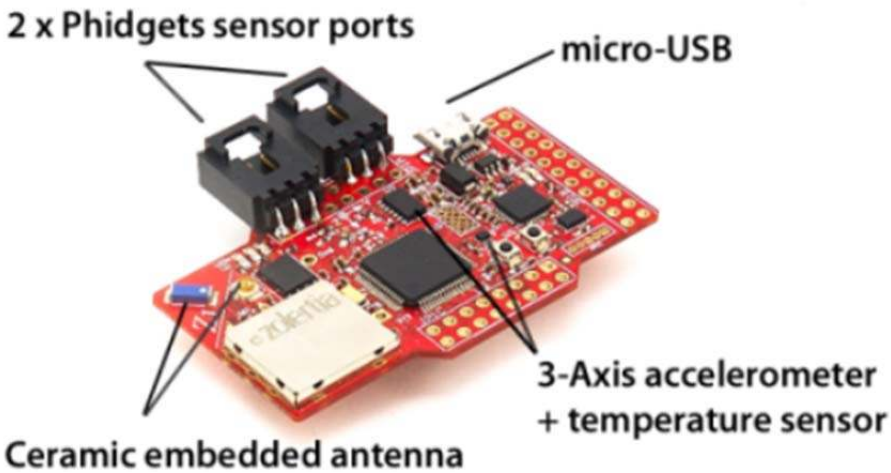

Figure 3.2. A Zolertia Z1 mote (da Silva, Segers, Braeken, Steenhaut, & Touhafi, 2018)

General features of this platform can be given as:

- Second generation MSP430F2617 low power microcontroller and CC2420 transceiver
- 16-bit RISC CPU @16MHz clock speed
- 8KB RAM
- 92KB Flash memory
- Operates at 2.4GHz with the data rate of 250Kbps

3.1.3. Network Stack

Although most common network frameworks TCP/IP and OSI (Open Systems Interconnection) are enough for almost every network application, because of different requirements, IoT network stack must be designed differently from the OSI model. Therefore, IoT network layers differ from the OSI model with layer scopes and protocols.

In the IoT network stack, the physical layer is operated by the IEEE 802.15.4 protocol. Similar to OSI Model, Medium Access Control (MAC) protocol is at the link layer, but between the physical layer and link layer, there is a different protocol which is called Radio Duty Cycling (RDC). At the adaptation layer, 6LoWPAN (IPv6 over low power wireless personal area networks) is used for communication tasks. The packet routing operations are controlled by a protocol called RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) and it is presented at the network layer with IPv6. In comparison to the TCP/IP model, UDP is the most used protocol for the transport layer when it comes to IoT network applications. And for the application layer, CoAP is used as an alternative for HTTP. Although there is Message Queue Telemetry Transport (MQTT) protocol at the application layer, because of its work area is constrained, CoAP is the most

used protocol in that layer (Sethi & Sarangi, 2017). Figure 3.3 illustrates the Contiki network stack:

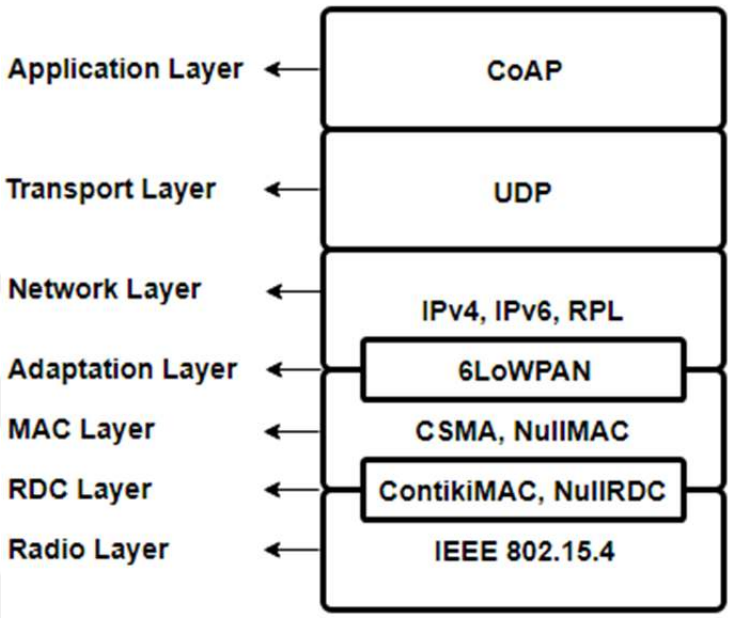

Figure 3.3. A typical IoT network stack and the protocols.

3.1.3.1. Physical Layer

This layer is at the bottom of the network stack and also called "Radio Layer". In this layer, operations are done bitwise and then these bits are converted to a signals or electrical current for sending to the transmission medium.

3.1.3.1.(1). IEEE 802.15.4

This WSN (Wireless Sensor Networks) protocol is developed for low data rate operations on the constraint devices and considers low cost, low energy, and short-range communications. This protocol is considered as a standard and its rules cover both radio and MAC layer (IEEE 802.15 WPAN™ Task Group 4 (TG4), 2010).

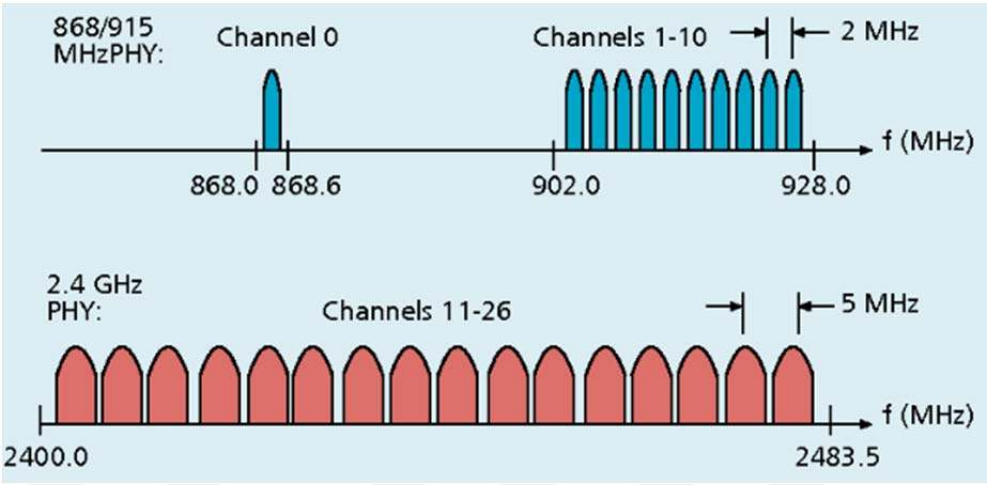

Figure 3.4. IEEE 802.15.4 channel structure

This layer has three different unlicensed band option for different areas (i.e., 868.0 to 868.6 MHz for Europe, 902 to 928 MHz for North America, and 2400 to 2483.5 for worldwide) that can be seen in Figure 3.4, and uses four different transmission types which generally utilize frame structure. This standard also has a control mechanism for transmissions called Cyclic Redundancy Check (CRC). Because constrained devices are concerned about power consumption, this protocol has a duty cycle mechanism and with this, it can organize the active or inactive time of devices. The duty cycle mechanism allows 99% inactive time for devices and it affects a tremendous amount of power economy (Buratti, Martalo, Verdone, & Ferrari, 2011).

3.1.3.1.(2). Packet Delivery Ratio

Packet delivery ratio (PDR) is a measure and calculated by the ratio of delivered data packets which are created and sent from a source to a destination (Rathee, Ahmad, Kerrache, & Azad, 2019). Cooja enables us to change TX (represents transmitted data from a link) and RX (represents received data from a link) rates. This is very useful because, in real life, no network is perfect and could

give the PDR value of 100. The advantage of this setting is giving an impression to the user a real-like simulation of the network.

3.1.3.2. Radio Duty Cycling (RDC) Layer

As said many times, the energy consumption is very important for IoT networks, and devices must consume less power and preserve their energy for long times as much as possible. For this purpose, between the MAC and link layer, there is a layer which is called Radio Duty Cycling (RDC). This layer ensures that the radio receivers of devices in the network are keeping off most of the time and awake in the case of establishing connections and transmitting or receiving data. This approach regulates the power consumption and there are some protocols in this layer that provide the devices are turned off 99% of the time (Dunkels A. , 2011).

RDC layer takes care of the most significant tasks in the network. Because the RDC driver must know when will the device is turned on or turned off, it must also know when the packets will be transmitted or received to schedule correctly the sleep and awake time intervals of the device. RDC drivers periodically listen to the medium and do its job when the activity is detected (MAC protocols in ContikiOS, 2014).

In Contiki-NG, there are several RDC drivers like ContikiMAC, NullRDC, X-MAC, CX-MAC (Compatibility X-MAC), LPP (Low Power-Probing). X-MAC is an older driver and CX-MAC is an alternative version of X-MAC. To be convenient to the thesis work, and measure the performance metrics efficiently, ContikiMAC and NullRDC are chosen for test scenarios.

3.1.3.2.(1). ContikiMAC

ContikiMAC is considered the best RDC driver due to its energy-efficient approach and it is the default RDC mechanism for Contiki operating systems. To speaking generally, ContikiMAC keeps the transceiver off 99% of the time because

most of the time in the network there is no communication and no need for staying awake all of the devices. This mechanism has a powerful time scheduling for turning on and off the transceiver and includes a lot of optimizations (fast sleep, phase-lock, etc.) about the energy and wake-up timing. According to the developer, this protocol designed with the consideration of the previous RDC protocols (Dunkels A. , 2011). The packet sending mechanism of ContikiMAC is illustrated in Figure 3.5.

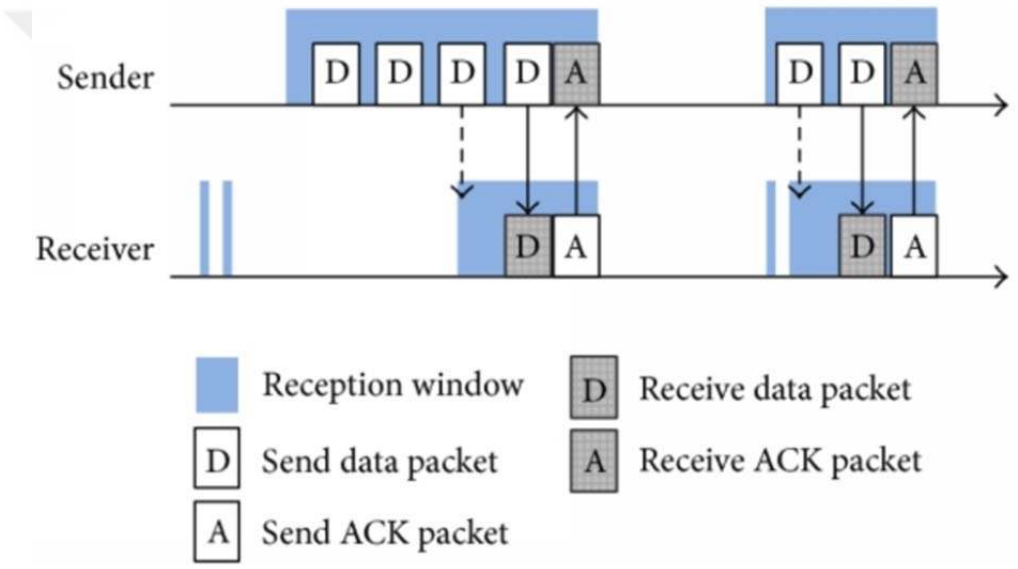

Figure 3.5. ContikiMAC packet sending mechanism (Ferri, Tang, Ma, Huang, & Wei, 2016)

ContikiMAC is designed for sending operations. It awakes the nodes at the time of sending a packet and node stay awake until the transmission is successfully done and receiving the acknowledgement (ACK) message. After this ACK message, it is time for sleeping the node until the next transmission. Because of this behavior, this protocol has an asynchronous mechanism.

3.1.3.2.(2). NullRDC

This protocol is not a real protocol and it depicts that the RDC layer removed. When this protocol is used in Contiki operating system, it is ensured that radio control mechanism is not implemented with any algorithm in the RDC layer. Packets are just pass through the RDC layer without any process and it increases the energy consumption when this protocol is activated (MAC protocols in ContikiOS, 2014). The nodes will stay awake all the time when this driver is used and the energy consumption is increased.

In this thesis, performance metrics are not based on energy. Using NullRDC may affect the metrics positively and, it is useful to be used in test scenarios (Thébaudeau, Change mac or radio duty cycling protocols, 2019).

3.1.3.3. Medium Access Control (MAC) Layer

MAC Layer (also called link layer or data link layer) is responsible for preventing the collisions on the network. It communicates with the RDC layer and uses this layer to transmit packets and receives packets from the RDC layer. If the collision (on the RDC layer or on the physical layer) detected the retransmission mechanism works and handles the situation properly (Halcu, Stamatescu, Stamatescu, & Sgârciu, 2016).

In Contiki-NG, there are several MAC drivers. NullMAC is simple and like NullRDC, it is a pass-through protocol and does nothing. Carrier-Sense Medium Access (CSMA) is a more advanced and standard protocol; it has retransmission and addressing mechanisms. Time-Slotted Channel Hopping (TSCH) is a version of frequency-hopping MAC and has scheduling and synchronization mechanisms. Bluetooth Low Energy (BLE) is experimental and used for radios that has Bluetooth (Duquennoy, The Contiki-NG configuration system, 2019).

3.1.3.3.(1). CSMA

This mechanism is the default for the Contiki-NG operating system and it does retransmission in the event of detecting packet collision by the RDC driver. Even though this protocol also has a job of carrier sensing, in the Contiki operating systems, this job is undertaken by RDC layer.

In Contiki operating system, CSMA works by depending on the RDC driver type. CSMA implements enqueuing based on packet transmission type (e.g., unicast, broadcast, etc.) and addressing the devices. Also, it has a list of the neighbors to keeping track of collisions, retransmissions, packet loss, etc. (MAC protocols in ContikiOS, 2014). The process mechanism and steps of CSMA/CD is illustrated in Figure 3.6.

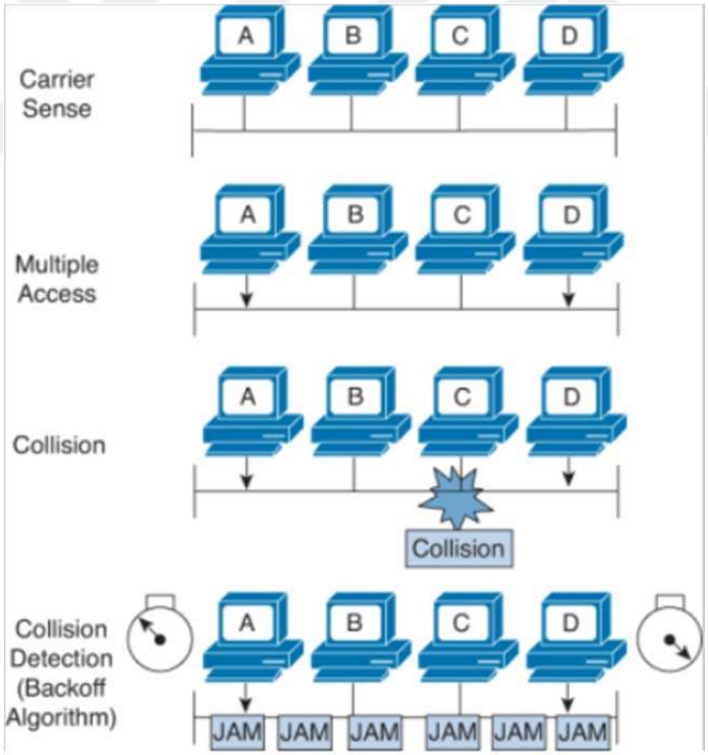

Figure 3.6. CSMA/CD Process (Cisco ICND1 Foundation Learning Guide: LANs and Ethernet, 2013)

CSMA uses a method called “exponential backoff” and RTS (Request to Send) / CTS (Clear to Send) to regulate retransmissions. This method has an algorithm for increasing the time of waiting for retransmitting the packet in order to find an appropriate transmission rate and is used in the case of another transmitting is detected at the same time. Packets are waiting for transmission until no activity on the medium is detected. Also, if the acknowledgement (ACK) system is online, MAC layer waits an ACK within the defined time interval (Hamadoun, Chalhoub, & Misson, 2016).

3.1.3.3.(2). NullMAC

NullMAC protocol simply removes the workload of the MAC layer and does nothing. It is a pass-through protocol so no packet transmission nor reception is made. This driver assumes that there is no collision occurred in the network therefore packet transmissions are instantaneous. This assumption of NullMAC driver may increase performance in some devices and it is very useful for testing and comparing the test scenarios (MAC protocols in ContikiOS, 2014). But for this thesis work, using this layer may create confusion in the interpretation of the results. Therefore, NullMAC layer has been removed from the test scenarios.

3.1.3.4. Adaptation Layer

Because IPv6 requires connections to be supported with a 1280-byte MTU (Maximum Transmission Unit), an adaptation layer required for the fragmentation and reassembly of the packets under the network layer of IEEE 802.15.4 links with a 127-byte MTU. Adaptation Layer (also known as 6LoWPAN Adaptation Layer) provides header compression, link layer forwarding when multi-hop is used by the link layer, packet fragmentation, and reassembly. Like IPv6, the 6LoWPAN adaptation layer also uses header stacking. This layer currently supports mesh addressing header, the fragment header, and the IPv6 header compression header (Dunkels & Vasseur, 2010).

3.1.3.4.(1). 6LoWPAN

The reason of 6LoWPAN is lightweight and it allows to ubiquitous connectivity and interoperability between different devices, it leads this protocol is now considered as de facto standard for IoT networks (Gomes, Salgado, Pinto, Cabral, & Tavares, 2018). The key idea behind the creation of this standard is IP should be applied to even the smallest devices (Mulligan, 2007) and this standard allows packet transmission over IEEE 802.15.4 based networks which uses 127 octets frame size.

The devices in a network that use 6LoWPAN and IPv6 must include border router to communicate outside because of header compression and different link layer protocols.

3.1.3.5. Network Layer

At the network layer, routers play a key role. The packages at this layer called datagrams which contain the information about the source and destination addresses that the router must know. This layer uses a routing protocol to forward datagrams to share information between devices. A routing protocol is very important because choosing the path must be at an optimal level (Rayes & Salam, 2017).

3.1.3.5.(1). IPv6

The number of devices connected to the Internet exceeds the capacity of Ipv4. Ipv4 can address 4.3 billion address and it is insufficient for addressing all of the devices in the world's network. But IPv6 uses 128-bit addressing (instead of 32 bit addressing of IPv4) which can address the 667 sextillions of device. IPv4 and IPv6 are not designed to work together so there are some transition mechanisms (e.g., NAT) to allow communication between this type of host. Because the number of devices in IoT is expected to be huge, the IPv6 is the preferred protocol in these networks (Rayes & Salam, 2017).

The advantages of IPv6 are more than that of IPv4. For example, the security, mobility, and configuration of devices have been considered when designing the IPv6. In addition to addressing larger spaces, IPv6 has some optimizations about address allocation, service delivery, and multicast addressing (Rayes & Salam, 2017). Also, the NAT problem is eliminated with IPv6.

3.1.3.5.(2). RPL

RPL (Routing Protocol for Low-Power and Lossy Networks) is a routing protocol that is responsible for discovering the neighbors using distance vectors and selecting a suitable path (Winter, et al., 2012). It is implemented at the top of the physical layer (IEEE 802.15.4) and MAC layer in two versions named RPL Lite and RPL Classic in Contiki-NG's IPv6 network stack and a standard mechanism for IoT networks. Using this protocol is inevitable when it comes to routing within the IoT networks and routing between the IoT networks.

The distance vector is a measurement of a direction costs of neighbor nodes (hops) or destination and determines which path will the packet goes on fastest. It maintains the routing table using some metrics like time, hops in the path, and using the information from the neighbors. Using these mechanisms, the network decides using which hops the packet will go through to destination with minimum cost (Pham & Sylvie, 2003).

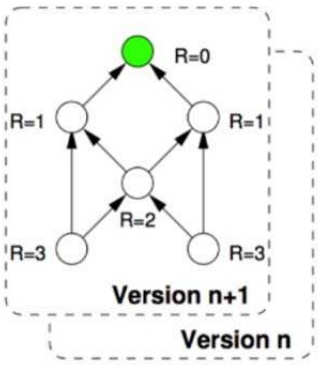

Figure 3.7. RPL Rank Mechanism (Kumar, 2018)

In addition to these, RPL uses another mechanism that has a tree topology and called Direct Acyclic Graph (DAG) or Destination-Oriented Direct Acyclic Graph (DODAG) (Duquennoy, Documentation: RPL, 2019). In Figure 3.7, the rank system in DODAG version is illustrated.

For the tasks that explained below, RPL uses Objective Functions (OF). It is very important to select the Objective Function that will be used by RPL to determine the best path because this function represents a solution about routing path based on Objective Function type.

3.1.3.5.(3). Objective Function Zero

Objective Function Zero (OF0) is designed with the consideration of RPL specifications and its main purpose is to add a node to a good connectivity provided DODAGs. When this type of DODAG root is detected, OF0 marks that root as "Grounded" and then looks for the nearest Grounded root in order to create a suitable routing path (Thubert, 2012).

OF0 uses a ranking system for neighbor nodes and it calculates and updates the rank using some specified metrics. Because of the main concern of OF0 is offering good connectivity, in some cases, the resulting hop-count and ranks are causes low connectivity. For these reasons, it is preferred that "The Minimum Rank with Hysteresis Objective Function" can be an alternative and it can offer better rank stability than OF0 (Thubert, 2012).

3.1.3.5.(4). The Minimum Rank with Hysteresis Objective Function

The Minimum Rank with Hysteresis Objective Function (MRHOF) is focused on minimizing the metric at the event of selecting the route and the rules are described in RFC 6719. In comparison to other OFs, this function uses extra metrics like latency, minimum rank, and expected transmission count (ETX), i.e., through a route. Before using the MRHOF, the desired metric must be selected first. For example, if the latency metric is selected, then the RPL will try to find the

lowest latency path. If no metric is selected, then MRHOF will use the ETX metric by default (Gnawali & Levis, 2012).

3.1.3.6. Transport Layer

The task of the transport layer is controlling and checking packet traffic, integrity, congestion, and reliability. This layer can be implemented with two protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Because of ensuring the compatibility of test environment components and memory overflow issues, TCP is disabled and is not used in test scenarios in this thesis.

3.1.3.6.(1). UDP

UDP is a transport layer protocol and it is mostly used for streams because it has no mechanism for reliability and connection like TCP includes. It is a connectionless protocol and it uses very little mechanisms. With this feature, it is lightweight and faster than TCP, and packet transmissions are very relaxed (Postel, 1980). For these reasons and the other protocol requirements, UDP is used in the test scenarios. The figure given below depicts the message format of UDP.

Figure 3.8. UDP Message Format (Kozierok C. M., n.d.)

The working mechanism of UDP is very simple. If a packet comes from the application layer to the transport layer, the data of this packet is encapsulated to the data section of UDP datagram. Then, source address, a destination address, length of the datagram and calculated checksum values make up the header of the datagram. Lastly, this datagram is sent to the network layer for transmission. These processes are unreliable (no guarantees the successful transmission) because UDP is connectionless and does not use a handshaking mechanism (Kozierok, 2005). These features of UDP makes the transmission faster. Also, the UDP header is only 8 bytes, there is more room for data in datagram, unlike the TCP.

3.1.3.7. Application Layer

The application layer is the seventh and topmost layer that provides services and applications to users interact with. In IoT networks, application layer protocols must use the resources of devices as little as possible. Using HTTP is also possible but this protocol consumes more system resource. Convenient protocols must be suitable for IoT requirements and in comparison, to HTTP, they must use the resources much less.

3.1.3.7.(1). CoAP

This protocol is specialized for IoT networks and WSNs; and it is defined in IETF RFC7252. It is implemented for constrained devices that can communicate with each other on the same or different networks. On networks that are created with constrained devices, it is important to provide simplicity, multicast, and low overhead. CoAP is a protocol which allows these requirements and mostly run on UDP networks (Roman & Lopez, 2009).

There are 4 message types of CoAP that can be listed as:

- CON (confirmable) is used for creating reliable communication when requested.

- NON (non-confirmable) is used for creating non-reliable communication.
- The CON message needs an ACK (acknowledgement) message but NON is not.
- If the CON message is failed, the RST (reset) message is sent.

To operate the RST message, an exponentially timeout time method is used. This method is called as exponential back-off mechanism. When the message gets no response, then retransmission time is calculated with this mechanism. There are six CoAP implementations in Californium.

The default CoAP mechanism is similar to the TCP congestion control function, but it is designed to use UDP for CoAP message transfer. In this function, clients have to wait for the ACK message. However, this mechanism is not suitable for Observing Resource (Obs) congestion control. Therefore, improvements have been made on it (Khunboa & Suwannapong, 2019). The CoCoA congestion control mechanism is an algorithm standardized by the IETF CoRE workgroup. This algorithm offers a Variable Backoff Factor (VBF) with a novel Round Trip Time estimation technique for a more controlled and dynamic RTO (Betzler, Gomez, Demirkol, & Paradells, 2016). CoCoA Strong (also known as CoCoA-S) is the version of this algorithm that uses a powerful RTO estimator and only allows strong RTT measurements.

Linux RTO was created by adding 2 mechanisms to the TCP RTO algorithm. These 2 mechanisms are applications that depend on the relationship between RTT and RTO. The first of these does not increase RTO if the current RTT is smaller than the previous RTT. Second, Linux RTO avoids the RTO estimator after repeatedly measuring constant RTT values (Betzler, Gomez, Demirkol, & Paradells, 2016).

While determining the test scenarios, it is primarily aimed to show the contribution of the study to the literature effectively. While selecting congestion

control mechanisms, by reviewing the literature, the algorithms that are tested many times, gave consistent results, and have stable working behavior are used. Therefore, in this thesis, four of them, Default CC, CoCoA, CoCoA Strong, and Linux RTO are used and Peak-Hopper-RTO and Basic RTO are not used.

3.2. Methods

This thesis covers the examination of the relationship about congestion between CoAP congestion control mechanisms and RPL's objective functions. Therefore, the main requirement for the simulations is some pre-implemented algorithms for CoAP and RPL. CoAP and RPL implementations are created with different programming languages and different software developing platforms. These different systems must work together so focusing on this task, some preparations are done using the Linux operating system which is suitable for this kind of task. These preparations will be explained in the sub-sections below in detail.

In this section, all the preparations for the simulations are described sequentially in sub-sections below. First of all, the newest version of the Contiki operating system is fetched from the relevant GitHub page and installed on the virtual machine. After the installation of the operating system is completed, some attempts to ensure that the scenarios run smoothly are performed, and all possible errors listed. Then all issues are cleared before running the simulations. After the vanishing of the errors, the necessary protocols for all scenarios are determined and these protocols are set up with the help of some scripts to run automatically during the simulations.

After the steps above, the test environment Cooja where the simulations are performed is set up to make the locations of the nodes and the creation of topology automatically. After all the preparations, a few test scenarios were run randomly and it is understood that the whole system is ready to use.

3.2.1. Preparations for Contiki-NG OS and Cooja Network Simulation Tool

As mentioned in the previous sections, simulations are decided to be done with Cooja, a powerful tool that runs on the Contiki-NG operating system and developed specifically for network simulations. Ubuntu operating system is recommended for the Cooja network tool and many other things such as Python and shell scripts to be used in the simulation to work more stable. In addition, it is better to host the Ubuntu operating system with a virtual machine in terms of flexibility. For this reason, using the VMWare program, the latest version of Ubuntu 18.04.3 LTS operating system was installed. A 64- bit operating system was chosen for the operation of the msp-gcc 4.7.3 compiler, which is used due to more memory support. In the same way, while the studies are in progress, the most recent version of Contiki-NG Release v4.4 was fetched from the relevant GitHub page and made the necessary configurations to be ready for use. All configurations made are examined in the next sections.

After the preparations required for the operation of the server devices to be used in the simulations were finished, the preparations were started on the client-side. Since Californium implementation will be used as a client and this project is prepared with JAVA, Eclipse program, which works stably on Ubuntu operating system and can be downloaded and made ready for use by using Ubuntu Software Center is used. This version was used as Eclipse IDE 2019-12 is the current version of Eclipse at the time of testing. The use of recommended and up-to-date versions for all programs is important for the stability of the tests.

After all the installations are completed, the Cooja program will be able to run smoothly. By opening a terminal in the folder where the Cooja program is, Cooja can be started with the command “ant run”. However, based on the previous studies, the “ant run_bigmem” command, which provides more memory support, was used instead of this command to prevent the Cooja program from leaking memory and causing different errors.

After opening the Cooja, a new simulation can be created and simulation can be done, and this simulation can be saved for later use. However, since the devices will be compiled with different protocols in the thesis study, simulations will not be saved and will be created continuously from scratch. Given the number of simulation scenarios to be made and the configurations that will change constantly, there are some scripts that have been written to create all simulation scenarios, since it will take a lot of time to repeat these processes continuously. These scripts are summarized in the next section.

3.2.2. Preparations for the Possible Errors

In order to use Cooja, there are a lot of issues that must be solved. In Contiki-NG, RPL border router and CoAP server examples exist, but these examples use memory relatively a lot (especially CoAP server) and when it comes to compiling and creating the motes (which are ready to use in Cooja) for simulation, RAM and ROM overflow errors arise. The reason for these examples of giving errors is that both the protocols and features that will be applied in the tests use memory a lot and the Z1 device to be employed in simulations has limited memory. To solve this problem, we need an advanced compiling system and new configurations. Upgrading the compiler and arranging the configuration files to solve these issues are explained below.

3.2.3. Upgrading msp-gcc to More Memory Support

In the Cooja, there are several predefined motes which are ready to use. Because of the code size limitations ofZ1 and Tmote Sky platforms, these are excluded from coap implementations. These motes are not allowed to be used at Cooja with coap examples. If used, the compiler will give an error indicates that the current msp-gcc version is not compatible with this type of operation. msp-gcc is a GCC (GNU Compiler Collection) toolchain which is used for compiling codes for motes that are employing Texas Instruments MSP430 architecture.

To solve this problem, the msp-gcc version must be upgraded to at least 4.7.0 version because it supports more memory (up to 1MB) and utilizes 20-bit registers and address space (Thébaudeau, MSP430X, 2019). In this work, msp-gcc is upgraded to version 4.7.3 and used.

3.2.4. Makefile Configurations of Coap Example Server

After upgrading the msp-gcc version, excluded motes must be included. To include these motes, related Makefile (which is for coap-example-server) must be changed. In this file, there is a section that excludes Z1 mote:

PLATFORMS_EXCLUDE = sky, z1

This line must be changed as the following for including Z1 mote and allowing msp-gcc to compile:

PLATFORMS_EXCLUDE = sky

After these changes, the Z1 mote is now allowed to compile.

3.2.5. Project Configurations

Because of using constraint devices on the simulator, memory capacity is very small and must be used efficiently. Any small detail will cause a buffer overflow. For example, if we try the run coap-example-server.c with only the above configurations, the compiler would say RAM or ROM overflow happened. To avoid this type of error, we must detect the configurations that use memory and could cause an overflow. According to the Contiki-NG wiki page (Oikonomou, 2018), reducing RAM and ROM usage with some configuration is possible. In this work, some of these configurations like disabling TCP, adjusting the log level to none with LOG_LEVEL_NONE parameter, and constrained the network parameters are made and RAM and ROM overflow are prevented.

After makefile configuration, compilation would still give an error that is related to the UIP_CONF_BUFFER_SIZE parameter, because it is now too small for COAP_MAX_CHUNK_SIZE parameter. UIP_CONF_BUFFER_SIZE is

changed to 140 by following the information on the Contiki-NG wiki page because it is indicated that lowering this value to 140 is appropriate in the case of using no large datagrams (Oikonomou, 2018).

Project configurations can be done using corresponding project configuration header files or makefiles but to keep track of these configurations, it is better to doing these arrangements in project-conf.h file which is responsible for the coap-example-server.c file. In addition to the previously described configurations, other configurations have been added to the project-conf.h file to reduce memory usage. By following the information on the Contiki-NG wiki page, project-conf.h file is arranged with the following variables:

- COAP_MAX_CHUNK_SIZE: 16
- NBR_TABLE_CONF_MAX_NEIGHBORS: 8
- NETSTACK_MAX_ROUTE_ENTRIES: 8
- QUEUE_CONF_NUM: 4
- UIP_CONF_BUFFER_SIZE: 140
- UIP_CONF_UDP: 1
- UIP_CONF_TCP: 0
- UIP_CONF_UDP_CONNS: 8
- UIP_CONF_TCP_CONNS: 0

Using project-conf.h file, we can change the protocol of each layer. For example; for the data link layer, we can change the MAC protocol types such as CSMA, NullMAC, or TSCH. In this thesis, determined protocols of layers will be changed and the effects of these changes will be recorded. Changing of these protocols are listed below:

Parameter	Option 1	Option 2
NETSTACK_CONF_RDC	nullrdc_driver	contikimac_driver
MAKE_MAC	MAKE_MAC_CSMA	
MAKE_NET	MAKE_NET_IPV6	
MAKE_ROUTING	MAKE_ROUTING_RPL_LITE	
RPL_OF_OCP	RPL_OCP_MRHOF	RPL_OCP_OF0

Figure 3.9. Parameters of different network layers

3.2.6. Creating and Configurating the Border Router

A border router creates an interconnection between two different systems and operates the packet forwarding between them (Liu, 2009). In this work, the border router plays a key role in connecting the virtual simulation environment to the real environment. Using the Tunslip interface, border router gets a global IPv6 address connecting through a serial server socket. When the simulation is started, it assigns IPv6 addresses to other motes that are in the network and detects the neighbors and routing links.

Creating the RPL border router is very simple and one of the creating steps can be seen in Figure 3.10. Once the simulation is added to the Cooja simulation platform, the border-router.c file which is under examples/rpl-border-router directory must be used for compiling. After compilation finishes, the border router is ready for including to the network.

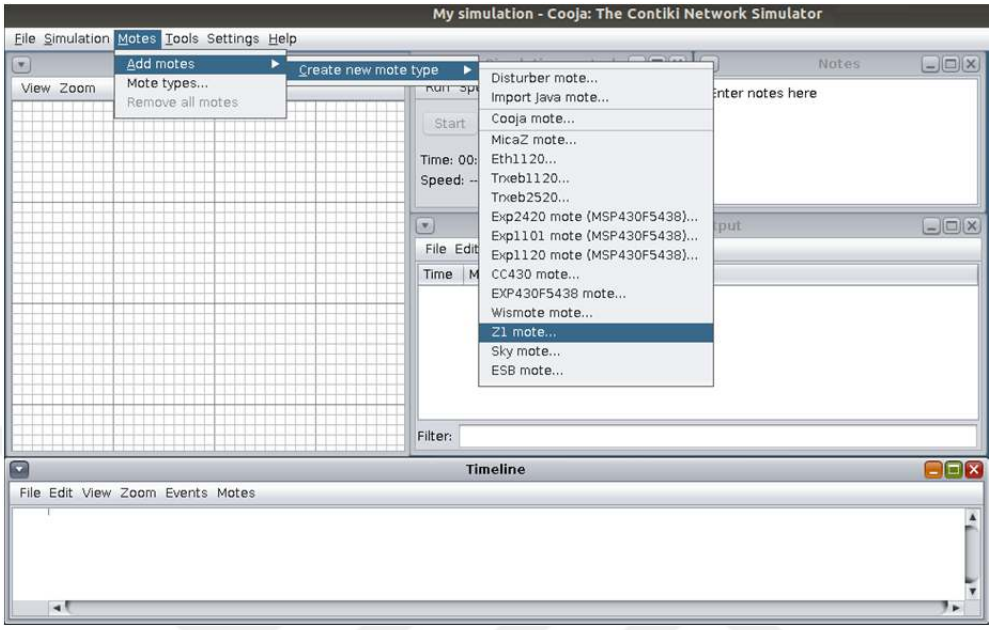

Figure 3.10. Adding a mote to the Cooja

When the border router is added to the network, there are several important setups that must be checked out. These setups will be explained in the next section.

3.2.7. Cooja Configurations

There are a lot of configurations that must be checked out to perform successful simulations on Cooja. First is adding a serial socket (server) to the border router. Because Tunslip will use this socket and border router will listen to the port number that is set on this socket to connect the Tunslip. Secondly, all motes must be in the transmission range in their neighborhood which can be determined on Cooja at the Network window. Also, one must be careful to place the motes that cannot be in their interference range. As a result of some researches, we decided that the distance between the devices can be 30 meters.

Perhaps one of the most important settings is getting the speed limit on Cooja as 100%. In this way, the tests can be simulated in real-time and more

realistic results can be obtained. Lastly, the TX/RX success ratio should be set to PDR value. This is also important for obtaining realistic results.

3.2.8. Using Objective Functions

In the Contiki operating system, there are two different RPL Objective Functions that are implemented by developers. The default objective function is MRHOF but depends on different requirements, it can be easily switched to OF0 (Objective Function Zero). To switch this, firstly the rpl-conf.h file which is in the network folder of Contiki must be modified. In this file, there is a line that determines the supported objective functions. By default, this parameter supports only MRHOF:

```
#define RPL_SUPPORTED_OFS {&rpl_mrhof}
```

To supporting OF0, we need to modify this line as follows:

```
#define RPL_SUPPORTED_OFS {&rpl_of0, &rpl_mrhof}
```

Lastly, we must add the RPL_CONF_OF_OCP parameter to the project-conf.h and set the desired OF. To use MRHOF, set this parameter to RPL_OCP_MRHOF, or use OF0, set the parameter as RPL_OCP_OF0.

3.2.9. Creating the Network Topology

The created network topology is grid topology and there are 10 nodes including one border router. The distance between the nodes is 30m and all of them are their radio coverage and they are not in their interference range. In network topology, the transmission range is 50m and the interference range is 100m.

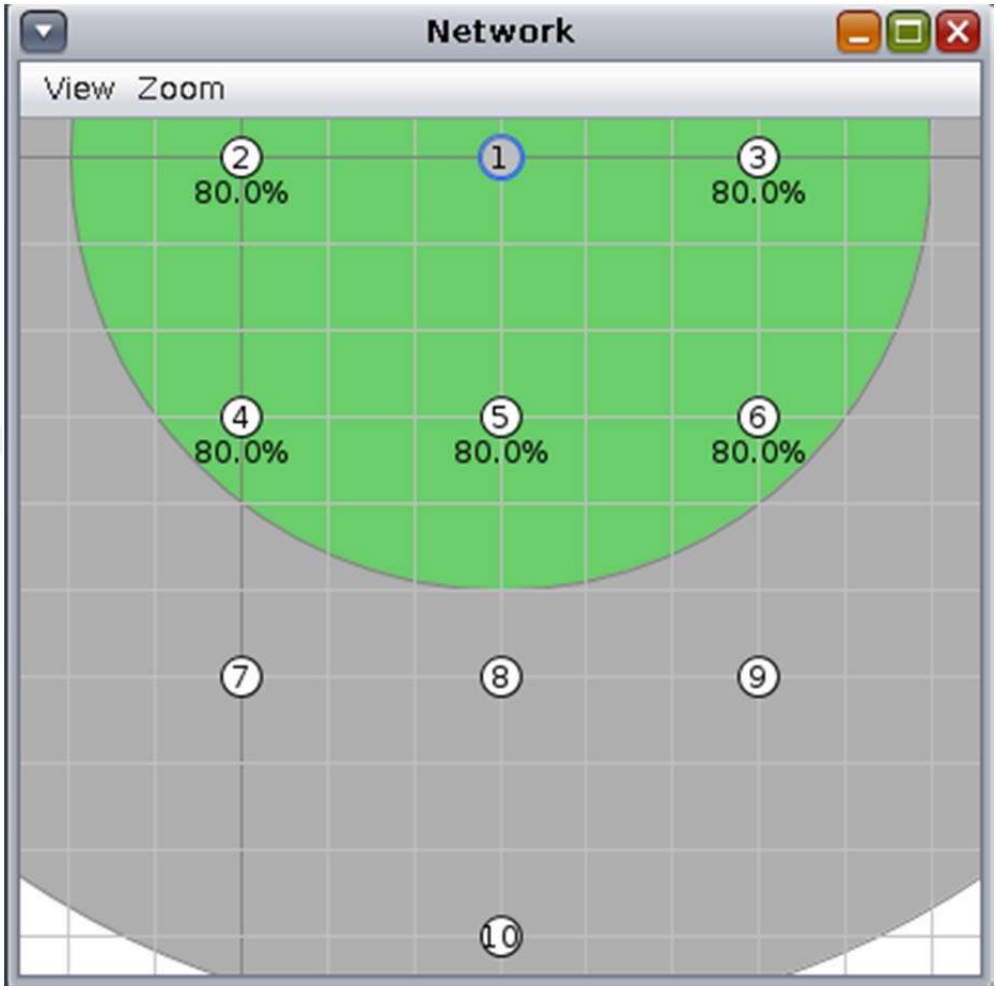

Figure 3.11. Network Topology

In the topology given in Figure 3.11, the mote numbered as 1 is border router and its coordinates are given especially. In this figure, the green area represents transmission range and the gray area represents the interference range. The ratio of 80.0% represents the TX ratio.

3.2.10. Creating Scripts for Different Test Scenarios

Since a large number of protocols and configurations will be used in simulations, some scripts have been created to make it easy to follow simulations

and configure them more quickly. All the scripts are modular and prepared to do a single job, providing flexibility and when necessary, running multiple scripts under a single script saves a lot of time. For example, 18 different test results can be obtained with a single run of this script. However, to get these results without using scripts, configurations would have to be adjusted 18 times manually.

Since all the protocols are located in the project-conf.h file or in different Makefile files, these scripts usually manipulate the text files so that the required configuration is included in where it should be written. For this purpose, scripts were written for each protocol. In addition, by opening Cooja, a different script was written to automatically compile all devices for the relevant scenario and place them in the coordinates determined on the topology.

All scripts are specially designed to separate repetitive jobs and provide a modular work order. As all tests can be implemented with a single script, it reduces the control over the simulations and makes it difficult to intervene when necessary. The scripts will have to be done in parts otherwise it will be necessary to perform the entire test from the beginning when implemented with a single script. The written scripts are summarized below:

- cooja: Runs the Cooja simulator.
- border: Adds a border router to Cooja. Since the location and codes of this node are different, this script has been written separately.
- coapserver: It ensures that all coap servers are compiled and created with the relevant code in the specified number and in the specified location, and placed in the simulation environment.
- contikimac, nullrdc: Sets the RDC layer protocol to ContikiMAC or NullRDC.
- ofzero, mrhof: Sets the objective function to OF0 or MRHOF.

- pdr80, pdr90, pdr100: It provides adjustment of PDR value (TX rate in Cooja) to 80, 90, or 100 in the simulation environment.
- setspeed: It sets the speed limit of the simulation to 100%, allowing the tests to be carried out in the real time period.
- socket: It ensures that the socket required for the Tunslip program is opened over the border router.
- tunslip: Opens Tunslip program with related parameters.
- startsim: Starts the simulation.
- simulate: This script is designed to make all the above scripts run consecutively. All scripts are written according to the order of operation and designed to allow changing when necessary.
- run3, run9: It is written so that 3 or 9 clients can work together to pull data from different IPv6 addresses at the same time.
- run: In this script, 3 and 9 clients are provided to work a certain number of times at certain time intervals. run3 and run9 scripts are used.

It is enough to run the Cooja, simulate, and run script to perform a test. Thus, 18 simulations start to produce results by working consecutively. The produced results were collected using different scripts. These scripts are written in Python programming language.

- read1: It holds the data from the first run.
- read2: It holds the data from the second run.
- read3: It holds the data from the third run.
- read: It provides the results of all runs in order and keeps them in a csv file.
- clearresult: It deletes all the results obtained from the previous runs which are stored as the csv file, and makes it ready for a new run.

- cleardata: It deletes the results of each client from previous runs, making it ready for a new run.

Since there is no standard for the measurement and evaluation criteria used for such experiments in the literature, the working behavior of the tests is determined by inspiring from the studies that are tried and obtained accurate results. Therefore in the tests to be carried out, the working time of the tests determined based on easy interpretation of the data and generates meaningful results to evaluate. Similar studies related to these tests examined and certain periods are determined at the stages in the tests. Three of the Californium clients will be run at the same time for 3 minutes, after 1 minute sleep time, then 9 of the Californium clients will be run at the same time for 3 minutes. In terms of the accuracy of the results, this process will be renewed 3 times, and after every 3 minutes of operation, there will be a waiting period of 1 minute for the packet traffic in the network to stop and stabilize completely and to avoid noise in the results. In order to do all these operations, different scripts were developed and the scenarios were run automatically in a row.

In terms of the accuracy of the results, this process will be renewed 3 times, and after every 3 minutes of operation, there will be a waiting period of 1 minute for the packet traffic in the network to stop and clearing the queues on the nodes completely and to avoid noise in the results. In order to do all these operations, different scripts were developed and the scenarios were run automatically in a row.

3.2.11. Determining the Metrics

To analyze the performance, there are a lot of metrics in the literature. For these simulations which include congested networks, two metrics which are throughput and average delay are used. These metrics support us to evaluate the network performance considering congestion.

3.2.11.1. Throughput

In the simulations for the thesis, 9 sinks and one border router were used. 9 sink nodes producing data continuously and serves the data to Californium clients via the border router. The amount of data that can be transferred to the client per unit time is the most basic measure in detecting the congestion. Therefore, throughput has been calculated to be used as a performance metric in these simulations.

The successful data transfer that takes place per unit time on a network is called throughput. Based on this definition, throughput is calculated on the Californium client using the JAVA programming language. Since 3 or 9 clients will be used in the simulations, more than one throughput value has been taken. Also, since each simulation is repeated 3 times, the number of throughputs received will be triple. Therefore, the throughput value of the system is accepted as the average of these three values. In cases where a client cannot receive any messages, the data of this client are ignored.

This metric is very important as it is learned from the throughput values whether the system is working properly. For example, TX value which determines the packet delivery ratio of the system on the simulator will be changed according to scenarios. As this value decreases, the throughput is expected to decrease. As the number of clients trying to fetch data at the same time increases, the throughput is expected to decrease. Considering these, it is examined individually whether the system is working correctly for each scenario. While large throughput values indicate that the system can transfer data easily, it is understood that the system cannot transmit data well as the value decreases.

3.2.11.2. Average Delay

Average delay is one of the most important performance measurement units that can be defined as the time it takes for a package delivering from source to destination. Delay-related measurements are very effective in determining the

performance of a network and accordingly and the selection of elements that creates the network such as the routing algorithm and flow control.

4. RESULTS AND DISCUSSIONS

In this section, the performance analysis of the simulated networks is presented. Because of the Contiki operating system use C programming language, new code blocks and parameters are created with this language. Also, Linux shell script (also known as bash script) and Python programming language were used to run the simulations and collecting the results.

These experiments were performed on VMware Workstation 12 Pro using Linux based operating system Ubuntu as a virtual machine. According to the requirements of tools, Ubuntu 18.04.3 LTS operating system were created with 4 GB of RAM and use the 8th generation Intel Core i5 1.60 GHz with two processors.

The experiments consist of seven main parts:

- Choosing of an operating system for IoT network
- Protocol determination for all layers
- Determining the topology and virtual mote
- Editing the configuration files
- Determination of performance metrics
- Programming clients for performance metrics and congestion control mechanisms
- Creating scripts to run simulations and collect results

To sum up, the parameters used in these simulations are as follows:

- Application Layer: Default CC, CoCoA, CoCoA Strong, Linux RTO
- Network Layer: RPL’s OF0, RPL’s MRHOF
- Transport Layer: UDP
- MAC Layer: CSMA

- RDC Layer: ContikiMAC, NullRDC
- PDR: %80, %90, %100
- Number of clients: 3, 9

The performance metrics used in these simulations are as follows:

- Throughput: The number of packets received successfully from the sink in a given time period.
- Average Delay: The average elapsed time between all CoAP requests and CoAP responses in one run. Only successful requests are calculated, failed requests are ignored.

There is 96 simulation scenarios, and each of them takes three minutes. To measure the metrics accurately, each simulation must be repeated at least 3 times so 288 test scenarios were operated. Also, between each simulation, there are 1 minute sleep periods to ensure that the network is completely stable.

In the next section, the data obtained from simulations will be examined from different perspectives. In the charts created with these data, the light-colored graphics columns show the tests done with OF0, and the dark-colored ones with MRHOF. For chart elements of the same color tone, the columns on the left show ContikiMAC, and the columns on the right show tests with NullRDC.

4.1. Throughput

Throughput is a frequently used metric to measure and evaluate the performance of a network. With this metric, the data transmission speed of devices in a network can be measured. In the network, the faster a device sends data, the more consistent the data the system will collect and the calculations made with this data will be more detailed and accurate. In addition, fast data transmission enables

faster releasing the resource space on the network for other devices and make resources less busy.

After the simulations, all results are obtained and the throughput of the network was measured, and in which cases the network performed better was examined. When analyzing the throughput results obtained from simulations, only successful packet transmissions were taken into account, and lost packets were ignored when making calculations.

THROUGHPUT (req/sec)													
CC - Client		OF0 / ContikiMAC			OF0 / NullRDC			MRHOF / ContikiMAC			MRHOF / NullRDC		
CC Algorithms	PDR / Client	80	90	100	80	90	100	80	90	100	80	90	100
Default CC	3	3.494	3.677	3.646	3.654	3.634	3.867	3.245	3.691	4.027	3.308	2.824	4.007
	9	1.023	1.244	1.333	1.104	1.274	1.348	0.802	1.196	1.401	0.663	0.777	0.943
CoCoA	3	3.649	4.144	4.640	3.935	4.296	4.764	3.186	2.710	4.819	2.558	3.510	5.019
	9	0.892	1.261	1.392	1.195	1.263	1.616	0.891	1.366	1.522	0.562	1.297	1.588
CoCoA Strong	3	4.206	5.084	5.646	4.080	4.896	5.260	3.052	3.461	3.798	4.420	4.878	5.616
	9	1.119	1.380	1.316	0.943	1.415	1.094	0.790	1.128	1.101	0.917	1.464	1.655
Linux RTO	3	4.859	5.064	5.945	4.307	3.820	4.094	3.785	4.846	4.029	4.107	2.060	4.542
	9	1.216	1.456	1.726	0.890	0.972	1.008	1.038	1.281	1.582	0.941	1.315	1.322

Figure 4.1. Throughput Result Table

Figure 4.1 shows the throughput of data obtained from all simulations. For each simulation, 3 throughput data were obtained if 3 clients were used and 9 if 9 clients were used. By taking the average of these data, the average throughput value obtained for related simulation scenario was calculated. Since each scenario was repeated 3 times, the results in Figure 4.1 were obtained by re-averaging these throughput values obtained for 3 reiterations. With the table given in Figure 4.1, we can easily see that how each variable affects the throughput data. There are some graphics prepared to support this table and interpret it more visually.

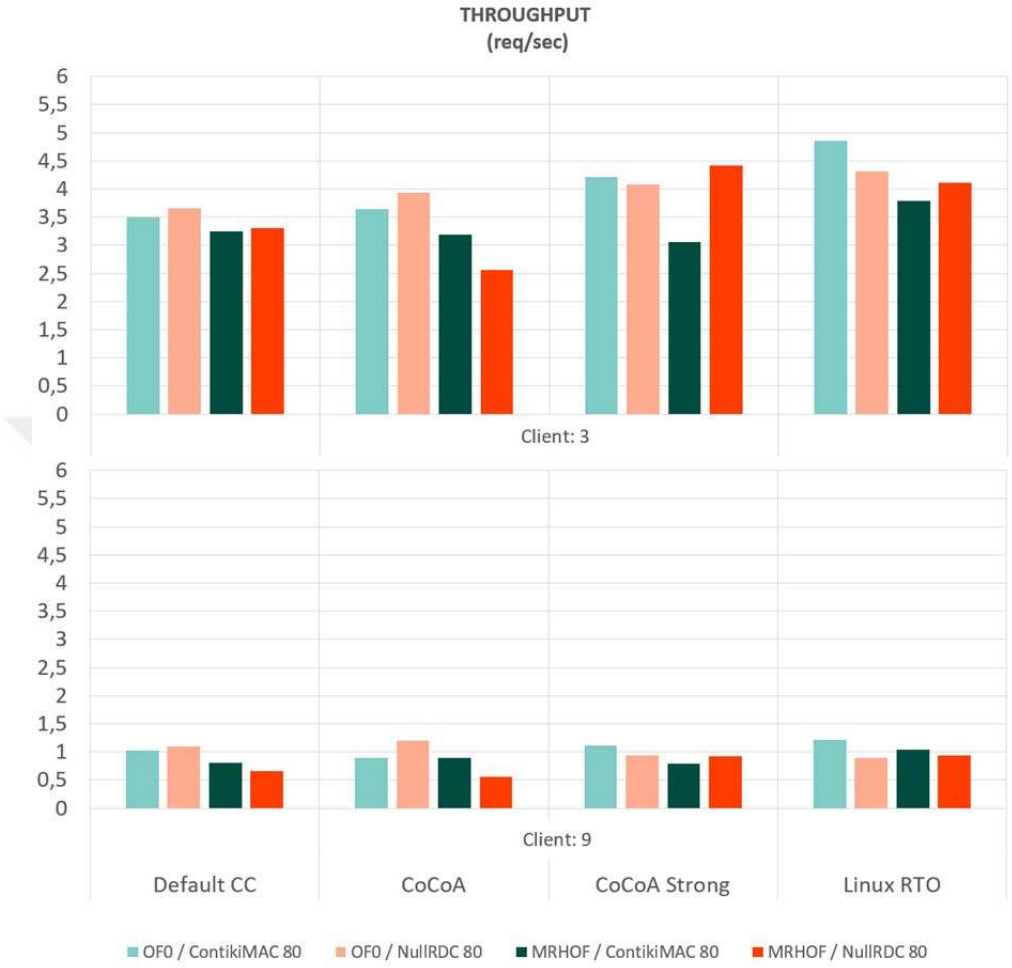

Figure 4.2. Throughput for PDR = 80

When the PDR value is 80 for 3 and 9 clients, the results are given in Figure 4.2. When this chart is analyzed, it can be seen that OF0 gives better results than MRHOF. Although CoCoA Strong and Linux RTO have very close values, Linux RTO stands out with a slight difference. When we compare RDC layer data, in general, scenarios where ContikiMAC is used, higher throughput is obtained compared to scenarios where NullRDC is used.

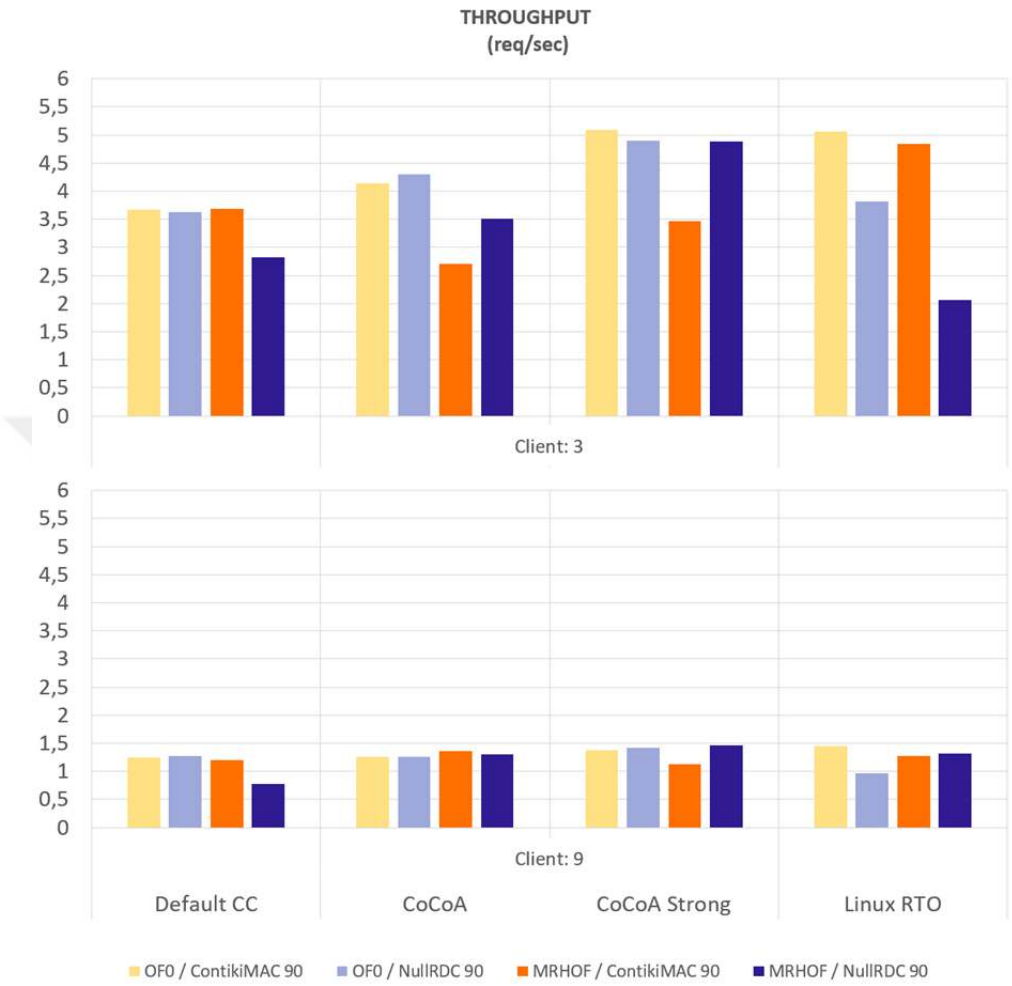

Figure 4.3. Throughput for PDR = 90

When the chart created with the data obtained from simulations where the PDR value is 90 is examined, it can be seen in Figure 4.3, OF0 is superior to MRHOF. When the congestion control mechanisms are examined, although they have very close values it can be seen that CoCoA Strong is slightly better than Linux RTO. Although the data in the RDC layer are similar, ContikiMAC stands out with a small difference.

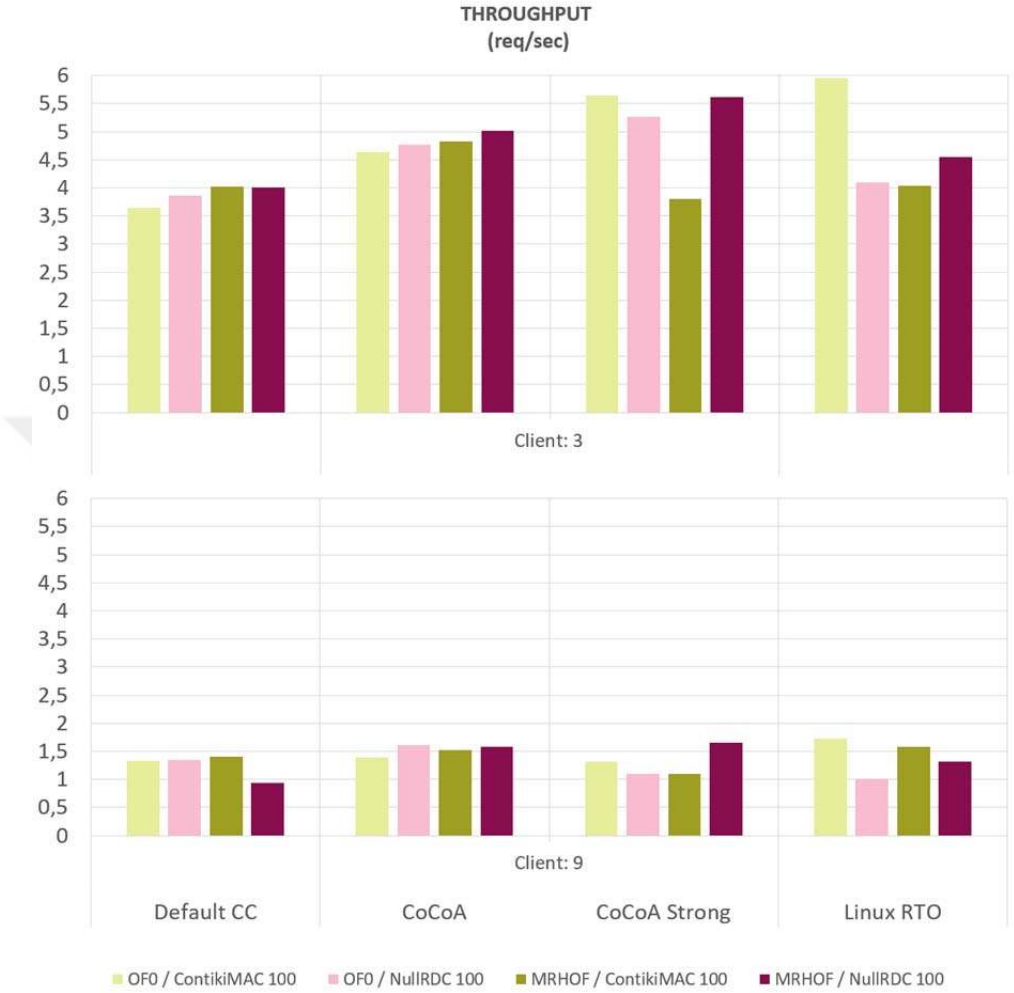

Figure 4.4. Throughput for PDR = 100

The Figure 4.4covers data where PDR = 100. The data obtained when the PDR value is 100 reveals a somewhat interesting situation. It is observed that Default CC and CoCoA pair, and Linux RTO and CoCoA Strong algorithm pair create similar shapes. MRHOF algorithm gave good results compared to OF for Default CC and CoCoA. When looking at the Linux RTO and CoCoA Strong algorithms, when NullRDC is used, MRHOF is superior to OF0. However, when using ContikiMAC, OF0 provided better performance than MRHOF.

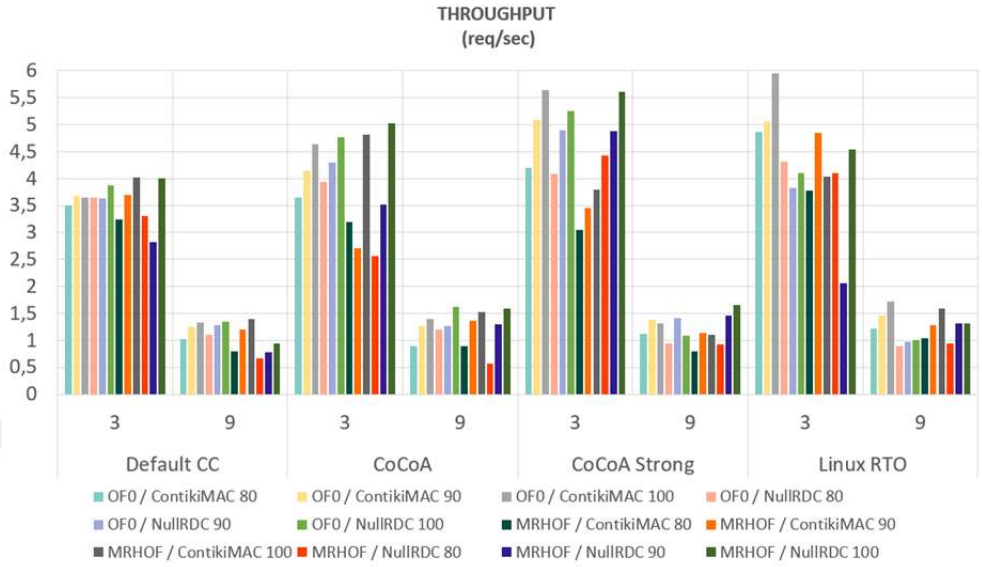

Figure 4.5. Throughput Results for Different Network Stacks

When we create a graph with the data which is in Figure 4.1, the graph in Figure 4.5 is obtained. With this graph, we can compare all simulation scenarios with each other according to their throughput results. As expected when the throughput of the network was examined, the highest throughput values were observed in the scenarios where 3 clients were running and PDR = 100, and as the PDR value decreased, the decrease in the number of packets received per unit time was almost a common feature for all scenarios. Again, the highest throughput values are seen in the scenario where the OF0 and ContikiMAC pair are used, whereas the lowest throughput values are realized in the scenarios including MRHOF and NullRDC using Default CC.

When we look at the congestion control mechanisms in general, it is observed that Default CC has the worst performance, and CoCoA Strong algorithm performs better than other algorithms. The Linux RTO algorithm has almost performed close to the CoCoA Strong algorithm, and the CoCoA algorithm comes after these two algorithms in terms of performance. In addition, the CoCoA algorithm is very close to CoCoA Strong in terms of performance in the scenarios

where 9 clients work together, and on average, CoCoA algorithm stands out with a slight difference from all congestion control mechanisms when the client number is 9.

OF0 performed better than the MRHOF algorithm when the congestion control mechanisms and other parameters were compared. When OF0 is used with ContikiMAC, it has been observed that NullRDC performs better, and the MRHOF algorithm performs better with NullRDC. In general, the ContikiMAC algorithm has been shown to give better results.

4.2. Average Delay

This metric is obtained by dividing the sum of the time elapsed for each data transmission by one minus the number of data transmissions. Calculating this metric helps us test the accuracy of the system, just like throughput. Similarly, in this metric, it is calculated on the Californium client using the JAVA programming language. We can understand that the system is working correctly, average delay increases when the number of clients which are fetching data at the same time increases.

AVERAGE DELAY (ms)													
CC - Client		OF0 / ContikiMAC			OF0 / nullRDC			MRHOF / ContikiMAC			MRHOF / nullRDC		
CC Algorithms	PDR / Client	80	90	100	80	90	100	80	90	100	80	90	100
Default CC	3	294.1	289.3	286.8	290.8	304.4	275.9	341.9	280.2	266.6	313.0	515.8	264.5
	9	1110.3	959.5	785.5	997.5	885.4	815.2	1890.8	914.7	790.0	1633.1	1524.7	1335.5
CoCoA	3	260.0	264.0	235.7	313.7	262.7	232.9	350.9	380.1	231.3	447.9	319.8	817.3
	9	1575.9	956.8	815.6	1183.4	887.4	677.4	1597.4	876.8	782.3	3048.0	1192.1	712.4
CoCoA Strong	3	238.3	195.1	177.8	232.9	205.9	190.3	368.5	320.0	280.4	229.5	245.0	194.4
	9	2312.0	1106.4	908.5	1465.6	769.5	989.3	1740.2	1027.4	1068.9	1999.7	879.4	709.4
Linux RTO	3	207.8	211.1	167.0	241.0	296.7	281.2	284.2	254.1	535.9	248.3	553.1	243.3
	9	1530.8	1647.1	1700.4	1998.9	1565.4	1312.8	2295.7	1768.5	923.9	2606.7	973.2	1012.2

Figure 4.6. Average Delay Result Table

Figure 4.6 shows the average delay data obtained from all simulations. For each simulation, 3 average delay data were obtained if 3 clients were used and 9 if 9 clients were used. By taking the average of these data, the average throughput value obtained for related simulation scenario was calculated. Since each scenario was repeated 3 times, the results in Figure 4.6 were obtained by re-averaging these average delay values obtained for 3 reiterations. With the table given in Figure 4.6, we can easily see that how each variable affects the average delay data. There are some graphics prepared to support this table and interpret it more visually.

Figure 4.7. Average Delay for PDR = 80

When the PDR value is 80 for 3 and 9 clients, the average delay results are formed in the chart given in Figure 4.7. When this chart is analyzed, it can be seen that OF0 cause lower average delay than MRHOF. When the congestion control mechanisms compared with each other, Default CC has lower average delay values than others. When we compare RDC layer data, in general, scenarios where ContikiMAC is used with MRHOF, the lower average delay is obtained compared to scenarios where NullRDC is used. In contrast, scenarios where NullRDC is used with OF0, the lower average delay is obtained even compared to ContikiMAC and MRHOF pair.

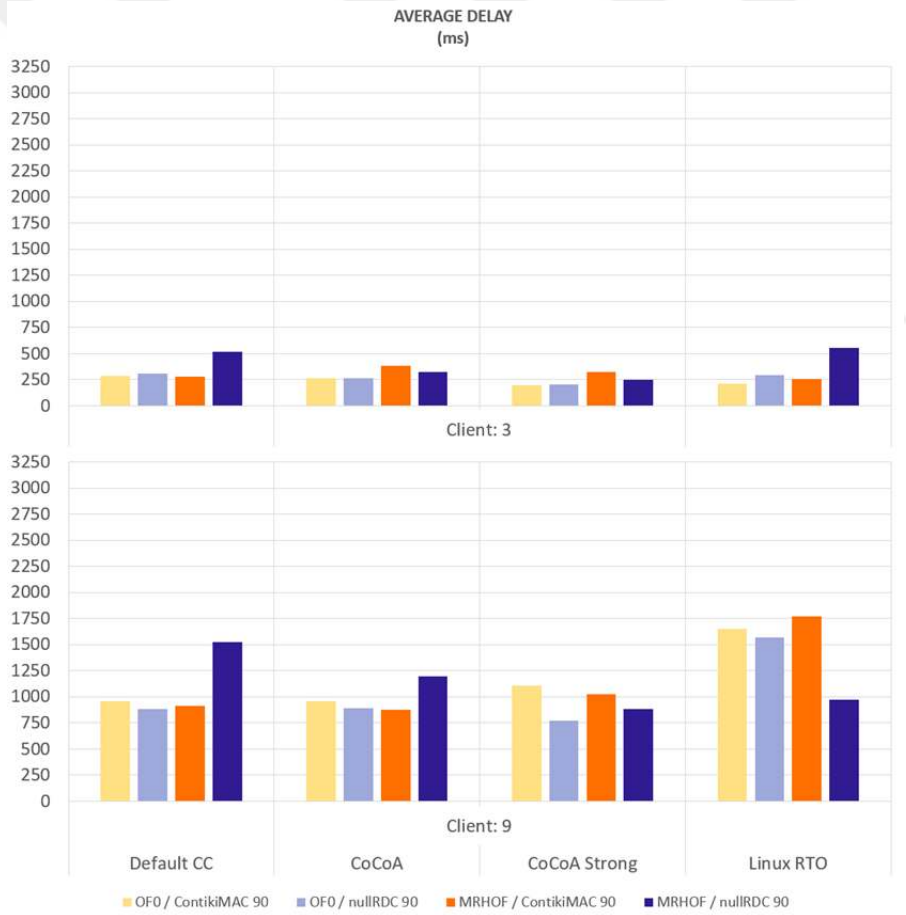

Figure 4.8. Average Delay for PDR = 90

If the PDR value is 90 is examined, it is seen in the Figure 4.8 that OF0 is superior to MRHOF. When the congestion control mechanisms are examined, although they have very close values it can be seen that CoCoA Strong is slightly better than other mechanisms. The data in the RDC layer are similar but NullRDC stands out with a small difference and better than ContikiMAC.

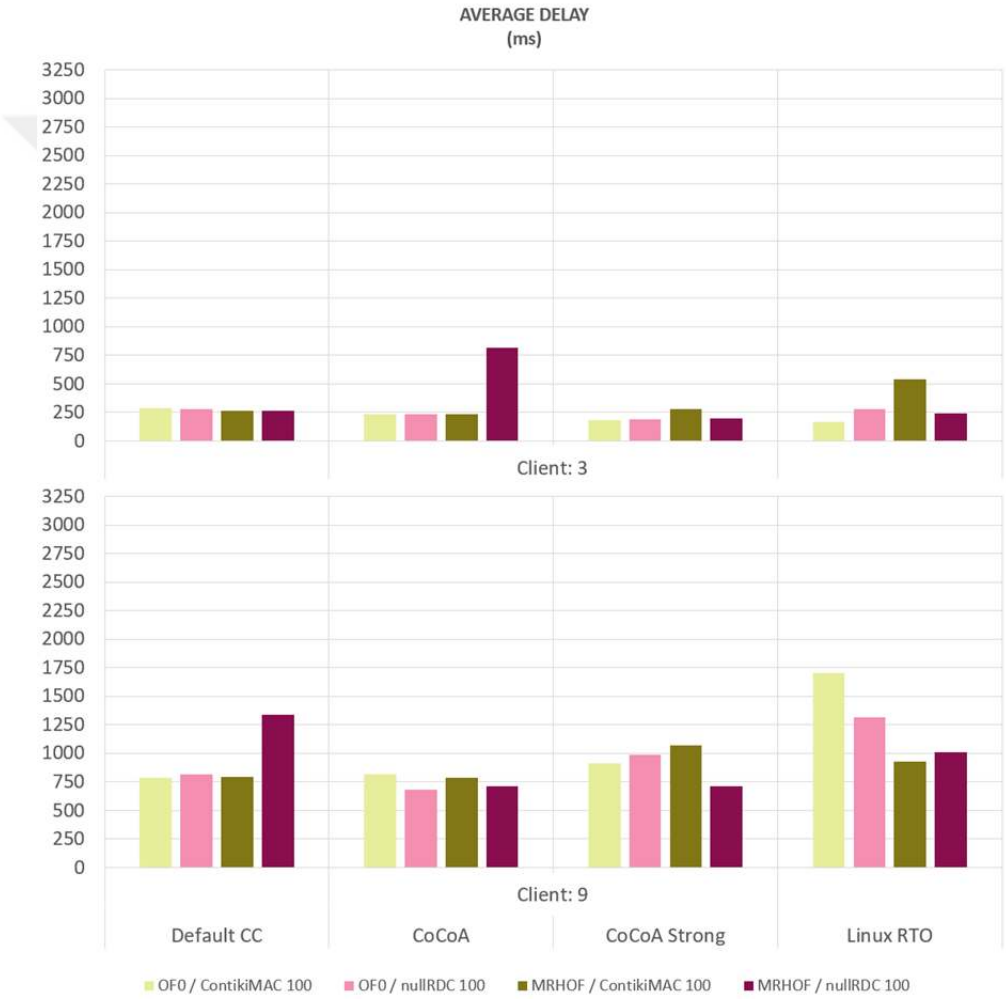

Figure 4.9. Average Delay for PDR = 100

The chart given in Figure 4.9 covers average delay data where PDR = 100. It is observed that OF0 algorithm gave good results compared to MRHOF especially seen for Linux RTO. When looking at the congestion control algorithms, NullRDC is generally has better performance than ContikiMAC.

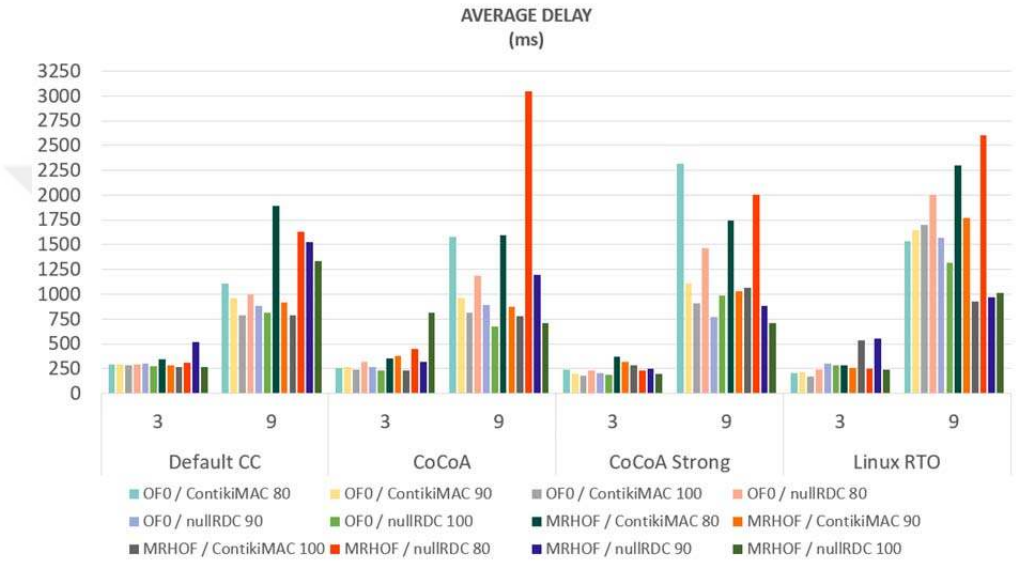

Figure 4.10. Average Delay Results for Different Network Stacks

When we convert the data in Figure 4.6, the graph in Figure 4.10 is obtained. With this graph, we can compare all simulation scenarios with each other. Although there are a few exceptions, the lowest average delay data was obtained in scenarios that 3 clients and PDR = 100. The attempt of 9 clients to pull data from the network at the same time caused quite a delay. In general, as the PDR value increases, the average delay time decreases as expected. All in all, the lowest average delay data obtained using OF0 and ContikiMAC, while the highest average delay was observed when using MRHOF and NullRDC.

When the congestion control mechanisms are examined, it is seen that CoCoA Strong has lower average delay values than other algorithms when the average of 3 and 9 client data is taken. Since the simultaneous operation of 9

clients increases the average delay unusually, 3 client data were examined, and it was seen that the CoCoA Strong algorithm had the best average delay values, followed by Linux RTO.

The performance of the objective functions was analyzed by comparing the average delay times, and OF0 was observed to create lower delay times than MRHOF. Especially, when using OF0 with 3 clients, ContikiMAC is better and when it is 9 clients, scenarios used with NullRDC gave better results. Although the results are quite close, NullRDC provided lower average delay times considering average data.

4.3. Relationship Between Objective Functions and Congestion Control Mechanisms

The objective of this study is to run RPL objective functions with all combinations with congestion control mechanisms and to investigate whether there is a relationship and how. When the results obtained from the simulations are evaluated, the best results in terms of both throughput and average delay are obtained when the CoCoA Strong algorithm is used. Again, when looking at all CC mechanisms, OF0 produced more efficient results than MRHOF. Due to its simplicity and using a fixed initial RTO value range, it was obvious that the Default CC algorithm will be underperformed. RTO calculation is very important because it distinguishes congestion control mechanisms. Because its RTO calculation method is powerful than the others, the CoCoA Strong algorithm is better than others.

When the client number is 3, CoCoA Strong gave better results than others, but when the client number is 9, CoCoA algorithm is better. Also results show that if the chosen congestion control mechanism is Linux RTO when the client number is 9, MRHOF is better than OF0.

The simulations in this study show that the use of OF0 in IoT networks will achieve more efficient results with all CC mechanisms. However, since the needs

of each network are different and it is known that the number of devices in the network to be installed has a significant effect on the results, each network stack needs to be revised according to its own needs.

4.4. Determination of Optimal IoT Stack

According to the results gathered from the tests, CoCoA Strong was found to be more efficient as the congestion control mechanism, and OF0 as the objective function, and ContikiMAC as RDC Layer was more efficient. However, this is a general situation, and since the tests included exceptional cases, it cannot be said that the use of these protocols will be correct for every network. Therefore, more than these tests are required to determine an optimal stack for the IoT network and should be based on the data to be obtained from the tests performed after these long periods of time. Since the requirement and setup of each network will be different, the network stack to be created must be installed accordingly.

The purpose of these tests is to compare the effect of congestion control mechanisms with objective functions. This study is not research about which protocol works best in which layer. But the results obtained from the experiments are examined and considering the algorithms that give the best results in each layer are compatible with the literature are taken as a criterion in terms of the accuracy of the tests. The number of tests is ideal for comparing congestion control mechanisms with objective functions. Similar congestion control mechanisms are not included in this study since they will give similar results and seeing the relationship will difficult when compared with the data obtained.

5. CONCLUSION

The congestion control and avoidance is still a major problem of WSNs and IoT networks in each stage of evolution of these networks. The aim of the IoT network is to connect and collect data from various sensors and devices. Because each device needs different requirements, traditional TCP congestion methods can be incapable. For these reasons, IoT networks need different protocols which are cooperatively work in each layer to handle the issue of congestion, packet losses, and data transmission insurance. In this work, by considering these problems, it has been investigated which objective function of RPL algorithm works more consistently with which congestion control mechanism and how the other parameters must be set.

In order to create simulations and combinations of protocols, recent works have been researched and studied. Then the simulations are created by taking into account of these criteria and combinations are arranged according to different protocols. The next step was choosing performance metrics. For this, a lot of studies is researched about this and the most accurate metrics were chosen. Then the simulation environment is prepared according to previous works and all issues are eliminated.

After the successful simulations, the results are gathered and analyzed. These evaluations were made taking into account of performance metrics that could make the congestion less occur in IoT networks. According to performance analysis, it was examined which protocol combination can be useful in the tested network topology. When examined these results, using OF0 with CoCoA Strong gives the highest performance, and using ContikiMAC as RDC layer will produce more effective results.

The performance analysis of different protocols is given in detail and tested for a variety of scenarios case in this study. Therefore, the results from these simulations can be helpful for creating a more powerful and less congestive

Internet of Things network. Depend on the requirements, the best combination of protocols can be chosen.

In future studies, it is planned to evaluate different techniques in the link layer like TSCH and 6TiSCH and application layer techniques like MQTT. In addition, for RPL, it is planned to develop an objective function that may be more efficient in working the application layer together and may reduce the congestion.

REFERENCES

A Quick Introduction to the Erbium (Er) REST Engine. (n.d.). Retrieved from GitHub: https://github.com/contiki-os/contiki/tree/master/examples/er-rest-example

Abuein, Q., Yassein, M. O., Shatnawi, M. Q., Bani-Yaseen, L., Al-Omari, O., Mehdawi, M., & Altawssi, H. (2016). Performance Evaluation of Routing Protocol (RPL) for Internet of Things. *International Journal of Advanced Computer Science and Applications*, *7*.

Akan, O., & Akyildiz, I. (2005). Event-to-sink reliable transport in wireless sensor networks. *IEEE/ACM Transactions on Networking, 13*(5), 1003-1016.

Al-Kashoash, H. A., Hafeez, M., & Kemp, A. H. (2017, June). Congestion-Aware RPL for 6LoWPAN Networks: A Game Theoretic Framework. *IEEE Internet of Things Journal, 4*(3), 760-771.

Balandin, S., Andreev, S., & Koucheryavy, Y. (2013). Internet of Things, Smart Spaces, and Next Generation Networking: 13th International Conference. *RPL Objective Function Impact on LLNs Topology and Performance* (pp. 341-344). Berlin: Springer.

Betzler, A., Gomez, C., & Demirkol, I. (2015). Evaluation of Advanced Congestion Control Mechanisms for Unreliable CoAP Communications. *In Proceedings of the 12th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks (PE-WASUN '15). Association for Computing Machinery*, (pp. 63-70). New York, NY, USA.

Betzler, A., Gomez, C., Demirkol, I., & Paradells, J. (2015). CoCoA+: An Advanced Congestion Control Mechanism for CoAP. *Ad Hoc Networks, 33*, 126-139.

Betzler, A., Gomez, C., Demirkol, I., & Paradells, J. (2016). CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine, 54*(7), 154-160.

Bormann, C., Ersue, M., & Keranen, A. (2014, May). *Terminology for Constrained-Node Networks.* RFC7228, Internet Engineering Task Force (IETF). Retrieved from tools.ietf.org/html/rfc7228

Brakmo, L., & Peterson, L. (1995). TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, *13*, pp. 1465-1480.

Build Tunslip6. (n.d.). Retrieved September 02, 2019, from FIT/IoT-LAB: https://www.iot-lab.info/tutorials/build-tunslip6/

Buratti, C., Martalo, M., Verdone, R., & Ferrari, G. (2011). *Sensor Networks with IEEE 802.15.4 Systems: Distributed Processing, MAC, and Connectivity* (1 ed.). Berlin: Springer.

Candoğan, O. (2020, February 1). *Araştırmacılar, IoT Cihazlarında Gecikmesiz Bağlantı Sağlamanın Yolunu Keşfetti*. Retrieved March 12, 2020, from Webtekno: https://www.webtekno.com/iot-cihaz-gecikmesiz-baglanti-yolu-kesfedildi-h84987.html

Cisco ICND1 Foundation Learning Guide: LANs and Ethernet. (2013). Retrieved April 13, 2020, from Cisco: https://www.ciscopress.com/articles/article.asp?p=2092245&seqNum=2

CoAP in Java. (n.d.). Retrieved September 13, 2019, from Californium (Cf) CoAP Framework: https://www.eclipse.org/californium/

Contiki-NG Home. (n.d.). Retrieved October 13, 2019, from Contiki-NG Wiki: https://github.com/contiki-ng/contiki-ng/wiki

Cooja Simulator. (2016, July 21). Retrieved October 14, 2019, from ANRG: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator#Relevant_Directories

da Silva, B., Segers, L., Braeken, A., Steenhaut, K., & Touhafi, A. (2018). A Low-Power FPGA-Based Architecture for Microphone Arrays in Wireless Sensor Networks. In N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, & P. Diniz, *Applied Reconfigurable Computing.*

Architectures, Tools, and Applications. ARC 2018. Lecture Notes in Computer Science (Vol. 10824, p. 286). Cham: Springer.

Demir, A. K., & Abut, F. (2018). Comparison of CoAP and CoCoA Congestion Control Mechanisms in Network Topologies. *GÜFBED CMES 2018 Sempozyum Ek Sayısı*, (pp. 53-60). Gümüşhane.

Development tools Archives. (n.d.). Retrieved March 23, 2020, from Zolertia: https://zolertia.io/product-category/development-tools/

Dunkels, A. (19, 07 17). *The Official Contiki OS Blog: 2013*. Retrieved from The Official Contiki OS Blog: http://contiki-os.blogspot.com/2013/

Dunkels, A. (2007). Poster Abstract: Rime - a lightweight layered communication stack for sensor networks.

Dunkels, A. (2011). The ContikiMAC Radio Duty Cycling Protocol.

Dunkels, A. (2015, 8 25). *Contiki: The Open Source OS for the Internet of Things*. Retrieved from Contiki: http://www.contiki-os.org/

Dunkels, A., & Vasseur, J. P. (2010). *Interconnecting Smart Objects with IP.* San Francisco, USA: Morgan Kaufmann Publishers Inc.

Dunkels, A., Eriksson, J., Finne, N., & Voigt, T. (2006). Cross-level sensor network simulation with cooja. *First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006).* Tampa, Florida, USA.

Duquennoy, S. (2019, February 14). *Documentation: RPL*. Retrieved from Contiki-NG Wiki: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-RPL

Duquennoy, S. (2019, February 18). *The Contiki-NG configuration system*. Retrieved December 09, 2019, from Contiki-NG Wiki: https://github.com/contiki-ng/contiki-ng/wiki/The-Contiki%E2%80%90NG-configuration-system

Fall, K., & Floyd, S. (1995). *Simulation-based comparisons of Tahoe, Reno and SACK TCP.* Technical Report.

Ferri, G., Tang, W., Ma, X., Huang, J., & Wei, J. (2016). Toward Improved RPL: A Congestion Avoidance Multipath Routing Protocol with Time Factor for Wireless Sensor Networks. *Journal of Sensors*, 3-4.

Floyd, S. (2003). *HighSpeed TCP for large congestion windows.* IETF. RFC3649.

Floyd, S., Henderson, T., & Gurtov, A. (2004). *The NewReno modification to TCP's fast recovery algorithm.* IETF, RFC3782.

Gnawali, O., & Levis, P. (2012). *The Minimum Rank with Hysteresis Objective Function.* IETF. RFC6719.

Gomes, T., Salgado, F., Pinto, S., Cabral, J., & Tavares, A. (2018). A 6LoWPAN Accelerator for Internet of Things Endpoint Devices. *IEEE Internet of Things Journal, 5*(1), 371-377.

Ha, S., Rhee, I., & L., X. (2008). CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review, 42*(5), 64-74.

Halcu, I., Stamatescu, G., Stamatescu, I., & Sgârciu, V. (2016). IPV6 Sensor Networks Modeling for Security and Communication Evaluation. In E. Pricop, & G. Stamatescu, *Recent Advances in Systems Safety and Security* (p. 251). Berlin: Springer International Publishing.

Hamadoun, T., Chalhoub, G., & Misson, M. (2016). Implementation of IEEE 802.15.4 Unslotted CSMA/CA Protocol on Contiki OS. *Annals of Telecommunications, 71*, 517-526.

IEEE 802.15 WPAN™ Task Group 4 (TG4). (2010, January 27). Retrieved December 12, 2019, from IEEE 802.15.4: http://www.ieee802.org/15/pub/TG4.html

Insense Examples. (n.d.). Retrieved March 23, 2020, from University of St. Andrews School of Computer Science.

Jacobson, V. (1988). Congestion Avoidance and Control. *SIGCOMM Comput. Commun. Rev., 18*(4), 314-329.

Jin, C., Wei, D., Low, S., Buhrmaster, G., Bunn, J., Choe, D., . . . Singh, S. (2005). FAST TCP: from theory to experiments. *IEEE Network. 19*, pp. 4-11. IEEE.

Kang, J., Zhang, Y., & Nath, B. (2007). TARA: Topology-Aware Resource Adaptation to Alleviate Congestion in Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, *18*, pp. 919-931.

Kaur, H., & Singh, G. (2017). TCP Congestion Control and Its Variants. *Advances in Computational Sciences and Technology, 10*(6), 1715-1723.

Kelly, T. (2003). Scalable TCP: improving performance in high-speed wide area networks. *ACM SIGCOMM Computer Communication Review, 33*, 83-91.

Khunboa, C., & Suwannapong, C. (2019). Congestion Control in CoAP Observe Group Communication. *Sensors, 19*(15), 3433.

Kozierok, C. M. (2005). *The TCP / IP Guide: A Comprehensive, Illustrated Internet Protocols Reference.* No Starch Press.

Kozierok, C. M. (n.d.). *UDP Message Format*. Retrieved April 14, 2020, from The TCP/IP Guide: http://www.tcpipguide.com/free/t_UDPMessageFormat.htm

Kumar, P. (2018, February 27). *RPL - Routing Protocol for Low Power and Lossy Networks*. Retrieved March 14, 2020, from Slideshare: https://www.slideshare.net/tspradeepkumar/rpl-routing-protocol-for-low-power-and-lossy-networks

Lamaazi, H., Benamar, N., & Jara, A. J. (2017). Study of the Impact of Designed Objective Function ın the RPL-Based Routing Protocol. In R. El-Azouzi, D. Menasche, E. Sabir, F. De Pellegrini, & M. Benjillai, *Advances in Ubiquitous Networking 2. UNet 2016. Lecture Notes in Electrical Engineering* (Vol. 397). Singapore: Springer.

Li, G., Li, J., & Yu, B. (2012). Lower bound of weighted fairness guaranteed congestion control protocol for WSNs. *In Proceedings of the IEEE INFOCOM*, 3046-3050.

Liu, D. (2009). *Cisco Router and Switch Forensics: Investigating and Analyzing Malicious Network Activity* (1 ed.). Syngress.

MAC protocols in ContikiOS. (2014, November 8). Retrieved December 23, 2019, from ANRG: https://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS

Mardini, W., Ebrahim, M., & Al-Rudaini, M. (2017). Comprehensive Performance Analysis of RPL Objective Functions in IoT Networks. *International Journal of Communication Networks and Information Security*, 9.

Mathis, M., & Mahdavi, J. (1996). Forward acknowledgement: refining TCP congestion control. *ACM SIGCOMM Computer Communication Review, 26*, 281-291.

Michopoulos, V., Guan, L., Oikonomou, G., & Phillips, I. (2012). DCCC6: Duty Cycle-awate congestion control for 6LoWPAN networks. *2012 IEEE International Conference Pervasive Computing and Communications Workshops* (pp. 278-283). Lugano, Switzerland: IEEE.

Michopoulos, V., Oikonomou, G. C., Phillips, I. W., & Guan, L. (2014). CADC: Congestion Aware Duty Cycle mechanism a simulation evaluation. *IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)* (pp. 340-344). Athens, Greece: IEEE.

Misra, S., Tiwari, V., & Obaidat, M. (2009). Lacas: learning automata-based congestion avoidance scheme for healthcare wireless sensor networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, (pp. 466-479).

Mulligan, G. (2007). The 6LoWPAN Architecture. *Proceedings of the 4th Workshop on Embedded Networked Sensors* (pp. 78-82). New York, NY, USA: Association for Computing Machinery.

Oikonomou, G. (2018, April 27). *Tutorial: RAM and ROM usage*. Retrieved October 14, 2019, from Contiki-NG Wiki: https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-RAM-and-ROM-usage

Pham, P., & Sylvie, P. (2003). Performance Analysis of Reactive Shortest Path and Multipath Routing Mechanism with Load Balance. *Twenty-Second Annual Joint Conference of the IEEE Computer and Communication Societies, INFOCOM 2003*, *1*, pp. 251-259.

Postel, J. (1980). *User Datagram Protocol.* IETF. RFC768. Retrieved from https://tools.ietf.org/html/rfc768

Pradeska, N., Widyawan, Najib, W., & Kusumawardani, S. S. (2016). Performance Analysis of Objective Function MRHOF and OF0 in Routing Protocol RPL IPv6 Over Low Power Wireless Personal Area Networks (6LoWPAN). *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, (pp. 1-6). Yogyakarta.

Qasem, M., Altawssi, H., Yassien, M. B., & Al-Dubai, A. (2015). Performance Evaluation of RPL Objective Functions. *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, (pp. 1606-1613). Liverpool.

Rathee, G., Ahmad, F., Kerrache, C. A., & Azad, A. M. (2019). A Trust Framework to Detect Malicious Nodes in Cognitive Radio Networks. *Electronics, 8*, 1299.

Rayes, A., & Salam, S. (2017). *Internet of Things From Hype to Reality: The Road to Digitization.* Springer.

Roman, R., & Lopez, J. (2009, April). Integrating Wireless Sensor Networks and the Internet: a Security Analysis. *Internet Research, 19*, 246-259.

Sethi, P., & Sarangi, S. R. (2017). Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering, 2017*, 10-12.

Sheu, J., Hsu, C., & Ma, C. A. (2015). Game Theory Based Congestion Control Protocol for Wireless Personal Area Networks. *In Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC)* (pp. 659-664). Taichung, Taiwan: IEEE.

Subir, V. (2015). *Internet Congestion Control* (Vol. 1). Morgan Kaufmann.

Supported host operating systems for Workstation Pro 12.x and 14.x (2129859). (n.d.). Retrieved October 2, 2019, from VMware Knowledge Base: https://kb.vmware.com/s/article/2129859

Texas Instruments. (2015). *CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications.* Dallas, Texas: Texas Instruments Incorporated.

The Contiki-NG build system. (n.d.). Retrieved October 13, 2019, from Contiki-NG Wiki: https://github.com/contiki-ng/contiki-ng/wiki/The-Contiki%E2%80%90NG-build-system

The Z1 Mote. (2018, July 18). Retrieved from Zolertia/Resources Wiki: https://github.com/Zolertia/Resources/wiki/The-Z1-mote

Thébaudeau, B. (2019, September 23). *Change mac or radio duty cycling protocols*. Retrieved October 25, 2019, from Contiki-NG Wiki: https://github.com/contiki-os/contiki/wiki/Change-mac-or-radio-duty-cycling-protocols

Thébaudeau, B. (2019, September 23). *MSP430X*. Retrieved October 13, 2019, from Contiki-NG Wiki: https://github.com/contiki-os/contiki/wiki/MSP430X

Thubert, P. (2012). *Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL).* IETF. RFC6552.

Tutorial: Running Contiki-NG in Cooja. (n.d.). Retrieved October 14, 2019, from Contiki-NG Wiki: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja#Create_a_Hello_World_simulation

Vilajosana, X., Tuset, P., Watteyne, T., & Pister, K. (2015). OpenMote: Open-Source Prototyping Platform for the Industrial IoT. *ADHOCNETS, 155*.

Wan, C. -Y., Campbell, A. T., & Krishnamurthy, L. (2005). Pump-Slowly, Fetch Quickly (PSFQ): a reliable transport protocol for sensor networks. *IEEE Journal on Selected Areas in Communications*, (pp. 862-872).

Wan, C. -Y., Eisenman, S. B., & Campbell, A. T. (2003). CODA: Congestion Detection and Avoidance in Sensor Networks. *In Proceedings of the 1st International Conference on Embedded Network Sensor Systems (SenSys '03)*, (pp. 266-279). New York, NY, USA.

Wang, J., Wen, J., Zhang, J., & Han, Y. (2011). TCP-FIT: An improved TCP congestion control algorithm and its performance. *2011 Proceedings IEEE INFOCOM* (pp. 2894-2902). Shanghai, China: IEEE.

Warren, S. (2012, 07 07). *Snapshots in VMware Workstation.* Retrieved August 21, 2019, from Techrepublic: https://www.techrepublic.com/blog/virtualization-coach/

Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., . . . Alexander, R. (2012). *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* IETF. RFC6550. Retrieved from https://tools.ietf.org/html/rfc6550

Woo, A., & Culler, D. E. (2001). A transmission control scheme for media access in sensor networks. *In Proceedings of the 7th annual international conference on Mobile computing and networking, MobiCom '01*, (pp. 221-235). New York, NY, USA.

BIOGRAPHY

Rıdvan SÖYÜ was born in Mersin, in 1991. He has completed his elementary education at 700. Yıl Primary School and secondary education at Mehmet Adnan Özçelik Anatolian High School. He has completed his university education at department of Computer Engineering of Gazi University in 2015. Since 2017, he is still working as a research assistant in Department of Computer and Software Engineering of Toros University in Mersin.