FEDERAL UNIVERSITY OF TECHNOLOGY - PARANÁ

OFFICE OF RESEARCH AND GRADUATE STUDIES

GRADUATE PROGRAM IN COMPUTER SCIENCE

HELBERT DA ROCHA

AN MQTT-SN-BASED PROTOCOL FOR QOS ADAPTATION IN WIRELESS SENSOR NETWORKS

MASTER'S THESIS

PONTA GROSSA

2018

HELBERT DA ROCHA

AN MQTT-SN-BASED PROTOCOL FOR QOS ADAPTATION IN WIRELESS SENSOR NETWORKS

Master's Thesis presented to Graduate Program in Computer Science of the Office of Research and Graduate Studies at Federal University of Technology - Paraná - Campus Ponta Grossa, as a partial requirement to obtain the title of Master in Computer Science.
Concentration Area: Information Systems and Computing

Advisor: Prof. Ph.D. Tania Lucia Monteiro

PONTA GROSSA
2018

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa
n.59/18

R672 Rocha, Helbert da

An MQTT-SN-BASED protocol for QoS adaptation in wireless sensor networks. / Helbert da Rocha, 2018.
48 f. : il. ; 30 cm.

Orientadora: Profa. Dra. Tania Lucia Monteiro

Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação. Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2018.

1. Internet. 2. Mensagens eletrônicas. 3. Rede de computador - Protocolos. 4. Sistemas de comunicação sem fio. 5. Detectores. I. Monteiro, Tania Lucia. II. Universidade Tecnológica Federal do Paraná. III. Título.

CDD 004

Elson Heraldo Ribeiro Junior. CRB-9/1413. 30/11/2018.

FOLHA DE APROVAÇÃO

Título de Dissertação Nº **7/2018**

AN MQTT-SN-BASED PROTOCOL FOR QOS ADAPTATION IN WIRELESS SENSOR NETWORKS

Por

Helbert da Rocha

Esta dissertação foi apresentada às **13 horas 30 minutos** de **08 de novembro de 2018**, na **Q204 / UTFPR - Reitoria**, como requisito parcial para a obtenção do título de MESTRE EM CIÊNCIA DA COMPUTAÇÃO, Programa de Pós-Graduação em Ciência da Computação. O candidato foi arguido pela Banca Examinadora, composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Prof. Dr. Marcelo Eduardo Pellenz (PUCPR)

Prof. Dr. Augusto Foronda (UTFPR)

Profª. Drª. Tânia Lúcia Monteiro (UTFPR) – *Orientador e presidente da banca*

Universidade Tecnológica Federal do Paraná
Câmpus · Ponta Grossa

Visto da Coordenadora:

Profª. Drª. Sheila Morais de Almeida
Coordenadora do PPGCC
UTFPR – Câmpus Ponta Grossa

A Folha de Aprovação assinada encontra-se na Coordenação do Programa

To my family.

ACKNOWLEDGMENTS

I wish to thank my family that is always by my side supporting me in all of the remarkable moments of my life, especially to my mother Neura, my father Neuri, my brothers Marcelo, Marcio, and Helder who contributed significantly to the person that I became.

A special thanks to my advisor Prof. Ph.D. Tania Lucia Monteiro for the guidelines and long hours of discussions that resulted in the development and conclusion of this thesis.

I would like to thank all the professors who make it possible to get here by sharing knowledge throughout the process, for the personal and professional growth.

Finally, to thank all my few and good friends and to all the other people who contributed in some form for the development of this thesis.

Sincerely,
Helbert da Rocha

“It all seems impossible
until it is done.”
Nelson Mandela

ABSTRACT

Rocha, Helbert da. **An MQTT-SN-BASED Protocol for QoS Adaptation in Wireless Sensor Networks**. 2018. 48 p. Master's Thesis (Master in Computer Science) - Federal University of Technology - Paraná. Ponta Grossa, 2018.

The Internet of Things (IoT) remains a concept that is being increased in the last few years. The principal idea is to connect smart devices through a network solution. In the next few years, IoT will be present in everyday objects, in people's life, almost everything will communicate through the Internet. The economic impact of IoT solution is expected to be, annually, billions of dollars. To provide data exchange from smart devices, some protocols are being used. The Message Queuing Telemetry Transport (MQTT) is one of the most common application protocols for IoT and Machine-to-Machine (M2M) communications. The MQTT implements the publish/subscribe paradigm that provides three Quality of Service (QoS) to ensure message exchange between the devices. However, MQTT protocol is developed over TCP stack and implements the TCP protocol to communicate. There is a version of MQTT for Sensor Network (SN), named of MQTT-SN, developed specially for exchanging messages in Wireless Sensors Networks (WSNs). As many smart devices will be connected on the same WSN, the network can be overloaded and the links may become unstable. This study presents a method to optimize the exchange messages and to increase message delivery during a communication process, between a publisher and a broker or between a publisher and the middleware, implementing the MQTT-SN protocol. The QoS Dynamic Adaptation Method (DAM) for sensor networks was developed on the publisher's side. It was focused to select the best Quality of Service between the three QoS levels implemented in the MQTT-SN protocol, based on network latency. The QoS DAM showed good performance in wireless networks, kept message delivery during the communication process, and it showed an impressive performance when compared with the normal QoS implemented in the MQTT-SN protocol.

Key-words: Internet of Things. MQTT-SN. QoS. Publish/Subscribe Paradigm. Adaptive QoS Method.

RESUMO

Rocha, Helbert da. **Um Protocolo Baseado em MQTT-SN para Adaptação da QoS em Redes de Sensores Sem Fio**. 2018. 48 p. Dissertação (Mestrado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

A Internet das Coisas (Internet of Things - IoT) é um conceito que vem crescendo nos últimos anos. A ideia principal é conectar dispositivos inteligentes por meio de uma solução utilizando à Internet. Nos próximos anos, a IoT estará presente nos objetos do dia-a-dia, na vida das pessoas, quase tudo irá se comunicar por meio da Internet. O impacto econômico da solução utilizando IoT, será de bilhões de dólares anualmente. Para promover a troca de mensagens entre os dispositivos inteligentes, alguns protocolos estão sendo utilizados. O Message Queuing Telemetry Transport (MQTT) é um dos protocolos mais comuns para a IoT e comunicação entre máquinas (Machine-to-Machine - M2M). O MQTT utiliza o paradigma publish/subscribe que provê três Qualidades de Serviço (Quality of Service - QoS) para garantir a troca de mensagens entre os dispositivos. Entretanto, o protocolo MQTT foi desenvolvido sobre a pilha de protocolos TCP e utiliza o protocolo TCP para realizar a comunicação. Há uma versão do protocolo MQTT para Redes de Sensores (Sensor Network - SN), chamado de MQTT-SN, desenvolvido especialmente para a troca de mensagens em Redes de Sensores Sem Fio (Wireless Sensor Networks - WSNs). Como muitos dispositivos inteligentes poderão ser conectados na mesma WSN isso pode ocasionar que a rede fique sobrecarregada e com links instáveis. Este estudo apresentou um método para otimizar a troca de mensagens e aumentar o número de mensagens entregues durante um processo de comunicação entre um publisher e um broker ou entre o publisher e um agente intermediário. O Método de Adaptação Dinâmica da Qualidade de Serviço (QoS Dynamic Adaptation Method - QoS DAM) para redes de sensores foi desenvolvido no lado do publisher com o foco de selecionar a melhor Qualidade de Serviço, baseando-se na latência da rede. O método QoS DAM apresentou boa performance em redes sem fio, mantendo a entrega das mensagens durante o processo de comunicação, também apresentou uma boa performance quando comparado com a implementação normal da Qualidade de Serviço presente no protocolo MQTT-SN.

Palavras-chaves: Internet das Coisas. MQTT-SN. QoS. Publish/Subscribe Paradigma. Método adaptativo da QoS.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
DAM	Dynamic Adaptation Method
DDS	Data Distribution Service
CBC	Cipher Block Chaining
CoAP	Constrained Application Protocol
CPU	Central Process Unit
dBm	Decibel-milliwatt
dBi	Decibel-isotropic
DNS	Domain Name System
ETSI	European Telecommunication Standards Institute
GB	Gigabyte
GHz	Gigahertz
GUI	Graphical User Interface
GW	Gateway
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
LLSec	Link-Layer Security
M2M	Machine-to-Machine
Mbps	Megabits per second
MQTT	Message Queuing Telemetry Transport
MQTT-SN	Message Queuing Telemetry Transport for Sensor Networks
MS	Millisecond
OS	Operating System
PDR	Packet Delivery Ratio

QoS	Quality of Service
REST	Representational State Transfer
RSMB	Really Small Message Broker
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
RTO	Retransmission Timeout
S	Second
SAs	Sensors and Actuators
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
DTLS	Datagram Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocols
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

LIST OF SYMBOLS

n	Number of messages
pt	Time of PUBLISH message
mt	Time of confirmation messages

TABLE OF CONTENTS

1 INTRODUCTION

The concept to communicate smart devices, through the Internet is named Internet of Things (IoT). It is a topic composed of technical, social and economic significance (ROSE; ELDRIDGE; CHAPIN, 2015). The smart devices can see, hear, think, talk to each other and exchange information to coordinate decisions (AL-FUQAHA *et al.*, 2015; TORRES; ROCHA; SOUZA, 2016). Many areas are using connected sensors through the Internet, e.g., environmental forecasting and monitoring, medical, military, transportation, crisis management, industrial automation, and others (ULLAH *et al.*, 2013). The worldwide impact of IoT and Machine-to-Machine (M2M), is related to connection of 100 billion devices, resulting in a global economic impact between \$3.5 and \$11 trillion by 2025 (ROSE; ELDRIDGE; CHAPIN, 2015; MANYIKA, 2015).

Currently, the appropriate protocol to exchange messages in the IoT and M2M communications is the Message Queuing Telemetry Transport (MQTT). It is capable to provide routing for small, cheap, low power and low memory devices working inside the vulnerable and low bandwidth networks (AL-FUQAHA *et al.*, 2015).

The MQTT protocol was developed in 1999 by Andy Clark of IBM and Arlen Nipper (Eurotech), it was standardized in 2013 at the OASIS (AL-FUQAHA *et al.*, 2015; BANKS; GUPTA, 2014). The MQTT protocol uses the publish/subscribe paradigm that provides a communication mechanism (one-to-one, one-to-many, and many-to-many) built on top of the TCP protocol. In this paradigm, the publisher publishes to a topic in the middleware named broker. A subscriber subscribes to a topic of his interest in the broker. When the publisher publishes a message to this topic inside the broker, the broker transfers the message from the topic to the subscriber. The messages are delivered using one of the three Quality of Service (QoS): QoS 0 sends a message without confirmation (fire or forget), the QoS 1 sends a message with a confirmation message, but duplicate messages can arrive, and QoS 2 sends exactly one message with three confirmation messages. There is a version of MQTT developed specially for Wireless Sensors Network (WSN) named of MQTT-SN (STANFORD-CLARK; TRUONG, 2013).

The MQTT-SN was developed by the IBM Company for networks with extremely specific purposes, e.g., WSNs. A typical WSN consists of a massive number of devices equipped with a limited amount of storage and processing capabilities. For this environment, it is significant a wireless communication between devices. This protocol can be used for networks where the devices may fail and be replaced frequently (STANFORD-CLARK; TRUONG, 2013).

The MQTT protocol requires underlying protocols, for example, TCP/IP which is unnecessarily complex for a simple, small footprint and low-cost devices as sensors. The MQTT-SN is a different protocol that uses the publish/subscribe paradigm. It was designed to be as close as possible of MQTT, adapted to the particularities of wireless communication environments. MQTT-SN was developed properly to work in low bandwidth and high link failures networks, it uses short length messages (STANFORD-CLARK; TRUONG, 2013).

The proposed method had the goal to optimize the process of exchange messages between a publisher and a broker or between a publisher and the middleware, e.g., a Gateway. Using the Message Queuing Telemetry Transport for Sensor Network (MQTT-SN) protocol. The method was developed on the publisher side of the communication. The user selects one of the three QoS provided by the MQTT-SN and the method will select the best QoS according to the network latency. The developed method, made possible an increase of message delivery compared with the original MQTT-SN without any special QoS treatment during the communication.

1.1 MOTIVATION

As an emerging modern technology, the Internet of Things (IoT) will provide the connectivity of billions or trillions of smart things in the following years. An important protocol that can be used in the application layer of communication is the Message Queuing Telemetry Transport (MQTT). The MQTT is a lightweight, open, simple, and with easy implementation developed by Andy Stanford-Clack of IBM and Arlen Nipper of Arcom (now Eurotech) in 1999. However, the MQTT protocol is too complex to communicate small, low bandwidth and battery devices like sensors and actuators (SAs)(LOCKE, 2013).

The alternative for Wireless Sensor Networks (WSNs) is the MQTT-SN protocol. The MQTT-SN was developed to be close as possible to MQTT, but adapted for sensors and actuators. It was designed to the particularities of the wireless communications environment. Moreover, it is adapted for limited bandwidth and high link failure. The optimization of the Quality of Service (QoS) from the MQTT-SN protocol to increase the network performance is essential. It is a significant advantage of this technology mainly by the fact that at the end of 2020 are expected more than 213 billion smart devices (GANTZ; REINSEL, 2012). Probably, the MQTT-SN protocol will be one of the solutions to interconnect many of them in WSNs.

1.2 OBJECTIVES

1.2.1 General Objective

The general objective of this work is to develop a Quality of Service Dynamic Adaption Method (QoS DAM) for the Message Queuing Telemetry Transport for Sensor Network (MQTT-SN) protocol.

1.2.2 Specific objectives

The specific objectives are:

- To study and understand the network behavior by each one of the three qualities of service, provided by MQTT and MQTT-SN in a proper environment;
- To develop tools that help to understand the protocols;
- To estimate a proper latency interval for QoS DAM;
- To provide a better network performance without overload the network using a proper QoS;
- To decrease message loss in WSNs implementing the MQTT-SN protocol.

1.3 CONTRIBUTIONS

This Thesis presents a new QoS Dynamic Adaptation Method to ensure message delivery and reduce packet loss implementing the MQTT-SN protocol. Furthermore, it presents two developed tools that can be used to comprehend how the Quality of Service, latency, and signal strength can affect the communication process by the MQTT protocol. The tools and the QoS DAM can be adapted for each environment with other variables and necessities.

1.4 OUTLINE OF THIS THESIS

The organization of the rest of the Thesis is structured as follows. Section 2 discusses the background and related works. The problem formulation is presented in Section 3. The solution proposed is presented in Section 4. The results are shown in Section 5. Finally, conclusions and future works are exposed in Section 6.

2 BACKGROUND

This Section presents some tools that were utilized to develop the method to reduce packet loss, the QoS Dynamic Adaptation Method for MQTT-SN protocol. The method was developed with the Python (ROSSUM; others, 2007) and C (SCHILDT, 1990) as programming languages to generate scripts for the environment of the test. Furthermore, a client called Mosquito from the author in (LIGHT, 2017) was utilized to receive messages from the broker.

2.1 APPLICATION LAYER

The Message Queuing Telemetry Transport for Sensor Networks is an application protocol. The application layer is defined as a layer that allows the user (human or software), to access the network. The main function of this layer is to provide services to final users. Some services provided are: e-mail service, access and transfer achieve, access system resources, surf the Web and network management (FOROUZAN, 2009).

The application layer has the own support protocols, to allow the application to work properly, e.g., DNS (Domain Name System), SMTP, MQTT, and others (TANENBAUM; WETHERALL; TRANSLATIONS, 2011).

Some IoT standards are proposed to help developers and service providers in the application layer. In addition, various groups created standards to support IoT protocols, e.g., World Wide Web Consortium (W3C), Internet Engineering Task Force (IETF), EPCglobal, Institute of Electrical and Electronics Engineers (IEEE) and the European Telecommunication Standards Institute (ETSI) (AL-FUQAHA *et al.*, 2015). Table 1 shows the most important protocols and their differences defined by these groups.

Table 1 – Comparison between the IoT Application Protocols

Application Protocols	RESTFul	Transport	Publish/Subscribe	Request/Response	Security	QoS	Header Size (Byte)
COAP	V	UDP	V	V	DTLS	V	4
MQTT	-	TCP	V	-	SSL	V	2
MQTT-SN	-	UDP / Others (e.g. ZigBee)	V	-	SSL	V	2
XMPP	-	TCP	V	V	SSL	-	-
AMQP	-	TCP	V	-	SSL	V	8
DDS	-	TCP / UDP	V	-	SSL / DTLS	V	-
HTTP	V	TCP	-	V	SSL	-	-

Source: Adapted from (AL-FUQAHA *et al.*, 2015; CRAGGS, 2015)

2.2 INTERNET OF THINGS (IOT)

The Internet of Things represents a paradigm of rapidly growing, corresponding to the integration of several technologies and communication solutions, identification and tracking technologies. Used to connected small, tiny, limited bandwidth everyday life objects, through the Internet, creating a synergy between diverse fields of knowledge, e.g., telecommunication, informatics, electronics, and social science. This technology will enable smart devices exchange information between them. Many areas will utilize the IoT, e.g., home appliances, surveillance cameras, monitoring sensors, actuators, displays, vehicles, etc (ATZORI; IERA; MORABITO, 2010; AL-FUQAHA *et al.*, 2015).

The Internet of Things will allow the connection of billion or trillion of smart devices through the Internet. Probably in 10 or 20 years, many operations will be possible to be done remotely, like a doctor examining a patient in another city, seeing the real-time health status information, e.g., blood pressure, heart rate, etc (LAMPKIN *et al.*, 2012).

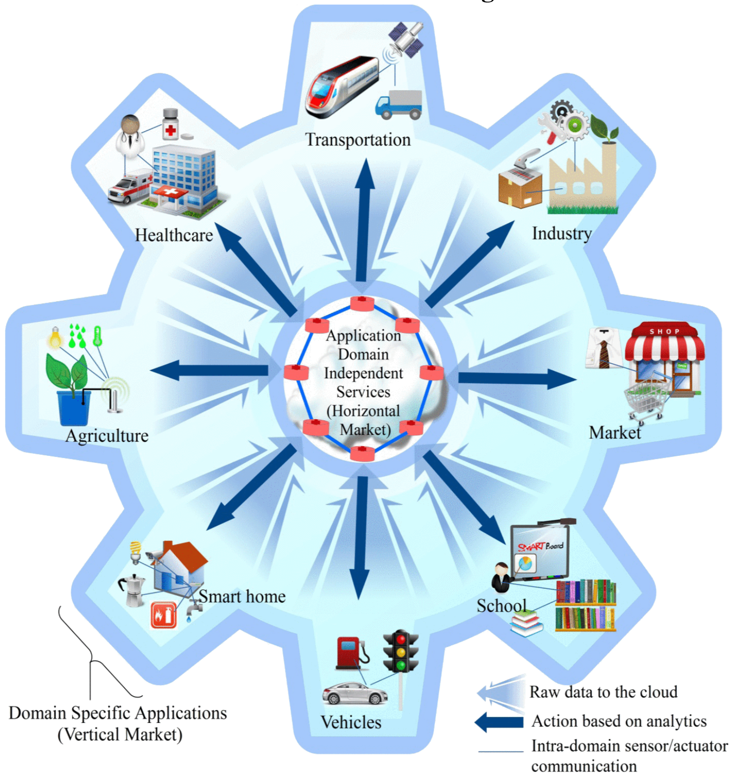
It is being expected that IoT will connect 212 billion smart objects around the world by the end of 2020. Figure 1 presents the application that can be developed, and the vertical market the objects will be connected (AL-FUQAHA *et al.*, 2015). The US National Intelligence Council includes the IoT in one of the "Disruptive Civil Technologies", presenting that: "by 2025 Internet nodes may reside in everyday things – food packages, furniture, paper documents, and more” (ATZORI; IERA; MORABITO, 2010).

The IoT is already present today, many places and objects contain sensors. There are some examples: buildings have sensors to save energy. Many homes are already autonomous, cars, taxis, and public lights have devices to improve safety and transportation. People have useful apps on their smartphones, also, many industries are currently connected to the Internet. Health care services are being used with home sensing to support medicines and wellness (STANKOVIC, 2014).

2.3 MESSAGE QUEUING TELEMETRY TRANSPORT (MQTT)

The MQTT message protocol was developed by Andy Stanford-Clack of IBM and Arlen Nipper of Arcom (now Eurotech) in 1999, and only in 2013 was standardized at OASIS (LOCKE, 2013). The MQTT allows the connection between embedded devices with networks applications and middleware. In addition, the MQTT is open and easy to implement because it uses the publish/subscribe pattern (BOYD *et al.*, 2014). The protocol was built on top of the TCP protocol. There are large specifications of MQTT: MQTT v3.1 and MQTT-SN (LOCKE, 2013). The MQTT-SN specification was defined specially for sensors networks and establishes a UDP mapping of MQTT. The benefits of the MQTT offers are (BOYD *et al.*, 2014, p. 6):

Figure 1 – The overall picture of IoT emphasizing the vertical markets and the horizontal integration between them


Transportation
Healthcare
Industry
Application Domain Independent Services (Horizontal Market)
Agriculture
Market
SHOP
Smart home
School
Vehicles
Domain Specific Applications (Vertical Market)
Raw data to the cloud
Action based on analytics
Intra-domain sensor/actuator communication

Source: (AL-FUQAHA *et al.*, 2015)

- To allow more connectivity to smart devices;
- To offer more options of connectivity optimized for sensors and remote devices;
- To deliver relevant data;
- To allow the scalability and management of solutions.

The MQTT protocol provides telemetry technology to meet the information challenges of the Internet users.

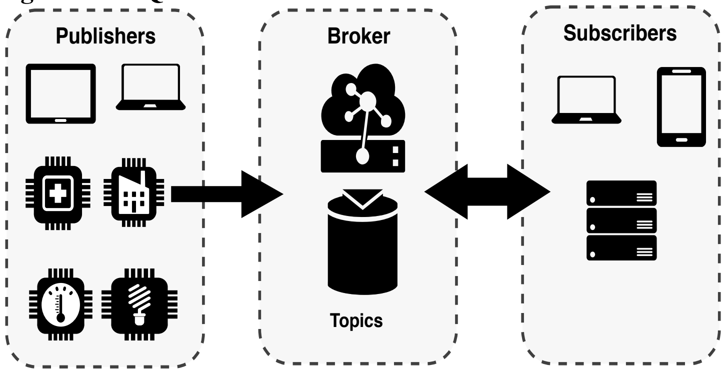
The telemetry technology enables to measure and monitor things from a distance. The improvements in this technology, at this time, made possible to interconnect measure and monitor devices inside of other locations, so it reduces the costs of developing applications that can run on smart devices to make them more useful (BOYD *et al.*, 2014; LAMPKIN *et al.*, 2012).

The telemetry is used on smart devices by people, business, and governments to interact smartly with their works. The information can help user decision, e.g., a person that is shopping groceries and wants to know what products there are at the moment in its pantry at home. The information comes from a variety of smart devices.

There are some challenges using telemetry technologies and communications protocols to transmit and receive information. A challenge is to get information from devices and deliver this information to people and application that want to use this information. At the time, it is important to allow the people or application to reply to the device with current instructions and requests. The challenge can increase with the devices distributed geographically, or if they have limited storage or computational abilities (BOYD *et al.*, 2014).

The MQTT consists of three components, subscribers, publishers, and brokers. Figure 2 illustrates the architecture of MQTT. To communicate, the device will register as a subscriber to a specific topic of its interest in the broker. When the publishers publish to the topic, the broker delivers the information to one or more subscribers (STANFORD-CLARK; TRUONG, 2013). Many applications use MQTT e.g., health care, monitoring, energy meter, Facebook notification (WANG, 2011). Furthermore, the MQTT represents one of the appropriate messaging protocol for IoT and M2M communications because provides routing for small, cheap, limited power and memory devices that belong to vulnerable and low bandwidth networks (BOYD *et al.*, 2014; AL-FUQAHA *et al.*, 2015).

Figure 2 – MQTT architecture

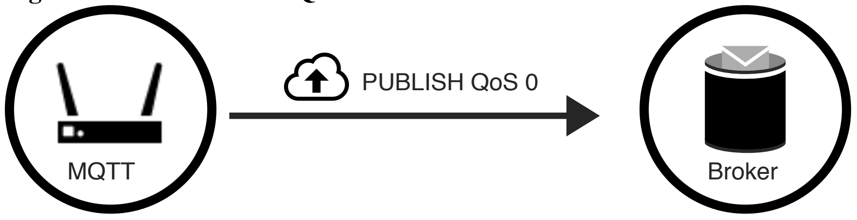


Source: Author

The MQTT implements the communication between the broker and the publishers and the broker with the subscribers. To start a communication a CONNECT packet is sent from the client (publisher or subscriber) to the server (broker) to start the connection, the server re-sponses with a CONNACK packet. To finish the communication, the client sends a DISCON-NECT packet. During the communication process three Quality of Service (QoS) levels may be used(LOCKE, 2010):

QoS 0 (At most once): the message is sent using the best effort on the TCP/IP network, the answer is not expected and the message is not sent again. The message can arrive at the server or not. The sender sends a PUBLISH packet as presented in Figure 3.

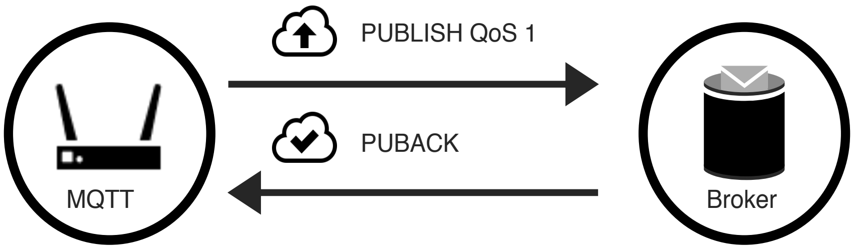
Figure 3 – At most once - QoS 0



Source: Author

QoS 1 (At least once): it is the default mode of message transfer (LOCKE, 2010). The message arrives at least once to the receiver, to ensure delivery at least once. If a failure is identified, there may be a failure on the communication link, or the message is not received after a specified period of time, the message will be resent by the sender. It allows the recipient to receive the message multiple times. After the message is processed, it is deleted from the receiver. The sender using QoS 1 sends a PUBLISH packet containing the Packet Identifier. The Publish packet retains a state of unacknowledged until the sender, receives the PUBACK packet from the broker/receiver. (LOCKE, 2010). Figure 4 shows message exchange between a publisher and broker using QoS 1 technique. After deleting the message the receiver sends the acknowledgment to the sender. The same occurs with the sender after receiving the acknowledgment from the receiver. Both sender and receiver delete the message after the communication.

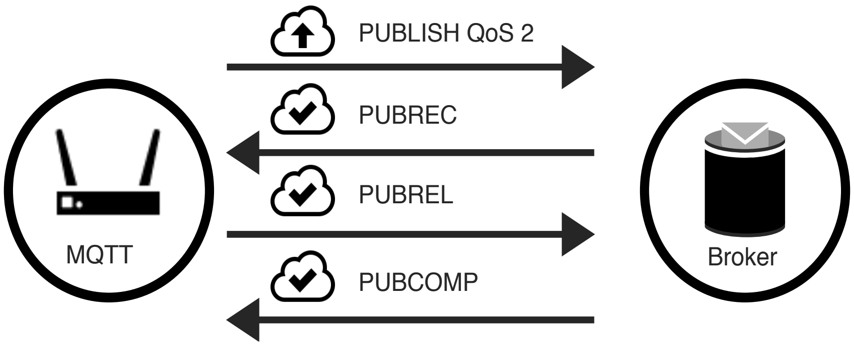
Figure 4 – At least once - QoS 1



Source: Author

Qos 2 (Exactly once): Duplicate messages and losses are not acceptable (LOCKE, 2010). It increases the network traffic, but it is acceptable because the QoS 2 is used to send relevant messages, it is the highest Quality of Service (LOCKE, 2010). This technique guarantees that a message is received once, when the receiver confirms the message has arrived. The QoS 2 PUBLISH packet involves two-step process. The PUBLISH packet is treated as unacknowledged until the sender receives the corresponding PUBREC packet from the receiver. The sender sends a PUBREL packet and waits from the receiver, the corresponding PUBCOMP packet. The PUBREL packet is treated as unacknowledged until the PUBCOMP packet reception. (LOCKE, 2010). Figure 5 represents messages exchange between a publisher and broker using QoS 2 technique.

Figure 5 – Exactly once - QoS 2



Source: Author

The MQTT specification (LOCKE, 2013) do not present parameters to be set in order to define an interval, and how many times the messages may be retransmitted.

2.4 MESSAGE QUEUING TELEMETRY TRANSPORT FOR SENSOR NETWORK (MQTT-SN)

The Message Queuing Telemetry Transport for Sensor Network (MQTT-SN) protocol was developed specially for Wireless Sensor Networks (WSNs), normally made up of low cost and easy developed environments. Typically, a WSN has a large number of sensors and actuators, from different device types, that present a limited amount of storage and processing capabilities. The devices are developed to detect and notify events through wireless links, used habitually in monitoring environment, traffic and building management, battlefield surveillance, and home automation (DAVIS; AUGé, 2018; STANFORD-CLARK; TRUONG, 2013).

Some problems like addressing schemes between the networks may be involved. For example, how to communicate an application residing on a TCP/IP-based network and a device running on a ZigBee-based wireless network? (STANFORD-CLARK; TRUONG, 2013)

The problem can be solved using data-centric communication approach. An excellent example of data-centric communication is the “Publish/Subscribe” message system, that is scalable and support dynamic network topology. A protocol that can be used is the MQTT protocol, optimized for communication with low bandwidth or networks with intermittent connection. However, the MQTT protocol requires an underlying network, e.g., TCP/IP, and in some situations, the protocol is unnecessarily complex for the very simple, small footprint and low-cost devices (STANFORD-CLARK; TRUONG, 2013).

The propose of the MQTT-SN is to be a publish/subscribe protocol for wireless sensor network, considering a version of MQTT that best applies to the particularities of wireless communication environment. Furthermore, it is optimized for implementation on low-cost, battery-operated devices with limited processing and storage resource (STANFORD-CLARK; TRUONG, 2013). The MQTT-SN was developed to not rely on network services. In addition, the MQTT-SN can be supported for any network which provides a bi-directional data exchange service between any node and a particular broker or gateway.

Some features of the MQTT-SN are (STANFORD-CLARK; TRUONG, 2013):

- To work with short message length and limited transmission bandwidth in a wireless network, the "topic name" in PUBLISH message is replaced for a two-byte “topic-id”. The clients register their topic name in the server/gateway and obtain the corresponding topic id;
- “Pre-defined” topic ids and “short” topic names are introduced. The short topics presents a length of two octets, so they are shorter enough for being carried simultaneously with data in PUBLISH messages;
- The discovery procedure is used to assist clients that do not know the server/gateway’s address to discover the network address;

- A new offline keep-alive procedure is defined to support sleeping clients. The battery-operated devices can go to a sleeping state, all messages designed to them are buffered at the server/gateway and delivered later when they wake up.

The differences between the MQTT and MQTT-SN protocols are presented in Table 2.

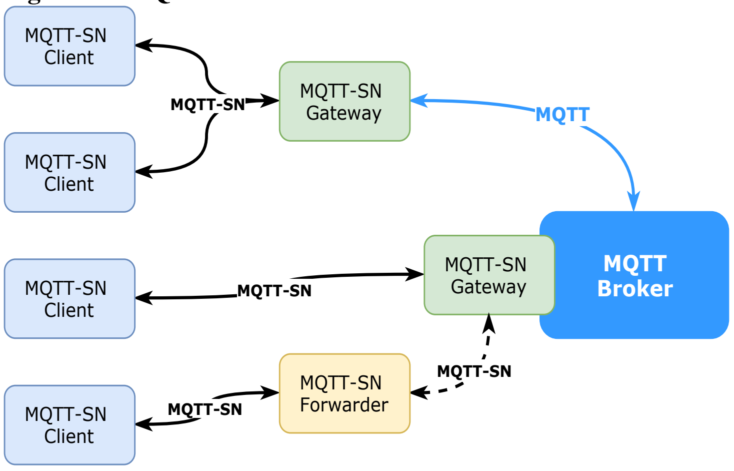
Table 2 – Difference between MQTT and MQTT-SN

Characteristic	MQTT	MQTT-SN
Transport	TCP	UDP / Other(ZigBee)
Replace topic for an Id	-	V
Pre-defined topics Id	-	V
Discovery procedures	-	V
Support for sleeping clients	-	V

Source: Author

The architecture of the MQTT-SN is illustrated in Figure 6. The classic architecture is composed of the MQTT-SN client (publisher), the MQTT-SN Gateway (GW), and the MQTT-SN forwarder. The MQTT-SN protocol transfer messages between an MQTT-SN client and a broker using an MQTT-SN GW as the middleware. The function of an MQTT-SN GW is to translate messages from MQTT to MQTT-SN or vice-versa, if it is a stand-alone topology. The MQTT-SN Forwarder is configured when the MQTT-SN GW is not present in the same network (STANFORD-CLARK; TRUONG, 2013).

Figure 6 – MQTT-SN architecture



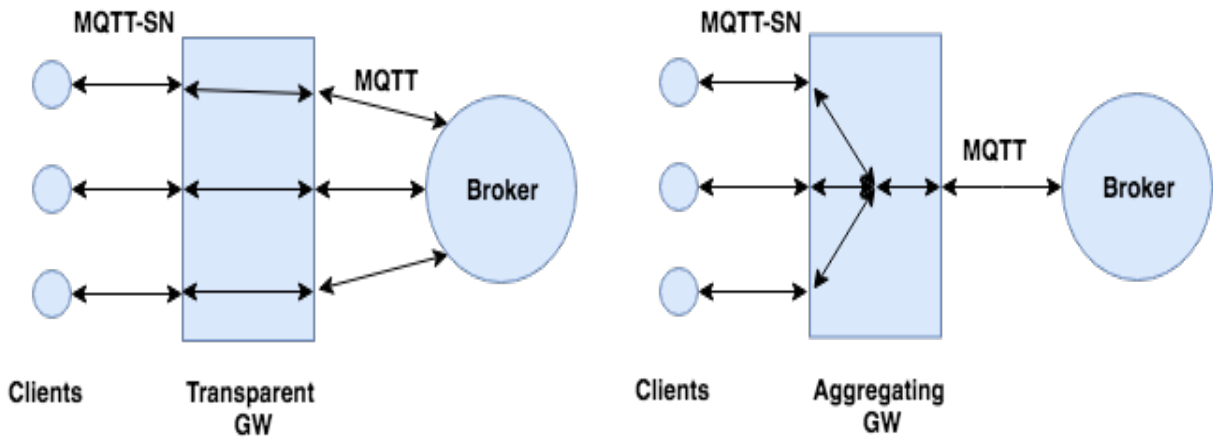
Source: Adapted from (STANFORD-CLARK; TRUONG, 2013)

To access a MQTT-SN GW that is not attached in the same network topology is used a MQTT-SN forwarder. The forwarder encapsulates the MQTT-SN frames when receives them from the gateway and sends to the clients. There are two types of GWs to translate frames (make a syntax translation) between MQTT and MQTT-SN, and vice versa. They are named Transparent and Aggregating GWs (STANFORD-CLARK; TRUONG, 2013).

The Transparent and Aggregating Gateways are shown in Figure 7. The Transparent GW will setup and maintain a connection from each MQTT-SN client to the MQTT server. The

MQTT connection is reserved exclusively for the end-to-end connections, and almost transparent message exchange between the client and the server. The Transparent GW will perform a "syntax" translation between the two protocols. All function and features implemented by the server can be offered to the client (STANFORD-CLARK; TRUONG, 2013).

Figure 7 – Transparent and Aggregating Gateways



Source: Adapted from (STANFORD-CLARK; TRUONG, 2013)

The implementation of Transparent GW is simpler, when compared to the Aggregating GW, because it requires the MQTT server to support a separate connection for each active client. However, the MQTT server might impose a limitation on the number of supported concurrent connections.

Aggregating GW will retain only one MQTT connection to the server. The Aggregating GW will decide which information will be given later to the server. This implementation is more complex if compared with the Transparent GW, but an Aggregating GW can be used with WSNs with a considerable number of Sensors and Actuators (SAs) because it reduces the number of MQTT connections (STANFORD-CLARK; TRUONG, 2013).

The MQTT-SN protocol was developed to be as close as possible to the MQTT protocol. However, it is adapted to the particularities of the WSNs. The messages are sent from a publisher to a topic in the middleware named broker. The broker transfers the messages to the subscripted subscribers to this topic in the broker. To transfer the messages both protocols, MQTT and MQTT-SN start with CONNECT message from the client to the server. The server responses with a CONNACK message. To finish the communication, the client sends a DISCONNECT message to the server. During the communication, three levels of Quality of Service are used to exchange the messages (BANKS; GUPTA, 2014; BOYD *et al.*, 2014; LAMPKIN *et al.*, 2012; LEE *et al.*, 2013).

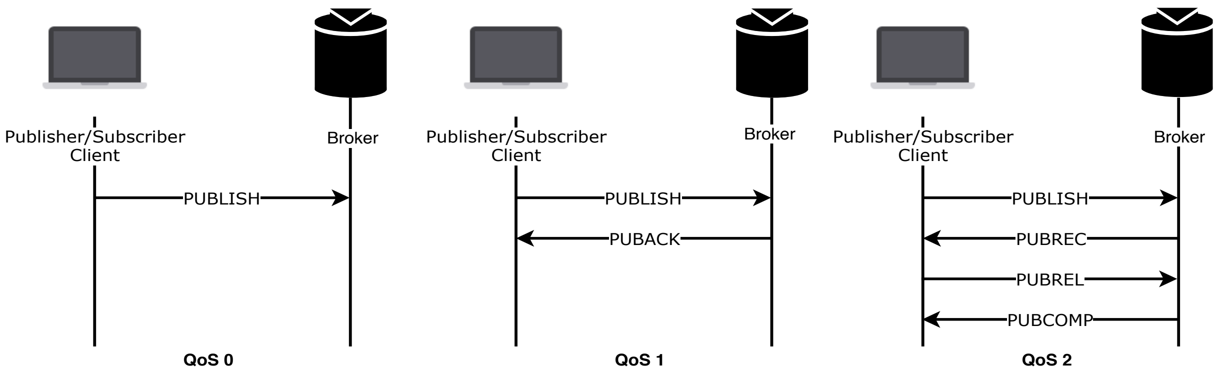
The QoS 0 (at most once) sends a PUBLISH message according to network capabilities. There is no retransmission if a sent message is lost. The message arrives once at the receiver or not at all (BANKS; GUPTA, 2014).

The QoS 1 (at least once) keeps the message in the client, sends a PUBLISH message and waits for a PUBACK from the broker, that is the message/packet confirmation. In the case the message does not arrive, a PUBLISH message will be sent until a PUBACK message is received, or the time to retransmit is over. This QoS ensures the message arrives at the subscriber/receiver at least once. However, duplicate messages can arrive at the receiver until the subscribe/client receives the confirmation message (PUBACK) (BANKS; GUPTA, 2014).

The QoS 2 (exactly once) keeps the message in the publisher, sends a PUBLISH message and waits for a PUBREC message. When the PUBREC message arrives, the publisher discards the message and keeps the PUBREC message, sends a PUBREL message and waits for a PUBCOMP message and finally discards the PUBREL message. It is the highest QoS level and appropriated to use when neither loss nor duplication messages are acceptable in the communication. This QoS promotes an increase of overhead (BANKS; GUPTA, 2014; BOYD *et al.*, 2014).

The QoS levels are illustrated in Figure 8.

Figure 8 – Packet Transmission QoS Levels


Publisher/Subscriber Client
Broker
PUBLISH
QoS 0
Publisher/Subscriber Client
Broker
PUBLISH
PUBACK
QoS 1
Publisher/Subscriber Client
Broker
PUBLISH
PUBREC
PUBREL
PUBCOMP
QoS 2

Source: Author

There is a specific QoS, named QoS Level-1, implemented in the MQTT-SN. The QoS Level-1 is defined to be used with simple devices that do not support any other QoS (STANFORD-CLARK; TRUONG, 2013). The QoS level-1 do not have the operations: connection setup; disconnect; registration; nor subscription. The client sends a PUBLISH message to a GW without knowing the GW address, also do not care if the GW address is correct, if it is alive, or if the message arrives at the GW.

The MQTT-SN specification defines best practices that can be adopted when the MQTT-SN is implemented. The best practices are presented in Table 3.

2.5 REALLY SMALL MESSAGE BROKER (RSMB)

The Really Small Message Broker (RSMB) was developed by Ian Craggs, that is a software engineer at the IBM. The RSMB is a small server with 80 KB of storage that can run with 200 KB of memory. Moreover, it uses lightweight MQTT publish/subscribe protocol to distribute messages between applications (CRAGGS, 2014). The reduced size of the RSMB enables mes-

Table 3 – Best practice values for timers and counters

Description	**Timer/Counter**	**Recommended value**
Time ADVERTISE message. The ADVERTISE message is broadcasted periodically by a gateway to advertise its presence	T_{ADV}	greater than 15 minutes
Number of ADVERTISE message	N_{ADV}	2 - 3
The SEARCHGW message. The SEARCHGW message is broadcasted by a client when it searches for a GW	$T_{SEARCHGW}$	5 seconds
The GWINFO message. A GWINFO message is sent as a response to a SEARCHGW	T_{GWINFO}	5 seconds
A client should wait for a time T_{WAIT} before restarting the registration procedure	T_{WAIT}	greater than 5 minutes
The time to retry a message starts when a client sends a message and stops when the expected GW's reply. In the case of the message times out and the expected GW's reply is not received, the client retransmits the message until it is received	T_{retry}	10 - 15 seconds
The Number of retries to send a message. In the case of the number of retransmissions end, the client aborts the procedure and assumes its MQTT-SN connection to the gateway is disconnected	N_{retry}	3 - 5

Source: Adapted from (STANFORD-CLARK; TRUONG, 2013)

saging from very small devices like sensors and actuators in networks with low bandwidth. It allows publishers to send messages to the broker and the broker transfers the messages to subscribers, who want to receive those messages. Furthermore, the RSMB can run in embedded systems to provide infrastructure in remote installations.

The RSMB transfers messages between applications over TPC/IP network connections. The data can be exchanged from a variety of sources (applications, other brokers, sensors, etc). The broker was written in C language, and it has a straightforward internal design and external features. Moreover, it provides MQTT clients in C and Java languages. The broker was developed to execute on some platforms (Linux for Intel-32-bit, Linux for Intel-64-bit, Linux on IBM, Linux for ARM XScale, Windows XP, and MacOSX).

2.6 RELATED WORKS

There are some related works in the literature which proposes publish/subscribe approaches for WSNs. The proposed papers satisfy the QoS reliability and others timeliness requirements. Some authors provide a comparison of the MQTT with other protocols.

In (DAVIS; AUGé, 2018) the authors presented a mechanism that establishes various QoS levels using the publish/subscribe model for wireless networks to provide reliable delivery of packet and timeliness. The authors proposed three QoSs: the QoS 2 and QoS 3 are based on the QoS 1. The QoS 1 calculates the Retransmission Timeout (RTO) and adjusts the RTO depending on the Packet Delivery Ratio (PDR). The QoS 2 provides data aggregation to reduce network congestion. The QoS 3 promotes the timeliness on the delivery packet. The propose of the QoS developed by the authors is to adapt to the different application requirements.

An implementation of the MQTT-SN protocol in an end-to-end e-health system was presented in (GOVINDAN; AZAD, 2015). The study promoted an end-to-end delay assurance and sufficient probability of content delivery in WSN and IoT systems. The sensors are simulated that are mounted on the human body to transfer data using the MQTT-SN protocol and a gateway.

The authors in (CHEN *et al.*, 2014) presented a survey with the technologies that are being used to connect Machine-to-Machine (M2M) networks. The authors examined the typical architectures of M2M home networks and discuss the performance of the present designs. In this survey is covered QoS, energy efficiency and security issues.

The author in (TONG; NGAI, 2012) presented a publish/subscribe platform for ubiquitous data access to mobile phones and sensors. The mobile phones are being used as mobile mules to relay subscriptions and to publish data between the Internet and the sensors. Also, the platform support data access from the mobile phones and sensors without any network infrastructure.

A simulation of the publish/subscribe model using OMNETT++ based on the Castalia simulator is presented in (TEKIN; SAHINGOZ, 2016). The simulation showed that the publish/-subscribe model decreases the number of messages transmitted, as a result, decrease the power consumption.

The author in (KATSIKEAS, 2016) implemented the MQTT for Wireless Sensors Nodes using the simulator Contiki OS to evaluate the security used on the IoT world with a real hardware platform. The study concluded that AES-CBC and LLSec are better implementations to provide security for MQTT.

In the literature, there is compassion between MQTT and other protocols. The authors in (HEđI; ŠPEH; ŠARABOK, 2017) compared the MQTT protocol and the CoAP (Constrained Application Protocol) for communication Machine-to-Machine and communication devices in the IoT world. The protocols are compared in alternative scenarios, and they concluded that MQTT is suitable for battery run devices and for applications that require massive updates with data that contains the same value.

The authors in (LEE *et al.*, 2013) analyzed the three QoS levels implemented on the MQTT protocol in wired and wireless networks using various sizes of payload. The authors analyzed the delay and message loss. The conclusion was that the delay and message loss maintained a direct relationship with the QoS level which was chosen to transfer the messages. The QoS 2 had the highest delay but had the least loss of messages. While the QoS 0 received the lowest delay and highest message loss.

The method developed in this work is distinctive from the other authors because it is focused on communication between the publisher and broker, it monitors the network latency and selects the proper QoS according to the network conditions.

3 METHODOLOGY

This Section presents the problem and the QoS Dynamic Adaptation Method using MQTT-SN as a solution. Two different tools were developed to understand how the QoSs work with MQTT. The tools are presented in APPENDIX A. The iPerf and Wireshark are network tools used to develop the QoS DAM method.

3.1 IPERF

The iPerf was originally developed by NLAND/DAST. However, the iPerf version 3 was developed by ESnet/Lawrence Berkeley National Laboratory. The iPerf is a tool used for measurements of the IP networks bandwidth. There are some protocols supported (TCP, UDP, SCTP with IPv4 and IPv6) (TIRUMALA *et al.*, 2006). Also, iPerf supports parameters related to timing and buffers. The tests provided by iPerf generate reports with the information about the bandwidth, loss and others parameters.

The iPerf tool is a cross-platform (Windows, Linux, Android MacOSX, FreeBSD, OpenBSD, NetBSD, Solaris). Some features are measured bandwidth, packet loss, delay, jitter, and multicast capable. The client and server can connect simultaneously, handling multiple connections, the tools can be configured to run at a specific time (TIRUMALA *et al.*, 2006).

3.2 WIRESHARK

The Wireshark is a network protocol analyzer. The project was started by Gerald Combs in 1998 and has been contributed by many networking experts around the globe (COMBS, 2006). The features of the Wireshark are a deep inspection of hundreds of protocols, live capture, and offline analyses. The analyzer is a multi-platform (Windows, Linux, macOSX, Solaris, FreeBSD). It is possible to use a GUI or a TTY-mode with TShark utility to capture network behavior, the output can be exported to XML, PostScript, CSV, or plain text.

3.3 PROBLEM STATEMENT

Regarding the IoT network solution, a massive part of the smart devices will be connected through Wireless Sensor Networks (WSNs). A suitable option to transfer messages in the IoT and Machine-to-Machine networks is the Message Queuing Telemetry Transport (MQTT) (AL-FUQAHA *et al.*, 2015). However, MQTT uses TCP/IP protocols that provide a complex kind of communication for extremely simple devices, not allowing connection to sensors that use other protocols like ZigBee (STANFORD-CLARK; TRUONG, 2013). Therefore, the MQTT for Sensor Networks (MQTT-SN) was developed to work in sensor networks with limited bandwidth, high link failures, and short message length.

Both protocols MQTT and MQTT-SN provide three Quality of Service (QoS) levels to transfer the messages. Each QoS level guarantees different level of message delivery. The QoS 0 does not provide a guarantee at all, if a message is lost it is not sent again. The QoS 1 ensures that the message arrives once, but duplicate messages may arrive. Finally, The QoS 2 guarantees that only a message will arrive, there will no be duplicated messages.

The impact of the QoS level in the communication is presented by the equations in Table 4.

Table 4 – Time to transfer messages according to the QoS level selected

Abbreviation	**Description**	**QoS**	pt	mt	**Equation**
pt	Time to send PUBLISH message	0	1	0	$n \cdot pt$
n	Number of messages	1	1	1	$n \cdot (pt + mt)$
mt	Confirmation messages (PUBACK, PUBREC, PUBREL, and PUBCOMP)	2	1	3	$n \cdot (pt + 3mt)$

Source: Adapted from (BOYD *et al.*, 2014; LAMPKIN *et al.*, 2012)

Considering a number of 10 messages, the time for pt equals $1s$ and time for mt equals $0.4s$. The QoS 0 requires 10 seconds to transfer 10 messages. The QoS 1 takes 40% more time to transfer the corresponding number of messages, while QoS 2 takes about 120%. When comparing QoS 2 and QoS 1 it is possible to see that QoS 2 takes 80% more time to send the corresponding number of messages.

The developed method chooses the proper QoS based on the network environment. By getting the latency value for specific intervals, it is calculated the median latency. Using this information the method chooses the best QoS, concerning to the original QoS chosen by the user.

4 SOLUTION PROPOSED

The goal of the developed method is to improve and guarantees the number of messages delivered in a WSN with unreliable wireless links and heavy traffic load. The latency intervals presented in Table 5 are proposed for testing the solution using the median latency (ML) variation. The latency intervals can be adapted for different wireless network environments. There are three scenarios proposed to select the best QoS based on the current performance of the WSN.

The latency intervals for the solution were based in the studies of the latency on wireless networks in (PATRO; GOVINDAN; BANERJEE, 2013; SUI *et al.*, 2016; SYED; HEIDEMANN; others, 2006). The method gets information about the latency in the network and decides which QoS is proper for the current moment.

The method works on the publisher's side. The user informs which QoS level wants to use to communicate. The method gets network latency values for a period of time and calculates the median latency variation to select the best QoS to consider. The median latency is calculated as in (SHRIVASTAVA *et al.*, 2004). Based on the calculated value, the method returns which are the best QoS to transfer the messages in a network environment, adapting it if necessary.

When the method changes the QoS of one or more messages, it will transfer all these messages to the new selected choice, until the latency is measured again. Based on the new value of the median latency the method can assign or not a new QoS level. That kind of operation will be processed until all messages are delivered to the broker.

According to the median latency value and the intervals presented in Table 5, the network status is considered. If it is considered as “Good”, the method changes the QoS selected by the publisher, if the initial one was QoS 1 or QoS 2 to QoS 0. Meaning that this change continues to guarantee messages delivery without overload the network. It is important to note that even with this change, the message will be delivered once.

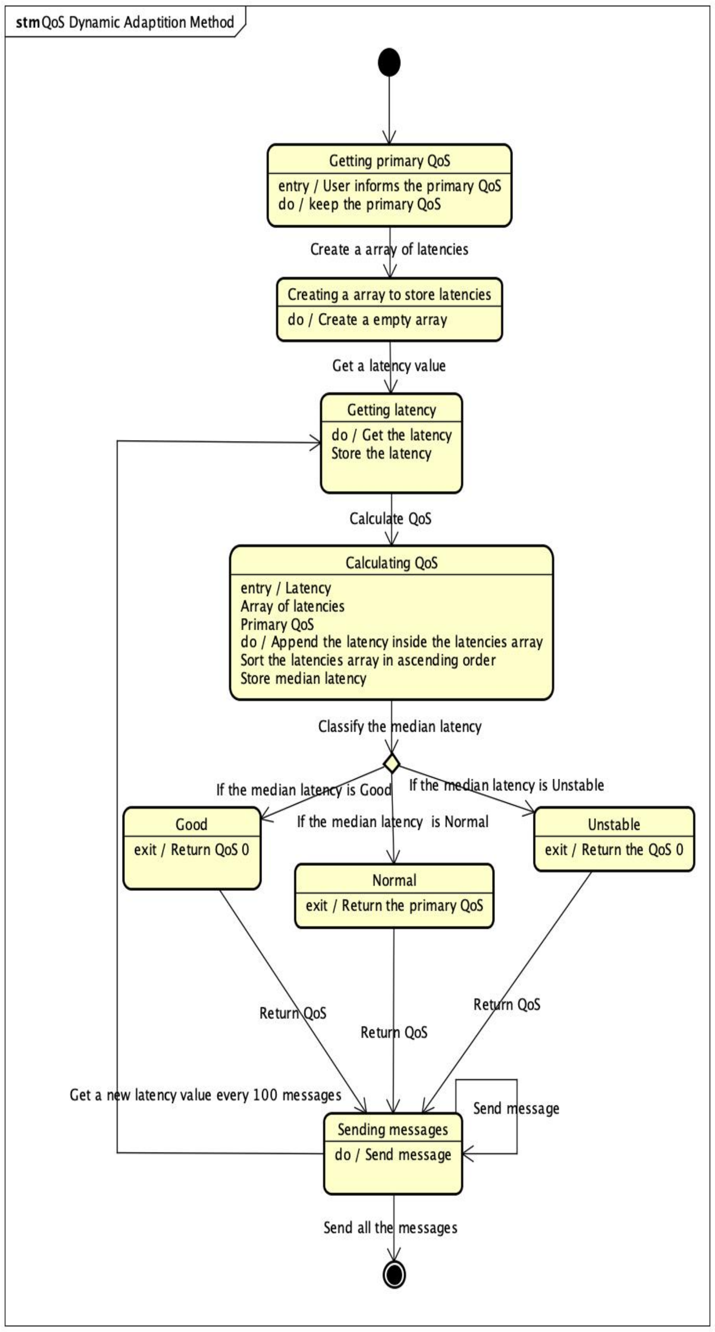
If the network is considered as "Unstable", the initial QoS selected by the user to publish is changed to QoS 0. The idea is to reduce the network load. To this unstable moment, by choosing QoS 0 the method offers more chances the message be delivered if compared to QoS 1 and QoS 2 choices. In the case of message of confirmation is lost and the time to retransmit the message is over, the client aborts the procedure and assumes that the gateway is disconnected (STANFORD-CLARK; TRUONG, 2013).

Table 5 – Latency interval for each scenario

Latency Interval	Good	Normal	Unstable
1	$ML \leq 10$ ms	10 ms $< ML \leq 20$ ms	20 ms $< ML$
2	$ML \leq 15$ ms	15 ms $< ML \leq 30$ ms	30 ms $< ML$
3	$ML \leq 30$ ms	30 ms $< ML \leq 60$ ms	60 ms $< ML$
QoS selected	QoS 0	QoS selected by the user	QoS 0

Source: Author

Figure 9 – QoS Dynamic Adaptation Method


stm QoS Dynamic Adaptition Method
Getting primary QoS
entry / User informs the primary QoS
do / keep the primary QoS
Create a array of latencies
Creating a array to store latencies
do / Create a empty array
Get a latency value
Getting latency
do / Get the latency
Store the latency
Calculate QoS
Calculating QoS
entry / Latency
Array of latencies
Primary QoS
do / Append the latency inside the latencies array
Sort the latencies array in ascending order
Store median latency
Classify the median latency
If the median latency is Good
If the median latency is Unstable
If the median latency is Normal
Good
exit / Return QoS 0
Normal
exit / Return the primary QoS
Unstable
exit / Return the QoS 0
Return QoS
Return QoS
Return QoS
Get a new latency value every 100 messages
Sending messages
do / Send message
Send message
Send all the messages

Source: Author

When median latency is in the interval considered as “Normal”, the method returns the current QoS to the original one, selected by the user to publish at the beginning of the communication. For example, if the user initially chooses QoS 2 to transfer the messages, and along with the communication process the network condition is classified as “Good” or “Unstable” the method changes the QoS 2 to QoS 0. When the network condition returns to “Normal” the method returns to use QoS 2. In case of the user initially selects QoS 0, the method does not realize any change. The state machine diagram is presented in Figure 9.

The method calculates which QoS will be used every time a new median latency is calculated. Hence, the goal is not to overload the network providing message delivery by the

QoS DAM.

4.1 TEST SCENARIO EVALUATION

The architecture scenario is composed of the devices presented in Table 6, a computer as an MQTT-SN publisher MacBook Air connected to a single broker. The broker MQTT-SN is installed in a Raspberry. The method was developed to work between the publisher and a broker. The Raspberry was configured as MQTT-SN GW to a subscriber that was a Mosquitto client tool. One more Raspberry was used to overload the network using the software iPerf with 10 Mbits/s of UDP packets. Finally, the last component of the test environment is a Wireless Router D-LINK DIR-611 with 300 Mbps configured with 25 % of its capacity, that represents 1.25 dBi antenna power gain. The test scenario is illustrated in Figure 10.

Table 6 – Devices specifications

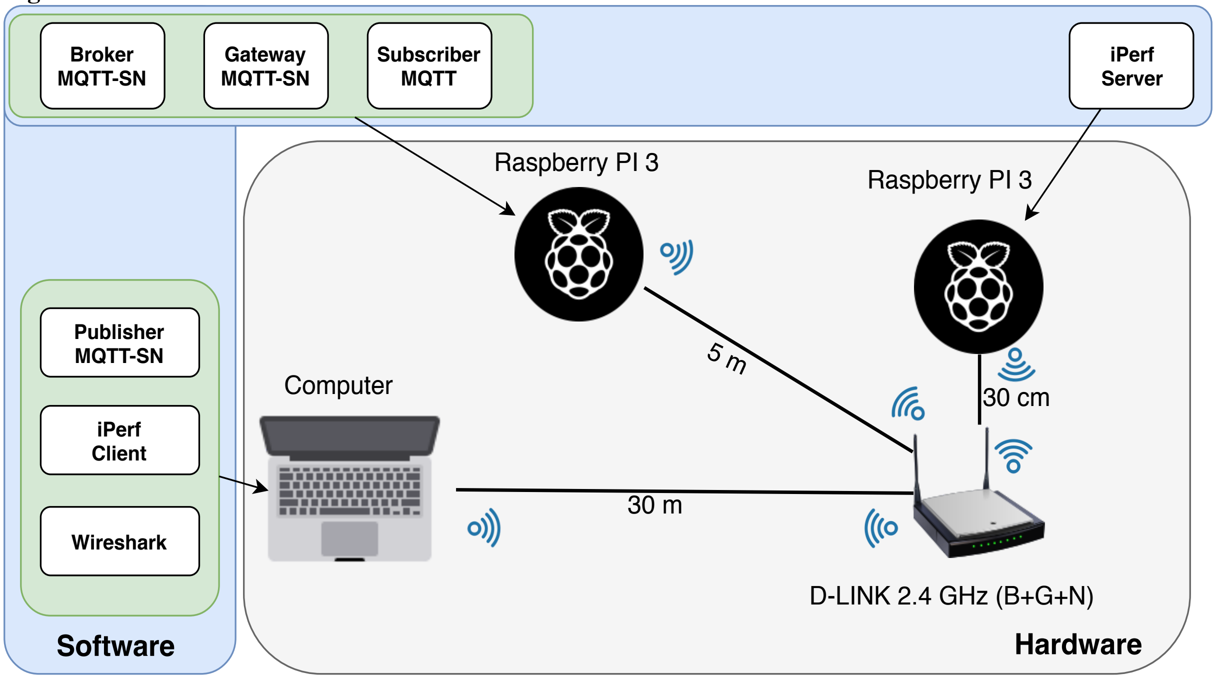
Device	Processor	Memory	Wireless	Storage	S.O.
MacBook Air (13-inch, Mid 2013)	1,3 GHz Intel Core i5	4 GB DDR3 (1600MHz)	AirPort Extreme/ Broadcom BCM43xx 1.0 802.11 a/b/g/n/ac	251 GB APPLE SSD SM0256F	macOS High Sierra 10.13.4
Raspberry PI 3 Model B	Quad Core 1.2 GHz Broadcom BCM2837 64 bit CPU	1 GB LPDDR2 (900 MHz)	2.4 GHz 802.11.b/g/n Wireless	microSDHC C4 16 GB	NOOBS
Raspberry PI 3 Model B	Quad Core 1.2 GHz Broadcom BCM2837 64 bit CPU	1 GB LPDDR2 (900 MHz)	2.4 GHz 802.11.b/g/n Wireless	microSDXC C10 16 GB	NOOBS
D-LINK DIR-611			2.4 GHz 802.11.b/g/n Wireless		

Source: Author

4.2 TEST SCENARIO IMPLEMENTATION

The tests were performed in a real network topology. The architecture scenario considers a computer connected to a single MQTT-SN broker. The position of the devices was fixed. The broker works as an MQTT-SN gateway and MQTT-SN broker. The method was developed

Figure 10 – Test scenario



Source: Author

to select the best QoS to send messages from the publisher to a broker. The scenario was constructed to simulate a network with links lossy by reducing the antenna power gain (dBi) and heavy traffic using iPerf to overload the network.

The radio signal strength is measured in dBm or decibel milliwatt that is a real continuous signal, whereas the signal strength indicator (RSSI) is implemented for each developer company (KAEMARUNGSI, 2006). Experiments conducted by the authors in (KAEMARUNGSI, 2006) and additional analysis of measurements done in all WLAN cards, from the scenario, lead to a considered dBm interval between -70 dBm and -80 dBm, representing a lower sensitivity condition in the test topology. The values are chosen to represent a topology with smart devices far from the router. It was considered a distance of 30 meters from the MQTT-SN publisher and the wireless router (SIBONI; SHABTAI; ELOVICI, 2018). A distance of 30 centimeters was considered from the second Raspberry and the wireless router. In this equipment was simultaneously executed the software iPerf to generate UDP traffic with 10 Mbits/s, to overload the network environment. The test script was written using Python language provided by the client library to RSMB (CRAGGS, 2014). A Mosquitto client tool (LIGHT, 2017) is used to receive messages implementing the MQTT protocol when the RSMB is used as an MQTT-SN gateway and MQTT broker at the same time.

The test was conducted, by transmitting 10 groups of 5,000 messages, by each of the three QoS levels implemented by MQTT-SN protocol. Each message has a specified size of 64 bytes. Every 100 messages the method estimate the latency using a *ping* command between the publisher and the broker. It was chosen 100 messages to generate a window where the method will adapt and do no lost time to publish the messages. The time to publish is measured, it concerns to the time taken to send 5,000 from the publisher to the broker considering the time to get the latency. The system computes the median latency. This median latency value was compared with the values presented in Table 5. At this point, the method chooses the best QoS to transfer the messages. When the median latency value is in the interval considered as "Normal" in Table 5, the QoS used in the communication is the QoS selected by the user. In case of the median latency value is classified as "Good" or "Unstable" the initial QoS changes to QoS 0.

The test scenario includes some specific procedures. As presented in Figure 10, all the connections were wireless. To manage the machines remotely it was established a connection between the Raspberries and the computer, using SSH protocol (YLONEN; LONVICK, 2005). The following step was to start the MQTT-SN RSMB broker in the Raspberry that is fixed 5m from the wireless router. In the other Raspberry was started the iPerf server tool to overload the network with 10 Mbits/s UDP packets. While the communication process occurs, the traffic in the background increases the latency and generates packet loss. In a sequence 5,000 messages are sent, using the MQTT-SN protocol without the method and later the MQTT-SN protocol using the developed method, QoS DAM, from the publisher to the broker. The data about the communication process was collected and analyzed. The steps are briefly listed:

1. To start the MQTT-SN in RSMB broker localized a 5 m of the wireless router through a SSH remote connection.
2. To start the iPerf server tool using SSH remotely.
3. To start the iPerf client tool, to overload the network with 10 Mbits/s UDP packets.
4. To start the Wireshark tool in the publisher computer.
5. To send 5,000 messages using the MQTT-SN protocol to the broker.
6. To get the number of messages implementing the Mosquitto client tool to manage the messages that arrive in the broker.
7. To stop Wireshark.
8. To stop iPerf.
9. To save the information from the publisher, subscribe and Wireshark. To extract the time interval to publish all the messages.

The goal of the test scenario was to verify how the proposed QoS DAM works under high traffic and latency variation.

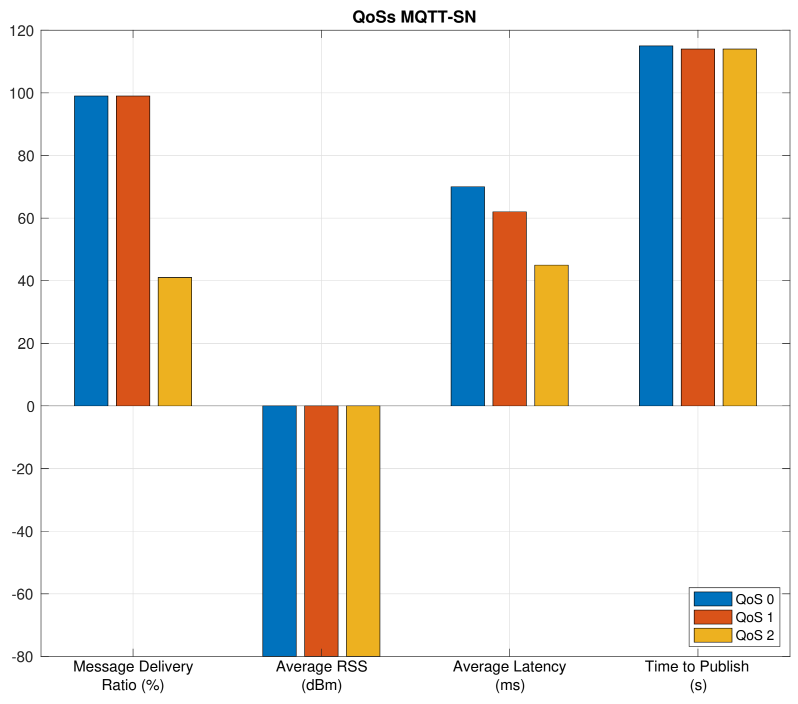
5 RESULTS

This Section presents the experimental results and discusses about the performance of the proposed method. Section 5.1 presents the results the original QoS mechanism provided by the MQTT-SN protocol. Sections 5.2 and 5.3 show the results for the proposed method.

5.1 RESULTS WITH THE ORIGINAL MQTT-SN

The average results obtained without using the QoS DAM are presented in Figure 11. The tests were repeated 10 times for each level of MQTT-SN QoS (QoS 0, QoS 1, and QoS 2). Figure 11 shows that initially using QoS 0 and QoS 1 the percentage of delivered messages were 99%. However, initially using QoS 2 the results showed merely 41% of messages delivered.

Figure 11 – Comparison of the tests using QoS provided by the original MQTT-SN



Source: Author

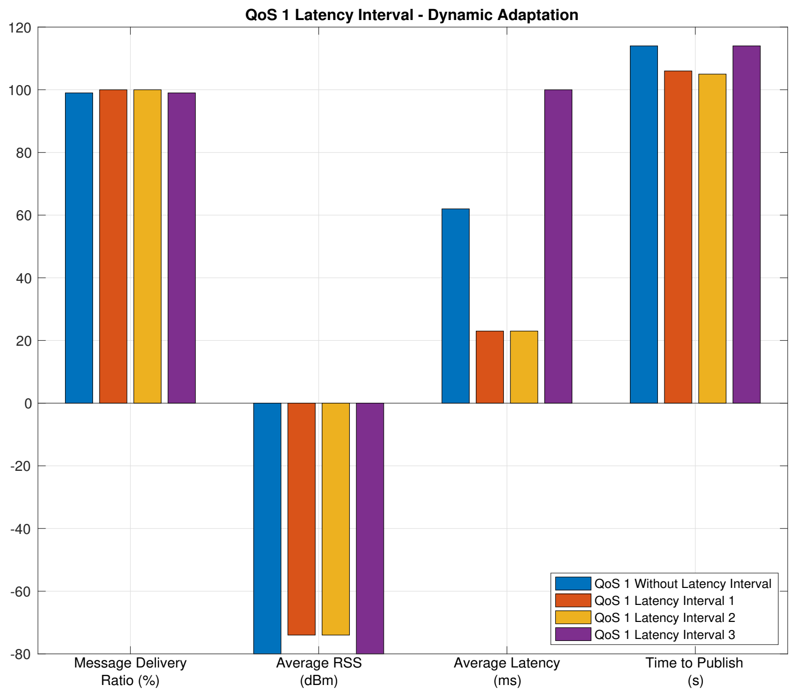
The RSS is the same for all the QoS (-80 dBm). Even with more latency during the communication process (latency of 70 ms) the MQTT-SN using QoS 0 delivered the same percentage of messages than the MQTT-SN using QoS 1 (latency of 62 ms). Although, the MQTT-SN using QoS 2 presented the lowest latency, the number of delivered messages was worse. The time to publish 5,000 messages was closer for all QoS techniques, approximately 114 seconds for QoS 1, and QoS 2. The time to publish all the messages using QoS 0 was 115 seconds.

It is possible to conclude the QoS 0 and QoS 1 presents a similar performance in the experimental scenario. The QoS 2 which requires a more complex communication process presents a reduced efficiency in terms of delivered messages.

5.2 RESULTS WITH QOS DAM – QOS 1

Figure 12 presents the results considering messages exchange from MQTT-SN, without any changes, and the results from the three different latency intervals in Table 5, considering QoS DAM method. Where QoS 1 was initially considered. The proposed method uses the median latency to select the best QoS to communicate. In this scenario, the method changes QoS 1 to QoS 0 if the median latency is considered “Good or Unstable” and return to the original one, QoS 1, if the network returns to latency values considered "Normal".

Figure 12 – Comparison between MQTT-SN QoS 1 and QoS DAM beginning with QoS 1



Source: Author

When considered latency intervals 1 and 2 it was delivered 100% of the messages. The intervals presented the lowest signal strength average (-74 dBm) because of the variation of signal strength during the communication process. The latency average (23 ms) and time to publish all the messages (106 s and 105 s, respectively) were also lower.

Considering Table 5 - latency interval 3 by the QoS DAM and MQTT-SN using initially QoS 1, the same percentage of messages delivered (99%) was presented. Also, the identical time to publish all the messages. However, latency interval 3 presented the best performance because the QoS DAM kept the percentage of messages delivered about 99%, with a higher latency (100 ms).

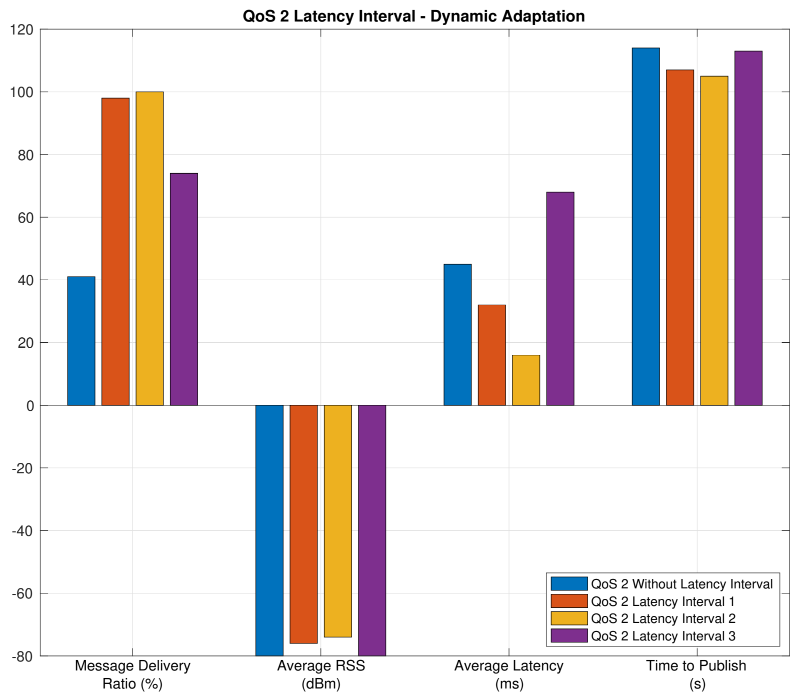
The QoS DAM using Table 5 - latency interval 3 had the best performance if compared with MQTT-SN and the two other latency intervals in Table 5. To the most pessimistic scenario QoS DAM maintained the performance.

5.3 RESULTS WITH QOS DAM – QOS 2

The method changes from QoS 2 to QoS 0 in case of the latency interval is classified as "Good" or "Unstable”. It returns from QoS 0 to QoS 2 when the latency interval returns to be classified as "Normal".

Figure 13 presents the results for the MQTT-SN protocol using QoS 2. Without any change during all the communication process, and the QoS DAM considering Table 5 - latency intervals.

Figure 13 – Comparison between MQTT-SN QoS 1 and QoS DAM beginning with QoS 2



Source: Author

The QoS DAM obtained a better performance when processing messages of QoS 2. Table 5 - latency intervals 1 and 2 maintained the performance in messages delivering (98% and 100% respectively). Moreover, the intervals got the lower latency (32 ms and 16 ms), and signal strengths (-76 dBm and -74 dBm), and the lowest time to publish all the messages (107 s and 105 s), respectively.

The Table 5 - latency interval 3 presented the worst latency but delivered about 74% of the messages with a signal strength average of -80 dBm and average latency of 68 ms. The time to publish was 113 s.

The MQTT-SN using QoS 2 presented the worst performance. It had 41% of the messages delivered. The signal strength average is the same as Table 5 - latency interval 3 (-80 dBm), however, the messages delivered that can be seen in Figure 13 for MQTT-SN using QoS 2 was 33% lower compared with QoS 2 latency interval 3. Also, MQTT-SN using QoS 2 had the worst time to publish all messages (114 s).

6 CONCLUSIONS

The goal of this study was to develop a method that can be used in Wireless Sensor Network (WSN) to keep the message delivery in networks with unstable connections and overload. The Quality of Service Dynamic Adaptation Method (QoS DAM) was developed to improve the message exchange process and the messages delivery time. It was chosen as the main network parameter, the latency interval.

We have performed an experimental and theoretical study of wireless sensors' communication, implementing the protocol Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN). The tests showed that considering latency intervals, to adjust QoS levels, the communication process may be improved in overloads networks. The number of messages delivered, mainly by QoS 2 the one that most penalizes the performance of the network, can be improved in overloaded networks.

The tests also showed that with a higher latency the QoS DAM method keeps the message delivery using the QoS 1 when it is compared with QoS 1 from the MQTT-SN.

6.1 FUTURE WORKS

There are some features that can be explored to improve the QoS DAM for the MQTT-SN also for MQTT:

- To explore more latency intervals.
- To implement the test on simulators.
- To employ Radio Signal Strength (RSS) interval to adapt the QoS.
- To use RSSI combined with latency to determine the interval to select the QoS.
- To implement the method in others protocols that use the paradigm publish/subscribe.
- To implement the method to set the Retransmission Timeout (RTO) and increase or decrease the window interval for message timeout.
- To exercise artificial intelligence to select the best latency interval to be considered when necessary to make a dynamic adaptation. Also, we want to study the implementation of artificial intelligence to select QoS based on the position of the sensors, as an example of the authors in (KARABOGA; OKDEM; OZTURK, 2012), producing a cluster in WSN using Artificial Bee Colony Algorithm.

REFERENCES

AL-FUQAHA, Ala *et al.* Internet of things: A survey on enabling technologies, protocols, and applications. v. 17, n. 4, p. 2347–2376, 2015.

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

BANKS, Andrew; GUPTA, Rahul. MQTT version 3.1. 1. v. 29, 2014.

BOYD, B. *et al.* **Building Real-time Mobile Solutions with MQTT and IBM MessageSight**. IBM Redbooks, 2014. ISBN 978-0-7384-4005-7. Available at: <https://books.google.com.br/books?id=R3tNBQAAQBAJ>.

CHEN, M. *et al.* A survey of recent developments in home m2m networks. v. 16, n. 1, p. 98–114, 2014. ISSN 1553-877X.

COMBS, Gerald. **Wireshark**. 2006. Available at: <http://www.wireshark.org>.

CRAGGS, Ian. **Really small message broker**. 2014. Available at: <https://www.ibm.com/developerworks/community/alphaworks/tech/rsmb>.

____. Conference, **MQTT-SN: MQTT for UDP, ZigBee and Other Transports | EclipseCon NA 2015**. 2015. Available at: <https://www.eclipsecon.org/na2015/session/mqtt-sn-mqtt-udp-zigbee-and-other-transports.html>.

DAVIS, Ernesto García; AUGé, Anna M Calveras. Publish/subscribe protocol in wireless sensor networks: improved reliability and timeliness. v. 12, n. 4, p. 1527–1552, 2018.

FOROUZAN, Behrouz A. **Comunicação de dados e redes de computadores**. [S.l.]: AMGH Editora, 2009.

GANTZ, John; REINSEL, David. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. v. 2007, n. 2012, p. 1–16, 2012.

GOVINDAN, K.; AZAD, A. P. End-to-end service assurance in IoT MQTT-SN. In: **2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)**. [S.l.: s.n.], 2015. p. 290–296.

HEđI, I; ŠPEH, I; ŠARABOK, A. IoT network protocols comparison for the purpose of IoT constrained networks. In: **Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017 40th International Convention on**. [S.l.]: IEEE, 2017. p. 501–505.

KAEMARUNGSI, Kamol. Distribution of WLAN received signal strength indication for indoor location determination. In: **Wireless Pervasive Computing, 2006 1st International Symposium on**. [S.l.]: IEEE, 2006. p. 6–pp.

KARABOGA, Dervis; OKDEM, Selcuk; OZTURK, Celal. Cluster based wireless sensor network routing using artificial bee colony algorithm. v. 18, n. 7, p. 847–860, 2012. ISSN 1572-8196. Available at: <https://doi.org/10.1007/s11276-012-0438-z>.

KATSIKEAS, S. A lightweight and secure MQTT implementation for wireless sensor nodes. 2016.

LAMPKIN, Valerie *et al.* **Building smarter planet solutions with mqtt and ibm websphere mq telemetry**. [S.l.]: IBM Redbooks, 2012.

LEE, Shinho *et al.* Correlation analysis of MQTT loss and delay according to QoS level. In: **Information Networking (ICOIN), 2013 International Conference on**. [S.l.]: IEEE, 2013. p. 714–717.

LIGHT, Roger A. Mosquitto: server and client implementation of the MQTT protocol. v. 2, n. 13, 2017.

LOCKE, Dave. Mqtt v3. 1 protocol specification. **International Business Machines Corporation (IBM) and Eurotech**, p. 42, 2010.

____. **MQTT: Enabling the Internet of Things**. 2013. Available at: <https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/tc_overview?lang=en>.

MANYIKA, James. **The Internet of Things: Mapping the value beyond the hype**. [S.l.]: McKinsey Global Institute, 2015.

PATRO, Ashish; GOVINDAN, Srinivas; BANERJEE, Suman. Observing home wireless experience through wifi aps. In: **Proceedings of the 19th annual international conference on Mobile computing & networking**. [S.l.]: ACM, 2013. p. 339–350.

ROSE, Karen; ELDRIDGE, Scott; CHAPIN, Lyman. The internet of things: An overview. p. 1–50, 2015.

ROSSUM, Guido Van; others. Python programming language. In: **USENIX Annual Technical Conference**. [S.l.: s.n.], 2007. v. 41, p. 36.

SCHILDT, Herbert. **C: The complete reference**. [S.l.: s.n.], 1990.

SHRIVASTAVA, Nisheeth *et al.* Medians and beyond: New aggregation techniques for sensor networks. In: **Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems**. ACM, 2004. (SenSys '04), p. 239–249. ISBN 1-58113-879-2. Available at: <http://doi.acm.org/10.1145/1031495.1031524>.

SIBONI, Shachar; SHABTAI, Asaf; ELOVICI, Yuval. Leaking data from enterprise networks using a compromised smartwatch device. In: **Proceedings of the 33rd Annual ACM Symposium on Applied Computing**. ACM, 2018. (SAC '18), p. 741–750. ISBN 978-1-4503-5191-1. Available at: <http://doi.acm.org/10.1145/3167132.3167214>.

STANFORD-CLARK, Andy; TRUONG, Hong Linh. Mqtt for sensor networks (mqtt-sn) protocol specification. v. 1, 2013.

STANKOVIC, John A. Research directions for the internet of things. v. 1, n. 1, p. 3–9, 2014.

SUI, Kaixin *et al.* Characterizing and improving wifi latency in large-scale operational networks. In: **Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services**. [S.l.]: ACM, 2016. p. 347–360.

SYED, Affan A; HEIDEMANN, John S; others. Time synchronization for high latency acoustic networks. In: **Infocom**. [S.l.: s.n.], 2006. v. 6, p. 1–12.

TANENBAUM, A.S.; WETHERALL, D.J.; TRANSLATIONS, O. **Redes de computadores**. PRENTICE HALL BRASIL, 2011. ISBN 9788576059240. Available at: <https://books.google.com.br/books?id=fnfwkQEACAAJ>.

TEKIN, Y.; SAHINGOZ, O. K. A publish/subscribe messaging system for wireless sensor networks. In: **2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP)**. [S.l.: s.n.], 2016. p. 171–176.

TIRUMALA, Ajay *et al.* **Iperf**. 2006. Available at: <https://iperf.fr/>.

TONG, X.; NGAI, E. C. H. A ubiquitous publish/subscribe platform for wireless sensor networks with mobile mules. In: **2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems**. [S.l.: s.n.], 2012. p. 99–108.

TORRES, Andrei BB; ROCHA, Atslands R; SOUZA, José Neuman de. Análise de desempenho de brokers mqtt em sistema de baixo custo. In: **Anais do XXXVI congresso da sociedade brasileira de computaç ao. Sociedade Brasileira de Computaçao**. [S.l.: s.n.], 2016.

ULLAH, Mohammad Hasmat *et al.* A collaboration mechanism between wireless sensor network and a cloud through a pub/sub-based middleware service. In: **The Fifth International Conference on Evolving Internet**. [S.l.]: Citeseer, 2013. p. 38–42.

WANG, Lucy. **Building Facebook Messenger**. 2011. Available at: <https://www.facebook.com/notes/facebook-engineering/buildingfacebook-messenger/10150259350998920>.

YLONEN, Tatu; LONVICK, Chris. **The secure shell (SSH) protocol architecture**. 2005.

APPENDIX A - DEVELOPED TOOLS

Two tools were developed to understand how the Quality of Service works with the MQTT protocol and MQTT-SN protocol.

6.2 CLIENT ECLIPSE MOSQUITTO

The Eclipse Mosquitto Broker is written in C language, however, the author provides an API, for developers, to develop client applications to Eclipse Mosquitto broker. This API was used to develop new functionalities needed for testing the Mosquitto broker in our proposal.

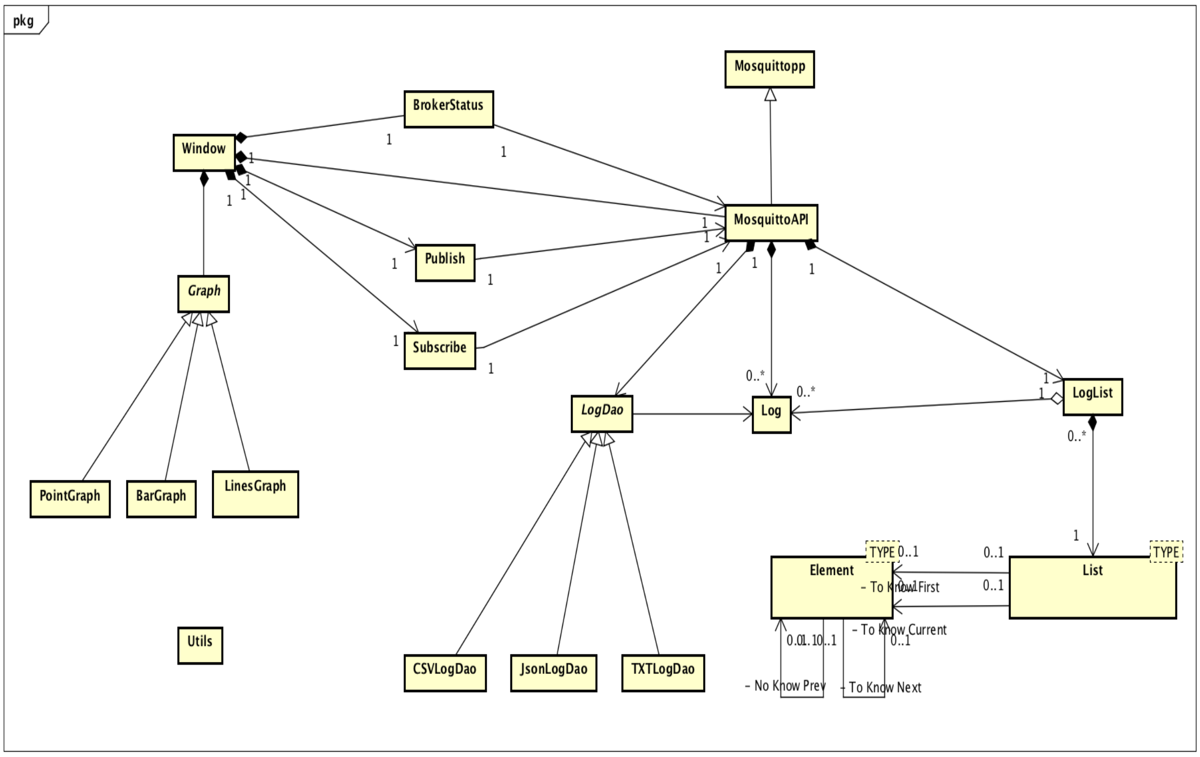
There are client applications that allow the user to test the Mosquitto broker e.g., (MQTT.fx[1], mqtt-spy[2], MQTT inspector[3], MyMQTT[4], HiveMQ WebSocket Client[5], MQTT Lens[6], and mosquitto_tools[7] However, these applications do not product statistics about the message exchange process. So, arises the need to create a tool to have these information.

The Client Eclipse Mosquitto tool was developed based on the Eclipse Mosquitto API for C++ developers. The new tool has some features to improve the management and provide a better comprehension of how the MQTT works. One of the main features, that was implemented, was a graphical presentation that allows the comparison of the QoS techniques. This tool is different from the others because is focused on the improvement of the QoS.

It was used Unified Modeling Language (UML) to prototype the project and create classes to allow the development of the tool, using the language C++ and Eclipse Mosquitto for C++ developers. Figure 14 presents the Class Diagram for the software that was developed to test the Eclipse Mosquitto API. The Diagram represents the mosquittopp, that is the API provided by the Eclipse Mosquitto broker. The class Mosquitto API uses the mosquittopp functions by heritance. The BrokerStatus, Publish, and Subscribe classes, are associated with the Mosquitto API class to use its functions: to get the broker status, publish and subscribe to a topic. Also, the same classes are associated with Window class, responsible to show the information. The Window class has the information to generate the graph about the time wasted to messages exchange. The Mosquitto API creates logs about any operation executed on the program. The logs are saved in three different formats, they are .txt, .csv[8], and .json[9].

[1] http://www.jensd.de/apps/mqttfx/
[2] http://kamilfb.github.io/mqtt-spy/
[3] https://itunes.apple.com/us/app/mqttinspector/id758868884?mt=8
[4] https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client
[5] http://www.hivemq.com/demos/websocket-client/
[6] https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm
[7] http://mosquitto.org/download/
[8] Comma-separated value (CSV), text format regulated by the RFC 4180
[9] JavaScript Object Notation (JSON), low format to exchange data.

Figure 14 – Class diagram of the client application



Source: Author

The developed software either presents a client interface to access the Eclipse Mosquitto API. The features available for the software are:

1. To provide the graphics interface to the user to configure the connection with the Mosquitto API. Figure 15 shows the graphic interface. This interface allows the user to set the broker address (local or remote), and the port that is used to connect.

2. To connect the Mosquitto API. It is shown in Figure 15.

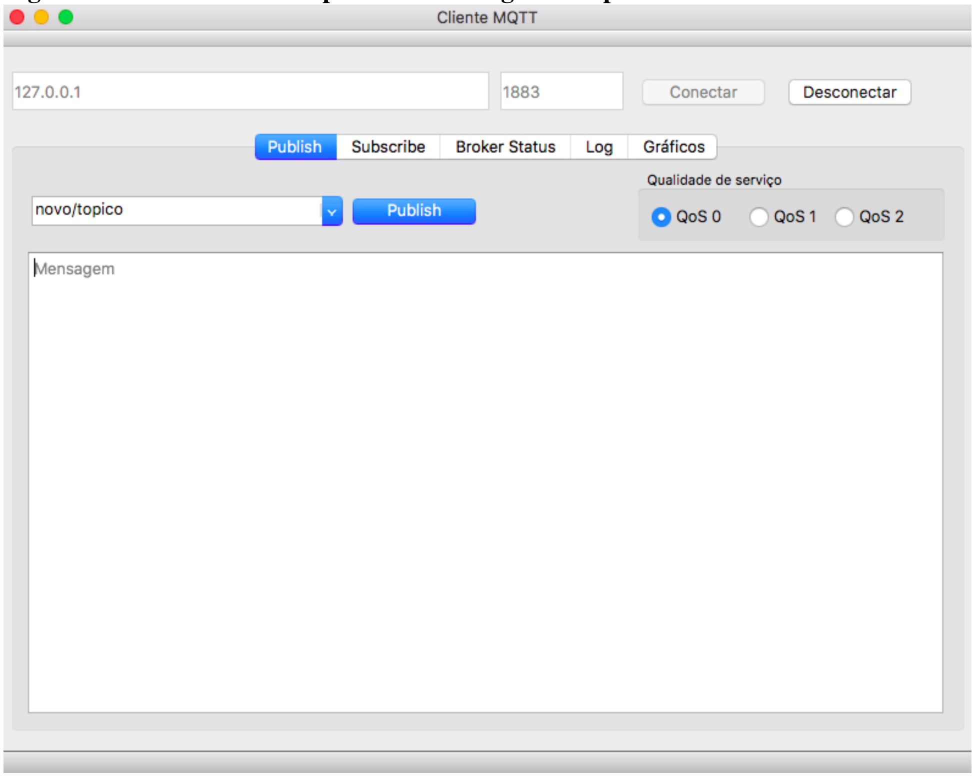
3. To allow a client to publish to send messages to a topic that is inside the broker, and select the QoS that the message needs to be exchanged, also in Figure 15.

To allow the client subscribes to a topic to receive messages from a specific topic present in the broker. Also, the user is able to visualize a list of the messages inside the broker, to access the content of the messages and information, such as date and messages content. The user can select the QoS to receive the messages. Figure 16 presents the functionality.

To access information about the broker status by the graphic interface, as shown in Figure 17. The information about the broker is obtained subscribing to predefined topics inside the broker.

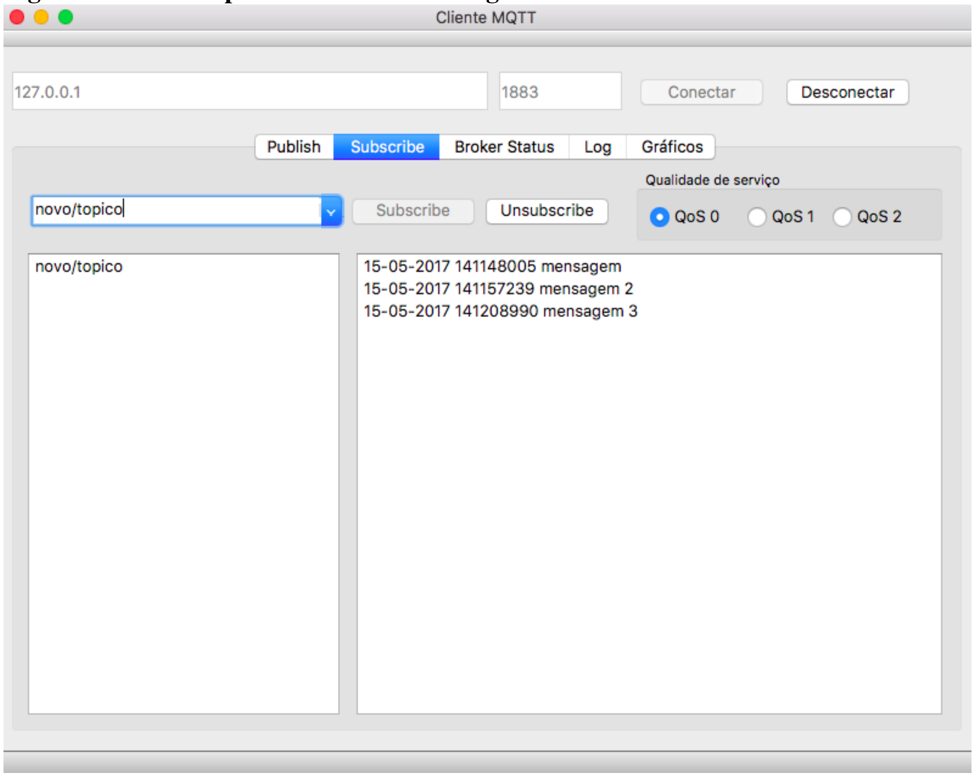
To show the time during the message exchanging and to provide a log of the of the messages exchanged. The functionality is presented in Figure 18.

Figure 15 – Initial screen to publish a message to a topic



Source: Author

Figure 16 – Subscription screen and message receiver

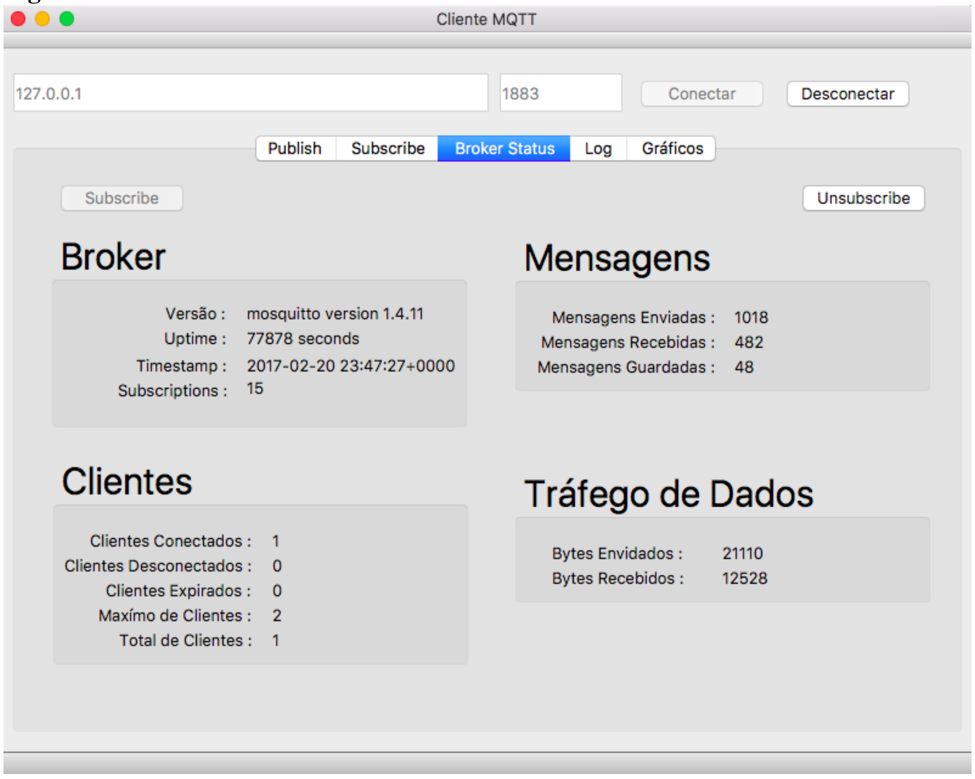


```
15-05-2017 141148005 mensagem
15-05-2017 141157239 mensagem 2
15-05-2017 141208990 mensagem 3
```

Source: Author

To save logs in three different formats; .txt, .csv, and .json. The files allow doing comparisons about the performance of exchanging messages using a chosen QoS technique. Figure

Figure 17 – Screen with information about the broker



Broker

```
      Versão :  mosquitto version 1.4.11
      Uptime :  77878 seconds
   Timestamp :  2017-02-20 23:47:27+0000
Subscriptions :  15
```

Mensagens

```
  Mensagens Enviadas :  1018
Mensagens Recebidas :  482
Mensagens Guardadas :  48
```

Clientes

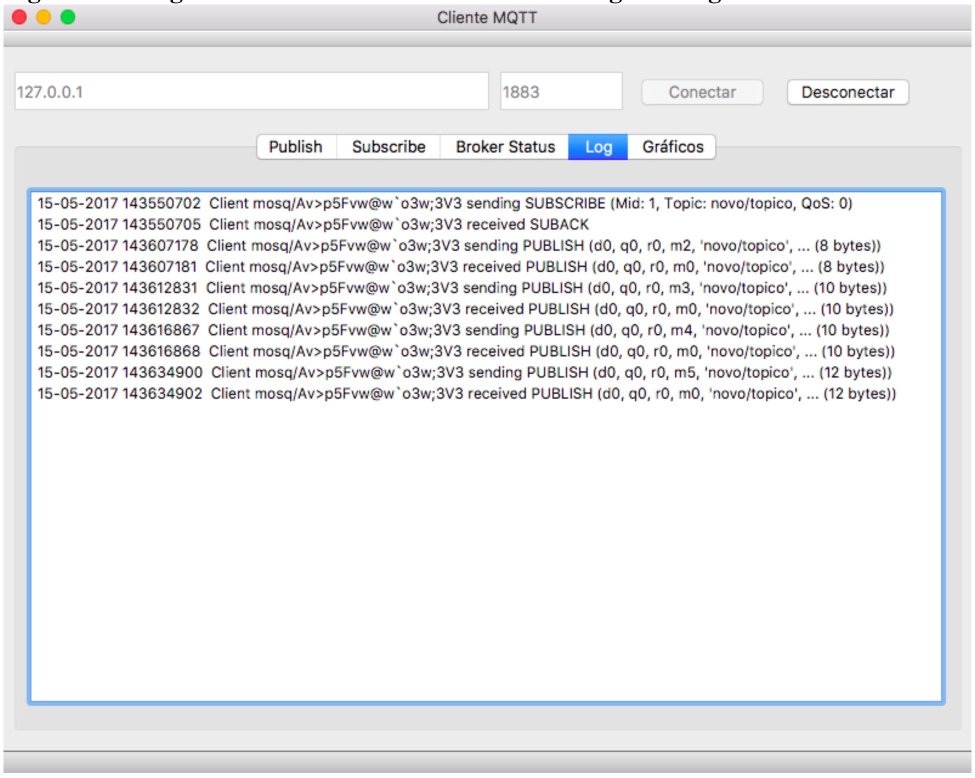
```
    Clientes Conectados :  1
Clientes Desconectados :  0
      Clientes Expirados :  0
     Maxímo de Clientes :  2
        Total de Clientes :  1
```

Tráfego de Dados

```
Bytes Envidados :  21110
Bytes Recebidos :  12528
```

Source: Author

Figure 18 – Log screen and information about exchange messages



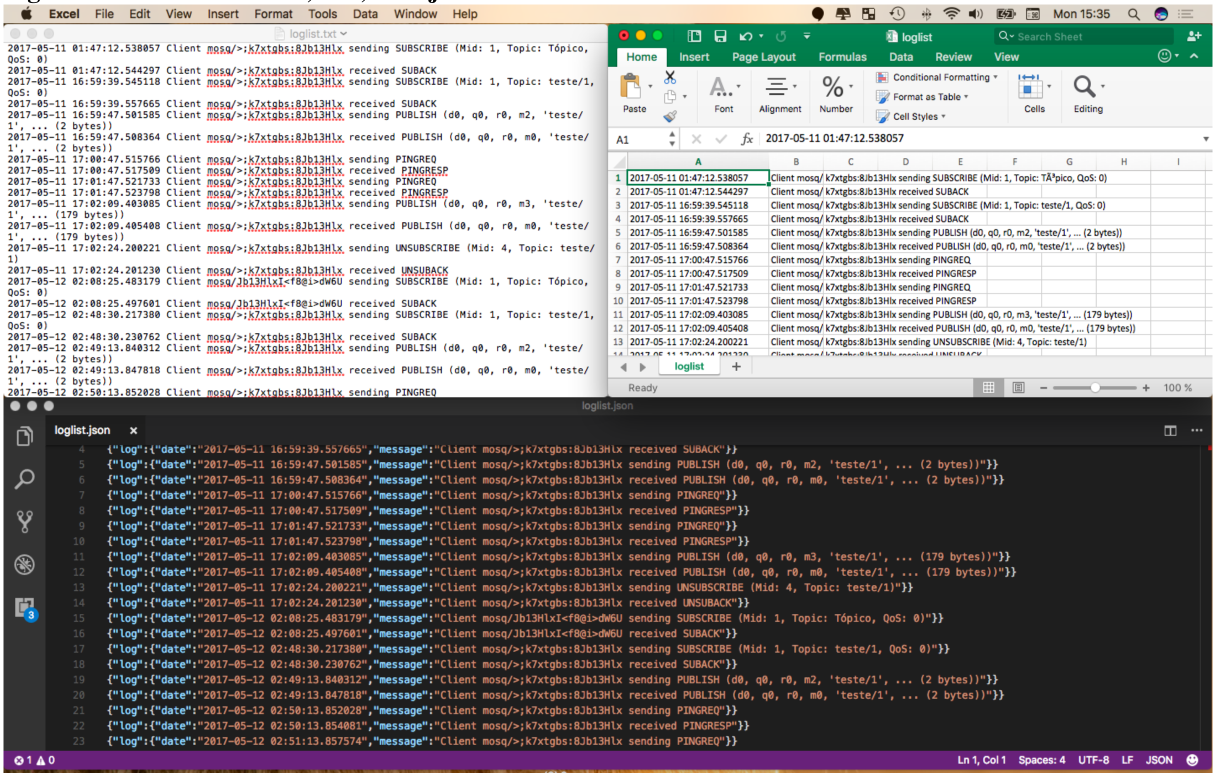
```
15-05-2017 143550702  Client mosq/Av>p5Fvw@w`o3w;3V3 sending SUBSCRIBE (Mid: 1, Topic: novo/topico, QoS: 0)
15-05-2017 143550705  Client mosq/Av>p5Fvw@w`o3w;3V3 received SUBACK
15-05-2017 143607178  Client mosq/Av>p5Fvw@w`o3w;3V3 sending PUBLISH (d0, q0, r0, m2, 'novo/topico', ... (8 bytes))
15-05-2017 143607181  Client mosq/Av>p5Fvw@w`o3w;3V3 received PUBLISH (d0, q0, r0, m0, 'novo/topico', ... (8 bytes))
15-05-2017 143612831  Client mosq/Av>p5Fvw@w`o3w;3V3 sending PUBLISH (d0, q0, r0, m3, 'novo/topico', ... (10 bytes))
15-05-2017 143612832  Client mosq/Av>p5Fvw@w`o3w;3V3 received PUBLISH (d0, q0, r0, m0, 'novo/topico', ... (10 bytes))
15-05-2017 143616867  Client mosq/Av>p5Fvw@w`o3w;3V3 sending PUBLISH (d0, q0, r0, m4, 'novo/topico', ... (10 bytes))
15-05-2017 143616868  Client mosq/Av>p5Fvw@w`o3w;3V3 received PUBLISH (d0, q0, r0, m0, 'novo/topico', ... (10 bytes))
15-05-2017 143634900  Client mosq/Av>p5Fvw@w`o3w;3V3 sending PUBLISH (d0, q0, r0, m5, 'novo/topico', ... (12 bytes))
15-05-2017 143634902  Client mosq/Av>p5Fvw@w`o3w;3V3 received PUBLISH (d0, q0, r0, m0, 'novo/topico', ... (12 bytes))
```

Source: Author

19 presents the data saved in different formats.

To show by a graphic the variation time, to exchange messages, between the broker and

Figure 19 – Files in the .txt, .csv, and .json formats



Excel File Edit View Insert Format Tools Data Window Help
Mon 15:35
loglist.txt
loglist
Search Sheet
Home Insert Page Layout Formulas Data Review View
Paste Font Alignment Number Conditional Formatting Format as Table Cell Styles Cells Editing
Ready
100 %
loglist.json
Ln 1, Col 1 Spaces: 4 UTF-8 LF JSON

```
2017-05-11 01:47:12.538057 Client mosq/>;k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: Tópico,
QoS: 0)
2017-05-11 01:47:12.544297 Client mosq/>;k7xtgbs:8Jb13Hlx received SUBACK
2017-05-11 16:59:39.545118 Client mosq/>;k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: teste/1,
QoS: 0)
2017-05-11 16:59:39.557665 Client mosq/>;k7xtgbs:8Jb13Hlx received SUBACK
2017-05-11 16:59:47.501585 Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m2, 'teste/
1', ... (2 bytes))
2017-05-11 16:59:47.508364 Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/
1', ... (2 bytes))
2017-05-11 17:00:47.515766 Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ
2017-05-11 17:00:47.517509 Client mosq/>;k7xtgbs:8Jb13Hlx received PINGRESP
2017-05-11 17:01:47.521733 Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ
2017-05-11 17:01:47.523798 Client mosq/>;k7xtgbs:8Jb13Hlx received PINGRESP
2017-05-11 17:02:09.403085 Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m3, 'teste/
1', ... (179 bytes))
2017-05-11 17:02:09.405408 Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/
1', ... (179 bytes))
2017-05-11 17:02:24.200221 Client mosq/>;k7xtgbs:8Jb13Hlx sending UNSUBSCRIBE (Mid: 4, Topic: teste/
1)
2017-05-11 17:02:24.201230 Client mosq/>;k7xtgbs:8Jb13Hlx received UNSUBACK
2017-05-12 02:08:25.483179 Client mosq/Jb13HlxI<f8@i>dW6U sending SUBSCRIBE (Mid: 1, Topic: Tópico,
QoS: 0)
2017-05-12 02:08:25.497601 Client mosq/Jb13HlxI<f8@i>dW6U received SUBACK
2017-05-12 02:48:30.217380 Client mosq/>;k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: teste/1,
QoS: 0)
2017-05-12 02:48:30.230762 Client mosq/>;k7xtgbs:8Jb13Hlx received SUBACK
2017-05-12 02:49:13.840312 Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m2, 'teste/
1', ... (2 bytes))
2017-05-12 02:49:13.847818 Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/
1', ... (2 bytes))
2017-05-12 02:50:13.852028 Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ
```

A1 2017-05-11 01:47:12.538057

	A	B	C
1	2017-05-11 01:47:12.538057	Client mosq/	k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: TÃ³pico, QoS: 0)
2	2017-05-11 01:47:12.544297	Client mosq/	k7xtgbs:8Jb13Hlx received SUBACK
3	2017-05-11 16:59:39.545118	Client mosq/	k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: teste/1, QoS: 0)
4	2017-05-11 16:59:39.557665	Client mosq/	k7xtgbs:8Jb13Hlx received SUBACK
5	2017-05-11 16:59:47.501585	Client mosq/	k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m2, 'teste/1', ... (2 bytes))
6	2017-05-11 16:59:47.508364	Client mosq/	k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/1', ... (2 bytes))
7	2017-05-11 17:00:47.515766	Client mosq/	k7xtgbs:8Jb13Hlx sending PINGREQ
8	2017-05-11 17:00:47.517509	Client mosq/	k7xtgbs:8Jb13Hlx received PINGRESP
9	2017-05-11 17:01:47.521733	Client mosq/	k7xtgbs:8Jb13Hlx sending PINGREQ
10	2017-05-11 17:01:47.523798	Client mosq/	k7xtgbs:8Jb13Hlx received PINGRESP
11	2017-05-11 17:02:09.403085	Client mosq/	k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m3, 'teste/1', ... (179 bytes))
12	2017-05-11 17:02:09.405408	Client mosq/	k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/1', ... (179 bytes))
13	2017-05-11 17:02:24.200221	Client mosq/	k7xtgbs:8Jb13Hlx sending UNSUBSCRIBE (Mid: 4, Topic: teste/1)

```
{"log":{"date":"2017-05-11 16:59:39.557665","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received SUBACK"}}
{"log":{"date":"2017-05-11 16:59:47.501585","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m2, 'teste/1', ... (2 bytes))"}}
{"log":{"date":"2017-05-11 16:59:47.508364","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/1', ... (2 bytes))"}}
{"log":{"date":"2017-05-11 17:00:47.515766","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ"}}
{"log":{"date":"2017-05-11 17:00:47.517509","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PINGRESP"}}
{"log":{"date":"2017-05-11 17:01:47.521733","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ"}}
{"log":{"date":"2017-05-11 17:01:47.523798","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PINGRESP"}}
{"log":{"date":"2017-05-11 17:02:09.403085","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m3, 'teste/1', ... (179 bytes))"}}
{"log":{"date":"2017-05-11 17:02:09.405408","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/1', ... (179 bytes))"}}
{"log":{"date":"2017-05-11 17:02:24.200221","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending UNSUBSCRIBE (Mid: 4, Topic: teste/1)"}}
{"log":{"date":"2017-05-11 17:02:24.201230","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received UNSUBACK"}}
{"log":{"date":"2017-05-12 02:08:25.483179","message":"Client mosq/Jb13HlxI<f8@i>dW6U sending SUBSCRIBE (Mid: 1, Topic: Tópico, QoS: 0)"}}
{"log":{"date":"2017-05-12 02:08:25.497601","message":"Client mosq/Jb13HlxI<f8@i>dW6U received SUBACK"}}
{"log":{"date":"2017-05-12 02:48:30.217380","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending SUBSCRIBE (Mid: 1, Topic: teste/1, QoS: 0)"}}
{"log":{"date":"2017-05-12 02:48:30.230762","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received SUBACK"}}
{"log":{"date":"2017-05-12 02:49:13.840312","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PUBLISH (d0, q0, r0, m2, 'teste/1', ... (2 bytes))"}}
{"log":{"date":"2017-05-12 02:49:13.847818","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PUBLISH (d0, q0, r0, m0, 'teste/1', ... (2 bytes))"}}
{"log":{"date":"2017-05-12 02:50:13.852028","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ"}}
{"log":{"date":"2017-05-12 02:50:13.854081","message":"Client mosq/>;k7xtgbs:8Jb13Hlx received PINGRESP"}}
{"log":{"date":"2017-05-12 02:51:13.857574","message":"Client mosq/>;k7xtgbs:8Jb13Hlx sending PINGREQ"}}
```

Source: Author

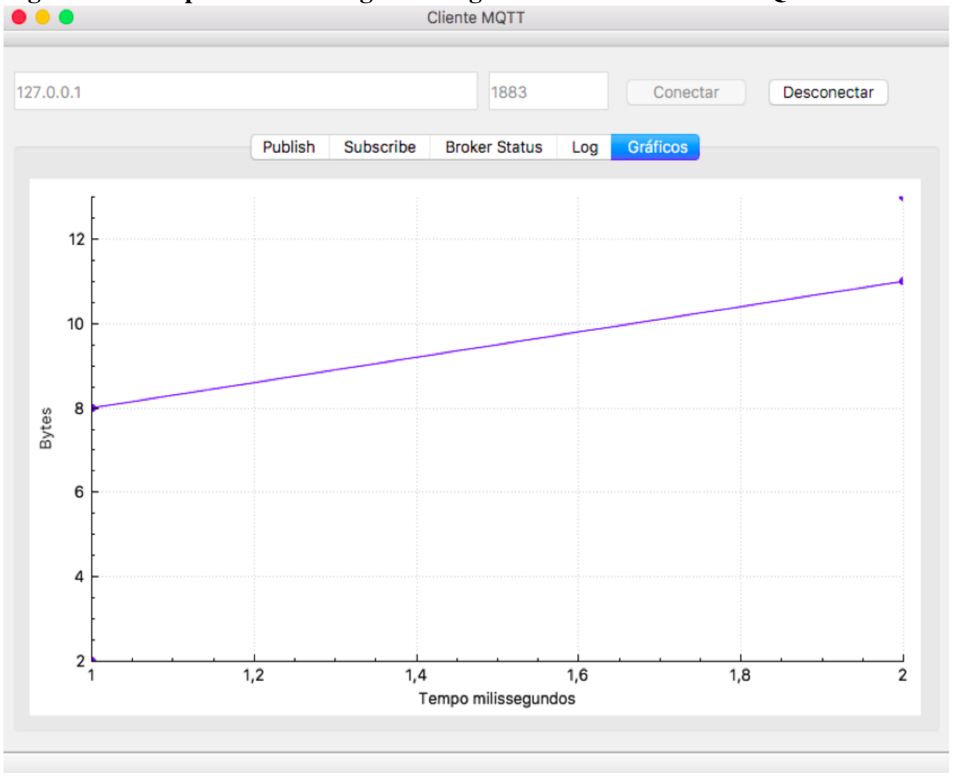
the clients, considering the number of bytes in a message. The time used for the QoS 0 is calculated when a client publisher sends a PUBLISH message and receive the broker answer. The time for QoS 1 is calculated when a client publisher sends a PUBLISH message and the broker sends a PUBACK message to info that the message arrived correctly. The time for QoS 2 is calculated when a client publisher sends PUBLISH message, the broker sends a PUBREC message to inform that the message arrived, the client sends a PUBREL message to inform that received the confirmation and the broker sends a PUBCOMP message to inform that receives a confirmation that the message arrived correctly to the client. Figure 20 shows a graph to understand the time wasted based on bytes transmitted in milliseconds.

6.3 SMARTPHONE CLIENT TOOL

There was a tool developed using WebSockets to connect an MQTT broker. The library Eclipse Paho JavaScript Client [10] is under Eclipse Foundation. The tool was developed to execute in Smartphone devices making possible to connect an MQTT broker, to send, and to receive MQTT messages. Also, the tool made the measurement of the device signal strength in dBm and the measurement of the latency. The tool was developed to study this two variables. The functionalities are presented in Figure 21.

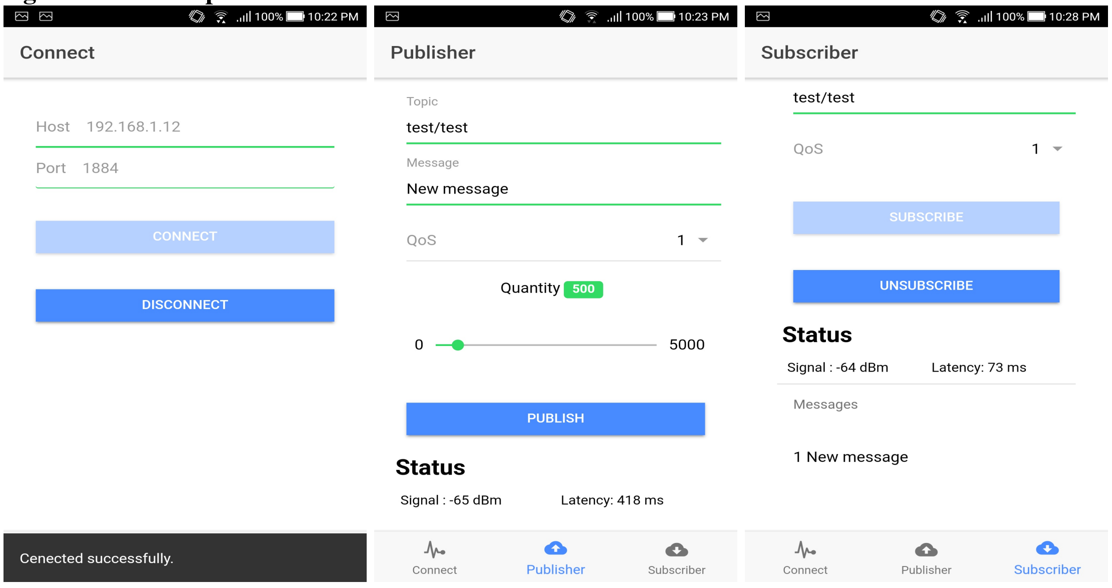
[10] https://www.eclipse.org/paho/clients/js/

Figure 20 – Graph of the message exchanged based on a chosen the QoS



Source: Author

Figure 21 – Smartphone client tool



Source: Author