Programa de Pós-Graduação em

Computação Aplicada

Mestrado/Doutorado Acadêmico

Vinicius Facco Rodrigues

HealthStack:
Providing an IoT Middleware for Malleable QoS Service
Stacking for Healthcare 4.0

São Leopoldo, 2020

Vinicius Facco Rodrigues

HEALTHSTACK:
Providing an IoT Middleware for Malleable QoS Service Stacking for Healthcare 4.0

Thesis presented as a partial requirement to obtain the Doctor's degree by the Applied Computing Graduate Program of the Unisinos University

Advisor:
Prof. Dr. Rodrigo da Rosa Righi

Co-advisor:
Prof. Dr. Cristiano André da Costa

São Leopoldo
2020

R696h Rodrigues, Vinicius Facco.

HealthStack : providing an IoT middleware for malleable QoS service stacking for Healthcare 4.0 / Vinicius Facco Rodrigues. – 2020.

145 f. : il. ; 30 cm.

Tese (doutorado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2020.

Advisor: Dr. Rodrigo da Rosa Righi.

Co-advisor: Dr. Cristiano André da Costa.

1. Internet of things. 2. Healthcare. 3. Distributed systems. 4. Quality of service. 5. Medical informatics. I. Título.

CDU: 004.738.5:614

Catalogação na Publicação (CIP):
Bibliotecário Alessandro Dietrich - CRB 10/2338

I dedicate this work to my parents.

You gave the only tool that
I need to help the world be
a better place: EDUCATION.
— VINICIUS FACCO RODRIGUES

ACKNOWLEDGEMENTS

I want to thank everyone that accompanied me on this incredible journey: especially my family, friends, colleagues, and professors. Thank you all for all the technical and mental support you gave me. Also, I want to thank some organizations that made this research possible. First, the Universidade do Vale do Rio dos Sinos (UNISINOS), in particular, the Programa de Pós Graduação em Computação Aplicada (PPGCA) for providing me this great opportunity. Second, the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), particularly the whole team of the Machine Learning and Data Analytics Lab (MadLab) for having me for six months in Erlangen, Germany. What an experience! Last but not least, the Siemens Healthineers for supporting the entire project.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (Capes) - Finance Code 001.

“*All we have to decide is what to do*
with the time that is given us.”.
(J.R.R. Tolkien)

ABSTRACT

Healthcare 4.0 is a new concept that originates from hospitals' evolution due to technological advances in medical activities. Nowadays, more and more doctors and healthcare administrators require real-time data analysis from sensors and surgery monitoring. In such settings, having real-time information may represent the difference between death or life. Currently, the analysis of data from medical settings takes place reactively. Actions to tackle problems with the patients' health are only taken when critical situations take place. With the arrival of the Internet of Things (IoT) in these environments, the data revolution can allow medical processes to generate many sorts of real-time data automatically. Specialized applications rely on centralized systems to transform medical data into precise feedback so that actuators, whether humans or not, can take the right actions. Although positive, centralized systems suffer from scalability problems. As the connected number of sensors and applications increase, the system might fail due to too many connections. Therefore, quality of service (QoS) is essential because, without it, the applications' results become unreliable. Although several approaches currently provide QoS for healthcare applications, it is still challenging to produce real-time data from sensors. More specifically, current studies present architecture models that employ different strategies to optimize the time to deliver data from sensors to the final users. Although presenting valuable contributions, they are restrained by the following limitations: (*i*) do not focus on hospital high critical environments; (*ii*) do not combine multiple strategies in different levels; (*iii*) do not put effort specifically in real-time data transmissions; and (*iv*) do not consider all relevant information for workflow analysis. Given the background, this study proposes HealthStack, a sensor middleware model for hospital settings. HealthStack collects data from sensors, stores them in a database, and delivers them to user applications meeting QoS requirements. HealthStack aims at reducing the delay and jitter in sensor data transmissions for user applications and, at the same time, reduce resource consumption. The model prototype was developed and tested in an operating room with depth cameras and an ultra-wideband (UWB) real-time location system (RTLS) for surgery workflow monitoring. This study presents scientific and technical contributions, and also contributions to society on behalf of hospital services. Its scientific contributions are twofold: a middleware model for healthcare environments with automatic QoS support for real-time data transmission; and a QoS strategy based on artificial neurons to select middleware components with poor performance. The experiments demonstrate that the strategy can improve the applications experienced jitter mean by 92.3% and delay mean by 28% for position data samples. Also, it resulted in a reduction of network, memory, and CPU consumption by up to 66.4%, 5.06%, and 48.3%, respectively. Besides the technical contributions, the solution offers a new level of reliability to time-critical applications directly impacting the patients' health. This research provides the improvement of medical services for patients, contributing to the hospital administration processes because they can access real-time data with QoS guarantees.

Keywords: Internet of things. Healthcare. Distributed systems. Quality of service. Medical Informatics.

LIST OF FIGURES

1 A multi-sensor system: applications with different requirements consume data from device sensors through a data service that may suffer scalability problems. The width of the lines connecting sensors and applications to the data service demonstrates the higher requirements. 27
2 Introduction of a QoS Manager to monitor the system and adapt QoS service stacks from the middleware. 30
3 Research development steps. Light blue boxes indicate steps that might occur more than once, representing a cycle of adjusts and implementations. . . 32
4 Example of monitored workflow steps in an operating room. Each circle represents one staff member, while the colors indicate their assignments (yellow for the anesthetist, red for the surgeon, green for the nurse). 34
5 An RFID based RTLS called LogiTrack, which provides hospitals, nursing homes, and clinics a method to accurately track, locate, and monitor assets and people, and trigger events in real time based on location and status (LOGITRACK RTLS, 2018). 36
6 Pose estimation technique employed by Kadkhodamohammadi et al. (2017). The lines represent body parts recognized by their thechnique. 39
7 Characteristics that differ hospital middlewares from traditional middlewares. Time and cost are related and critical for safety in these environments. . . . 46
8 Filtering process sequence. Filter II removes most of the articles due to their focus be different than healthcare. 47
9 Comparinson of sensor middlewares: (a) typical approach; and (b) HealthStack main idea. HealthStack comprises a Manager to monitor metrics, application requirements, and provide services for user applications and sensors. 59
10 HealthStack components connecting sensors to applications. The light blue boxes represent the contributions of HealthStack. 60
11 Deployment of the architecture in an actual hybrid operating room. Three ToF camera devices and several RTLS tags track the surgery workflow. The server processes the data and provides visualization for remote physicians and hospital administrators. 61
12 HealthStack architecture. Light blue boxes represent the HealthStack components, while white boxes represent existing software and hardware. 62
13 Packet traveling path and timestamps. 65
14 HealthStack message types and their contents. All messages use the same network header, which identifies the packets. 66
15 Communication process the components perform to produce data samples. . 68
16 Communication sequence that the Manager starts with the Core and Collector instances to monitor their measurements. 69
17 Application communication process to subscribe and receive data from a specific sensor data stream/topic. 70
18 HealthStack's closed-loop model with a Manager in charge of monitoring and adapting the middleware according to the workload. From the user perspective, a denotes the number of user applications. From the middleware perspective, n denotes the number of nodes running a Collector instance acquiring data from sensors. 72

19 QoS Manager main monitoring cycle. The idea is to monitor the components' metrics and organize QoS services according to the measurements. The Manager collects metrics from Core and Collector instances and organizes their QoS service stacks when QoS violations occur for user applications. 73
20 A look inside the QoS Manager. The component receives measurements from the middleware components and evaluates them for decision-making. The output is QoS service changes in the components' stacks, if necessary, to guarantee QoS. 74
21 Example of a monitored metric and its calculated aging. The figure highlights two specific points in which the raw value of the metric exceeds a QoS limit. In these points, the calculated aging smooths these values resulting in no violations. 78
22 A look at the whole QoS model operation. The QoS Manager performs three main monitoring tasks: (1) monitors applications' metrics; (2) monitors Core, and Collector instances; (3) computes PA for Core and Collector instances; and (4) manages service stacks from Core and Collector instances. 80
23 Matrices from an example scenario with three applications and three Collectors. Red boxes indicate QoS violations. The values $delay_{th1}$, $delay_{th2}$, and $jitter_{th3}$ are the threshold values expected by the applications. The values $delay_1$ and $delay_2$ are current delay measures for applications 1 and 2, while $jitter_1$ is the current jitter measure for application 3. 82
24 HealthStack prototype modules and their corresponding technologies. 88
25 Deployment overview of the hardware, middleware, and software components among the evaluation infrastructure. 90
26 Experiment laboratory setup in which Sewio RTLS system and Microsoft Kinects are deployed. The laboratory also contains a Siemens SIREMOBIL Compact L C-Arm. 91
27 The sequence of application connections in the eight workload scenarios. One application connects at each minute, requesting a Data type, a QoS type, and a QoS threshold. The application index refers to the sequence of the application connection to the middleware. 93
28 Workload scenarios based on the QoS threshold and the number of connected applications over time. 95
29 Delay results considering the two different sensor types: (a) depth and (b) position. Each bar is calculated with the mean delay of each application. . . 98
30 Jitter results considering the two different sensor types: (a) depth and (b) position. Each bar is calculated with the mean jitter of each application. . . . 99
31 Proportion of the number of monitoring observations in which the applications QoS was violated. At each observation, for each application, the $violation_time$ increments, and at the last observation, $violation_proportion$ divide results from dividing the $violation_time$ by the total number of observations. The bars represent this proportion of all 32 applications in each scenario. 100
32 Resource consumption of the $node_2$ for all workload scenarios. The values correspond to the mean of all samples from the monitoring observations. . . 101
33 Resource consumption of the $node_0$, in which Collector instances 0 runs. . . 102
34 Resource consumption of the $node_1$, in which Collector instances 1 runs. . . 102

35 Delay and Jitter means of all observations from Collector instances 1 and 2. 103
36 Measures of delay and jitter over time for applications 1, 2, 3, and 4 from scenarios S1 and S1'. 105
37 Resource consumption over time of Core and Collector instances from scenarios S1 and S1'. 107
38 Looking at the Service Orchestration details. Weights $w5()$ and $w6()$, Potential of Adaptation $PA()$, and number of QoS services enabled over time of Collector instances 1 and 2 from scenarios S1 and S1'. The figures show the weights' variation, and the instant QoS services are stacked for each Collector instance. The QoS service sequence occurs as follow: 1st Data Frequency Rata, 2nd Data Compression, and 3rd Data Prioritization. 108
39 The sequence of application connections in the new workload scenario. One application connects every six seconds requesting a Data type, a QoS type, and a QoS threshold. The application index refers to the sequence of the application connection to the middleware. 109
40 Delay results for the new evaluation scenario considering the two different sensor types: depth and position. Each bar is calculated with the mean delay of each application. 110
41 Jitter results for the new evaluation scenario considering the two different sensor types: depth and position. Each bar is calculated with the mean jitter of each application. 111
42 All nodes' resource consumption over time, including the number of connected applications in the middleware, and the services delivered for each node. In the last row, the y-axis represents whether a service is active (zero) or not (one). 114
43 Application ID 4 delay and jitter measurements over time with QoS Service Stacking disabled versus enabled. The goal is to visualize the impact on measurements when QoS services are enabled. 115

LIST OF TABLES

1 Technical details of the communication standards and protocols. 38
2 Technical details about the camera devices. 40
3 QoS metrics for healthcare applications. 43
4 Databases for the inclusion criteria. 46
5 Summary of the researches and their strategies addressing QoS in healthcare solutions. 48
6 Description of the network header's fields. 66
7 Description of the data attributes from sensor data messages. 67
8 QoS services short description. 75
9 Input variables the QoS Manager collects to employ the QoS Service Stacking. 77
10 Libraries and software used in the prototype development. 89
11 Equipment installed in the environment. 90
12 Technical details from the sensors installed in the simulated operating room. 92
13 Prototype parameters. 96
14 Average resource consumption of each node CPU, memory, and network. Additionally, the last column represents the variation in resource consumption when enabling the QoS Service Stacking strategy. 112
15 Variation in average delay and jitter of all applications when enabling the QoS Service Stacking. Improvements are highlighted in blue boxes. 117
16 Variation in resource consumption of all applications when enabling the QoS Service Stacking. Improvements are highlighted in blue boxes. 117

LIST OF ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
AODV	Ad hoc On-Demand Distance Vector
BLE	Bluetooth Low Energy
COVID-19	Coronavirus Disease 2019
CPU	Central Processing Unit
CV	Computer Vision
DICOM	Communications in Medicine
ECG	Electrocardiogram
ELE	Enhance Living Environment
ETSI	European Telecommunications Standards Institute
FPS	Frame Per Second
HTTP	Hypertext Transfer Protocol
I4.0	Industry 4.0
IC-FUC	Instituto de Cardiologia - Fundação Universitária de Cardiologia
ICE	Integrated Clinical Environment
ICU	Intensive Care Unit
ID	Identification
IoT	Internet of Things
IoHT	Internet of Health Things
IP	Internet Protocol
ITU-T	International Telecommunication Union's Telecommunication Standardization Sector
JSON	JavaScript Object Notation
MAC	Media Access Control
MAPE	Monitor-Analyze-Plan-Execute
MEM	Memory
MQTT	Message Queue Telemetry Transport
NET	Network
NFC	Near Field Communication
NTP	Network Time Protocol
QoS	Quality of Service
RFID	Radio Frequency Identification

RGBD	RGB-Depth
RTLS	Real-Time Location System
SES	Simple Exponential Smoothing
SDN	Software Defined Network
SL	Structured Light
SLA	Service Level Agreement
SV	Stereo Vision
TCP	Transmission Control Protocol
TDMA	Time-Division Multiple Access
ToF	Time-of-Flight
UTC	Coordinated Universal Time
UWB	Ultra-wideband
VM	Virtual Machine
WBAN	Wireless Body Area Network
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

LIST OF SYMBOLS

cm	Centimeter
GHz	Gigahertz
Hz	Hertz
Kbps	Kilobits Per Second
m	Meter
Mbps	Megabits Per Second
MHz	Megahertz
ms	Milliseconds
s	Seconds

CONTENTS

1 INTRODUCTION **23**
1.1 Motivation 24
1.2 Research Problem 26
1.3 Goals 29
1.4 Research Plan 31
1.5 Document Organization 31

2 BACKGROUND **33**
2.1 Workflow Monitoring 33
2.2 Sensing Technologies 35
2.2.1 Real-Time Location Systems 35
2.2.2 Computer Vision Techniques 38
2.3 Quality of Service 41
2.4 Summary 44

3 RELATED WORK **45**
3.1 Search Methodology 45
3.2 State-of-the-Art 47
3.2.1 Wireless Body Area Network 50
3.2.2 Telemedicine 52
3.2.3 Internet of Things 53
3.2.4 Other Approaches 53
3.3 Discussion and Research Opportunities 54
3.4 Summary 55

4 THE HEALTHSTACK MODEL **57**
4.1 Design Decisions 57
4.2 Architecture 60
4.2.1 Application Classes 63
4.2.2 Time Synchronization 64
4.3 Communication Protocol 65
4.3.1 Middelware Communication Process 65
4.3.2 Application Communication Process 68
4.4 Quality of Service Model 70
4.4.1 Managing QoS 71
4.4.2 QoS Services 73
4.4.3 Definition of Input Variables 76
4.4.4 Dynamic QoS Service Stacking 78
4.5 Summary 84

5 EVALUATION METHODOLOGY **87**
5.1 Prototype Implementation 87
5.2 Infrastructure Setup 88
5.3 Workload Model and Evaluation Scenarios 90
5.4 Parameters 94
5.5 Summary 96

6 RESULTS **97**
6.1 Applications' Delay and Jitter 97
6.2 Resource Consumption 100
6.3 Applications' Delay and Jitter Time Series 104
6.4 Enhancing the Experiments 108
6.4.1 Applications' Average Delay and Jitter 110
6.4.2 Resource Consumption 112
6.4.3 Applications' Measurements Time Series 115
6.5 Discussion 116
6.5.1 Main Results 116
6.5.2 Limitations 118
6.6 Summary 119

7 CONCLUSION **121**
7.1 Lessons Learned 122
7.1.1 Lesson 1: Build a Strong Partnership 122
7.1.2 Lesson 2: Consider the Complexity of the Environment in the System Design . . 122
7.1.3 Lesson 3: Tailor the Technology to the Target Environment 123
7.1.4 Lesson 4: Critically Analyze the Results to Find Solutions 124
7.2 Main Contributions 124
7.3 Publications 125
7.4 Limitations and Future Work 132

REFERENCES **135**

1 INTRODUCTION

The future of the Internet of Things (IoT) has arrived. In the last few years, many authors argued that the IoT paradigm would reach more than 50 billion connected devices by 2020 (FEKI et al., 2013). Well, we finally reached 2020, and what does that mean for the healthcare scenario? The vast number of connected devices, producing many kinds of information, was one of the foundations during the rise of Industry 4.0 (I4.0) (OZTEMEL; GURSEV, 2020). More specifically, one of the pillars of I4.0 is the combination of IoT and Artificial Intelligence (AI) technologies (OZTEMEL; GURSEV, 2020). While IoT brings the sensors and the data generation, AI applications process data in real-time to provide insights, recognize actions, and even predict outcomes (BAIG; HOSSEINI; LINDÉN, 2016). Besides, the arrival of IoT to hospital wards tailored the new Internet of Health Things (IoHT) concept, which regards the many sensors generating information from patients (COSTA et al., 2018). Following I4.0, currently it is already common to read terms as “Healthcare 4.0”, “Health 4.0”, “Hospital 4.0”, and “Surgery 4.0” to refer to the revolution of hospitals due to the arrival of IoT technologies in healthcare environments (ACETO; PERSICO; PESCAPé, 2020; THUEMMLER; BAI, 2018; FEUSSNER et al., 2017; AFFERNI; MERONE; SODA, 2018). In this new paradigm, the main focus relies on the personalization of clinical services to increase patients’ quality of care (ACETO; PERSICO; PESCAPé, 2020).

In Healthcare 4.0, sensor devices spread all over the hospital settings can provide data to support medical decisions. Wearable sensors attached to patients, medical staff, and even equipment collect important information, such as physiological (e.g., pulse rate, heart rate, etc.) and physical (e.g., movements, position, etc.) data. Many sensors can provide information from single individuals, equipment status, or even entire processes. These data can provide information to support decision-making processes and medical data analytics (ACETO; PERSICO; PESCAPé, 2020; GIATRAKOS et al., 2019). Combined, such a variety of information can produce more valuable insights. New information and monitoring systems capture, store, and process data in real-time for medical decisions. The main goal is to support the medical team in the decision-making process locally or remotely. More importantly, tracking individuals, equipment, and environment data allows information systems to give the medical team feedback to avoid medical errors. For instance, real-time solutions can monitor procedure workflows, detect critical scenarios, and avoid delays to tackle them (MOGHIMI; WICKRAMASINGHE; ADYA, 2020; ALBAHRI et al., 2019). In such scenarios, data analytics can indicate flaws in the surgery processes that can be corrected. Even more, real-time analysis can indicate critical situations that may lead to medical errors. As another example, in medication processes, the medical team must administrate the right doses to the right patients at the right times. Therefore, tracking medications and patients, including dose timing, can avoid human errors that may occur (MAGALHÃES et al., 2015).

Given the background, response time for critical situations is decisive to save lives in health-

care. With several sensors distributed in several hospital settings, it emerges the need for a system that can gather all information and make it available to specialized applications. Given the importance of the sensor information, providing the right data to the right person at the right time is vital. More importantly, faster actions to treat a health crisis increase the probability of better outcomes. In this context, users require the system to respect levels of performance. Such a performance level is commonly called Quality of Service (QoS), and it directly impacts the user experience (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014; OODAN et al., 2003).

In the era of Healthcare 4.0, hospitals contain several sensors and equipment that take distinct processing and network requirements to generate heterogeneous datasets. Such heterogeneity increases the complexity of applications with individual QoS requirements to make sense of the incoming data. The variety of sensors and applications in healthcare requires real-time systems to be aware of data's heterogeneity and relevance to choosing the best processing and transmission strategies. That means that the solution must adapt its strategies depending on the data and application requirements (delay, jitter, etc.). For instance, medical workflow data analysis depends on the arrival of data with acceptable delay to predict situations before they happen (ZHOU et al., 2020). In mission-critical telerobotic applications, fluctuating delay and jitter can lead to instabilities and failures (SZYMANSKI; GILBERT, 2010). Furthermore, not only humans can profit from real-time feedback in healthcare, but also autonomous agents using robotics and machine learning (AHMED; LE MOULLEC, 2017; CHEATLE et al., 2019). With quick access to information, proper actions can be taken to reduce risks for the patients' lives.

1.1 Motivation

Currently, the analysis of data from medical settings takes place reactively (COSTA et al., 2018). Actions to tackle problems with the patients' health are only taken when critical situations take place. A physician decides the actions to take based solely on traditional examination data from the patient. For instance, in the intensive care unit (ICU), patients are placed in beds side by side in individual blocks. Each block contains special equipment that monitors a specific patient. Next to the patient's bed, a display system outputs all sensors' measurements that monitor the patient. This display also provides configurable parameters to physicians set thresholds to the patient's measurements. Regularly, a physician passes by to check for abnormalities. He takes note of the patient's file and only comes back in the next checkup interval. In case of an emergency (e.g., a cardiac arrest), the patient's monitor sounds an alarm, and the medical staff has to rush to control the situation. This situation exemplifies the reactiveness of medical processes, in which action is taken after an emergency occurs. Further, this process is individual for each patient. It does not contain a centralized analysis of everything that happens in the medical environment. Ideally, decision-making should employ a global analysis of the medical processes, generated in real-time using data from all patients and the medical environment. In

this context, patients' health parameters remote monitoring is becoming crucial (PONCETTE et al., 2020). That would allow a proactive analysis of data to predict critical situations before they occur. Furthermore, this view enables the acquisition of further knowledge, for example, the spread of disease within the hospital setting.

In such scenarios, emerging Healthcare 4.0 brings a series of opportunities. The numerous sensor devices scattered all over the hospital, generating information from various assets, including the patients and the medical team, play an essential role in the hospital revolution. On top of such infrastructures, it is feasible to employ AI technologies (SONG et al., 2020; PONCETTE et al., 2020). Currently, AI applications emerge to handle this increasing data volume. Those applications generate valuable information and perform predictions that might save lives, bringing proactiveness to the processes. Specialized applications transform medical data into precise feedback so that actuators, whether humans or not, can take the right actions. Among many strategies, three stand out: (*i*) Data Prediction (NYCE; CPCU, 2007); (*ii*) Pattern Recognition (TVETER, 1997); and (*iii*) Data Correlation (WILLIAMS et al., 2018). Data prediction enables forecasting of measurements and situations, anticipating problems, and the required countermeasures. In turn, Pattern Recognition offers strategies to identify situations that already occurred in the past. Hence, one only needs to verify what was done when such situations occurred and whether the action was valid. Lastly, Data Correlation combines information from multiple parameters to identify the source of specific situations. For instance, data correlation strategies are currently employed for ICU patients with Coronavirus Disease 2019 (COVID-19) to identify the relation of many indicators (LIU et al., 2020). The outbreak of new diseases brings the public health systems the necessity to discover new treatments by analyzing the patient's response to current methods.

It is clear that AI brings many advantages to clinical analysis; however, it depends on various sensor data sources to work correctly. Collecting data from physically distributed heterogeneous sources is challenging, and it imposes complexity on application design. To close this gap, centralized systems are common strategies present in many sectors. They provide a global data analysis by collecting distributed data and providing them to applications. Although positive, centralized systems suffer from scalability problems. Scalability is the system's ability to maintain its performance and QoS regardless of the input workload and internal processing (BONDI, 2000). Centralized systems concentrate data processing at a single point. As the connected number of sensors and applications increases, the system might fail due to too many connections. That would cause instabilities in the flow of data to different systems, which can be critical, depending on the application. Therefore, a sensor data middleware for medical settings must provide QoS to the applications. Data from sensors should arrive at the corresponding destination in time and steadily to all of that correctly work. Imagine an application that processes data in real-time to trigger an alarm in the operating room (a visible red light, for instance). The application generates a warning if it predicts that the patient's heart rate will exceed a certain threshold of five seconds in the future. If the sensor's data delays ten seconds

to arrive or even does not arrive, the application will not work.

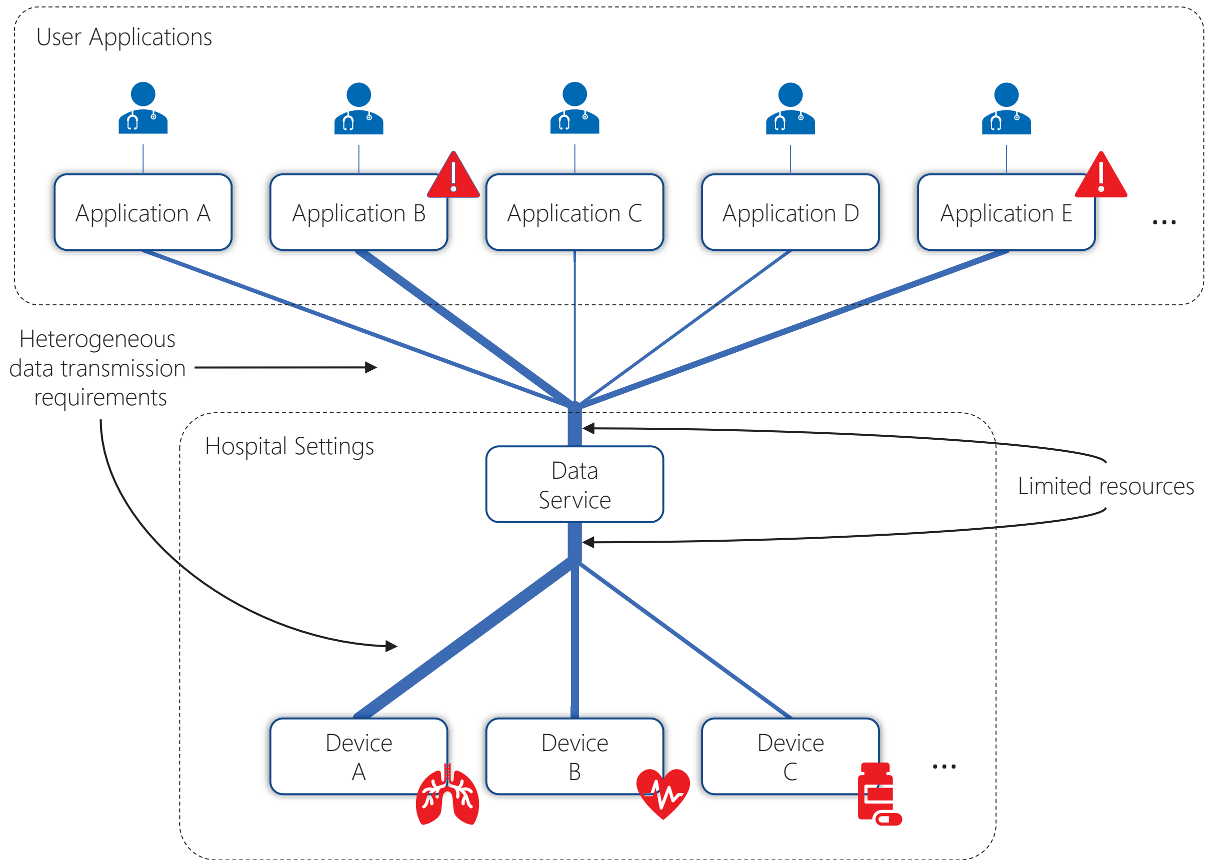
As another example, some applications also might depend on a predictable and stable time interval between the arrival of data packets. An application that requests to receive ten data samples per second in a timely way expects to receive one sample every 100 milliseconds. Variations in the time arrivals, called jitter, can produce an undesirable effect in the user application. A typical example regards video stream applications. Image sensors generate image frames that applications project on screens one after another to see what is going on in the procedures. Suppose the time between each frame deviates too much from the desired interval. In that case, the user will perceive a loss of quality since the video will not be synchronized appropriately (STEINMETZ, 1996). This effect would be easily perceived by the medical staff that examines the videos, impacting the user experience and the analysis's quality.

There are two main reasons why QoS is essential for medical applications. First, most applications handle multimedia data that follows a video Codec standard, such as Digital Imaging and Communications in Medicine (DICOM) (MUSTRA; DELAC; GRGIC, 2008). Video streams require a cadence of data arrival to make sense. Therefore, high jitter can impact user experience. Second, some applications deal with sensitive health data. Such applications require that the system does not fail to deliver these data. Otherwise, the application will not identify critical situations, which may lead to severe consequences. Issues of this kind can represent the difference between the life and death of a patient. Therefore, it is essential to build a middleware between sensors and applications that efficiently manages the communication process. The middleware should provide QoS guarantees that the correct data arrives at the right destination at the right time. Hence, it should take actions that focus on maintaining performance without imposing penalties. It is not desirable to decrease delay but, at the same time, cause an increase and jitter (and vice-versa).

1.2 Research Problem

Figure 1 depicts the main research problem the current study focuses on. The number of user applications and sensors connected to the system can vary dynamically over time. As the data service has limited resources, overload situations may occur, reducing the system's performance. Such situations can directly impact the applications' perceived performance, which can be invalid depending on the degree of such performance loss. The figure demonstrates a scenario in which the data service collects data from three different devices with different data transmission requirements. At the users' level, five applications collect data at different rates and from different sensors, which leads to different data transmission requirements. The width of the lines connecting sensors and applications to the data service demonstrates the higher requirements. Besides, the lines connected to the data service demonstrate its limited resources that all connections share. In the example, applications B and E have higher requirements and are affected by the system's performance loss.

Figure 1: A multi-sensor system: applications with different requirements consume data from device sensors through a data service that may suffer scalability problems. The width of the lines connecting sensors and applications to the data service demonstrates the higher requirements.



Source: elaborated by the author.

Although there are several approaches to provide QoS for healthcare applications, it is challenging to produce and deliver real-time data from sensors (SISINNI et al., 2018). Such systems consist of complex distributed infrastructures that may suffer from many problems, such as transmission interference and energy constraints. On one side, we have an increasing number of sensors producing data. On the other side, we have an increasing number of applications requesting data from different sensors with different QoS requirements. That leads to scalability problems and may affect the QoS experienced by applications and the infrastructure's resource consumption. It is not trivial to deploy monitoring systems that inherit the present infrastructure in the current state of hospital settings. Therefore, achieving real-time data transmission requires new approaches to adapt to these new environments. The current literature presents many studies that employ QoS strategies for healthcare architectures. More specifically, studies present architecture models that employ different strategies to optimize the time to deliver data from sensors to the final users.

In summary, these models present methods in the network level including **scheduling protocols** (LIU; YAN; CHEN, 2017; SAMANTA; MISRA, 2018; SAMANTA; LI; CHEN, 2018; IRANMANESH; RIZI, 2018; VENKATESH et al., 2019), **data prioritization** (BANOUAR

et al., 2017; POORANI et al., 2017; LIU; LIU; CHEN, 2017; SAMANTA; LI; CHEN, 2018; GUEZGUEZ; REKHIS; BOUDRIGA, 2018; IRANMANESH; RIZI, 2018; WANG et al., 2019; AL-TARAWNEH, 2019; VENKATESH et al., 2019; KHALIL; MBAREK; TOGNI, 2019; GOYAL et al., 2020; IBRAHIM et al., 2020), and **routing protocols** (AHMED; LE MOULLEC, 2017; GUEZGUEZ; REKHIS; BOUDRIGA, 2018; ZITTA et al., 2018; IRANMANESH; RIZI, 2018; TSENG; WANG; YANG, 2020; ZUHRA et al., 2020; WAHEED et al., 2020; VADIVEL; RAMKUMAR, 2020; IBRAHIM et al., 2020). Additionally, some solutions have a different approach employing resource management schemes to improve the system performance as a whole, including the **energy efficiency** (HASSAN; ALRUBAIAN; ALAMRI, 2016; AGIRRE et al., 2016; BANOUAR et al., 2017; POORANI et al., 2017; LEE; JUNG; LEE, 2017; MAATOUGUI; BOUANAKA; ZEGHIB, 2017; CELDRÁN et al., 2018; WANG; SUN; JI, 2018; GOYAL et al., 2020). Although these studies present valuable contributions, they are restrained by the following limitations:

(*i*) Strategies do not focus on hospital high critical environments;

(*ii*) Solutions do not combine multiple strategies in different levels;

(*iii*) Initiatives do not put effort specifically in real-time data transmissions;

(*iv*) Studies do not consider all relevant information for workflow analysis;

(*v*) Studies do not focus on the scalability problem of the system they propose.

Maintain a desirable QoS for real-time applications that consume data from the healthcare workflow is challenging since the number of sensors and user applications might dynamically change. The growth of the number of sensors and devices increases the complexity of real-time information monitoring. The more the number of nodes, the higher the amount of information that the middleware must handle. Besides handling many information data sources present in the environment, the middleware must also manage user application requests, demanding real-time data. Therefore, the middleware should identify and handle such situations to keep a certain level of QoS, not impacting the user experience. At the same time, it is also essential to take into consideration computational resource consumption. It is not desirable to tackle one problem by increasing resource consumption considerably and, therefore, the total cost. Thus, taking all of this into consideration, the following statement defines the current thesis hypothesis to guide this research:

*An **adapting-driven** middleware for sensor-based healthcare environments, with **dynamic sensors and applications** connected to the system, can **efficiently** provide an **acceptable quality of service** for end-user applications.*

First, **adapting-driven** refers to policies the middleware employs to change the system behavior to improve or maintain performance. These strategies range from parameter adjustments

to replications of middleware components to maintain scalability. Second, **dynamic sensors and application** refers to the number of applications and sensors connected to the system, which can change over time. The amount of data and requests that the middleware manages depends on how many sensors and applications are connected. Both sensor data and user application requests change over time, increasing or decreasing its load on the system. Third, **efficiently** refers to resource consumption and cost characteristics. Employing more computational resources to achieve its goal is not ideal since it consequently increases the total cost. Finally, **acceptable quality of service** refers to the system's final performance to the user application. An acceptable performance means that the average performance of the target QoS metric for the applications is equal or lower than a threshold the application expects. Based on the hypothesis, the following research question arises:

Which self-adaptative strategies a healthcare middleware, with dynamic sensor and application connections, needs to efficiently provide timed data for applications?

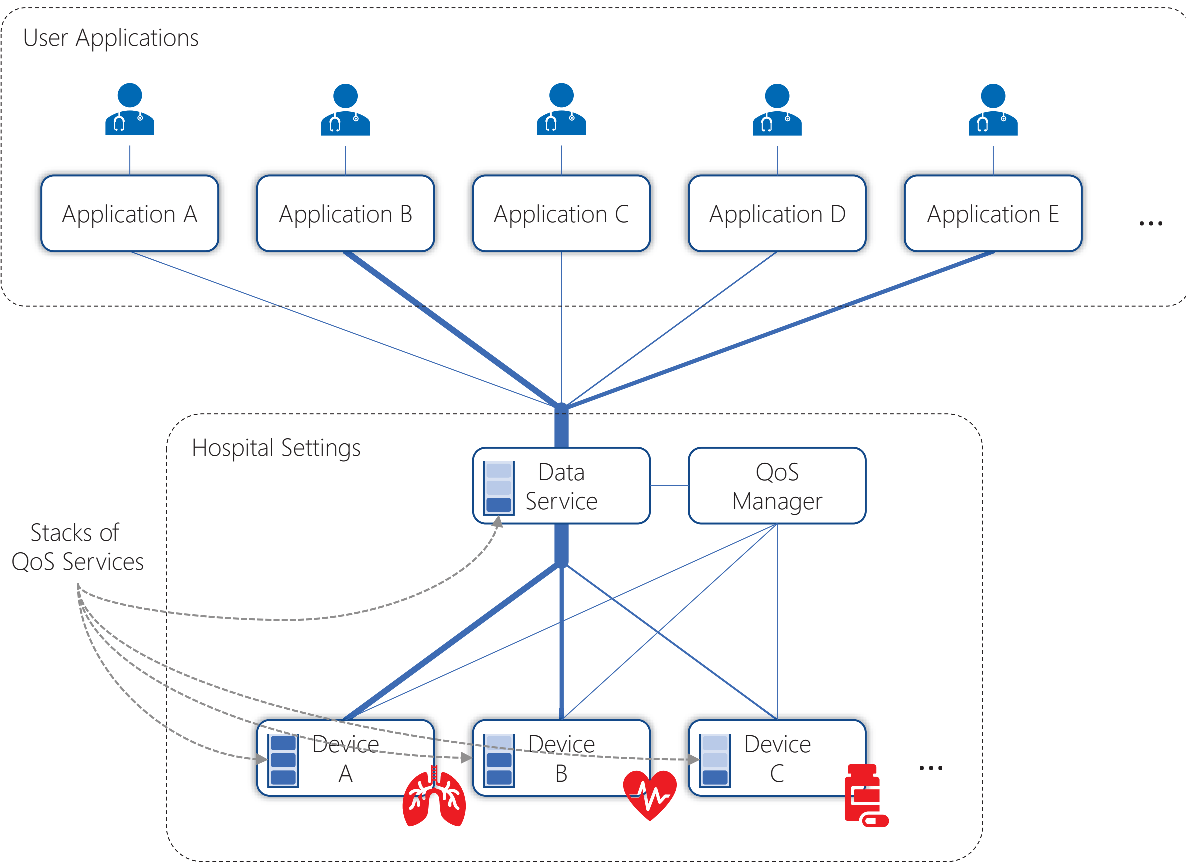
The above question is the central question of the current research that, jointly with the hypothesis, guide this study. The next section presents the research goals defined to answer the research question and test the research hypothesis. More specifically, it describes this study's proposal to addresses the hypothesis and the research question.

1.3 Goals

Based on the hypothesis and the research question above, this study proposes the HealthStack middleware model. Figure 2 illustrates how this research study intends to solve the research problem. In summary, HealthStack introduces two main ideas: (*i*) QoS Manager; and (*ii*) QoS service stacks. HealthStack implements different QoS strategies called QoS services that the QoS Manager can stack for each middleware component. The QoS Manager monitors the middleware performance and organizes the QoS service stacks accordingly. Its main goal is to enable QoS services to improve resource utilization and reserve more resources for applications with higher requirements without impacting the others. The QoS services comprise different QoS strategies, such as data prioritization, resource elasticity, data compression, and data frequency rate.

HealthStack is a sensor-based middleware for healthcare environments that provides QoS strategies to improve its performance and resource consumption. To achieve that, HealthStack introduces the QoS Service Stacking, a self-adaptive strategy to handle overload situations that may impact the middleware's performance. This strategy aims at reorganizing QoS services for sensors on-the-fly according to workload changes. HealthStack proposes separate components to extract data from individual sensor data sources. Through these individual components, HealthStack can stack QoS services applied to the sensor data to improve performance and resource consumption. The self-adaptative strategies are agnostic to the applications' point of

Figure 2: Introduction of a QoS Manager to monitor the system and adapt QoS service stacks from the middleware.



Source: elaborated by the author.

view. HealthStack performs reconfigurations to accommodate all applications in a transparent manner. Its leading role is to access data from several sensors and provide these data to applications with QoS guarantees, not requiring them any adaptions to profit from the QoS strategy. Accordingly, the following statement defines the principal goal of this research:

Develop an adaptable middleware model for healthcare employing different QoS strategies to provide timed data for applications, also improving hardware utilization, regardless of the sensors and applications load on the system.

To complete the principal goal, the research must fulfill several secondary objectives. They support the development of the model and the applications necessary for its validation. The following objectives define how this study is going to achieve the main goal:

(i) Perform literature research regarding middlewares for healthcare to find the existing lacks in the area and which QoS strategies they employ;

(ii) Develop a distributed system that supports multiple QoS strategies to collect data from sensors, accept applications connections, and send real-time data to them;

(*iii*) Determine the user application model with types of applications to define standard QoS requirements in case the user does not specify them;

(*iv*) Define a time synchronization strategy between the middleware components and the user clocks to be possible to calculate the approximate delay to deliver data at the application side;

(*v*) Specify the communication protocols employed in the middleware to allow future research to extend the middleware capacity;

(*vi*) Define an artificial neuron-based algorithm to evaluate all metrics from the middleware and applications, and select the QoS services to improve performance for applications;

(*vii*) Provide a lower average delay and jitter for applications, and at the same time reduce resource consumption.

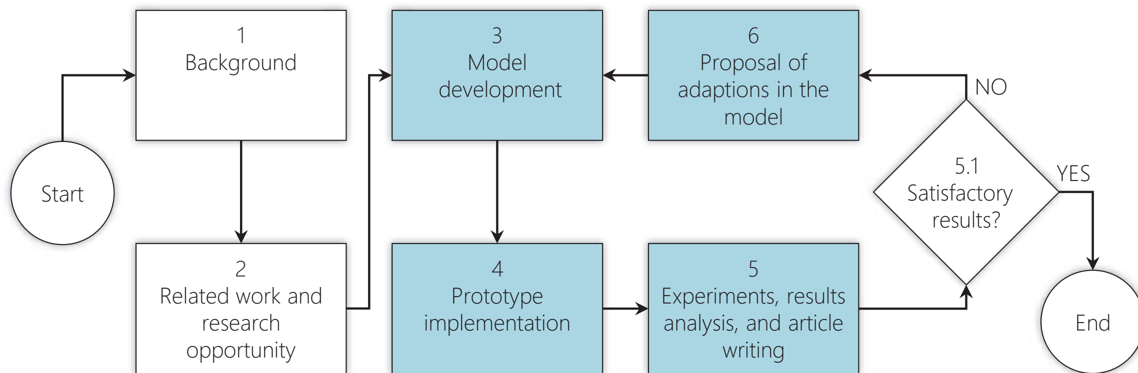
1.4 Research Plan

This study follows a research plan composed of six steps to achieve its goals. Four steps integrate an adaption and implementation cycle, as shown in Figure 3. First, step 1 represents the study of fundamental concepts related to the research topic to form this document's background. Then, step 2 consists of literature research in the context of the topic research. Following, steps 3, 4, 5, and 6 establish a closed-loop to perform adaptations and implementations of the proposed model. Step 3 is the development of the model focusing on the goals presented in the previous section. In turn, step 4 encompasses the implementation of the prototype following the model definitions. Subsequently, step 5 consists of an experimental evaluation of the prototype, analysis of results, and article production based on these analyzes. Depending on these results, in step 6, it is possible to propose adaptions in the model and start the cycle again in step 3. Otherwise, when the results and production of articles are satisfactory, the research ends. The current research development fell one time in the "NO" answer for question 5.1. Therefore, the current steps compose the current study: 1, 2, 3, 4, 5, 6, 3, 4, and 5.

1.5 Document Organization

This document is structured in seven chapters, and they present the outcomes of the research steps depicted in Figure 3. Chapter 2 presents the fundamental concepts required to understand the subjects that this study approaches, resulting from step 1. Next, Chapter 3 introduces the literature research presenting the state-of-the-art middlewares for healthcare, which is obtained through step 2. In turn, Chapter 4 presents the core of this document, which is the HealthStack model defined in step 3. The chapter introduces the design decisions, the architecture of the model, and the QoS model. Following, Chapter 5 proposes the evaluation methodology to

Figure 3: Research development steps. Light blue boxes indicate steps that might occur more than once, representing a cycle of adjusts and implementations.



Source: elaborated by the author.

validate the model proposal, including the prototype implementation from step 4. Chapter 6 presents the results of the experiments according to step 5. The analysis of results leads to step 6; therefore, Chapter 6 also presents some adaptations in the methodology and results from complementary experiments. Finally, Chapter 7 presents the final remarks of this study, presenting the contributions, limitations, future work, and the publications achieved.

2 BACKGROUND

This chapter introduces the essential terminologies and concepts that this document uses in the coming chapters. It shows the importance of workflow monitoring in medical settings and the definition of QoS and its importance. Section 2.1 describes the workflow monitoring process, which is essential for tracking medical processes. Following, Section 2.2 presents technologies employed for tracking and monitoring in healthcare scenarios. The section presents the most employed technologies in the current literature. Finally, QoS is an important concept to understand the current study. Therefore, Section 2.3 introduces the QoS aspects in the context of telecommunication technologies. Then, Section 2.4 summarizes the ideas present in the chapter.

2.1 Workflow Monitoring

A workflow is a set of activity steps with a partial ordering aiming to achieve a particular objective (ELLIS, 1999). Specifics of such a definition can vary depending on the application scenario. In the particular case of hospital environments, workflows are quite dynamic (ROJAS et al., 2016). They must be continuously updated due to new regulations, and the introduction of new treatment methods, medications, and technologies (RUTLE et al., 2013). Niazkhani et al. (2009) present a structure to describe the workflow of healthcare activities. Their model encompasses six main questions regarding an activity: **who, what, when, where, how,** and **which resources.** The present document focuses on four of the aforementioned questions: **who, where, how,** and **when**. The first three questions can be answered by three respective aspects related to actors in a hospital: (*i*) their **identity**; (*ii*) their **location**; and (*iii*) their **state**. In turn, the last question can be answered with the inclusion of timing information when observing these aspects. Thus, in this study, the triple **identity, location, state** defines an **activity** of an arbitrary actor and a **sequence of activities ordered by time** from one or more actors defines a **workflow**.

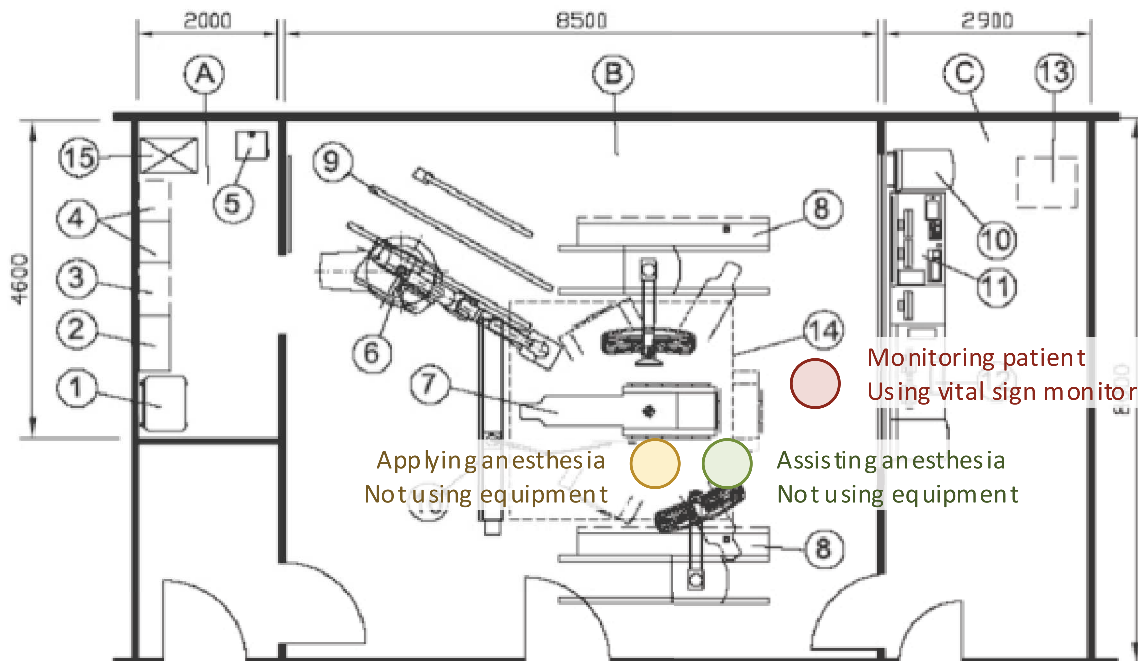
The two remaining questions, **what** and **which**, do not relate to the scope of monitoring workflow or provide information already obtained with the other four. The **what** question defines which specific task is being executed by an actor. When considering monitoring, it aligns with the final steps of the process. That is, the analysis of identity, location, state, and activity data should enable the characterization of all tasks being undertaken in the environment. In turn, **which** relates to the resources (i.e., the medical equipment) used by each actor to conduct activities.

The optimization of medical workflows is a fundamental process to improve clinical services (HAZLEHURST et al., 2003). In turn, an essential element in this process is the **monitoring** of activities to document clinical procedures (NIAZKHANI et al., 2009). Traditional methods involve manual document analysis and staff testimony (MALHOTRA et al., 2007). As a result,

it can generate imprecise results, leading to suboptimal management decisions (RUTLE et al., 2013). In turn, automated monitoring is non-intrusive and achieves reliable results if correctly planned and implemented (VANKIPURAM et al., 2011). Consequently, it offers a sound basis for decision-making on workflow changes.

An example of a healthcare workflow is a surgical procedure that takes place in an operating room. Such a workflow would vary greatly depending on factors like the type of procedure and patient condition. For the sake of brevity, a "generic" procedure comprised of two steps is assumed: (*i*) anesthesia; (*ii*) operation. A workflow monitoring system aims to identify these steps, the actors that participate in them, and their sequence of actions. Figure 4 illustrates an example of these steps with some of the information expected to be acquired by the system. As shown, some of the data expected to be acquired without manual intervention are the staff's location in the room, their current activities, and interactions with equipment.

Figure 4: Example of monitored workflow steps in an operating room. Each circle represents one staff member, while the colors indicate their assignments (yellow for the anesthetist, red for the surgeon, green for the nurse).



Source: adapted from Nollert and Wich (2009).

An architecture to monitor healthcare workflows is bound to result in a dataset of considerable size. The acquisition of actual knowledge about the workflows requires the analysis of collected data. Depending on the monitoring system scale, such analysis may present challenges similar to those found in Big Data (JAGADISH et al., 2014). In fact, the vast amount of data generated by current healthcare systems already presents a Big Data scale (FANG et al., 2016). The application of Big Data concepts to such systems can potentially improve the quality of care of patients. Additionally, the combination of Big Data and Smart System concepts

can lead to new solutions for various aspects of healthcare (PRAMANIK et al., 2017).

Fang et al. (2016) summarize the main challenges of Big Data on healthcare. These include: (*i*) high heterogeneity of structured and unstructured data formats used to store medical data; (*ii*) lack of standardization for datasets; (*iii*) difficulty to process such datasets, especially regarding storage and computational power; and (*iv*) security and privacy considerations when dealing with sensitive patients' data. Such challenges will arise to any system tailored to analyze a large healthcare dataset, including monitoring workflow in such spaces.

2.2 Sensing Technologies

Sensing devices are ubiquitous in hospital environments, but their primary goal is to evaluate patients' clinical conditions. In turn, recent work shows that sensors can be applied to recognize workflow elements, including those from healthcare environments (VANKIPURAM et al., 2011). Two main areas encompass these studies. The first one is the IoT, which presents considerable work tailored for healthcare applications (ISLAM et al., 2015). IoT technologies focus on monitoring the **identity** and **location** aspects. In particular, Real-Time Location System (RTLS) is considered a promising method to monitor and document medical activities (BOULOS; BERRY, 2012). The second area is Computer Vision (CV), which presents several studies related to human activity recognition using data collected from cameras (SÁNCHEZ; TENTORI; FAVELA, 2008). It has a greater focus on the **state** aspect. Thus, a combination of sensing technologies based on RTLS and Computer Vision can be used to capture the workflow of a hospital environment, enabling its analysis and improvement.

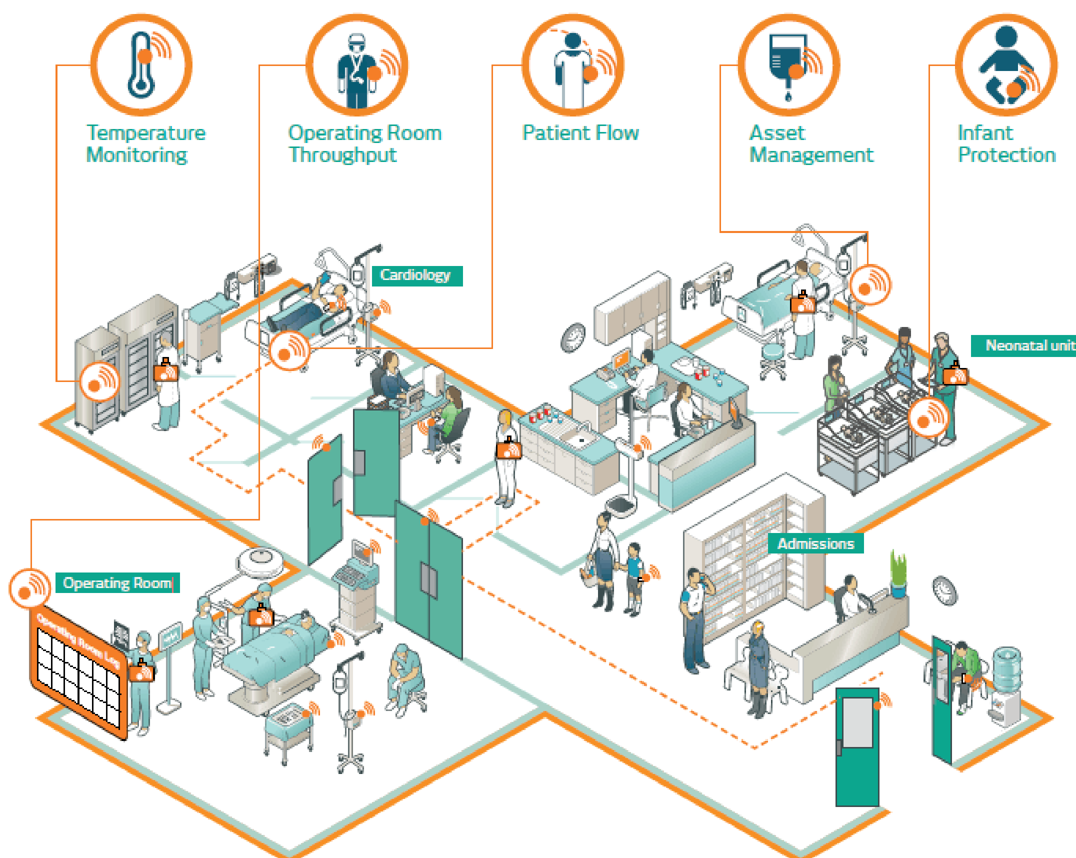
2.2.1 Real-Time Location Systems

RTLS are solutions for indoor identification and location tracking of persons and assets. Such systems consist of a set of fixed readers or anchors reading wireless signals from tags (BOULOS; BERRY, 2012). The system applies position estimation methods to these signals, outputting the tag position in its coordinate system. Thus, by assigning a tag to a specific target, it is possible to monitor its location within a building facility (indoor spaces). Nowadays, specifically in the healthcare landscape, solutions are employing Wireless Sensor Networks (WSN) technologies in RTLS, such as Wi-Fi, Radio Frequency Identification (RFID), Bluetooth, and Ultra-wideband (UWB) (TAN et al., 2015; ADAME et al., 2018; DECIA et al., 2017; REGOLINI et al., 2019; KOLAKOWSKI; DJAJA-JOSKO; KOLAKOWSKI, 2017). Particularly, UWB technologies surpass low-frequency technologies when it comes to interference since it implements adaptive frequency hopping (LEE; SU; SHEN, 2007).

WSNs can provide location information, which is a crucial factor in understanding the application context (LIU et al., 2012). The low cost of sensor technology has eased the proliferation of WSNs in many areas, such as healthcare and smart buildings (MAINETTI; PA-

TRONO; VILEI, 2011). A WSN is a network of tiny devices that cooperate using wireless protocols to collect information about a target physical environment (RAWAT et al., 2014). WSNs comply with a wide range of solutions, thus being characterized by their high heterogeneity (MAINETTI; PATRONO; VILEI, 2011). Data gathered by different devices can be stored and combined locally or sent to other networks, such as the Internet (RAWAT et al., 2014). A typical WSN usage scenario is to employ technologies such as Wi-Fi or Bluetooth for indoor location purposes (SCHEUNEMANN et al., 2016; XIAO et al., 2016). WSNs applied to the healthcare field aim to improve monitoring systems and services (ALEMDAR; ERSOY, 2010). For example, Wheeler (2007) demonstrates the value of a WSN that can report the location of patients, medical staff, and critical equipment. Another example is presented in Figure 5, which depicts the Logitrak[1] solution from Logi-Tag Systems (LOGITRACK RTLS, 2018).

Figure 5: An RFID based RTLS called LogiTrack, which provides hospitals, nursing homes, and clinics a method to accurately track, locate, and monitor assets and people, and trigger events in real time based on location and status (LOGITRACK RTLS, 2018).



Source: adapted from LogiTrack RTLS (2018).

These technologies generally employ sensor devices embedded with memory, processor, and wireless communication protocols to transmit data (RAWAT et al., 2014). A WSN uses a collection of devices to produce information about things and the context. However, a WSN

[1]https://logi-tag.com/real-time-location-system/

itself is not able to identify a target object independently (RAWAT et al., 2014). Conversely, RFID is a modern technology aiming to provide object identification in a short-range (WANT, 2006). Therefore, combining WSN and RFID technologies is an attractive solution for better information monitoring (RAWAT et al., 2014; MITROKOTSA; DOULIGERIS, 2009).

Unlike printed code technologies, RFID (ISOs 15693 and 14443) permits short-range identification without requiring visibility between readers and tags. RFID tags have exclusive identifications (ID) and can store and transmit information about the manufacturer, environment, and technical parameters. They segment into two principal standards: active, which require a power source like batteries, and passive, which do not require a power source (WANT, 2006). Proposals extensively apply RFID solutions aiming at activity recognition since RFID is a mature and low-cost technology (ALEMDAR; ERSOY, 2010). In turn, the creation of the Near Field Communication (NFC) Forum (NFC Forum, 2017) in 2004 is a direct result of attempts to spread RFID applications further. This forum's principal goal is to bring together existing mobile RFID standardization efforts and introduce short-range communication capabilities into RFID (WANT, 2011). It also aims to standardize mechanisms in which sensor devices can exchange information in very short distances. NFC technologies that operate in the high-frequency band at 13.56 MHz (ISO 14443, ISO 18092) support tag readings from distances of 10cm (WANT, 2011).

Regarding wireless communication protocols, UWB (PORCINO; HIRT, 2003), Wi-Fi (IEEE Computer Society, 2007) and Bluetooth (Bluetooth, 2017a; IEEE Computer Society, 2005) are part of the short-range wireless field (LEE; SU; SHEN, 2007; RAWAT et al., 2014). UWB is a radio frequency technology that provides information exchange by transmitting data through continuous short radio pulses. Wi-Fi is a well-known protocol that allows data transmission in higher ranges with larger data throughput. However, these factors result in higher values of energy consumption. Lastly, Bluetooth aims to cover wireless communication in short ranges focusing on low-cost devices. This protocol works in the 2.45 GHz band employing frequency hopping strategies to increase performance.

The IEEE standard 802.15.4 (IEEE Computer Society, 2016) aims at wireless communication devices with low energy consumption, cost and data rate (LEE; SU; SHEN, 2007; RAWAT et al., 2014). Many communication technologies apply this standard (RAWAT et al., 2014), for example Bluetooth Low Energy (BLE) (Bluetooth, 2017b), 6LoWPAN (RFC 4944) (MONTENEGRO et al., 2007) and ZigBee (ALLIANCE, 2017; FARAHANI, 2008). BLE is an attractive choice for WSN applications that require high data transmission in short distances between devices. Similar to BLE, ZigBee is a wireless communication technology for applications that focus on low energy consumption and cost (LEE; SU; SHEN, 2007; RAWAT et al., 2014). Likewise, the 6LoWPAN standard adapts the IPv6 over IEEE 802.15.4 networks focusing on low energy approaches. Recent work promotes the adoption of this standard instead of proprietary, rigid ones (MAINETTI; PATRONO; VILEI, 2011). Table 1 summarizes technical aspects from the aforementioned technologies.

Table 1: Technical details of the communication standards and protocols.

Technology	RFID	NFC	Wi-Fi	Bluetooth	UWB	ZigBee	6LoWPAN	BLE
Specification	ISO 15693, ISO 14443, ISO 18000	ISO 14443, ISO 18092	IEEE 802.11	IEEE 802.15.1	IEEE 802.15.3a	IEEE 802.15.4	IEEE 802.15.4, RFC 4944	IEEE 802.15.4
Frequency band	< 100 MHz, 868 MHz, 915 MHz, 2.45 GHz	13.56 MHz	2.4 GHz, 5 GHz	2.4 GHz	3.1-10.6 GHz	868-928 MHz, 2.4 GHz	868-928 MHz, 2.4 GHz	2.4 GHz
Max signal rate	–	424 Kbps	54 Mbps, 540 Mbps	3 Mbps	110 Mbps	250 Kbps	250 Kbps	1Mbps
Nominal range	30 cm, 1 m, 3-5 m	10 cm	100 m	10-100 m	10 m	10-100 m	1-100 m	200 m

Source: adapted from Antunes et al. (2018).

2.2.2 Computer Vision Techniques

RGB-Depth (RGBD) devices have become very popular and accessible in recent years, being extensively used in CV applications. Since they encompass color and depth data, there is usually no real advantage in choosing RGB cameras over RGBD. Some even have readily available implementations of skeletal tracking, hand tracking, and gesture recognition, which are strictly related to an actor's **state**. The analysis of actions and activities from one or more individuals is a fundamental aspect of workflow monitoring. It falls in the scope of surveillance, one of the main applications of CV-based human motion capture and evaluation (MOESLUND; HILTON; KRüGER, 2006). The study from Moeslund, Hilton, and Krüger (2006) reviews an extensive body of work that laid the foundation for current human motion analysis. Authors identify the following main contribution from early work in this field:

(*i*) image segmentation and tracking methods that enable recognition of subjects in uncontrolled, outdoor environments with frequent of object occlusion;

(*ii*) pose estimation techniques based on probabilistic approaches that enabled high accuracy on challenging natural, uncontrolled images;

(*iii*) model-based techniques to recognize complex human body movements for activity tracking;

(*iv*) use of pose estimation techniques to recognize simple human activities, such as sitting, walking, and running.

In recent years, the field of CV and human motion analysis further advanced at a higher pace due to breakthroughs in computer processing capabilities. These enabled the implementation of algorithms with previously prohibitive complexity. Consequently, CV methods can analyze images and models of higher complexity and extract a richer set of information as a result. Kadkhodamohammadi et al. (2017) is an example in which the authors employ CV techniques to

estimate the pose of medical staff in an operating room. Figure 6 shows their results employing a 3D pictorial structures approach.

Figure 6: Pose estimation technique employed by Kadkhodamohammadi et al. (2017). The lines represent body parts recognized by their thechnique.

Source: adapted from Kadkhodamohammadi et al. (2017).

According to their working principles, the majority of RGBD cameras can be generalized into three categories: Stereo Vision, Structured Light, and Time-of-Flight. These different depth imaging approaches attribute advantages and limitations of their own to the equipment. While hardware components and techniques may vary drastically among devices, their principles remain the same. Table 2 presents some camera devices and their technical details, including the approach adopted by each one. In the next sections, a brief overview of each category is presented.

2.2.2.1 Stereo Vision

Stereo cameras are usually composed of two (sometimes more) RGB camera components aligned over a common baseline at known positions and orientation. Analogous to human eyes, the cameras' disposition ensures slightly different scene perspectives while preserving a substantial overlap of their fields of view. Using the acquired color images, also called stereo pairs, Stereo Vision (SV) algorithms can synthesize depth images. Depth pixel values are computed through triangulation, requiring its position to be identified in both stereo pair images. The process of identifying corresponding pixels in two RGB images, known as the correspondence problem (HUSSMANN; HAGEBEUKER; RINGBECK, 2008), is the major challenge tackled by SV algorithms.

SV solutions can be implemented using any pair of RGB cameras since most of the complexity lies in algorithms rather than hardware. Among the advantages of SV are high reso-

Table 2: Technical details about the camera devices.

Devices	Type	Data Modalities		Depth Image			Communication
		RGB Image	Depth Image	Max Resolution	Max FPS	Range	
Stereolabs ZED	Stereo Vision	✓	✓	4416 x 1242 (15 fps), 3840 x 1080 (30 fps), 2560 x 720 (60 fps), 1344 x 376 (100 fps)	100	0.7 - 20m	USB 3.0
Microsoft Kinect v1	Structured Light	✓	✓	640 x 480	30	0.8 - 4m (default), 0.4 - 3m (near mode)	USB 2.0
Intel RealSense R200	Structured Light	✓	✓	640 x 480	60	0.6 - 3.5m (indoor), 0.6 - 10m (outdoor)	USB 3.0
Intel RealSense SR300	Structured Light	✓	✓	640 x 480	60	0.2 - 1.5m	USB 3.0
Asus Xtion Pro Live	Structured Light	✓	✓	640 x 480 (30 fps), 320 x 240 (60 fps)	60	0.8 - 3.5m	USB 2.0/3.0
Occipital Structure	Structured Light		✓	640 x 480 (30 fps), 320 x 240 (60 fps)	60	0.4 - 3.5m	USB 2.0
Microsoft Kinect v2	Time-of-Flight	✓	✓	512 x 424	30	0.5 - 4.5m	USB 3.0
Creative Senz3D	Time-of-Flight	✓	✓	320 x 240	30	0.15 - 1m	USB 2.0
PMD CamBoard Pico Flexx	Time-of-Flight		✓	224 x 171	45	0.1 - 4m	USB 2.0/3.0
Heptagon Taro	Time-of-Flight		✓	120 x 80	-	Up to 5m	USB, SPI, UART, I2C, Optocoupler I/O, WiFi, Zigbee, Bluetooth

Source: adapted from Antunes et al. (2018).

lution and the fact that it does not require energy emission (HUSSMANN; HAGEBEUKER; RINGBECK, 2008). This passive approach benefits outdoor usage and prevents equipment interference, making multicamera implementations quite feasible.

2.2.2.2 Structured Light

Like SV, Structured Light (SL) cameras also perform triangulation based on their components' position. However, instead of relying on the scene to provide meaningful features to solve the correspondence problem, it projects light patterns over the observed area (SCHMALZ et al., 2012). Since projected patterns are known beforehand, instead of a camera pair, it is possible to build an SL device with a single camera and a light projector (CHUA; REN; ZHANG, 2014). Depth results from deformations observed in the patterns (HARTMANN; SCHLAEFER, 2013; PAULY et al., 2015).

Addressing the main issues found in passive stereo, this active approach offers a robust solution to deal with untextured surfaces. On top of that, projected patterns may lead to captured images, which are trivial to match (SCHARSTEIN; SZELISKI, 2003), requiring less computationally intensive algorithms. Also, projectors can act as light sources in low light environments,

making such devices more robust in that regard as well.

2.2.2.3 Time-of-Flight

Time-of-Flight (ToF) cameras are a recent technology compared to triangulation based alternatives and are becoming increasingly popular (FüRSATTEL et al., 2016; BAUER et al., 2013). Usually, the principal components of these devices are a light emitter and an image sensor. The emitter dispatches modulated light pulses or continuous-wave to the observed scene (FOIX; ALENYA; TORRAS, 2011). Light reflects at the sensor, which demodulates it in parallel for each pixel. Devices emitting discrete pulses calculate distance based on pulses' time to travel forth and back to the camera. Meanwhile, continuous-wave solutions measure phase differences between the original signal and the received one.

One of ToF cameras' main advantages is that they are immune to environmental lighting conditions, working even in the dark (FüRSATTEL et al., 2016). Solid colored surfaces also pose no challenge as this strategy completely avoids the correspondence problem (HUSSMANN; HAGEBEUKER; RINGBECK, 2008). Lastly, the parallel acquisition of the whole pixel matrix at the hardware level makes it a very efficient solution.

2.3 Quality of Service

The concept of QoS in the telecommunication field is not new. In the last century, the telecommunication advance allowed humans to send high volumes of information from one physical place to another (LINDSEY; SIMON, 1991). Telecommunication technologies introduced distributed systems network connected performing data transmission remotely. For instance, in telephony, it is possible to capture and transmit back and forth the human voice in nearly real-time, allowing people to talk to each other, even being remotely distant. In such a setting, users easily perceive poor performance impacting their experience of using the system. For instance, too many packet losses in voice transmission practically make the conversation not feasible. In this context, the QoS of a system is strongly related to its performance, impacting user experience (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014; OODAN et al., 2003).

Nowadays, QoS is a well-established concept that addresses the overall performance of services provided by systems through data transmissions, such as cloud computing, computer networks, and telephony. Several standards and recommendations from different institutes define the term QoS in their documents (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014). In the E.800 recommendation, the International Telecommunication Union's Telecommunication Standardization Sector (ITU-T) defines QoS as a telecommunication service's ability to satisfy its user needs (ITUT, 2008). The document shows that QoS is a perceived characteristic by users of a telecommunication system. Basically, QoS refers directly to system communication performance, which impacts the user experience. That is, data transmission problems cause the

service to offer poor performance and a bad experience for the user.

In turn, the European Telecommunications Standards Institute (ETSI) defines QoS as a level of quality of a service (ETSI, 1994). Mainly, the document makes a relation between QoS and network performance. It introduces four points of view to clarify the E.800's generic definition of QoS: user's QoS requirements, QoS offered by the service provider, QoS achieved by the service, and QoS perceived by the user. According to the document, QoS is expressed through parameters that can be technical, such as packet loss, or non-technical, such as time of availability.

Both definitions have the user in common experiencing the final system's performance. It demonstrates that a system's QoS is centered on the system's ability to achieve the user's expected performance. To measure QoS, most of the parameters fall in one of the two possible groups (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014; OODAN et al., 2003): (*i*) network-related (or technical-related) parameters; (*ii*) non-network related (or non-technical related) parameters. For the first one, commonly, several performance metrics measure the QoS of a system, such as throughput, delay, jitter, and packet loss rate (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014; OODAN et al., 2003). These metrics have in common that they refer specifically to the system's network performance, indicating data transmission problems. For example, the delay metric is a measure of time to transmit data from one point to another in the network. In turn, jitter refers to the expected time interval between two consecutive data receptions. Multimedia streaming helps the understanding of jitter. For instance, in video transmission, each video frame is transmitted individually and in the same recording sequence. The interval each frame arrives at the destination should be the same for a smooth visualization. Irregular intervals between packets directly impact the video frame rate visualization, which the user instantly perceives.

Besides network-related metrics, QoS can also be measured through non-network related performance, such as service provisioning time, availability, reliability, maintainability, or even satisfaction rate (VARELA; SKORIN-KAPOV; EBRAHIMI, 2014; OODAN et al., 2003). In these cases, they do not represent a single metric measurement, but the system's assessment from a point of view. For instance, availability is a characteristic of a system to be available when needed. From the users' point of view, they can require a system to be available 99% of the time. From the provider's point of view, this translates to network parameters such as the number of successful replied requests. As another example of a non-network related performance metric is user satisfaction with the service. In this particular case, it is essential to understand the user's satisfaction level. This can leads to bad ratings of the system, which can be correlated with technical parameters at the provider's side. What all non-network related parameters have in common is that they are of easy understanding by the users. That helps them define their requirements, which are further translated to technical terms by the service provider.

In the context of healthcare applications, QoS refers to specific metrics and also limit val-

ues for them. Many studies present different types of metrics that target different health data types (MUKHOPADHYAY, 2017; NANDA; FERNANDES, 2007; MALINDI; KAHN, 2008; LEE et al., 2011; SKORIN-KAPOV; MATIJASEVIC, 2010). To illustrate that, Table 3 presents some traditional healthcare data types and their related QoS metrics. As the table shows, besides audio and video information, equipment telemetry and patient metrics are fundamental data types in healthcare. Such information is valuable to track the patients' health within the medical workflow. Electrocardiogram (ECG), pulse rate, and blood pressure are examples of this type of data. They represent the patient status and require much less bandwidth for data transmission. Considering their metrics, they have essential requirements in data transmission delay because time is an important factor in handling a medical crisis. Although the table defines a fixed number of data types, currently, the advance of Healthcare 4.0 brings new data types into the healthcare scope as location information, for instance. Therefore, the information available for medical decisions is continuously growing, and the specific QoS metrics for them are not available yet.

Table 3: QoS metrics for healthcare applications.

Data Type	**Data Rate**	**QoS Metric**		
		Maximum Delay	**Maximum Jitter**	**Packet Loss (%)**
Voice	4-25 Kbps	150-400 ms	-	3
High Quality Voice	384 Kbps	100 ms	50 ms	-
Diagnostic Sound	32-256 Kbps	100-300 ms	-	1
High Quality Video	0.64-5 Mbps	100-300 ms	30-50 ms	0
Uncompressed Image	30-40 MB	-	-	0
Region of Interest Image	15-19 MB	-	-	0
ECG	24 Kbps	1 s	-	0
Pulse Rate	2-5 Kbps	1 s	-	0
Blood Pressure	2-5 Kbps	100 ms	25 ms	-
Telemetry (diagnostic)	25.6 Kbps	200 ms	200 ms	-
Telemetry (alarms)	5.1 Kbps	200 ms	200 ms	-
Infusion Pump (status)	1 Kbps	200 ms	1s	-
Infusion Pump (alert)	1 Kbps	200 ms	1s	-
Barcode Medication Administration	0.8 Kbps	500 ms	500 ms	-
Electronic Medical Record	4.1 Mbps	200 ms	5 ms	-

Source: elaborated by the author.

2.4 Summary

This chapter presented a few concepts to a better understanding of this document focusing mainly on three subjects: (*i*) healthcare workflow monitoring; (*ii*) sensing technologies in the scope of healthcare; and (*iii*) definition of QoS. The healthcare workflow consists of a series of tasks performed to achieve a given objective in the hospital. These tasks characterize activities executed by the medical staff team in several hospital facilities. Specifically, medical settings are essential environments, and their workflow comprises critical tasks that influence patients' health. The workflow monitoring of such environments provides valuable information that can improve the performance of medical services.

Different technologies are employed in healthcare to track and monitor assets, equipment, and people. Nowadays, two areas emerge as the leading solutions to monitor hospital services: (*i*) RTLS; and (*ii*) Computer Vision. RTLS solutions focus primarily on tracking the position of tags or badges in real-time. A subject under tracking carries a tag that is assigned to it. Therefore, the tag positioning is associated with the person or equipment that is carrying the tag. On the other hand, Computer Vision techniques are less intrusive since they do not need to bear with a device. These techniques are most suitable for activity recognition in which human pose estimation strategies attempt to extract the position and the skeletal tracking of the subjects from image frames.

Monitoring data from healthcare environments can improve medical services significantly. However, users' great demands can decrease the ability of systems to deliver services for user applications. This situation is a common problem in telecommunication distributed systems. The concept of QoS defines the level of performance of such systems that can impact the user experience. QoS can be seen as the performance properties of a system in delivering its services. These properties can be both technical and non-technical. For instance, in the first group, time delay in data transmission is a technical metric that measures lags in transmission channels. In the second group, the availability of a system describes its capacity to be available when needed. Several other properties can measure the QoS of a system, making it essential to deliver quality services for users.

3 RELATED WORK

This chapter reviews the current state-of-the-art regarding the use of QoS strategies in the healthcare scope. It focuses on this specific scope to find related literature that considers the challenges of deploying computing systems in clinical environments. Such environments deal with critical information from patients and medical staff that require confidentiality and responsibility in its use. Section 3.1 describes the search strategy to gather the most relevant articles for the current study. Following, Section 3.2 describes the details of the selected articles from the search strategy. In turn, Section 3.3 discusses the articles showing the main gaps. Finally, Section 3.4 summarizes the content present in this chapter.

3.1 Search Methodology

The literature review in this study adopts the principles of systematic reviews (BIOLCHINI et al., 2005; KITCHENHAM; CHARTERS, 2007) to achieve reproducibility and high-quality results. It targets providing an overview of solutions that employ QoS strategies in healthcare systems and platforms. The current methodology consists of two criteria:

(*i*) **Inclusion criteria**: definition of search parameters, such as keywords, year range, and databases;

(*ii*) **Exclusion criteria**: definition of removal filters to select only the most relevant articles.

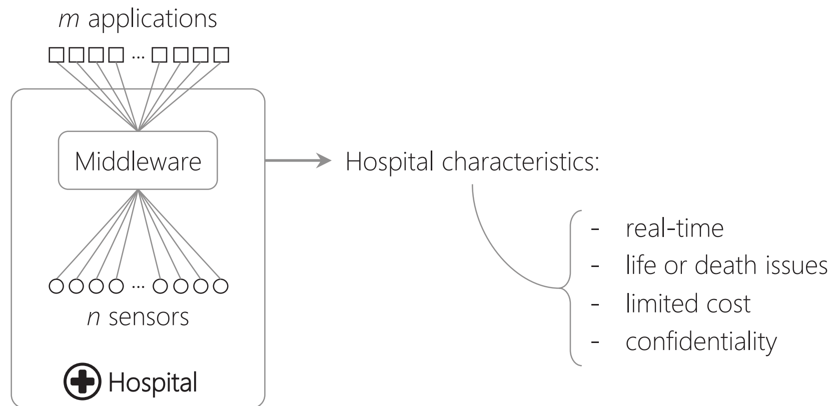
Concerning the hospital scope, middlewares have specific characteristics, as presented in Figure 7. Real-time, safety, and cost are primarily related to clinical activities; therefore, middlewares for hospitals differ from traditional middlewares, which do not have these characteristics. Thus, to limit the scope of the article search strategy, the search string is defined as follows:

(“QoS” in the title) AND (“health” OR “healthcare” OR “hospital” in the metadata)

The inclusion policy consists of considering articles that explicitly describe a QoS strategy in a healthcare scenario, presenting their methods and algorithms. Writing paper guidelines suggest that the article’s title should present the aim of the research (MACK, 2012). Therefore, the current research opts to find articles that mention the word “QoS” to hit research focusing specifically on QoS strategies.

The search strings’ combination as a search string to be used in the target databases represents the inclusion criteria. Besides, to guarantee that the study reflects the most recent findings on monitoring technologies, the scope is limited to the last five years (in the date of this writing, studies published between January 2015 and November 2020). Considering the related article sources, the **scope** of the literature search encompasses the selection of literature databases.

Figure 7: Characteristics that differ hospital middlewares from traditional middlewares. Time and cost are related and critical for safety in these environments.



Source: elaborated by the author.

It is narrowed to sources that: (*i*) index articles from relevant conferences and journals from Computer Science and Medicine; and (*ii*) include a broad selection of venues to maximize the number of returned articles. Based on these criteria, Table 4 presents the seven databases that compose the inclusion criteria.

Table 4: Databases for the inclusion criteria.

Database	**URL**
ACM Digital Library	`https://dl.acm.org/`
Google Scholar	`https://scholar.google.com.br/`
IEEE Xplore	`https://ieeexplore.ieee.org/`
Microsoft Academic	`https://academic.microsoft.com/`
PubMed	`https://www.ncbi.nlm.nih.gov/pubmed/`
ScienceDirect	`https://www.sciencedirect.com/`
Springer Link	`https://link.springer.com/`

Source: elaborated by the author.

The inclusion criteria result in a set of articles, called raw literature corpus. Following the methodology, the exclusion criteria select the most relevant articles from the raw corpus in this study. The following removal filters form the criteria:

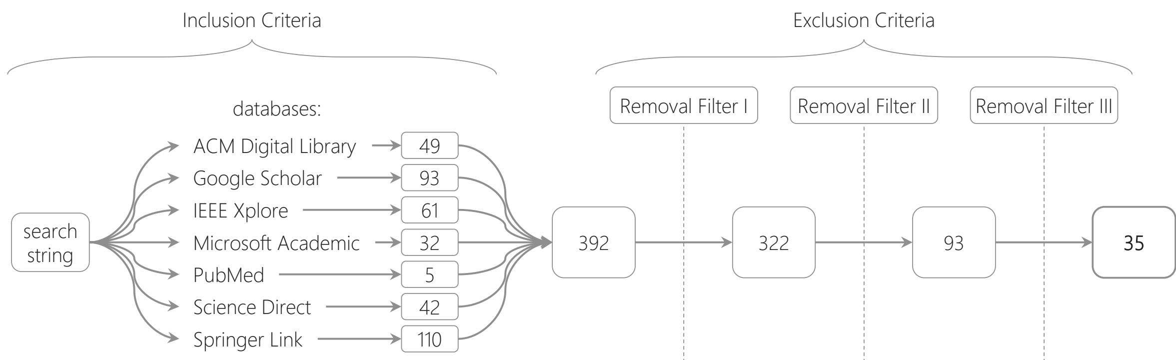
I. Duplicate removal: The remaining studies from individual databases are grouped, and duplicates are eliminated;

II. Title and abstract review: The title and abstract from each study are reviewed, and those that do not address sensing technologies applied to healthcare monitoring are removed;

III. Full-text analysis: The remaining studies' full text is further analyzed to remove articles not expressly related to the current study.

Figure 8 depicts the filtering process sequence to achieve the final corpus. The total corpus resulting from the first search methodology step consists of 392 articles. After applying steps I and II, the corpus reduces to 93 items. The article analysis from step III consists of reading each article's abstract to check whether the article is related. This analysis resulted in a set of 35 related articles, which the next sections describe.

Figure 8: Filtering process sequence. Filter II removes most of the articles due to their focus be different than healthcare.



Source: elaborated by the author.

3.2 State-of-the-Art

Table 5 summarizes the characteristics of each article showing their primary focus and the QoS strategies they employ. From the analysis of the state-of-the-art regarding QoS strategies in the healthcare domain, most of the strategies focus on three main fields:

(*i*) **Wireless Body Area Network (WBAN)** (65.7%) (SAMANTA; MISRA, 2018; AHMED; LE MOULLEC, 2017; PANDIT et al., 2015; HU et al., 2015; BRADAI et al., 2015; SAMANTA; LI; CHEN, 2018; IRANMANESH; RIZI, 2018; WANG; SUN; JI, 2018; GUEZGUEZ; REKHIS; BOUDRIGA, 2018; HASSAN; ALRUBAIAN; ALAMRI, 2016; RAZZAQUE et al., 2015; JACOB; JACOB, 2015; LEE; JUNG; LEE, 2017; PURI; CHALLA; SEHGAL, 2015; BAI et al., 2019; LIU; LIU; CHEN, 2017; LIU; YAN; CHEN, 2017; POORANI et al., 2017; TSENG; WANG; YANG, 2020; GOYAL et al., 2020; ZUHRA et al., 2020; WAHEED et al., 2020; IBRAHIM et al., 2020);

(*ii*) **Telemedicine** (14.2%) (POORANI et al., 2017; WANG et al., 2019; AGIRRE et al., 2016; MAATOUGUI; BOUANAKA; ZEGHIB, 2017; SODHRO et al., 2019);

(*iii*) **IoT** (14.2%) (ZITTA et al., 2018; KHALIL; MBAREK; TOGNI, 2019; BANOUAR et al., 2017; GATOUILLAT; BADR; MASSOT, 2018; VADIVEL; RAMKUMAR, 2020).

The following subsections group the articles' analysis according to these fields. Also, articles that do not fall in at least one of these fields form an additional subsection.

Table 5: Summary of the researches and their strategies addressing QoS in healthcare solutions.

Paper	Year	Real-time	Focus	QoS Strategy	Description
(JACOB; JACOB, 2015)	2015		WBAN	Data prioritization, and sleeping scheduler (energy)	Sleeping mechanisms for WBAN sensors to improve energy performance
(RAZZAQUE et al., 2015)	2015	✓	WBAN	Service differentiation (dynamic data priorization), error control mechanism	Error recovery in WBAN multi level.
(PANDIT et al., 2015)	2015	✓	WBAN	Dynamic data priorization	QoS MAC protocol for WBAN.
(PURI; CHALLA; SEHGAL, 2015)	2015		WBAN	Data classification/priorization, time slot allocation protocol	Time slot allocation strategy in the MAC layer for WBASNs.
(BRADAI et al., 2015)	2015		WBAN	Data classification/priorization, adminission control	QoS-aware architecture for remote signal monitoring in WBANs.
(HU et al., 2015)	2015		WBAN	Multi-level data priorization	Multi-level QoS strategy in the MAC layer for WBAN.
(HASSAN; AL-RUBAIAN; ALAMRI, 2016)	2016	✓	WBAN	Resource elasticity	Cloud model to provide medical services from WBANs.
(AGIRRE et al., 2016)	2016		Telemedicine	Admission control	Middleware for remote health monitoring.
(AHMED; LE MOULLEC, 2017)	2017		WBAN	Channel selection	Channel optimization approach for WBANs.
(BANOUAR et al., 2017)	2017		IoT	Data prioritization, resource elasticity	QoS mechanism for IoT middlewares to enhance living environments.
(POORANI et al., 2017)	2017		WBAN, Telemedicine	Cloud elastiticy, data priorization (alerts)	Remote monitoring system of WBAN.
(LEE; JUNG; LEE, 2017)	2017		WBAN	Energy control algorithm, data rate adaptation	Energy control algorithm for WBAN.
(LIU; LIU; CHEN, 2017)	2017		WBAN	Data priorization	Transmission rate adaptation strategy in WBANs.
(LIU; YAN; CHEN, 2017)	2017		WBAN	Scheduling protocol	Slot time and order allocation strategy for WBANs.
(MAATOUGUI; BOUANAKA; ZEGHIB, 2017)	2017		Telemedicine	MAPE control loop, resource elasticity	Cloud-based remote patient monitoring system.

Continued on next page

Table 5 – continued from previous page

Paper	Year	Real-time	Focus	QoS Strategy	Description
(SAMANTA; MISRA, 2018)	2018	✓	WBAN	Connectivity stablishment, packet scheduling protocol (data priorization)	Dynamic connection establishment and packet scheduling for WBANs.
(SAMANTA; LI; CHEN, 2018)	2018		WBAN	Scheduling protocol with data prioritization, admission control algorithm	Packet scheduling algorithm for data transmission in WBANs.
(CELDRÁN et al., 2018)	2018	✓	ICE	Load balancing	ICE architecture to manage security, privacy, QoS, and high availability.
(GUEZGUEZ; REKHIS; BOUDRIGA, 2018)	2018	✓	WSN, WBAN	Data prioritization, admission control algorithm, routing protocol	Sensor cloud architecture for health services that combines WSN and WBAN.
(ZITTA et al., 2018)	2018	✓	IoT	Multi-channel transmission	IoT infrastructure to monitor QWL.
(IRANMANESH; RIZI, 2018)	2018		WBAN	Data prioritization, admission control algorithm, routing protocol, time slot allocation protocol	QoS solution for co-existing WBAN.
(WANG; SUN; JI, 2018)	2018		WBAN	Power control model	Power control scheme for multi-WBANs.
(GATOUILLAT; BADR; MASSOT, 2018)	2018		IoT	Feedback and adaptation loop	IoT self-adaptive system for critical monitoring
(SODHRO et al., 2019)	2019	✓	Telemedicine	Video smoothing	QoS control algorithm to 5G telemedicine.
(WANG et al., 2019)	2019		Telemedicine	Data priorization	Slicing framework for eHealth media applications over 5G networks.
(AL-TARAWNEH, 2019)	2019		Medical network	Service differentiation, data categorization, and data prioritization	Medical grade QoS solution.
(VENKATESH et al., 2019)	2019	✓	SDN	Service differentiation, routing protocol	QoS-aware multipath algorithm for medical data transmission in a SDN.
(BAI et al., 2019)	2019		WBAN	Clock synchronization	Protocol for QoS and energy efficiency in WBAN.
(KHALIL; MBAREK; TOGNI, 2019)	2019	✓	IoT	Traffic differentiation	IoT architecture for data access.

Continued on next page

Table 5 – continued from previous page

Paper	Year	Real-time	Focus	QoS Strategy	Description
(TSENG; WANG; YANG, 2020)	2020	✓	WBAN	Channel selection	Dynamic channel and superframe selection scheme in IEEE 802.15.6 WBANs to avoid interference.
(GOYAL et al., 2020)	2020	✓	WBAN	Data prioritization, data rate adaptation	Energy efficiency transmission rate approach for WBANs with data classification.
(ZUHRA et al., 2020)	2020	✓	WBAN	Routing protocol	Strategy for selecting an optimal end-to-end route which considers a composite metric (link quality metric).
(WAHEED et al., 2020)	2020		WBAN	Routing protocol	Modification in the Ad hoc On-Demand Distance Vector (AODV) routing protocol for medical data.
(VADIVEL; RAMKUMAR, 2020)	2020		IoT	Routing protocol	Bio-inspired based routing protocol for IoT-based healthcare applications using cognitive radio ad hoc networks.
(IBRAHIM et al., 2020)	2020		WBAN	Routing protocol and data prioritization	WBAN architecture with a single sink node and routing protocol for high priority data transmission.

3.2.1 Wireless Body Area Network

In the literature, the acronym for body area networks varies between BAN, BSN, WBAN, or WBSN. The current study adopts WBAN for all cases to maintain consistency and make the reading easier. WBAN consists of several sensors attached to a patient's body, acquiring physiological information and forwarding it to a central node. The central node then transmits this information through a network connection to a server. Many strategies focus on delivering QoS in such architectures. Samanta and Misra (2018) propose a dynamic QoS connection establishment. They focus on a master sensor moving and changing the connection from an access point to another. Ahmed and Le Moullec (2017) propose a channel optimization approach that consists of an algorithm to choose the best channel for the communication between the central node and the gateway. Moving nodes perform automatic handover between different gateways when the patient changes from one location to another. Tseng, Wang, and Yang (2020) also propose a channel optimization approach. They focus on medical emergency settings with multiple WBANs interfering with each other's data transmission. Their strategy consists of looking

at the historical state of channels to choose the most appropriate data transmission channel.

Focusing on energy consumption, Pandit et al. (2015) propose an energy-efficient Media Access Control (MAC) protocol to classify the data into six different classes that define the transmission priorities. The class of a packet might change dynamically according to context. Also employing data classification, Hu et al. (2015) employ a service prioritization strategy in data transmission on different levels: user, data, and time. They focus on the IEEE 802.15.4 MAC beacon mode protocol focusing on high priority data transmissions. Likewise, Bradai et al. (2015) propose PMAC (Priority MAC) and a QoS aware architecture for remote signal monitoring. The coordinator node classifies data into three categories for data prioritization: emergency, normal, on-demand. The strategy also employs an admission control mechanism that checks if it is possible to provide the required QoS for new incoming data. In turn, Goyal et al. (2020) focus on the energy efficiency aspect. The strategy employs a Genetic Algorithm to adjust the transmission rate of each node. Additionally, the study proposes the classification of packets in high or normal priority data.

Co-existing WBANs pose a challenge in data transmission because of mutual interference. In this context, Samanta, Li, and Chen (2018) focus on co-existing WBANs sharing access point bases. The authors propose a packet scheduling algorithm for high critical priority data transmissions. They also present an admission control strategy to guarantee QoS employing an optimization problem solution using Lagrangian Multipliers. Iranmanesh and Rizi (2018) propose MultiBodyQoS, a QoS solution for co-existing WBANs. MultiBodyQoS detects interference and applies several methods to mitigate the problem: channel change, different time slots allocation, admission control, and data prioritization. In turn, Wang, Sun, and Ji (2018) propose a transmission power adaption scheme. The authors design a Nash bargaining game model scheme for power control. They focus on the problem of interference between multiple co-located WBANs.

With a different scope, Guezguez, Rekhis, and Boudriga (2018) propose an admission control mechanism to choose between a WSN route or a 4G connection to send the data from the WBAN to the cloud. The solution employs traffic classification to guarantee real-time data streaming: high-priority and best effort. Also, employing cloud computing, Hassan, Alrubaian, and Alamri (2016) propose a cloud model to provide medical services employing elasticity strategies. They propose a virtual machine (VM) allocation strategy to improve cost, energy efficiency and reduce response time for remote WBANs. Differently, Razzaque et al. (2015) employ a service differentiation strategy, which defines two different prioritization levels: critical and non-critical. The authors propose an error recovery mechanism based on network conditions to deal with network packet transmission errors.

Among several challenges, WBANs undergo the problem of constraint resources. As its architecture contains several nodes with low computing and energy capacity, guaranteeing system availability is the target of many studies. For instance, Jacob and Jacob (2015) propose a sleeping mechanism for WBAN sensors to improve energy performance. Also, the strategy

changes data acquisition from cyclic pooling to priority pooling in emergencies. Also, Lee, Jung, and Lee (2017) present an energy control algorithm that adjusts the transfer rate from sensors according to users' QoS requirements. In turn, Puri, Challa, and Sehgal (2015) focus on an energy-efficient time slot allocation strategy in the MAC layer for WBANs. Their solution calculates the time slot duration according to the payload size and employs data packets classification for transfer prioritization: emergency, critical, reliability constrained, delay constrained, and normal. Bai et al. (2019) focus on energy consumption to improve network lifetime and performance. The authors propose a protocol for clock synchronization since synchronization between the clocks is essential for nodes to wake-up at the right time.

Liu, Liu, and Chen (2017) propose a transmission rate adaption strategy in WBANs. The strategy evaluates the quality of the link to decide changing or not the transmission rate. Besides, the solution provides a high priority queue to handle abnormal signals. The same authors also propose a slot time and order allocation strategy for WBANs in a different article (LIU; YAN; CHEN, 2017). They employ a TDMA-based (Time-Division Multiple Access) MAC protocol to avoid collisions by scheduling sensors to transmit data to the central node.

Finally, some studies propose new routing protocols for WBANs. Zuhra et al. (2020) propose a routing protocol to optimize the end-to-end route in data transmission. They focus on defining a strategy that addresses a composite metric to evaluate the quality of each route. Waheed et al. (2020) introduce a new routing protocol as an extension for the Ad hoc On-Demand Distance Vector (AODV) protocol (DAS; PERKINS; ROYER, 2003). The authors propose new metrics to evaluate the quality of links. Lastly, Ibrahim et al. (2020) present a WBAN architecture with a single sink node. The system implements a routing protocol for high priority data transmission to decrease energy consumption and improve network performance. Their strategy directly routes critical packets to the sink node and routes data with other priorities for different paths.

3.2.2 Telemedicine

In the field of Telemedicine, strategies focus on remote monitoring systems employing different technologies. Poorani et al. (2017) propose a real-time system to monitor health information from WBANs. The solution classifies health data into five categories for data prioritization: very low, low, normal, high, and very high. The method consists of a severity scheduler in the cloud to deliver alerts for doctors. Also, Agirre et al. (2016) propose a middleware for health monitoring based on multiple nodes acquiring information from multiple sensors. They present a case of study for a remote fire emergency system to monitor one subject in a home. The authors designed a QoS manager that performs three operations: admission control, QoS monitoring, and composition algorithm, including task prioritization.

Focusing on cloud computing, Maatougui, Bouanaka, and Zeghib (2017) present a self-adaptive cloud-based remote patient monitoring system. The system employs the MAPE (Monitor-

Analyze-Plan-Execute) loop model performing adaptations based on QoS constraints. In their context, QoS is a Service Level Agreement (SLA) input using response time, the number of service invocation failures, and cumulative service invocation cost as metrics in a threshold-based strategy. In the field of 5G networks, two articles present some advances. First, Wang et al. (2019) propose a slicing framework for e-health media applications over 5G networks. The solution employs traffic classification and priority queues in the inbound traffic. Second, Sodhro et al. (2019) present a QoS control algorithm to 5G networks. The strategy consists of a window rate adjusted to smooth video playbacks on the client's side.

3.2.3 Internet of Things

In the context of healthcare solutions, some strategies focus on IoT infrastructures to support medical applications. Zitta et al. (2018) propose an IoT infrastructure that employs a multi communication channel approach to guarantee QoS. The strategy focuses on the quality of work in medical environments. The approach consists of redundant real-time data transmission to the server through two or more channels. Khalil, Mbarek, and Togni (2019) present QBAIoT, an IoT architecture for e-health services. The authors propose in the architecture a service differentiation for data transmission composed of four classes: real-time mission-critical, real-time non-critical mission, streaming, and best-effort. QoS is achieved through data prioritization transmission for real-time classes. In turn, Vadivel and Ramkumar (2020) introduce the emergence of IoT in healthcare applications. In this scope, the authors focus on specific data transmission in cognitive radio ad hoc networks. They propose a routing protocol to select alternate routes over the failures of nodes and active routes.

With a different approach, Banouar et al. (2017) propose a QoS mechanism for IoT middlewares to enhance living environments (ELE). The strategy adds a field to the HTTP (Hypertext Transfer Protocol) header to define one of the three priority levels: high, medium, and low. They propose an algorithm that evaluates the header's value and applies some methods to deal with requests: rejection, delaying, or scheduling. Also, they propose resource elasticity methods in the middleware components to improve its performance. Lastly, Gatouillat, Badr, and Massot (2018) propose an IoT self-adaptive system using a patient health monitoring case of study. The architecture consists of gateways connected to one or more sensors, which the feedback and adaption loop mechanism monitors. The solution monitors parameters from the sensors, such as battery level, and triggers adaptions in defined situations.

3.2.4 Other Approaches

Finally, three studies push the research into different bounds than the previous ones. First, Al-Tarawneh (2019) present a medical-grade QoS solution using IEEE 802.11e WLAN (Wireless Local Area Network). The solution employs a service differentiation defining different

priority levels depending on the type of data. Second, Venkatesh et al. (2019) propose a QoS-aware multi-path algorithm for medical data transmission in a Software Defined Network (SDN) framework. They focus on the time delay of medical data employing differentiated packet flows. Third, Celdrán et al. (2018) propose an Integrated Clinical Environment (ICE) architecture to manage security, privacy, QoS, and high availability. The strategy employs a QoS policy that evaluates the sampling rate of data and, based on a threshold, deploy new access points to balance the load. The authors explore edge computing to run medical applications and guarantee low latency.

3.3 Discussion and Research Opportunities

The state-of-the-art shows that articles that present QoS strategies focus mainly on three fields of study: (*i*) WBAN; (*ii*) Telemedicine; and (*iii*) IoT. In particular, the great majority of the studies focus on WBAN technology. WBAN is a promising solution that is gaining much attention in the last few years in the healthcare scope. In the WBAN field, the authors focus on improving data transmission quality from sensors to the master node and from the master node to the server. Some strategies also focus on providing energy efficiency strategies in data transmissions to improve the network lifetime. One of the main problems in this field is the co-existence of multiple WBANs, which might interfere with each other. Therefore, the researchers seek to mitigate these problems by employing scheduling algorithms to avoid collisions. In the fields of Telemedicine and IoT, authors propose architectures to provide remote access to medical data. The main challenge these solutions tackle regards guaranteeing service quality to the remote user. Considering the QoS strategies authors employ in their strategies, five main strategies are commonly used, independently the focus of the research:

(*i*) data prioritization;

(*ii*) admission control;

(*iii*) resource elasticity;

(*iv*) routing protocols;

(*v*) scheduling protocols.

Several studies employ data prioritization to improve the time delay for data transmission. The strategy consists of defining different classes for each data type and prioritizing transmission for data packets with high priority levels. Some approaches employ admission control algorithms to guarantee that new data transmissions do not decrease the network's quality. Therefore, the system only admits new connections before applying an evaluation algorithm that checks the network's current state and estimates the new state. Resource elasticity is a common strategy in cloud environments, and some researchers employ this idea to provide

QoS. In such strategies, solutions employ VM allocation and migration strategies to increase some modules' performance in overloaded situations. Finally, authors also employ routing and scheduling protocols to improve network performance and decrease transmission delay. The former consists of algorithms that define the best route to transmit data according to the current network status. On the other hand, the latter consists of algorithms that control the time slot allocation for each data transmission and their order according to the amount of data and data type. Although targeting different fields and strategies, the current literature has some gaps that can be further explored. Among them, three main points can be explored and are highly related to the current study:

(*i*) Focus on hospital high critical environments;

(*ii*) Combine multiple strategies in different levels;

(*iii*) Give emphasis specifically in real-time data transmissions.

First, most strategies focus on remote health sites, like nursing homes, and only WBANs for patient health monitoring. Within a hospital, there are many critical locations, such as operating rooms, in which a system can provide useful data from patients and physicians. Second, a healthcare system has two main actors: the sensors at the hardware level, generating information, and the users at the application level, which consume and process data from the system. Strategies focus mainly on the first level to provide QoS in data transmission from the sensor to the network. Furthermore, although some initiatives present concerns regarding real-time, they do not focus intensely on this issue. In general, solutions that focus on real-time only consider improving time delay from priority packets or providing an architecture that supports real-time data transmissions.

3.4 Summary

This section presented an overview of the current literature in the scope of Qos for healthcare. Several studies propose new systems and architectures for data monitoring in the healthcare scenario in the last few years. Hospitals administrators and stakeholders are paying attention to the technological advance and how IoT technologies can revolutionize healthcare environments. Looking at this scope, most of the recent studies concentrate their efforts on providing QoS for WBAN architectures, IoT systems, and Telemedicine solutions. In all cases, the main QoS strategies the literature investigates are data prioritization, admission control, resource elasticity, routing protocols, and scheduling protocols. Although diverse techniques, the studies focus on employing them in particular cases, not combining them dynamically, neither considering high critical environments with many data sources. Besides, the strategies do not focus on the whole workflow of medical processes. Assuring data transmission of assets and physicians is not on the scope of these strategies. The lack of studies in these fields raises a research opportunity that the current study targets.

4 THE HEALTHSTACK MODEL

This chapter details the HealthStack model and the main contributions of this research. HealthStack is a QoS-aware real-time middleware that focuses on hospital facilities. The middleware collects data from sensors, stores it in a database, and delivers it to user applications meeting QoS requirements. Its main characteristic is its ability to provide QoS for both user applications and sensors according to the system load. The number of sensors and user applications may change over time, and HealthStack analyzes their effects on the applications' performance in real-time. Its main goal is to guarantee that medical applications keep consuming real-time data regardless of the system load. The model contains a manager component in charge of monitoring and adaptation tasks to provide the multi-level QoS. HealthStack is based on the publish-subscribe paradigm, in which applications subscribe to data streams from sensors to receive updated readings in real-time. In the healthcare scope, applications are sensitive to jitter and delay depending on the application's purpose. Therefore, it is essential to guarantee that sensor data arrives at the applications' side according to their requirements. HealthStack's differential approach covers the following topics:

(*i*) automatic and transparent QoS management to the application user;

(*ii*) a mechanism for applications to request and receive sensor data with or without QoS requirements;

(*iii*) the dynamic QoS Service Stacking strategy to enable different QoS services to acquire sensor data, improving the middleware performance and resouce consumption.

The next sections present characteristics and algorithms that HealthStack employs. Section 4.1 presents the design decisions of the model development. Next, Section 4.2 describes the model architecture and its components. Section 4.3 defines the middleware's communication protocol. Then, Section 4.4 presents the HealthStack QoS model. This section defines QoS parameters, algorithms, and strategies HealthStack employs in its model, including the Service Orchestration model. Finally, Section 4.5 summarizes the concepts from this chapter.

4.1 Design Decisions

Real-time is a significant issue in healthcare since data support medical staff and hospital administrators to monitor medical processes (KURMOO et al., 2020; DJELOUAT et al., 2020). HealthStack provides soft real-time data to user applications and for storing purposes in the context of data acquisition. One of the characteristics of soft real-time systems is that they have a soft deadline, which means that an occasional delay in the data production does not have a catastrophic impact on the environment (KOPETZ, 1997). Currently, healthcare environments fit in this concept since data acquired from sensors do not trigger immediate actions. That is,

no actions are made based on single sensor data measurements. Instead, physicians and administrators always analyze these data for decision support. HealthStack focuses on guaranteeing real-time data avoiding data loss and respecting QoS requirements, and at the same time improving resource consumption. In summary, the HealthStack design comprises the following decisions:

(*i*) user applications can inform QoS thresholds of delay and jitter;

(*ii*) the middleware allows configuration of default QoS parameters for each sensor; therefore, if the application user does not inform QoS requirements, the middleware assumes the default;

(*iii*) programmers do not need to adapt the application to profit from the middleware QoS services;

(*iv*) the middleware components are interconnected in a private wired network environment to improve network performance;

(*v*) the clock from the middleware architecture components are synchronized to an internal time server in the same network.

(*vi*) the model focus on medical sensors for medical workflow monitoring;

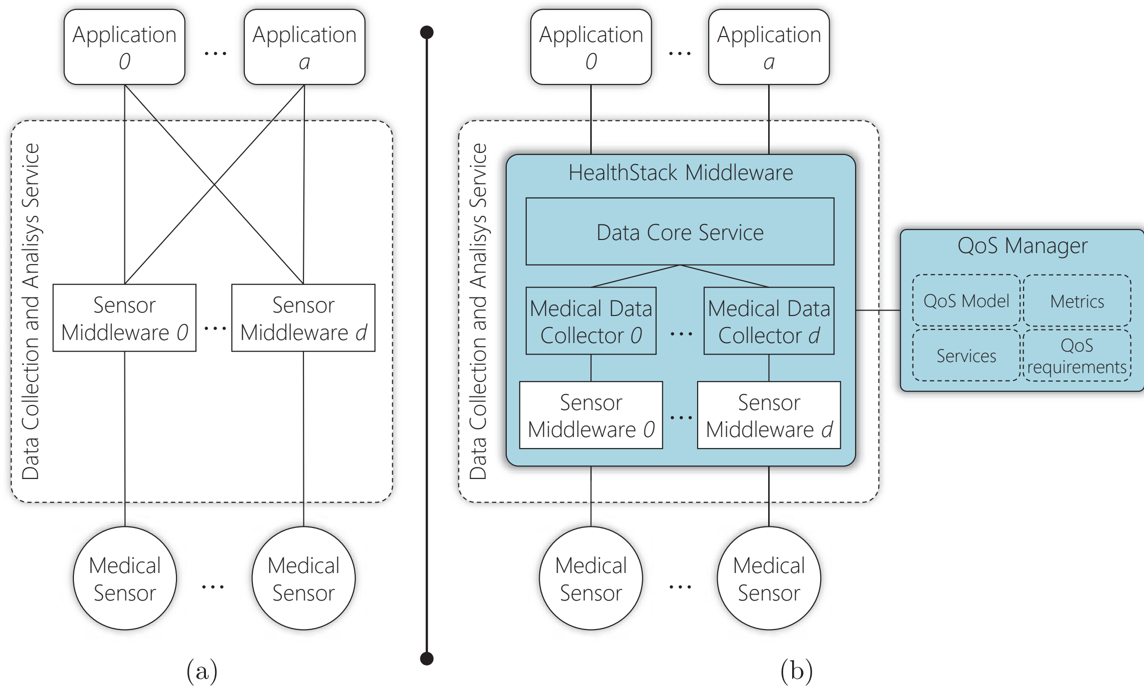
(*vii*) QoS focuses on supporting soft real-time data;

(*viii*) the deployment allows configuration of sensor priorities and the type of data they produce;

(*ix*) data from sensors include the data source identification, data type, timestamp, and raw information from the sensor.

The model has two main actors involved in the production and consumption of data: (*i*) sensors; and (*ii*) user applications. HealthStack implements QoS strategies, called QoS services, to meet QoS requirements by monitoring different metrics in both sensor and user application levels. On the one hand, at the user level, applications that consume middleware data can define QoS requirements that the middleware monitors. If the user application does not provide its requirements, HealthStack sets default requirements depending on the data the application requests (details in Subsection 4.2.1). On the other hand, the middleware defines requirements to guarantee data acquisition at the sensors level, even if there are no user applications requesting data. Besides, the middleware also adjusts sensors parameters to improve hardware utilization and energy consumption through an additional manager component that periodically monitors the system's health. Figure 9 depicts the HealthStack idea in comparison to a traditional middleware without QoS support. The figure shows the QoS Manager as an additional component, managing QoS requirements, metrics, and services. Additionally, the Manager employs a Service Orchestration model, called QoS Service Stacking, to address QoS violations.

Figure 9: Comparinson of sensor middlewares: (a) typical approach; and (b) HealthStack main idea. HealthStack comprises a Manager to monitor metrics, application requirements, and provide services for user applications and sensors.



Source: elaborated by the author.

HealthStack monitors specific metrics related to time to deliver packets (delay) and time variation between data packets (jitter). In addition to these two, it also monitors computing resources such as CPU (Central Processing Unit), memory, and network. Based on the results of such monitoring, the HealthStack QoS Service Stacking adds or removes services to the middleware components to meet the QoS requirements. The service adaptations do not impact the user application execution, which does not need to take any actions. This set of services comprises vertical elasticity, data rate adaptation, data prioritization, and compression strategies. The model adopts vertical elasticity in the operating system level of nodes running components from HealthStack architecture. HealthStack employs this elasticity model because replication of components would require network connection reorganization, which directly impacts data availability. On the other hand, data rate adaptation and compression are strategies focused on changing the middleware's behavior. They impact the system delay and network utilization, improving response time when many sensors and applications are connected to the middleware. In turn, data prioritization is a common QoS strategy that differentiates packets from different sources with priority levels. HealthStack provides different data queues for each priority. Then, it provides more or fewer resources to process them depending on the queue.

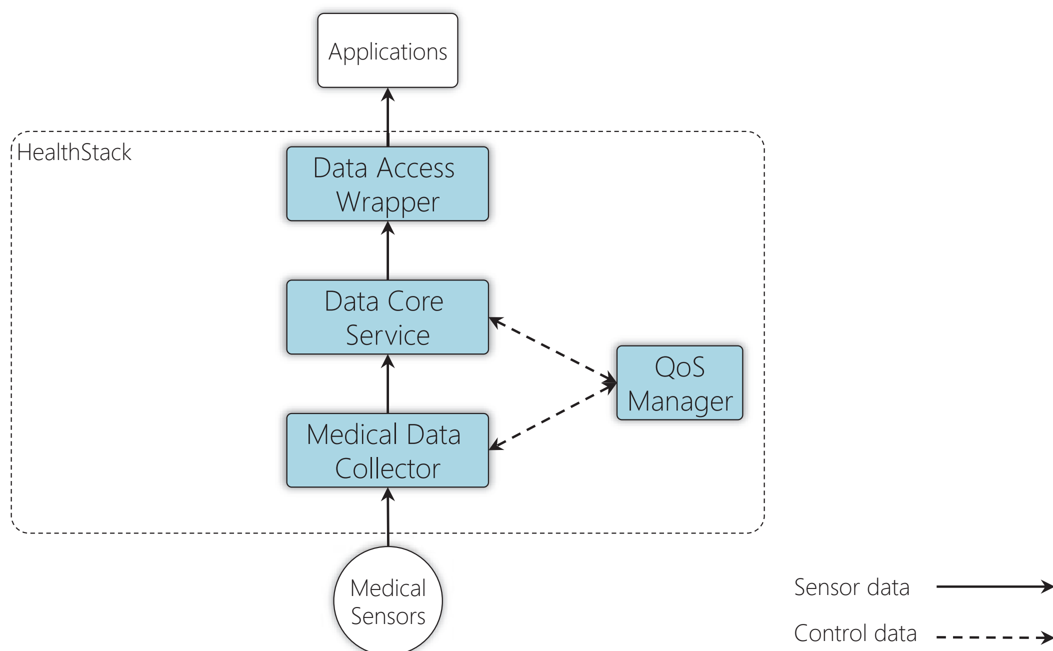
Regarding sensor data acquisition, HealthStack design comprises a distributed system that extracts raw samples from several sensor devices. It encodes them into JavaScript Object Notation (JSON) format with information regarding the source, type, and timestamp, and delivers

it to connected applications. Also, HealthStack stores each sample in a database for offline queries. Further, HealthStack requires some previous configurations considering the different types of sensor devices attached to the system. HealthStack is aware of the applications' QoS parameters and provides QoS services to improve delay, jitter, and resource consumption. Additionally, different sensors generate different types of data that require distinct priority requirements. HealthStack requires at the deployment point the configuration of the priorities for each sensor, the classes they belong, and also the default QoS parameters for each class.

4.2 Architecture

HealthStack's architecture introduces four distributed components that have specific roles in the middleware: (*i*) Data Core Service (Core); (*ii*) Medical Data Collector (Collector); (*iii*) Data Access Wrapper; and (*iv*) QoS Manager. Figure 10 briefly shows their organization linking applications to sensors. Instead of connecting directly to sensors and implementing their Application Programming Interface (API), the applications connect to the Core through the Data Access Wrapper. The Core is the central component that collects data from multiple Collector instances. In turn, a Collector instance extracts data from each sensor. Finally, the QoS Manager manages the Core and Collector instances to provide QoS services.

Figure 10: HealthStack components connecting sensors to applications. The light blue boxes represent the contributions of HealthStack.

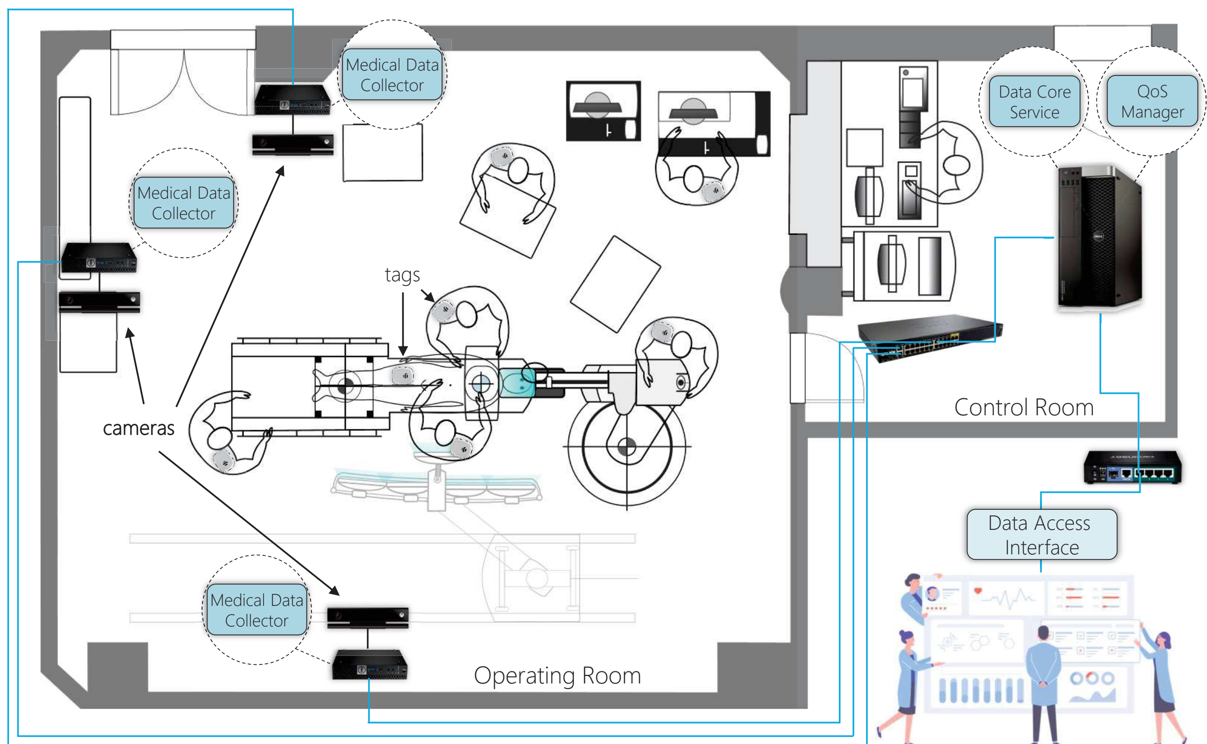


Source: elaborated by the author.

This research comprises a clinical partner that provides a real hybrid operating room for experiments. At the deployment point of view, Figure 11 depicts the overview of the architecture installed at the clinical partner Instituto de Cardiologia - Fundação Universitária de Cardiologia

(IC-FUC)[1], Porto Alegre, Rio Grande do Sul, Brazil. The architecture was deployed in a real hybrid operating room to monitor and track surgeries. The figure depicts how the components are distributed and how the data is provided to applications. It shows three ToF camera devices and several RTLS tags for workflow monitoring. The Collector instances acquire image frames from the cameras and position samples from the medical team, and forward them to the Core. The Core processes the information and provides data visualization for remote physicians and hospital administrators. The applications access the data through the Data Access Wrapper that provides specific functions in the publish-subscribe model. Finally, the QoS Manager monitors the whole architecture's parameters and modifies them to improve QoS for applications and resource consumption.

Figure 11: Deployment of the architecture in an actual hybrid operating room. Three ToF camera devices and several RTLS tags track the surgery workflow. The server processes the data and provides visualization for remote physicians and hospital administrators.



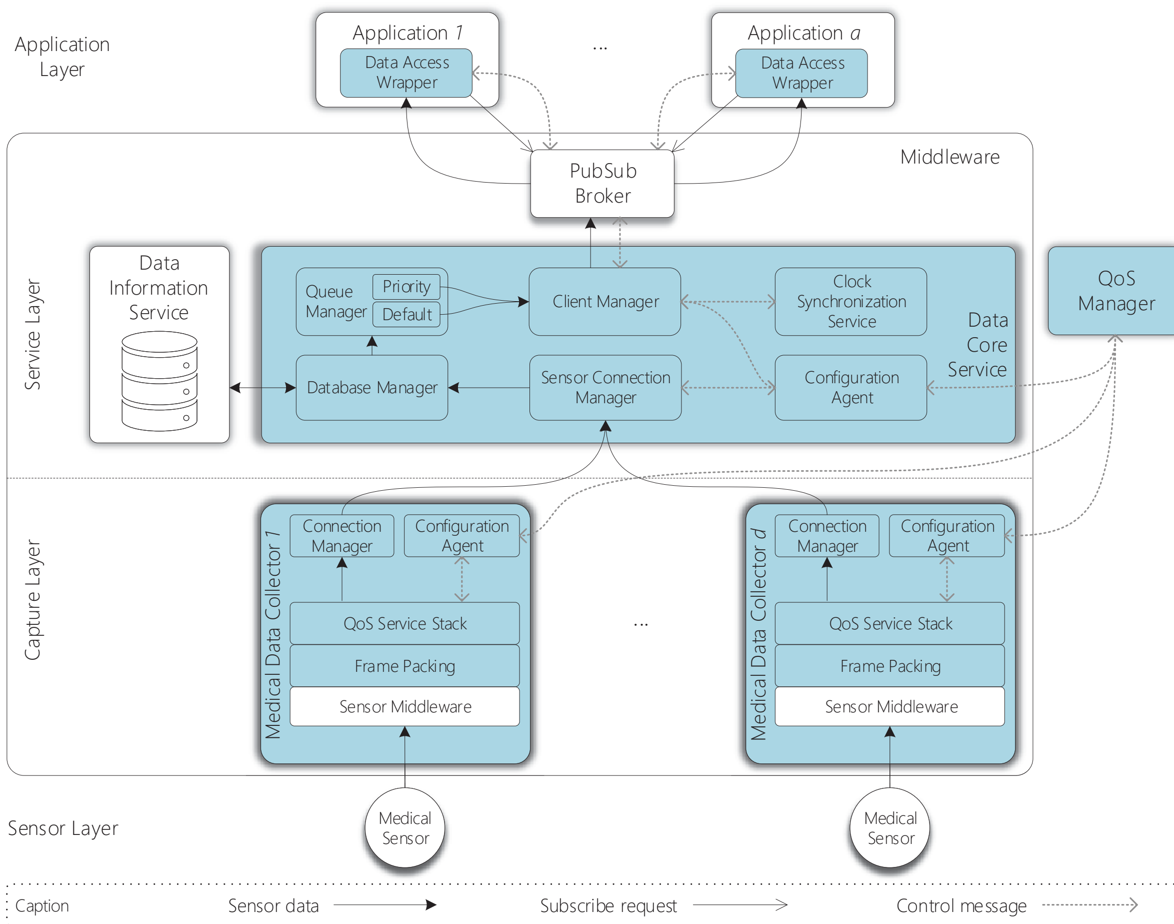
Source: elaborated by the author.

Going deeply into the architecture and communication details, Figure 12 illustrates the organization of the middleware components, highlighting the contributions in light blue boxes. The Core is the central component in processing incoming data from sensors and dispatching them to both Data Information Service and PubSub (publish-subscribe) Broker. Several Manager sub-components compose the Core to perform operations accordingly: Sensor Connection Manager; Database Manager; Queue Manager; and Client Manager. The Sensor Connection Manager is in charge of managing Collector connections, and their incoming data flows. The

[1]http://www.cardiologia.org.br/

Database Manager stores each incoming frame and sends them to the Queue Manager that reads the frame header to define which queue the data will be included. In turn, the Client Manager monitors the queues flushing their data to the Broker. The middleware comprises d Collectors corresponding to the number of sensor data sources. A sensor data source might provide data from one or several sensors of the same type. For instance, RTLS provides an API to access many tracking tags, while a camera sensor provides images from only one camera. Each Collector extracts raw data from a singular sensor data source and forwards it to the Core at a defined rate. In a scenario with two camera sensors (two image data sources) and an RTLS solution (one position data source), the middleware has three Collector instances.

Figure 12: HealthStack architecture. Light blue boxes represent the HealthStack components, while white boxes represent existing software and hardware.



Source: elaborated by the author.

Two parameters define each Collector's behavior: pri_d, the priority value of the attached sensor, and fps, the number of data samples per second in Hertz (Hz) to extract from the sensor. HealthStack allows the definition of a priority value (pri_d) for each sensor. pri_d should be a positive integer value ($pri_d > 0$) that the Core uses in its decision-making process to calculate the priority level of each sensor (see Section 4.4.4). At each $fps \div 1000$ milliseconds (ms), the Collector acquires the data through the sensor vendor API, packs it as a frame, applies

QoS services, and sends it to the Core. The QoS Service Stack is its primary sub-component in which applies QoS services for each data sample according to configurations performed automatically by the QoS Manager. Reconfigurations are transparent and do not interrupt the Collector's primary process, consisting of extracting and sending information to the Core.

On top of all components, the QoS Manager monitors several metrics and performs the QoS Service Stacking process. Its main goal is to address QoS violations by stacking QoS services for the middleware components. It carries this out by monitoring the applications' QoS level and performance measurements from all middleware components individually. The Manager uses a communication protocol exclusive for requesting metrics from the Collector and Core instances. It establishes a connection with each one and subscribes to their metrics. Then, they send updates of CPU, memory, network, delay, and jitter. Regarding the Core instance, it also sends to the Manager the QoS requirements of each application and their current delay and jitter. To acquire these values, the Data Access Wrapper sends updates to the Core at each second. The one-second interval is chosen to compute the mean delay, and jitter of all samples arrived for each second as the application has a defined number of frames per second to receive. The Manager periodically acquires measurements from all components and the applications' QoS level, and it uses these data to determine which services should be stacked. These services are employed directly to data samples in the Collector instances through the QoS Service Stack sub-component. Section 4.4 describes in detail this module and its functions.

4.2.1 Application Classes

The middleware requires all sensors to be defined from one of the two existing classes to guarantee that all sensors have default QoS values. HealthStack allows the configuration of such parameters at the deployment phase, making the system flexible for different deployments. Default QoS parameters are essential since some applications might request data without providing QoS parameters. In such cases, it is also crucial to monitor QoS violations even if the application does not provide QoS requirements. Therefore, the middleware can maintain a minimum QoS level to those applications respecting different default values. The model defines two application classes according to the type of data they require from the middleware:

(*i*) health monitoring systems;

(*ii*) workflow monitoring systems.

On the one hand, health monitoring systems target monitoring health parameters from patients to monitor their conditions. They collect vital signs and process them to provide insights into current conditions and forecast possible scenarios. Therefore, these applications focus on identifying critical situations that are occurring or might occur with the patient. For instance, an application may process health parameters to identify patterns and trends in these measure-

ments to alert the medical staff. This class of applications comprises sensor of blood pressure, heart rate, respiratory rate, temperature, and all sensors related to the patient.

On the other hand, workflow monitoring systems aim at tracking medical processes within a critical environment. These systems focus on acquiring image information from the environment and also position estimations from the hospital staff. Such information is essential to monitor phases of processes and control in real-time treatments and medications. Additionally, external applications may process such information to assess the medical staff's current activities individually in real-time. This class of applications covers sensors ranging from cameras to presence sensors and RTLS.

4.2.2 Time Synchronization

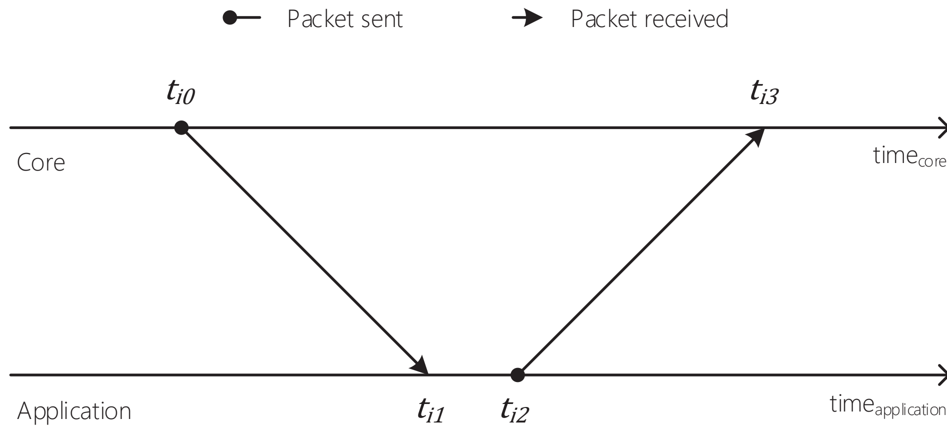
Clock synchronization in the architecture is important because sensor data packets contain a timestamp value in the Unix time format that defines the instant the data was generated. Different components from the architecture use this value to measure the data transfer delay. The components acquire the local time from the computing node in which it is running. Each node has its local clock used to measure the node's local time. Different nodes have different local times, and, therefore, clock synchronization is required between them so that events can be analyzed based on standard time. This standard time is frequently referred to as the approximate global time representing a global time for all nodes of a distributed system (KOPETZ; OCHSENREITER, 1987). In the architecture, several clocks are observed from the nodes that run the Core and Collector instances and the applications. The model performs two different synchronization processes: (*i*) middleware synchronization; and (*ii*) application clock calibration. In the middleware, since all modules work in a private network, they synchronize their local clocks to a primary time source (global clock) present in the same network through the Network Time Protocol (NTP) (MILLS, 1991) following the Coordinated Universal Time (UTC) standard. The service layer provides an NTP time server, which all nodes use to synchronize their clocks timely. Particularly, this time server runs at the same node where the Core runs.

The model does not require the application users to synchronize their operating systems' clock with the middleware clock to allow the end-delay measuring. After an application starts a new connection, a time calibration protocol takes place as a hand-shake process. The Data Access Wrapper hides this process from the application user, making it entirely transparent to the application. At each new application connection event, the Core calculates the clock offset and saves it to a local state. Every time the model calculates the time delay, it corrects its value applying the time shift after acquiring the application operating system clock. The calibration process consists of approximating the time difference between the application local clock to the global clock. This approximation is based on the NTP protocol as presented by Mills (1991).

The Core's sub-component Clock Synchronization Service performs the calibration process by sending a set of i roundtrip packets to determine the clock offset. These packets contain

four timestamps: t_{i0}, t_{i1}, t_{i2}, and t_{i3}. Figure 13 demonstrates message transmission and where the timestamps are acquired. t_{i3} and t_{i0} are the times before Core sends the packet and after receiving it back, respectively. t_{i1} and t_{i2} are the time instants the application received the packet and sent it back to the Core. The Core only sends a new packet after receiving the last one back. It uses the timestamps to calculate the clock offset O_i for each packet i according to equation 4.1. After calculating O_i for i packets, the Core employs the minimum filter to choose the offset from the packet with a lower delay.

Figure 13: Packet traveling path and timestamps.



Source: adapted from Mills (1991).

$$O_i = \frac{(t_{i1} - t_{i0}) + (t_{i2} - t_{i3})}{2} \tag{4.1}$$

4.3 Communication Protocol

This section describes the communication protocol of HealthStack in two levels: middleware; and application. Section 4.3.1 focuses on the middleware components communication protocol. It defines the messages the components exchange. In turn, Section 4.3.2 defines the user application communication protocol and its methods.

4.3.1 Middelware Communication Process

There are three different messages HealthStack components might transmit between them: (*i*) configuration data; (*ii*) metrics measurements; and (*iii*) sensor data. The middleware requires that all messages arrive at their destination without errors. Therefore, HealthStack employs the Transmission Control Protocol (TCP) from the Internet Protocol (IP) for all network transmissions. HealthStack employs TCP since it provides reliability guarantees on data transmission. Data transmission consists of messages including a 9-byte network header, which identifies the messages (details in Table 6), and a variable payload that contains one of the three messages.

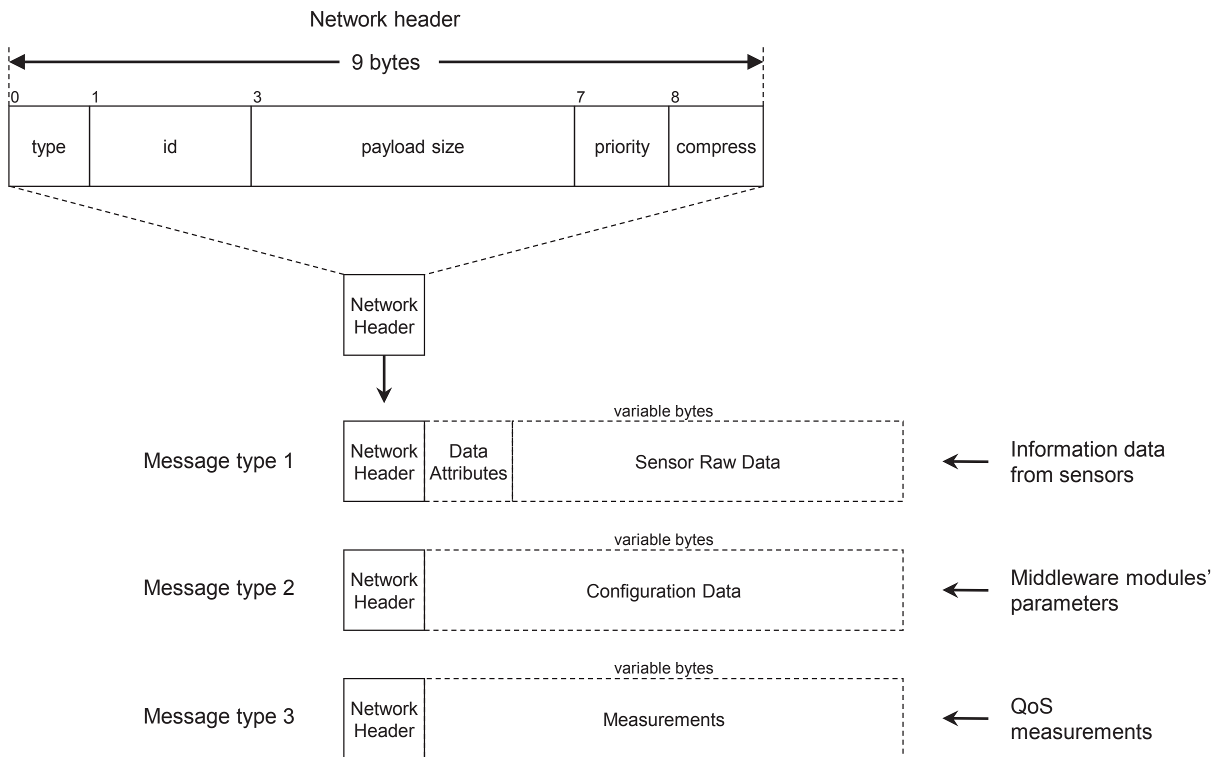
Figure 14 depicts the fields of the network header and the composition of each type of message. While the header is fixed for all messages, each payload is different depending on the message type.

Table 6: Description of the network header's fields.

Field	Description
Type	The payload type.
ID	Identification of the request.
Payload Size	The size of the payload in bytes.
Priority	The priority of the packet.
Compression	Identification of whether the payload is compressed or not.

Source: elaborated by the author.

Figure 14: HealthStack message types and their contents. All messages use the same network header, which identifies the packets.



Source: elaborated by the author.

Sensor data messages (type 1) represent the central information that HealthStack transmits. Medical Data Collector instances gather sensor raw information data from hardware sensors through their API or a vendor's provided service. The Collector packs it into a JSON, which is the payload of this message, to transmit it. This package contains six specific data attributes: (*i*) `Sensor_ID`; (*ii*) `Device_ID`; (*iii*) `Data_Collector_ID`; (*iv*) `Sample_Counter`; (*v*)

`Timestamp`; and (*vi*) `Type`. Table 7 organizes the details of each one of them briefly. The `Sensor_ID`, `Device_ID`, and `Data_Collector_ID` fields identify the source of the sensor data. Medical Data Collector instances might extract data from different physical sensors. Therefore, this set of IDs identify the sources individually. The `Sample_Counter` defines the sample sequence of the sensor data, and the `Timestamp` is the instant of time that the data was extracted from the physical sensor. Finally, the `Type` defines the data type, which can be, for instance, a position or an image frame. Jointly to these fields, it is attached the raw sensor information data composing a sensor data message.

Table 7: Description of the data attributes from sensor data messages.

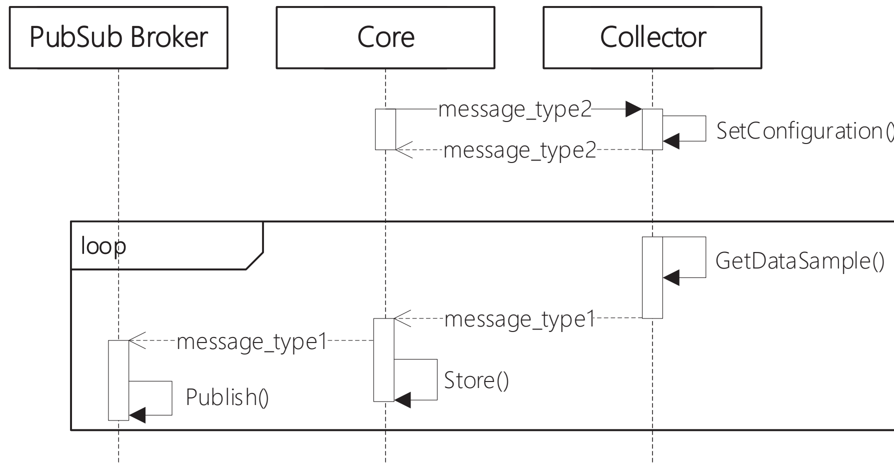
Field	**Bytes**	**Description**
`Type`	1	Data type.
`Timestamp`	8	Time in milliseconds that the data is collected from the sensor.
`Sample_Count`	4	Sequence number of the collected data.
`Data_Collector_ID`	2	Identification of the Collector instance that collected the data.
`Device_ID`	2	Identification of the device from which the data was collected.
`Sensor_ID`	2	Identification of the sensor from which the data was collected.

Source: elaborated by the author.

In turn, configuration data messages (type 2) contain component configurations and might be used for two reasons. First, to check the current configuration of a specific component instance. Second, to change the components' configurations. Its payload is also a JSON, but with the Collector's fps parameter and a flag for each QoS service to activate or deactivate them. The QoS Manager uses it to update the components' service stacks on-the-fly by communicating with the Configuration Agent sub-component. The configuration also contains the network address and port the Collector uses to connect to the Core and send the data samples. Figure 15 illustrates the communication process the components perform to produce data samples. First, the Core sends a message type 2 to each Collector informing their configurations. Second, the Collector sets its local configuration and replies it to the Core. After that, the Collector starts to collect data samples from its sensor, and at each new data sample, it packs into a message type 1 and sends it to the Core. When receiving data samples, the Core dispatches them to the PubSub Broker and stores them in the database. This process occurs in a loop and is the same for each Collector instance.

Finally, metric measurement messages (type 3) contain samples of metrics from the middleware components. It also contains a JSON as its payload, which encompasses the metrics measurements. The QoS Manager uses these samples to evaluate the middleware's status and monitor QoS violations for user applications. Based on that, the Manager employs the QoS Service Stacking model to guarantee QoS to applications. Figure 16 illustrates the communication

Figure 15: Communication process the components perform to produce data samples.



Source: elaborated by the author.

sequence that the Manager starts with the Core and Collector instances. It first starts a TCP connection with each component and requests their measurements. The components send a reply to the Manager with their local configurations to acknowledge the request. Then, the components acquire their local measurements and send them in a message type 3 to the Manager in a loop. Finally, the Manager receives each message and processes it for decision-making. This process repeats for each Collector instance.

4.3.2 Application Communication Process

At the applications side, the Data Access Wrapper plays an essential role in the middleware's communication process. It works as a wrapper of the publish-subscribe communication protocol allowing the programmer to use the traditional calls without adapting the application. Because of the Wrapper, the application does not need to connect to the sensor data sources directly. Instead, it only needs to access the Core through the Wrapper to consume data from many sensors in a standard data format. To do so, programmers must import HealthStack's communication library that offers the middleware QoS services transparently. It provides three calls:

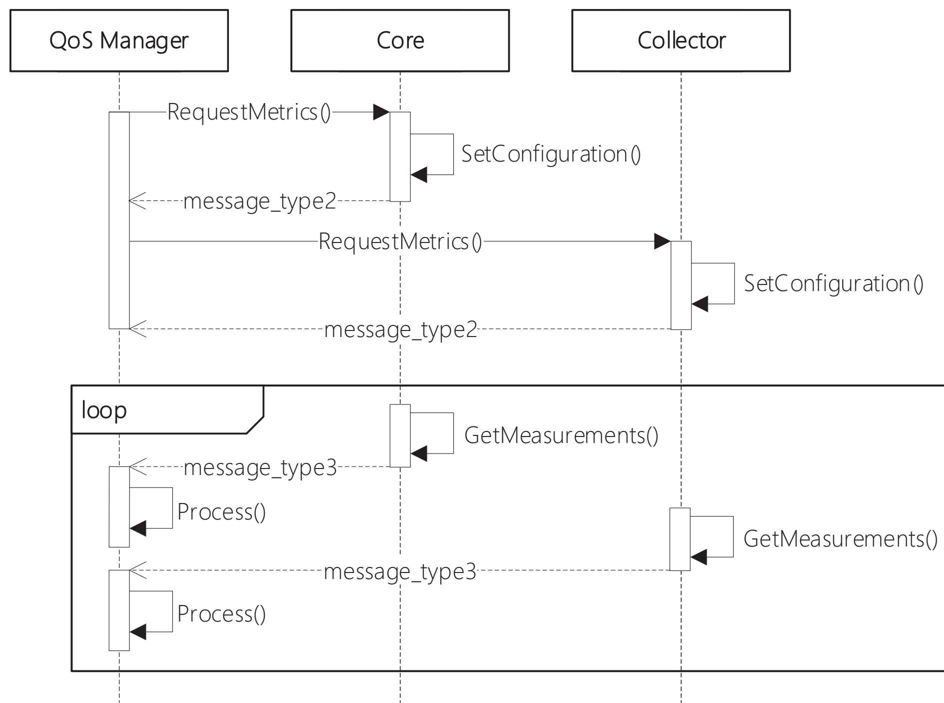
(*i*) `bool Connect(ADDRESS, PORT)`;

(*ii*) `bool SubscribeData(CONNECTION, SOURCE, FPS, DELAY_QOS, JITTER_QOS)`;

(*iii*) `bool ReceiveData(CONNECTION, &DATA, &BYTES)`.

The `Connect` method starts a TCP connection with the server `ADDRESS` at port `PORT`. The `ADDRESS` is the network address of the server that runs the PubSub Broker, and `PORT` is the listening port of the service. For instance, the address `mqtt.eclipse.org` is a

Figure 16: Communication sequence that the Manager starts with the Core and Collector instances to monitor their measurements.



Source: elaborated by the author.

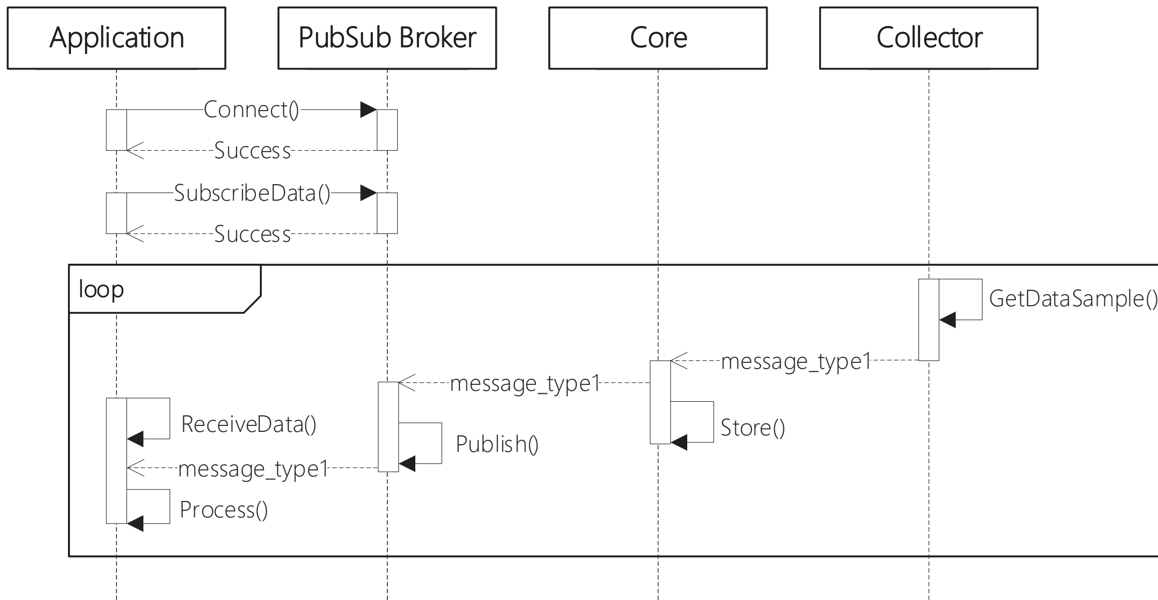
public MQTT (Message Queue Telemetry Transport) broker that listens the port `1833` [2]. The method `Connect` returns a `CONNECTION` object that is used as a parameter in the other two methods. The `SubscribeData` sends a subscription request of data from sensor `SOURCE` to the connection `CONNECTION`. The parameters `FPS`, `DELAY_QOS`, and `JITTER_QOS` are optional and, if defined, represent the frames per second, the maximum delay, and maximum jitter. In the case of `ReceiveData`, the function receives a data sample from the connection `CONNECTION`.

Figure 17 exemplifies the application calls and demonstrates the communication process from generating a data sample until delivering it to the application. The application connects the PubSub Broker and, after it, subscribes to a specific data stream. Then, the application only needs to keep receiving data that the Core publishes on the related topic. As mentioned in the last subsection, the Collector sends each new data sample to the Core, which stores it and publishes it in the PubSub Broker. The Broker is in charge of forwarding this data to applications that have subscribed to the corresponding topic.

The application can subscribe to topics and receive data without knowing the additional operations in the communication layer. Suppose the application does not provide a specific QoS threshold. In that case, HealthStack assumes the delay as the target QoS with a predefined value according to the application class defined by the sensor. Besides providing specific calls,

[2] https://mqtt.eclipse.org/

Figure 17: Application communication process to subscribe and receive data from a specific sensor data stream/topic.



Source: elaborated by the author.

the Wrapper unpacks incoming data according to the data sample's services. In particular, the Collector's data may be compressed, and uncompressing is the Wrapper's role. Finally, the Wrapper also measures the packet delay and jitter to send this information to the Core periodically. To do so, the Wrapper performs the clock calibration described previously in Subsection 4.2.2.

4.4 Quality of Service Model

HealthStack focuses on providing strategies to adapt the middleware to meet both user application and sensor requirements. As the application users have the system's final perception, it is crucial to guarantee QoS meeting their demands. For instance, a defined frame per second (FPS) must be respected not to impact the user experience. Thus, the middleware must be able to adapt itself to guarantee that. HealthStack provides multi-level QoS strategies to ensure that data is provided respecting specific QoS parameters. Besides providing strategies focusing on user-specific parameters, HealthStack also provides QoS parameters and services in the architecture's sensor level. Since HealthStack generates and stores data in real-time for the future, even if no user applications are consuming real-time data, the middleware must guarantee continuous data acquisition and storage. Thus, the HealthStack Core also demands QoS from Medical Data Collector instances.

The main goal of HealthStack is to provide real-time data to guarantee the user experience and, at the same time, improve hardware utilization. The model defines QoS parameters

focusing on middleware performance to provide online services for user applications. Considering healthcare environments, the middleware provides security for patients and medical staff by providing valuable information. Besides real-time, the middleware also provides data consumption from past events, critical for workflow analysis. It leads to improvements in the healthcare process and medical staff tasks.

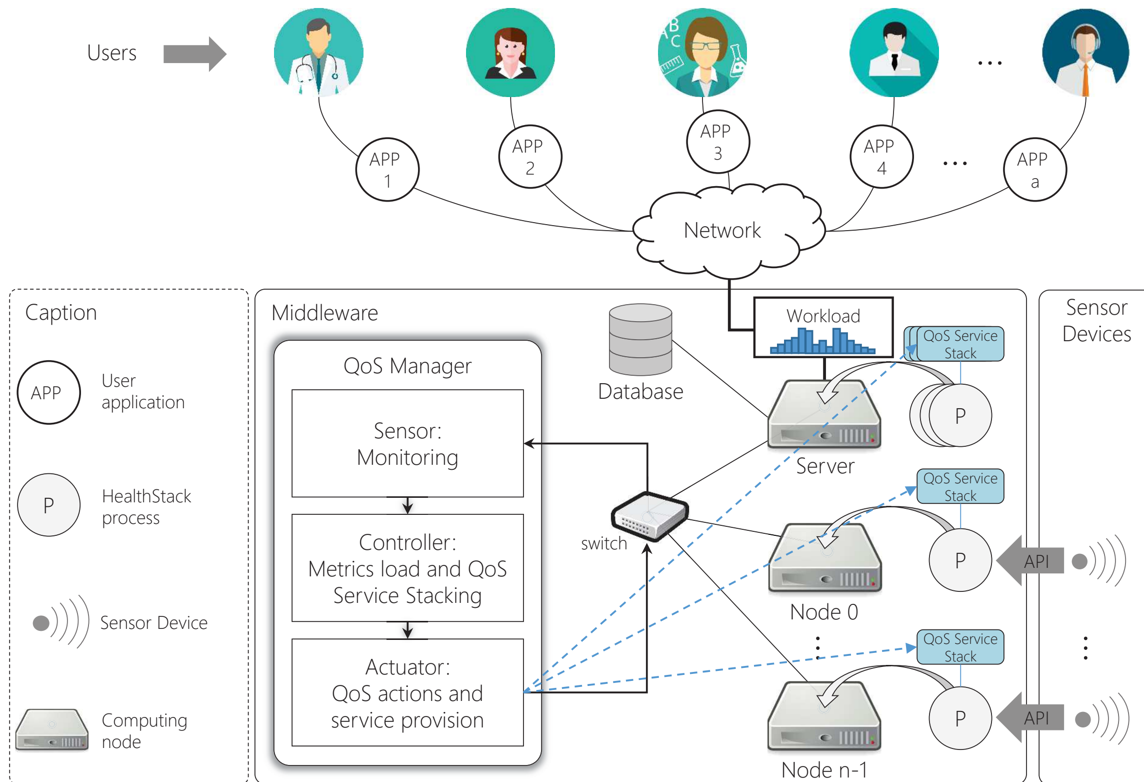
The HealthStack QoS model is designed as a closed feedback-loop architecture (GHANBARI et al., 2011; LIM et al., 2009). Figure 18 illustrates the architecture components and their organization showing the main control tasks of the QoS Manager. The middleware components can be distributed among computing nodes in a cluster within the hospital facilities or a single server. The QoS Manager has access to each HealthStack component instance regardless of their locations among servers or clusters. The architecture is composed of a server and n nodes in which the HealthStack components perform their roles depicted in Figure 12. Control theory is an engineering and mathematics branch focused on dynamical systems behavior and how they are affected by feedback (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). Therefore, service provisioning decisions should be made based on system performance according to application requirements. The service provisioning is obtained by the QoS Service Stacking model (detailed in Section 4.4.4), which evaluates a series of QoS metrics and defines the set of QoS services for each component of the middleware. The QoS Manager presents three primary functions that characterize control systems: a sensor to acquire monitoring data, a controller to evaluate measurements, and an actuator to provide modifications and services.

4.4.1 Managing QoS

The QoS Manager monitors the metrics and delivers the different component instances' QoS services. Its main task consists of collecting metrics from each process to compute it and decide whether actions are required to meet QoS requirements. In particular, the HealthStack components have individual QoS service stacks that might change over time, according to the QoS Manager decisions. Figure 19 depicts this task showing the QoS Manager monitoring cycle. Each process has particular metrics which the Manager evaluates individually. The Manager executes this procedure at a given time interval for two main reasons. First, continuous monitoring can increase considerable data traffic in the network and consequently impact sensor data transfer. Second, that prevents the Manager to take actions based on metric outliers that can occur in short periods. At each monitoring cycle, the QoS Manager decides the QoS services available for each component. These actions only take place if the QoS of applications is not respected.

Figure 20 further details the QoS Manager depicting its inputs and outputs. The Middleware Interface interacts with the middleware to collect metrics and send updates. It allows the collection of metric measurements from components and applications. Subsection 4.4.3 describes in details all input variables the Manager receives. The Middleware Interface also provides

Figure 18: HealthStack's closed-loop model with a Manager in charge of monitoring and adapting the middleware according to the workload. From the user perspective, a denotes the number of user applications. From the middleware perspective, n denotes the number of nodes running a Collector instance acquiring data from sensors.



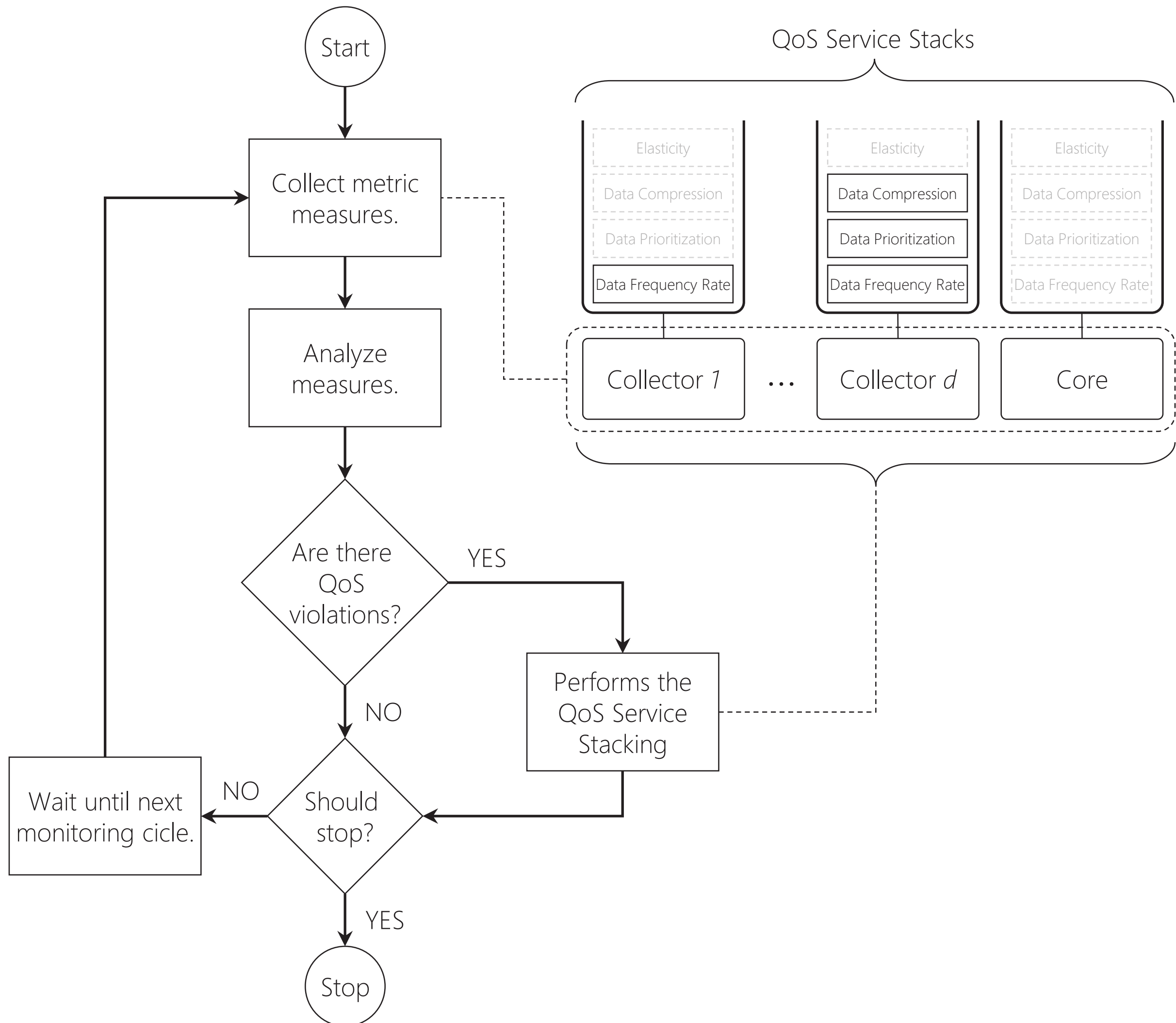
Source: elaborated by the author.

methods to send configuration updates to change parameters from the middleware components to activate QoS services. All input and output messages respect the communication protocol defined in Subsection 4.3.1. In particular, it allows receiving messages type 2 and sending messages type 3.

Following, the Metric Monitoring is in charge of collecting each QoS metric measurement periodically at a given time interval. Thus, the main component, called Dynamic Service Stacking, analyzes these measurements. It contains the main strategies the Manager applies to adapt the middleware QoS service stacks. Data and Performance Analyzer evaluates the middleware's QoS metrics, comparing them with QoS requirements to generate violation events. Thus, as the name suggests, the Service Stacking Definition defines the proper QoS service stacks for each component. After defining the QoS services, the Engine calls the Service Provider component to deliver the needed QoS services. This component can call either the Middleware Interface to send parameters or the Resource Management.

Algorithm 1 details the Manager's operations and procedures that occur periodically. The procedures from lines 4 and 5 collect data from the middleware and compute the QoS metrics. The qos_service_stacking() computes the service stacks for each component according to the $qos_metrics$. This procedure defines the QoS services the Service Provider must deliver to

Figure 19: QoS Manager main monitoring cycle. The idea is to monitor the components' metrics and organize QoS services according to the measurements. The Manager collects metrics from Core and Collector instances and organizes their QoS service stacks when QoS violations occur for user applications.



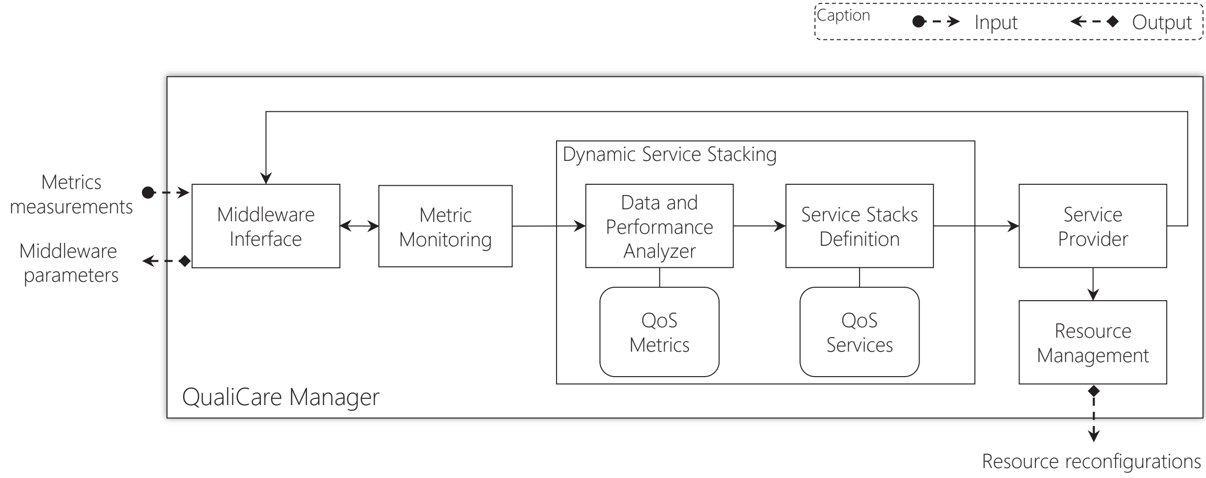
Source: elaborated by the author.

tackle QoS violations through the procedure provide_services(). The monitoring cycle ends in the procedure sleep(), which blocks the process in a given time interval.

4.4.2 QoS Services

The QoS Service Stack is in charge of transparently change QoS services from the Collector. The stack can receive four different QoS services: (*i*) Data Prioritization; (*i*) Data Frequency Rate; (*i*) Data Compression; and (*i*) Resource Elasticity. These services aim at improving the Collector capabilities to provide sensor data quickly to the Core. Hence, improving the performance of applications. Table 8 shortly summarizes the services, and the next subsections describe them individually.

Figure 20: A look inside the QoS Manager. The component receives measurements from the middleware components and evaluates them for decision-making. The output is QoS service changes in the components' stacks, if necessary, to guarantee QoS.



Source: elaborated by the author.

Algorithm 1 QoS Manager main tasks.

Input: $component_addresses[]$, $\&running_flag = true$, $monitoring_interval$

Output: QoS service stacks adaptations.

1: $cycle \leftarrow 0$;
2: **while** $running_flag$ **do**
3: $measurements[] \leftarrow$ collect_monitoring_data($component_addresses[]$);
4: $qos_metrics[] \leftarrow$ compute_metrics($measurements[], cycle$);
5: $qos_service_stacks[][] \leftarrow$ qos_service_stacking($qos_metrics[], qos_services[]$);
6: **if** $qos_service_stacks > 0$ **then**
7: provide_services($qos_service_stacks[][]$);
8: qos_service_stacks.clear();
9: **end if**
10: sleep($monitoring_interval$);
11: $cycle + +$;
12: **end while**

4.4.2.1 Data Prioritization

The model allows data classification through an additional 1-byte field in the message header (see Subsection 4.3.1). This flag defines the package classification as high priority or low priority, but more can be included. The Manager delivers this QoS service by defining the flag from

Table 8: QoS services short description.

QoS Service	**Description**
Data Prioritization	Packet priority differentiation.
Data Frequency Rate	Sampling rate adaptation.
Data Compression	Lossless data compression.
Resource Elasticity	Computing resources provisioning.

Source: elaborated by the author.

packages of a particular Collector instance to high priority. This QoS service can not be activated for more than one Collector simultaneously. It works like a token, and only one Collector can hold it at a time. Therefore, two or more Collectors can not produce high priority packages simultaneously. According to the flag, incoming messages in the Core are scheduled into two different queues. For each queue, a time slice factor (TSF_q) defines the proportion of the time that data from the queue q will be consumed. For instance, if the high priority queue has a factor (TSF_h) of 0.7 (70%) the low priority queue factor (TSF_l) is 0.3 (30%). At each package processing time, the core chooses one of the queues based on a lottery strategy (SHARMA; ADARKAR; SENGUPTA, 2003; WALDSPURGER; WEIHL, 1994) in which it generates a random value between the interval $[0, 1]$. If the value falls between the interval $[0, TSF_h]$, the core consumes a package from the high priority queue. Otherwise, it consumes a package from the low priority queue. Such a strategy aims at reducing the risk of starvation in the low priority queue since high priority packages can arrive at a higher rate than low priority packages.

4.4.2.2 Data Frequency Rate

Collector instances send data to the Core at different frequencies according to their configurations. For each Collector, fps defines the number of frames per second to send to the Core. The minimum value for this parameter is 1 Hz, and the maximum depends on the sensor capacity. The higher the fps, the higher the rate of transmitted data. Therefore, changing this parameter causes effects on the sensor's throughput and its bandwidth requirements. To stack the Data Frequency service, the Manager can adapt the individual fps of each Collector on-the-fly. In summary, when this service is active, the Collector's fps is 1 Hz.

4.4.2.3 Data Compression

This service consists of enabling or disabling lossless data compression in the Collector instances. The strategy aims at reducing the amount of data transmitted between the Collector instances and the Core to avoid network overload situations. At any time, the Manager can change the Collector configurations stacking this service. It happens through a flag $compression$ in the

Collector configuration. Before transmitting the data to the Core, the Collector checks the status of $compression$. In case it is true, the Collector compresses the data payload for transmission. Network packet headers contain a specific flag that defines whether the payload data from the message is compressed (see Subsection 4.3.1 for more details in the packet header).

4.4.2.4 Resource Elasticity

Elasticity is a popular concept in cloud platforms, which refers to a system's capacity to adapt resource provisioning according to workload changes automatically (GALANTE; BONA, 2012). Modules from the architecture can be deployed in VM instances allowing vertical elasticity. In this elasticity model, the Manager resizes VM instances by increasing or decreasing allocated resources (e.g., CPU and memory). Physical machines can run one or more VMs encapsulating modules from the architecture. The amount of resources each VM receives depends on the physical machine's available resources and the number of VMs sharing the same physical machine. For instance, a particular physical machine can host only one VM, and the allocated share of resources to this particular VM can vary up to 100%. In a different scenario, a particular physical machine can host two or more VMs, and these VMs share the available resources dynamically. This service provides resource scalability in the infrastructure resources to improve the system capacity according to user application demands. Overload situations require a large number of resources during periods. However, in periods in which high demand for resources is not required, the allocated resources can be released, improving the system energy efficiency (MORENO; XU, 2011).

4.4.3 Definition of Input Variables

The QoS model is one of the QoS Manager component's attributions, which employs a control loop monitoring strategy, as aforementioned. To do these tasks, it monitors several metrics, here, called input variables. Table 9 describes all variables the Manager collects and uses as input in the QoS Service Stacking model. At the application's side, the two variables $delay_a$ and $jitter_a$ measure the current values for delay and jitter that the middleware delivers for application a. In the Core scope, three variables measure the usage load of CPU, memory, and network resources: cpu_{d+1}, mem_{d+1}, and net_{d+1}. The index uses a plus one since the architecture comprises d nodes, one for each Medical Data Collector, and an additional Core server. Finally, Collectors have seven different variables since they are responsible for acquiring and packing samples from sensors. As for the Core, the first three variables regard computational resources. Next, $delay_d$ and $jitter_d$ correspond to Collector instance d regarding the arrival of its samples at the Core. The remaining two variables, $conn_d$ and pri_d, define how many applications consume data that Collector d generates and the sensor priority level parameter for this Collector.

Table 9: Input variables the QoS Manager collects to employ the QoS Service Stacking.

Source	**Variable**	**Description**
Application	$delay_a$	Time interval between the instant the application a receives the sample and the time it was sampled by the Collector.
	$jitter_a$	Time interval between two consecutive samples' reception.
Core	cpu_{d+1}	CPU usage level of the Core instance.
	mem_{d+1}	Memory usage level of the Core instance.
	net_{d+1}	Network usage level of the Core instance.
Data Collector	cpu_d	CPU usage level of the Collector instance d.
	mem_d	Memory usage level of the Collector instance d.
	net_d	Network usage level of the Collector instance d.
	$conn_d$	Number of applications consuming data generated by the Collector instance d.
	pri_d	Priority value for the sensor device from the Collector instance d.
	$delay_d$	Time interval between the instant the Core receive the sample and the time it was sampled by the Collector instance d.
	$jitter_d$	Time interval between the receive of the last two consecutive samples by the Core from the Collector instance d.

Source: elaborated by the author.

Resource management based on periodic monitoring is a common strategy for web and cloud applications (CHIU; AGRAWAL, 2010; IMAI; CHESTNA; VARELA, 2012). Thus, the QoS Manager collects and evaluates each variable periodically in a monitoring cycle c at a given time interval t. At each monitoring cycle, the Manager collects all metrics from the middleware and, for each one, computes its corresponding input variable employing the Aging concept (TANENBAUM; VAN STEEN, 2007). Except for the $conn_d$ and pri_d variables, the Manager applies a Simple Exponential Smoothing (SES) (HYNDMAN; ATHANASOPOULOS, 2018) filter to the raw readings of the variables. This strategy aims at smoothing the values to reduce noise that might occur in the variable samples. Peaks can lead the decision-making to perform operations incorrectly. For instance, Figure 21 depicts a given metric, monitored over time, and its smoothed value. Highlighted points represent situations in which aging avoids wrong actions based on detached peaks. We employ this strategy mostly to avoid stacking services for false-positive QoS violations that might occur due to outlier measurements.

Equation 4.2 presents how the Manager calculates SES for each variable. $V(m, c)$ returns the value of the variable m in the monitoring cycle c. SES is based on the Simple Exponential Smoothing method, which assigns a higher weight for recent readings (HERBST et al., 2013). Let $SES(m, c)$ be the smoothed value of the metric m in the monitoring cycle c. $SES(m, c)$ computes the aging value for the metric m, which is one of the variable collected from the

Figure 21: Example of a monitored metric and its calculated aging. The figure highlights two specific points in which the raw value of the metric exceeds a QoS limit. In these points, the calculated aging smooths these values resulting in no violations.

Source: elaborated by the author.

middleware components presented in Table 9 (except for the $conn_d$ and pri_d). Therefore, each variable has its value processed before being used in the evaluation.

$$SES(m,c) = \begin{cases} \frac{V(m,c)}{2} & if\ c = 0 \\ \frac{SES(m,c-1)}{2} + \frac{V(m,c)}{2} & if\ c \neq 0 \end{cases} \tag{4.2}$$

For instance, let $c = 6$, to compute the variable $delay_d$ for a given Medical Data Collector 1, the Manager makes the following call: $SES(delay_1, 6)$. To compute this value, let $V = \{38, 25, 22, 23, 25, 20\}$ be the last six samples of $delay_1$ (20 is the oldest and 38 is the newest). SES is achieved the same as $\frac{38}{2} + \frac{25}{4} + \frac{22}{8} + \frac{23}{16} + \frac{25}{32} + \frac{20}{64}$, which results 29.9. The peak in the last observation is smoothed according to the values of the other most recent samples.

4.4.4 Dynamic QoS Service Stacking

Given all input variables, the Manager employs a Service Orchestration process to identify QoS violations and perform adaptations in the middleware. This process's primary goal is to select a Collector instance at each monitoring cycle and stack QoS services if there is at least one QoS violation at the applications' side. At first glance, it seems trivial to select the Collector acquiring the data the application is consuming. However, in scenarios where many applications have QoS violations, selecting a single Collector is challenging since they may be consuming data from different Collectors. To address this issue, HealthStack proposes the Potential of Adaptation (PA) metric to compute the probability of selecting a given Collector. PA is modeled as an artificial neuron based on the concepts of artificial neural networks. It employs the sigmoid neuron, which uses the sigmoid activation function to produce its output (NIELSEN, 2015). The sigmoid neuron takes several input variables, $X = \{x_1, x_2, ..., x_n\}$, weights, $W = \{w_1, w_2, ..., w_n\}$, and a bias value, b. Its output is produced by $\sigma\,(W \cdot X + b)$,

where σ is the logistic sigmoid function, and is defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

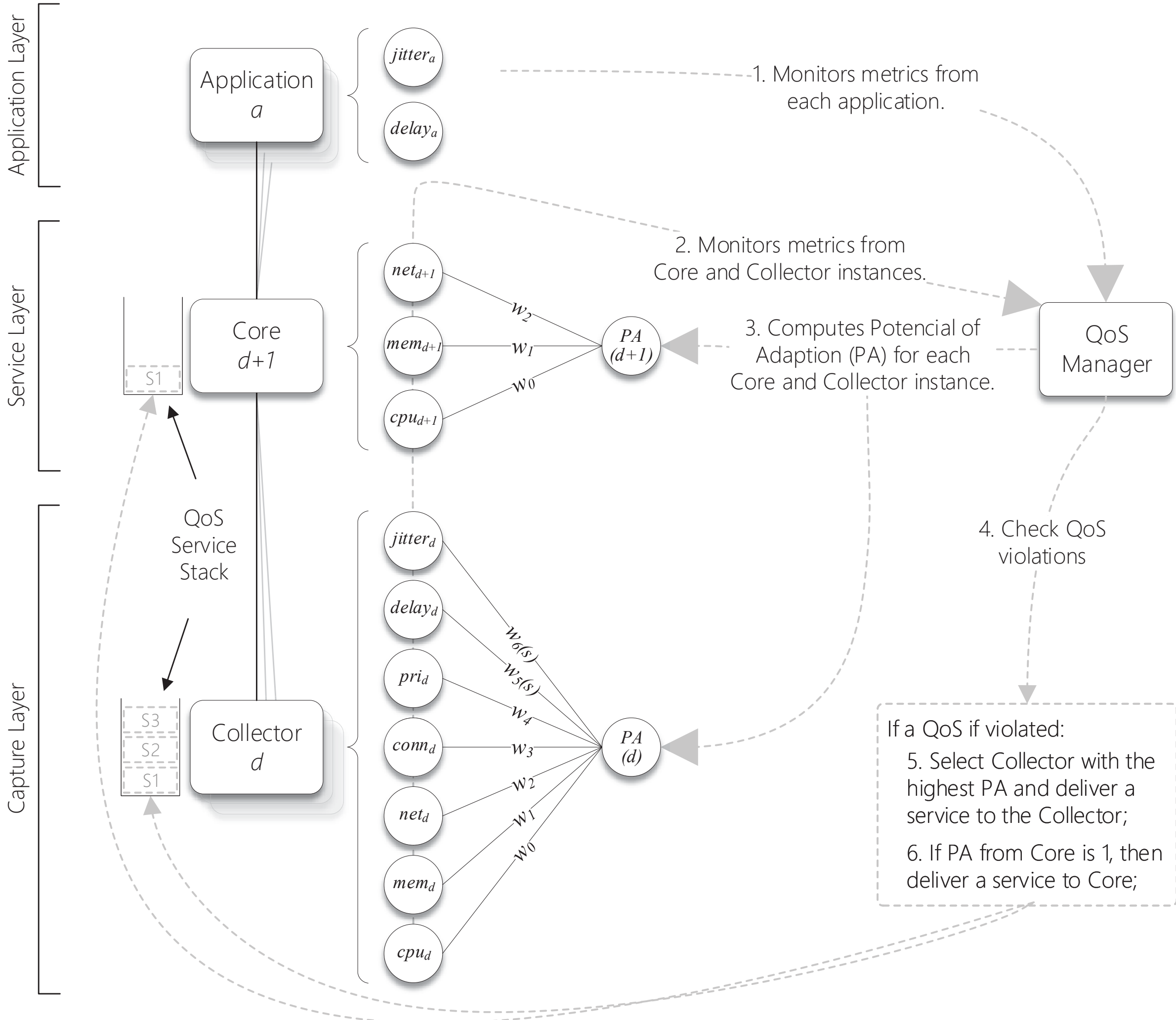
Figure 22 illustrates the overall QoS model process. The Manager computes PA for all instances and verifies if there are current QoS violations. If true, it selects the Collector with the highest PA and stacks a new QoS service for it. Also, the Core PA indicates the computing resources load from the Core instance. A Core PA equal to one indicates a high computing load, and, in this case, the Manager also stacks a new service for the Core, if possible. Accordingly, Equation 4.4 defines PA. It takes all measurements as input variables ($i_0,\ i_1,\ \ldots,\ i_j$) and corresponding weights ($w_0,\ w_1,\ \ldots,\ w_j$) combining them in a weighted sum $\sum_0^j w_j i_j$. The final result is achieved by applying the sigmoid activation function to the resulting value. The Collector with the highest PA is the one with a higher probability of having its services adapted. The main contribution of this strategy relies on the weights applied to the delay and jitter input variables. While traditional artificial neurons have fixed weights for all input values, PA has two weights adapted on-the-fly according to QoS values' measurements. It allows giving higher weights to Collectors that produce data to applications with QoS violations.

$$PA(d) = \sigma \begin{pmatrix} & w_0 & \times & cpu_d \\ + & w_1 & \times & mem_d \\ + & w_2 & \times & net_d \\ + & w_3 & \times & conn_d \\ + & w_4 & \times & pl(d) \\ + & w_5(d) & \times & delay_d \\ + & w_6(d) & \times & jitter_d \\ + & 1 & \times & b \end{pmatrix} \tag{4.4}$$

According to Equation 4.4, $PA(d)$ computes the PA value for the Collector instance d. In the equation, the weights w_0, w_1, w_2, w_3, and w_4 are fixed parameter values. In turn, functions $w_5(d)$ and $w_6(d)$ compute the corresponding weights to be applied to $delay_d$ and $jitter_d$ inputs. Differently from all metrics multiplied by a given weight, for the input variable pri_d, the function $pl(d)$ computes a priority level based on all sensor priorities. In the equation below (Equation 4.5), $pl(d)$ from the Collector d is the result of dividing its priority value pri_d by the sum of the priority values from all d Collectors' sensors. This equation results in a value in the interval (0,1]. The sum of all priority values of all Collectors is always equal to 1.

$$pl(d) = \frac{pri_d}{\sum_{i=1}^{d} pri_i} \tag{4.5}$$

Figure 22: A look at the whole QoS model operation. The QoS Manager performs three main monitoring tasks: (1) monitors applications' metrics; (2) monitors Core, and Collector instances; (3) computes PA for Core and Collector instances; and (4) manages service stacks from Core and Collector instances.



Source: elaborated by the author.

Before further explaining the mathematical expressions to compute $PA(d)$, $w_5(d)$, and $w_6(d)$, it is worth taking the time to define some matrix notations to be used. That is important since some operations can be visualized as matrix operations, which may ease the understanding. Let M be a generic matrix as follows:

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1c} \\ m_{21} & m_{22} & \dots & m_{2c} \\ \dots & \dots & \dots & \dots \\ m_{r1} & m_{r2} & \dots & m_{rc} \end{bmatrix}$$

Following the notation from Bernstein (2009), $M_{[r,\cdot]}$ denotes the submatrix of M obtained by retaining the row r and all columns. $M_{[\cdot,c]}$ denotes the submatrix of M obtained by retaining

the column c and all rows. For instance, given a row $r = 1$, the submatrix $M_{[r;\cdot]}$ considering the total of columns $c = 3$, is:

$$M_{[1;\cdot]} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \end{bmatrix}$$

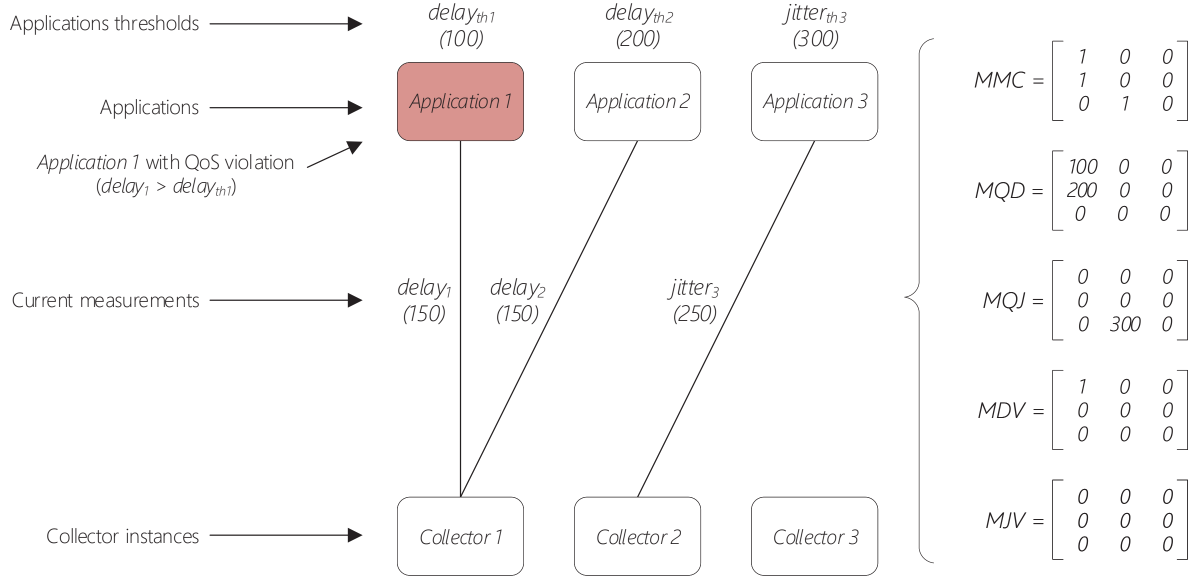
That being defined, we can now map the relations between all a applications and all d Collector instances using matrices. Therefore, HealthStack defines five relation matrices with equal dimensions $a \times d$ ($rows \times columns$): MMC (Matrix of Middleware Connections); MDV (Matrix of Delay Violations); MJV (Matrix of Jitter Violations); MQD (Matrix of QoS Delay); and MQJ (Matrix of QoS Jitter). Respectively, a and d represent the number of connected applications and the number of Collectors in the architecture. New application connections and disconnections respectively increase and decrease the number of rows of the matrices. Changes in the number of sensors increase or decrease the number of columns of the matrices. For each matrix, M_{ij} represents the element corresponding to the relation between the application i and the Collector j.

The values that are stored in each matrix vary according to the matrix. MMC stores binary elements representing if a given application i is consuming data from a given Collector j. As the model comprises two QoS policies (delay and jitter), the matrices MQD and MQJ store delay and jitter threshold values, respectively, for each relation between all application connections and Collectors. Their elements are positive integer values if a threshold is defined. Otherwise, the non-set value corresponds to 0. Additionally, MDV and MJV, likewise MMC, also store binary elements, but in their case, they represent the application QoS violations respectively for delay and jitter. For instance, given an element $mdv_{ij} = 1$, the delay QoS of the application i, regarding data generated by Collector j, is currently being violated.

Figure 23 illustrates an example of relations between applications and Collectors and the resulting matrices. According to the figure, applications 1 and 2 are consuming data produced by the Collector 1, and application 3 from Collector 2. The first two applications have delay QoS policies with threshold values 100 ms and 200 ms, respectively. Differently, application 3 has a jitter QoS policy with a threshold equal to 300 ms. While the QoS is respected for applications 2 and 3, the same is not valid for application 1 because $delay_1$ (150 ms) is higher than the defined threshold (100 ms). In the figure, the setup results in the relation matrices at the right side of the figure. According to the structure depicted, it results in MMC with binary elements mmc_{11}, mmc_{21}, and mmc_{32} defined as 1. Considering that there is only one QoS violation (delay for application 1 from Collector 1), only the element mdv_{11} is set to 1 in MDV. Two applications define a delay QoS policy resulting in mqd_{11} and mqd_{21} from MQD to have their values accordingly the defined thresholds. Finally, as only one application defines a jitter QoS policy, MQJ has only the element mqj_{32} defined.

Now that we have defined all matrix operations and relation matrices, we can understand how to achieve PA. Returning to the aforementioned Equation 4.4, $w_5(d)$ and $w_6(d)$ are not fixed weights but weight functions. Equations 4.6 and 4.7 demonstrate how these functions are

Figure 23: Matrices from an example scenario with three applications and three Collectors. Red boxes indicate QoS violations. The values $delay_{th1}$, $delay_{th2}$, and $jitter_{th3}$ are the threshold values expected by the applications. The values $delay_1$ and $delay_2$ are current delay measures for applications 1 and 2, while $jitter_1$ is the current jitter measure for application 3.



Source: elaborated by the author.

computed, respectively. They differ only in which matrices they operate to achieve the final result since $w_5(d)$ uses the matrices regarding delay (MDV and MQD) and $w_6(d)$ uses the matrices regarding jitter (MJV and MQJ). By analyzing $w_5(d)$, the equation is the product of two parts. The left part introduces the importance unit function iu (see Equation 4.8). The importance unit represents the percentile importance of a single time unit to a baseline value from an input vector V. In summary, the baseline is the lowest positive value element from a vector. In case the vector does not contain values higher than zero, the importance unit is equal to zero. In the delay scope, iu calculates a single time unit's load from a Collector d regarding the lowest QoS delay active currently. For instance, let the time unit be milliseconds and $delay_d = 50$. With a lowest QoS delay equals to 100, each time unit corresponds to 0.01, $i.e.$, 1%. Thus, in this case, the load of the delay of Collector d is equal to 0.5, $i.e.$, 50%. That would be enough to define the weight value to $delay_d$ in PA. However, that would be the same as considering a weight $w_5 = 1$ to the current load of $delay_d$, which can be achieved by $\frac{delay_d}{lowestQoSdelay}$.

$$w_5(d) = iu(MQD_{[\cdot,d]}) \times \left(1 + \alpha \times \sum_{i=1}^{a} mdv_{id}\right) \tag{4.6}$$

$$w_6(d) = iu(MQJ_{[\cdot,d]}) \times \left(1 + \alpha \times \sum_{i=1}^{a} mjv_{id}\right) \tag{4.7}$$

$$iu(V) = \begin{cases} 0 & if \sum v_i = 0 \\ \frac{1}{\min V^*} & if \sum v_i > 0 \qquad (where \quad V^* = V - \{0\}) \end{cases} \tag{4.8}$$

To address this, now that we have covered the left part of the function $w_5(d)$, the right part of the equation aims at defining the increase factor to be applied in the importance unit. Instead of directly applying the importance unit to the measure, getting the current load, HealthStack firstly increases iu by an increase factor computed according to the number of QoS violations. To do so, HealthStack introduces the adaptation rate parameter α (alpha greek letter), which defines how significant $w_5(d)$ should get for each QoS violation. According to the equation, α is added the number of delay QoS violations regarding Collector d (given by $MDV_{[\cdot,d]}$). The more QoS violations, the higher the $w_5(d)$ result. By achieving the importance unit for a time unit, HealthStack can increase this value according to QoS violations.

For instance, lets $\alpha = 0.1$, if we have just one QoS violation for Collector d, $w_5(d)$ is the result of iu multiplied by 1.1, resulting in the final calculus of PA the current load plus 10% ($1.1 \times delay_d$). Following the example from Figure 23, lets the measured $delay_1 = 150$, lower delay QoS equals to 100, $w_5(d) = 0.01 \times 1.1 = 0.011$, which results in $0.011 \times 150 = 1.65$. In other words 150 represents a load of 150% regarding 100, and 165% is the result of 150% plus 10% ($\alpha = 0.1$). As the final result of $w_5(d)$ is multiplied by the $delay_d$ measure, HealthStack can enhance its value in the final calculus of PA in case we have QoS violations. Otherwise, HealthStack applies only the iu value as the right part of the function returns 1. Lastly, HealthStack also has to consider scenarios in which any applications consume data produced by Collector d. In this case, $delay_d$ is not considered in PA since iu will be 0 and, consequently, $w_5(d)$ will return 0 due to $MQD_{[\cdot,d]}$ being filled with zeros.

The expression $\alpha \times \sum_{i=1}^{a} mdv_{id}$ in Equation 4.6 can be represented as a matrix multiplication of $MDV_{[\cdot,d]}$ by a matrix of one row and a columns populated by α in all elements, which results in a single value:

$$\begin{bmatrix} \alpha & \dots & \alpha \end{bmatrix} \times \begin{bmatrix} mdv_{1d} \\ \dots \\ mdv_{ad} \end{bmatrix} = \begin{bmatrix} \alpha \times mdv_{1d} + \dots + \alpha \times mdv_{ad} \end{bmatrix}$$

The same is valid for the expression $\alpha \times \sum_{i=1}^{a} mjv_{id}$ in Equation 4.7, but in its case using $MJV_{[\cdot,d]}$:

$$\begin{bmatrix} \alpha & \dots & \alpha \end{bmatrix} \times \begin{bmatrix} mjv_{1d} \\ \dots \\ mjv_{ad} \end{bmatrix} = \begin{bmatrix} \alpha \times mjv_{1d} + \dots + \alpha \times mjv_{ad} \end{bmatrix}$$

Finally, we can now represent PA as a vector (one-dimensional matrix) multiplication of a single-row weight matrix by a single-column variable matrix, as presented in Equation 4.9. The first matrix contains all weights and weight function results distributed in one row and c

columns. On the other hand, the second matrix contains all input variables organized in r rows and one column to match the corresponding weights in the multiplication. It is important to note that the number of columns of the first matrix matches the second matrix's rows ($c = r$).

$$PA(d) = \sigma \left(\begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5(d) & w_6(d) & 1 \end{bmatrix} \times \begin{bmatrix} cpu_d \\ mem_d \\ net_d \\ conn_d \\ pl(pri_d) \\ delay_d \\ jitter_d \\ b \end{bmatrix} \right) \tag{4.9}$$

Simplifying, considering W as the matrix of weights and I the matrix of input variables, we get:

$$PA(d) = \sigma \left(W \cdot I + b \right)$$

4.5 Summary

HealthStack is a QoS-aware real-time middleware that focuses on hospital facilities. The middleware collects data from sensors, stores it in a database, and delivers it to user applications meeting QoS requirements. Its main characteristic is its ability to provide QoS for both user applications and sensors according to the system load. To connect sensors with user applications, HealthStack introduces four distributed components that have specific roles in the middleware: (*i*) Data Core Service (Core); (*ii*) Medical Data Collector (Collector); (*iii*) Data Access Wrapper; and (*iv*) QoS Manager. These components can exchange three different messages: (*i*) configuration data; (*ii*) metrics measurements; and (*iii*) sensor data. The middleware requires that all messages arrive at their destination without errors. Therefore, HealthStack employs the TCP/IP protocol for all network transmissions. The Data Access Wrapper plays an essential role in the middleware's communication process on the applications side. It works as a wrapper of the publish-subscribe communication protocol allowing the programmer to use the traditional calls without changing the application. Because of the Wrapper, the application does not need to connect to the sensor data sources directly. Instead, it only needs to access the Core through the Wrapper to consume data from many sensors in a standard data format.

The Core is the central component that processes incoming data from sensors and dispatches them to both the database and the PubSub Broker. The middleware comprises d Collectors corresponding to the number of sensor data sources. Two parameters define each Collector's behavior: pri_d, the priority value of the attached sensor, and fps, the number of data samples

per second to extract from the sensor. On top of all components, the QoS Manager monitors several metrics and performs the QoS Service Stacking process. Its main goal is to address QoS violations by stacking QoS services for the middleware components. The QoS Manager implements a closed feedback-loop architecture, and it has access to each HealthStack component instance regardless of their locations among servers or clusters. The QoS Manager presents three primary functions that characterize control systems: a sensor to acquire monitoring data, a controller to evaluate measurements, and an actuator to provide modifications and services. The Manager monitors several middleware metrics and applications and delivers different services for each component by performing the QoS Service Stacking strategy. In particular, the HealthStack components have individual dynamic service stacks that might change over time, according to the QoS Manager decisions. At each cycle, the QoS Manager decides the services that should be available for each component. The stack can receive four different QoS services: (*i*) Data Prioritization; (*i*) Data Frequency Rate; (*i*) Data Compression; and (*i*) Resource Elasticity.

The QoS Service Stacking strategy works on several monitoring metrics. At the application's side, the two variables $delay_a$ and $jitter_a$ measure the current values for delay and jitter that the middleware delivers for application a. In the Core scope, three variables measure the usage load of CPU, memory, and network resources: cpu_{d+1}, mem_{d+1}, and net_{d+1}. Finally, Collectors have seven different variables since they are responsible for acquiring and packing samples from sensors. As for the Core, the first three variables regard computational resources: cpu_d, mem_d, and net_d. Next, $delay_d$ and $jitter_d$ correspond to Collector instance d regarding the arrival of its samples at the Core. The remaining two variables, $conn_d$ and pri_d, define how many applications consume data that Collector d generates and the sensor priority level parameter for this Collector. Thus, the QoS Manager collects and evaluates each variable periodically in a monitoring cycle c at a given time interval t. Except for the $conn_d$ and pri_d variables, the Manager applies an SES filter to the variables' raw readings.

Given all input variables, the Manager employs the QoS Service Stacking strategy to identify QoS violations and perform the middleware adaptations. This process's primary goal is to select a Collector instance at each monitoring cycle and stack services if there is at least one QoS violation at the applications' side. However, in scenarios where many applications have QoS violations, selecting a single Collector is challenging since they may be consuming data from different Collectors. To address this issue, HealthStack proposes the Potential of Adaptation (PA) metric to compute the probability of selecting a given Collector. It employs the sigmoid neuron, which uses the sigmoid activation function to produce its output (NIELSEN, 2015). Based on all input variables, QoS services are stacked only when QoS violations occur, and the higher PA indicates which Collector should be treated first.

5 EVALUATION METHODOLOGY

This chapter presents implementation details and the environment setup employed to execute experiments. Moreover, it also describes the evaluation scenarios modeled to test the architecture and its features. Hospital 4.0 and IoHT are new areas of study (ACETO; PERSICO; PESCAPé, 2020; COSTA et al., 2018), and, for that reason, it is not possible to compare the current proposal quantitatively against other solutions. However, the experiments consider well-known metrics and QoS requirements commons in health solutions, such as delay and jitter. This research defines different application workloads to cover several scenarios in which applications differ in their QoS requirements.

The next sections detail the evaluation methodology as follows. Section 5.1 presents the details of the implementation and design of the HealthStack prototype. It describes all technical details employed in the development of the solution. Next, Section 5.2 shows the infrastructure in which the middleware is deployed. It also gives details on the hardware installed to perform experiments. In turn, Section 5.3 defines the user application, experimental workloads, and evaluation scenarios. Section 5.4 describes all parameter values in the experiments. Finally, Section 5.5 summarizes the concepts present in this chapter.

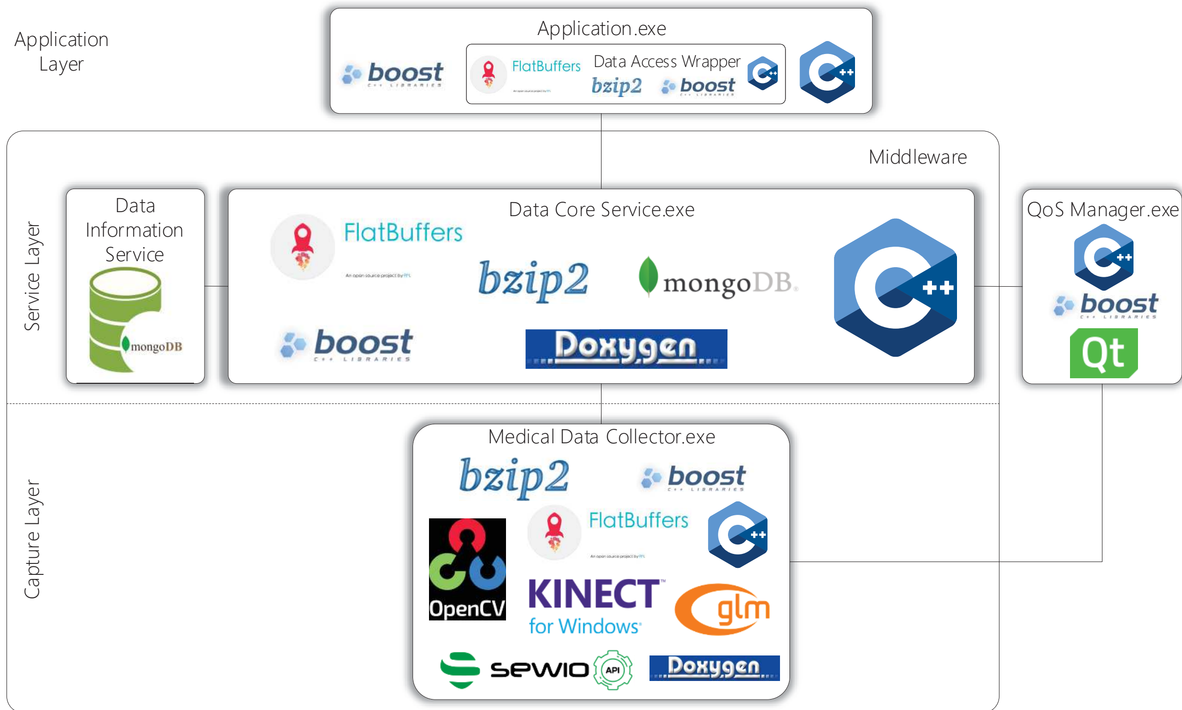
5.1 Prototype Implementation

A complete prototype of HealthStack was developed in C++, covering all components: Core, Collector, Manager, and Application. Figure 24 depicts the modules developed to cover HealthStack architecture. The figure illustrates all technologies the prototype employs for each module. In particular, the prototype supports Microsoft Kinect's API[1], and Sewio RTLS solution[2]. The Collector components use these APIs to extract data from the sensors. The development included third party libraries and software for different purposes, as listed in Table 10. In particular, Flatbuffers provides easy to use data serialization model for data transmission over the network. A publish-subscribe strategy is implemented directly in the Core Client Manager using the communication protocol developed for the Core and Collector instances in the application scope. The protocol is essential for communication between components, and it follows the publish-subscribe paradigm. Therefore, this is extended to the applications, also including the QoS model services.

The Collector prototype implements a data compression strategy to offer the Data Compression service through the Bzip2 library. It uses the Huffman coding (MILLER; VANDOME; MCBREWSTER, 2009) lossless strategy that allows compression of data without losing data. Regarding the Data Information Service, the MongoDB stores configurations and sensor data samples in JSON format. Finally, the source code from the prototype is hosted in a private

[1]https://developer.microsoft.com/pt-br/windows/kinect
[2]https://www.sewio.net/

Figure 24: HealthStack prototype modules and their corresponding technologies.



Source: elaborated by the author.

account on GitLab platform[3]. This account is private since the prototype composes a Unisinos project protected by a confidentiality agreement. However, it is possible to make available the QoS model's code at Github[4]. The repository contains the C++ classes implementing the PA strategy presented in the previous section.

5.2 Infrastructure Setup

Besides having a prototype running at the project's clinical partner, the HealthStack architecture is also deployed in a simulated hybrid operating room in the Software Innovation Laboratory – SOFTWARELAB, at UNISINOS. The room is equipped with advanced imaging instruments, including a Siemens SIREMOBIL Compact L C-Arm[5]. In this laboratory, it is possible to perform several experiments before deploying the software in production at the hospital. The environment is composed of an isolated network to avoid network interference and guarantee high performance. The network equipment is based on a TCP/IP Gigabit Ethernet. Figure 25 depicts the organization of the infrastructure and the distribution of the middleware components. Regarding the hardware, Table 11 lists all equipment employed in the SOFTWARELAB operating room. All equipment are interconnected by the same Gigabit switch to avoid network hops and improve performance. Besides, a server at the university datacenter

[3]https://gitlab.com/
[4]https://github.com/viniciusfacco/healthstack
[5]https://www.healthcare.siemens.com.br/surgical-c-arms-and-navigation/mobile-c-arms/siremobil-compact-l

Table 10: Libraries and software used in the prototype development.

Item	Version	Description	URL
CMake	3.11.0	CMake to build the Visual Studio project.	`https://cmake.org/download`
Boost	1.66.0	Set of libraries and utilities for C++.	`https://sourceforge.net/projects/boost/files/boost`
Bzip2	1.0.6	Data compression library.	`http://www.bzip.org/`
OpenCV	3.4.1	Computer vision libraries.	`https://sourceforge.net/projects/opencvlibrary`
Qt	5.9.1	Cross-platform framework for developing UI in C++.	`https://www.qt.io/download`
GLM	0.9.8.5	Header only C++ library for graphics software.	`https://github.com/g-truc/glm/tags`
Doxygen	1.8.14	Software to generate the documentation.	`http://www.stack.nl/~dimitri/doxygen/download.html`
Kinect v2	2.0 - 1409	Kinect v2 API.	`https://www.microsoft.com/en-us/download/details.aspx?id=44561`
Flatbuffers	1.9.0	Library for data serialization.	`https://github.com/google/flatbuffers`
MongoDB	4.0.5	NoSQL database integration.	`https://www.mongodb.com/`

Source: elaborated by the author.

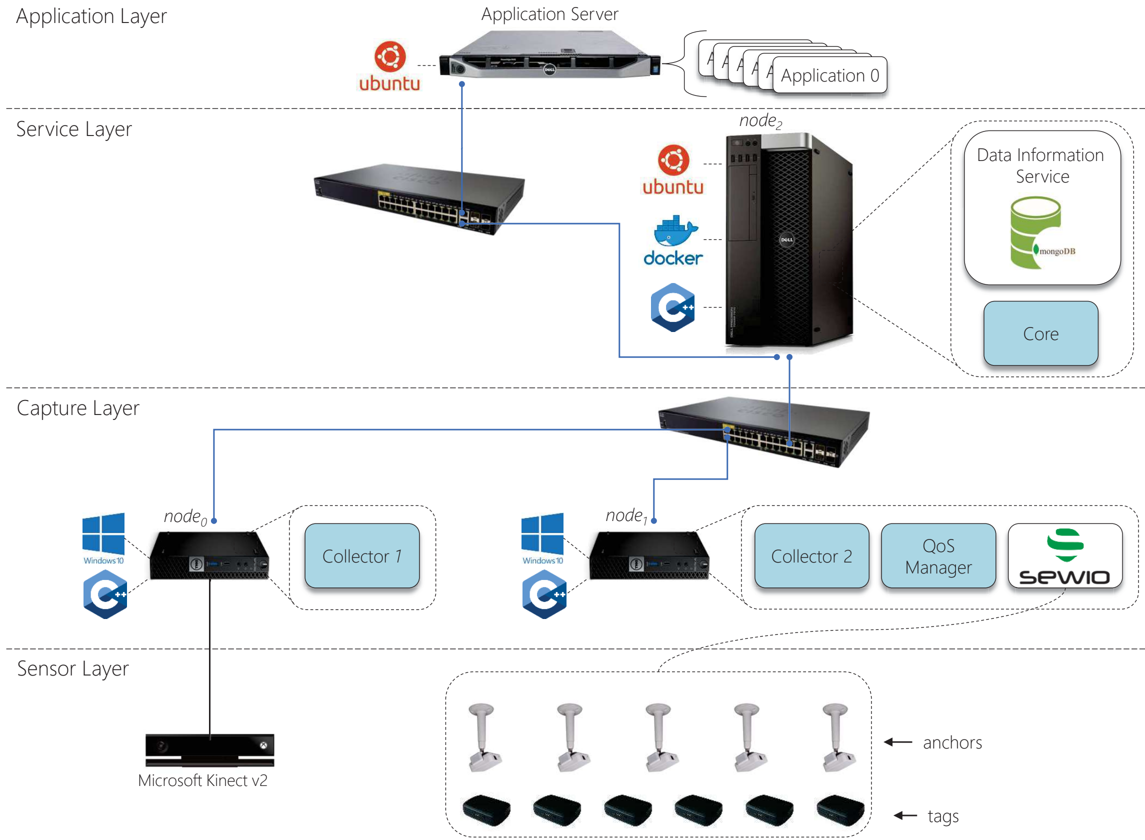
dispatches applications to connect the middleware.

Considering the architecture's components, one Core instance and the MongoDB are deployed in the $node_2$ computer. Also, $node_2$ as Docker[6] installed and MongoDB database is deployed in a container. Additionally, two Collector instances are deployed in two different computers, each one collecting one type of data. On the one hand, in the $node_0$, a Microsoft Kinect version 2 provides depth image data. On the other hand, in the $node_1$, the Sewio RTLS server tracks several UWB tags in the environment. Sewio is an indoor RTLS solution that employs UWB technology to track subjects in real-time. The solution consists of several anchors and tags installed in the room. The anchors are connected via a wired network, and the Sewio server is running on $node_1$ accesses them to compute the tag positions. In $node_1$, a Collector instance accesses the Sewio server through its API to acquire the tag positions. Moreover, although independent from the node of execution, the QoS Manager also runs in $node_1$. It only requires network access to all nodes running the middleware components.

Regarding the deployment of the sensors, Figure 26 depicts the experimental environment in which is installed the Sewio solution and the Kinects' setup. The figure shows two Kinect devices positioned over tripods at the left and right parts of the figure. At the figure's top left and right, two Sewio anchors are attached to the ceiling. Anchors must be positioned at higher positions of the room to improve the RTLS solution accuracy. Finally, Table 12 summarizes all

[6] https://www.docker.com/

Figure 25: Deployment overview of the hardware, middleware, and software components among the evaluation infrastructure.



Source: elaborated by the author.

Table 11: Equipment installed in the environment.

Node	Model	CPU	RAM	Network	Software and Components
$node_0$	Dell Optiplex 3050 Mini	i7-6560U 2.20 GHz	8GB	1 Gbps	• Microsoft Kinect • Depth Collector • QoS Manager
$node_1$	Dell XPS 13 9350	i5-7500T 2.70 GHz	8GB	1 Gbps	• Sewio API • Position Collector
$node_2$	Dell Precision	i7-7820X 3.60GHz	32GB	1 Gbps	• Core • Docker • MongoDB

Source: elaborated by the author.

devices and sensors available for experiments with further technical details.

5.3 Workload Model and Evaluation Scenarios

There is no workload characterization for applications focused on consuming data from medical middlewares in the literature to the best of our knowledge. Therefore, to evaluate HealthStack, it was needed to develop an application to serve as a user application that connects

Figure 26: Experiment laboratory setup in which Sewio RTLS system and Microsoft Kinects are deployed. The laboratory also contains a Siemens SIREMOBIL Compact L C-Arm.

Source: elaborated by the author.

the middleware and requests data. Several instances of this application can start simultaneously anywhere, simulating many users requesting data. To simulate different circumstances, the application has the following parameters:

(*i*) Data type: the type of data the application consumes from the middleware;

(*ii*) QoS type: the target QoS metric the user requires;

(*iii*) QoS threshold: the limit value to be respected for the QoS metric.

The first two parameters have limited options for applications according to what the middleware offers. Therefore, considering that the middleware deployment covers an RTLS system and image cameras, the data type has two alternatives: (*i*) depth image data; and (*ii*) position data. In turn, the QoS type offers two metrics: (*i*) delay; and (*ii*) jitter. Thus, each new application instance provides one of these alternatives for these parameters. The third parameter, the QoS threshold, does not have limited options since this is a user's decision. Each user application can have particular QoS requirements, and they can differ from each other. In particular, this last parameter refers to the limit for delay or jitter the user requires, depending on the second parameter's choice.

Given the three parameters, by starting several application instances with different values, it is possible to create specific workloads. With that in mind, this study modeled several different

Table 12: Technical details from the sensors installed in the simulated operating room.

Hardware	Description
RTLS Anchors	Compliant with UWB PHY IEEE 802.15.4a Decawave UWB Radio, 6 channels 3-7GHz Dimensions: 70 x 74 x 25 mm Driven by MCU ARM Cortex M4 Configurable via web RTLS Manager Anchor's wireless sync via UWB Ethernet used as a backhaul Firmware upgrade via Ethernet Native web interface For Indoor Use
Li-ion Tags	Compliant with UWB PHY IEEE 802.15.4a Decawave UWB Radio, 6 channels, 3-7GHz Dimensions: 70 x 50 x 21 mm Driven by Ultra Low Power ARM EFM32G M3 Battery included, Li-ion 600mA Configuration via RTLS Manager Firmware upgrade and Charging via USB User LED and Charging LED indication Unique 6 bytes ID
Piccolino Tag	Compliant with UWB PHY IEEE 802.15.4a DecaWave UWB Radio, 6 channels, 3-7GHz Dimensions: 29x37x11 mm Driven by Ultra Low Power ARM EFM32G M3 Coin Battery CR 2450 600mA Configuration Wirelessly via RTLS Manager User LED indication Unique 6 bytes ID
Kinect v2	Depth Estimation Strategy: Time-of-Flight Resolution (width x height): 512 x 424 Field-of-View (horizontal x vertical): 0.50 - 4.5 m Range: 70° x 60° Frames per second: 30

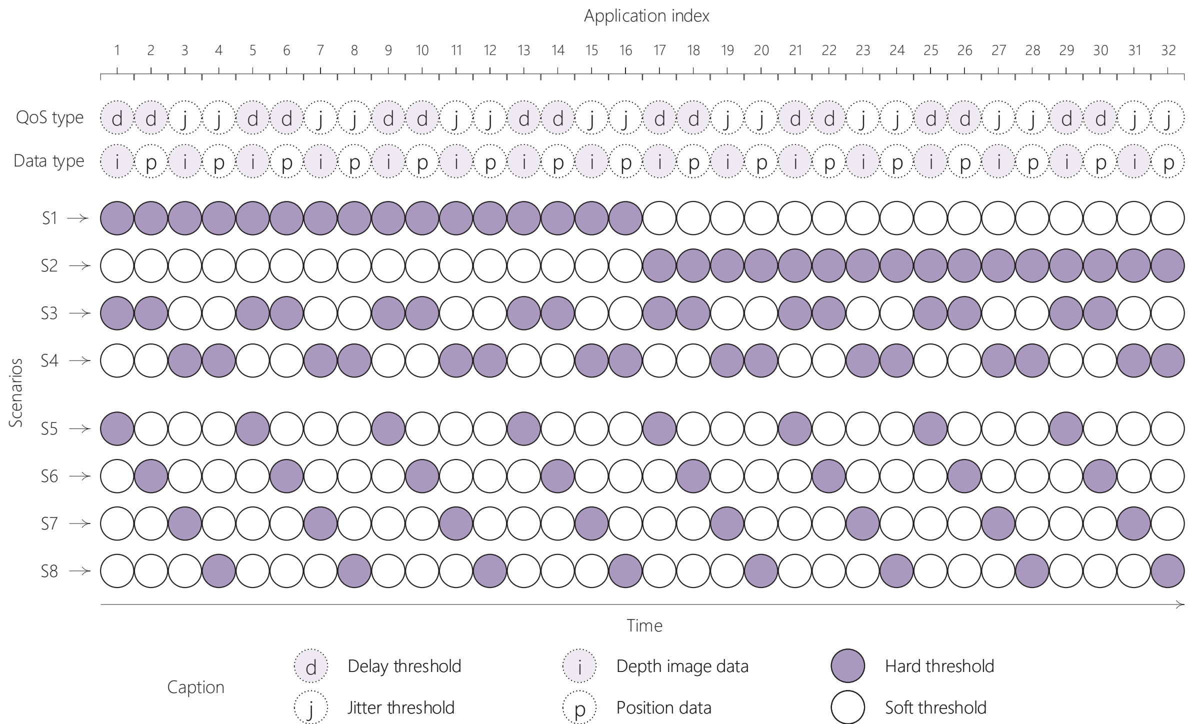
Source: elaborated by the author.

scenarios with varying workload changes to cover a broad set of applications. Eight workload scenarios composed of 32 applications were modeled: S1, S2, S3, S4, S5, S6, S7, and S8. The experiments comprise the execution of each 32-application scenario with and without QoS support to compare the results. The different scenarios intend to simulate distinct situations and, therefore, cover different application setups.

For simplicity, in the results, executions in which QoS support is enabled are referred to as S1', S2', S3', S4', S5', S6', S7', and S8'. For all scenarios, each application instance connects the middleware in different instants and starts requesting data at a given rate. The 32 applications connect to the middleware sequentially, with an interval of 60 seconds between each new connection. Once connected, they remain consuming data until the end of the experiment. Figure 27 depicts eight different workload scenarios and their parameters. The figure presents an application index that refers to the sequence that the application connects to the middleware. As each application connects every 60 seconds, the index also represents the minute it connects.

In all scenarios, the choices of QoS type and data type follow the same sequence. As the figure shows, application 1 defines the data type to depth image data, and application 2

Figure 27: The sequence of application connections in the eight workload scenarios. One application connects at each minute, requesting a Data type, a QoS type, and a QoS threshold. The application index refers to the sequence of the application connection to the middleware.



Source: elaborated by the author.

defines it to position data. This sequence keeps repeating until the last application. The same occurs for the parameter QoS type. However, in this case, the parameter varies in pairs of two. Applications 1 and 2 define the QoS type do delay, while applications 3 and 4 define it as jitter. The following applications follow the same pattern.

Finally, the third parameter is primarily essential to create different workloads. The QoS Manager considers this value in its decision-making process. Therefore, lower thresholds represent more necessity for QoS and, consequently, higher workloads. For that reason, for each QoS metric, two values represent high and low requirements. Here, the lower value is indicated as “Hard threshold,” while the higher is indicated as “Soft threshold.” The definition of the applications QoS values is based on the literature and which QoS values other authors use for applications in the healthcare scope (MUKHOPADHYAY, 2017; NANDA; FERNANDES, 2007; MALINDI; KAHN, 2008; LEE et al., 2011). Considering the values employed by the literature, the two threshold values for each QoS type are: (*i*) for delay, 100 ms (hard) and 300 ms (soft); and (*ii*) for jitter, 25 ms (hard) and 50 ms (soft).

As the QoS threshold has such importance in the workload definition, it is essential to understand the different scenarios with them in mind. For that reason, Figure 28 depicts the eight scenarios showing the number of connected applications with each QoS threshold over time. All scenarios present an ascending number of applications. However, some scenarios have more ap-

plications with Hard threshold requirements than others. For example, in S1, S2, S3, and S4, by the end, the total number of applications with Hard and Soft threshold is equal to 16. On the other hand, in S5, S6, S7, and S8, in the end, the total of applications with Soft threshold is 24, and Hard threshold is 8. It is important to analyze this particular figure together with Figure 27. Although S5, S6, S7, and S8 present similar graphs, they shift in which applications define their QoS threshold to Hard and Soft.

5.4 Parameters

Some technical decisions are imposed on the implementation of the model prototype. Table 13 presents the values used for each parameter in all experiments. The “Collection interval” is the time interval for the Manager to collect variables from applications and middleware components. The Collection interval is set at one second to ensure that the Manager receives a significant number of samples to use in the QoS Service Stacking process. Combined with this parameter, the “Monitoring interval” is the time interval for the Manager to execute the middleware status evaluation and QoS Service Stacking. A fine-grained monitoring interval can reduce the SLA violation rate in large scale infrastructures (TAN; VENKATESH; GU, 2013). That is why the “Collection interval” is defined as the lowest value possible. As the Manager employs SES to all measures, the Monitoring interval is set to six seconds, which means that during this interval, there are six new samples of each variable available. With a monitoring window of six observations, the last six values represent 98.4% in the final calculus of SES when the Manager wakes up. In turn, the choice for a cool-down period of 60 seconds considers ten monitoring interval windows. Besides, three QoS services are implemented and they are stacked in the following sequence: (*i*) Data Frequency Rate; (*ii*) Data Compression; and (*iii*) Data Prioritization.

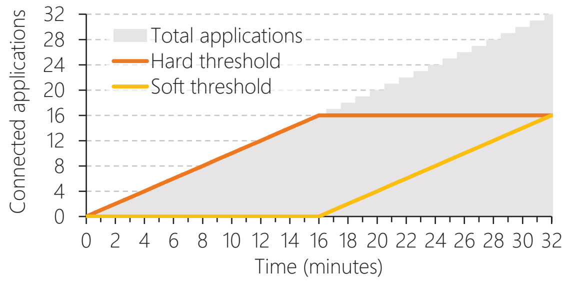
In the QoS Service Stacking strategy, several weights should be defined. Such parameters represent the level of importance of the metrics to which they are applied. A weight equal to 1.0 means that the measurement which multiplies this weight is fully considered. On the other hand, a 0.5 weight considers only 50% of the measurement it multiplies. For instance, let cpu_d of a given Collector d be 0.8, and w_0 be 0.5, when computing PA it adds 0.4 ($w_0 \times cpu_d$) to the final calculus. Defining all five weights could be challenging; however, it depends on which value should receive more importance in the final calculus of PA. In the experiments, the weights were defined as follows:

(*i*) 1 divided among the weights applied to the three resources input variables (w_0, w_1, and w_2). Which means that the values of resources are converted to 33% of the measurement;

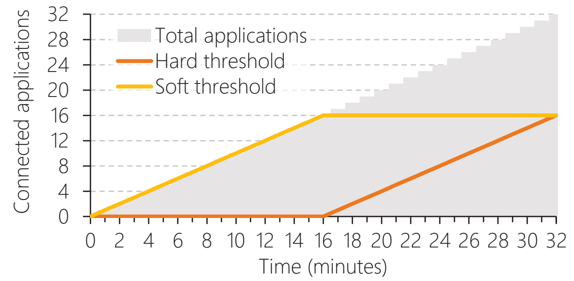
(*ii*) a tenth for the weight applied to the number of connections (w_4), which means that each new connection adds 0.1;

(*iii*) a full unit (1.0) for the weight applied to the priority, therefore, the priority is used as a

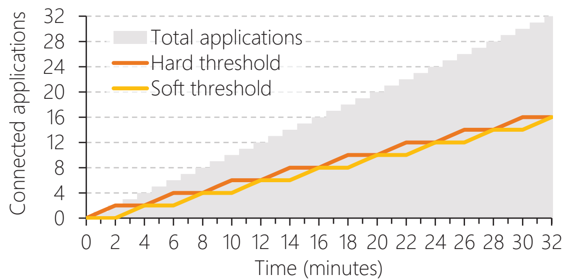
Figure 28: Workload scenarios based on the QoS threshold and the number of connected applications over time.



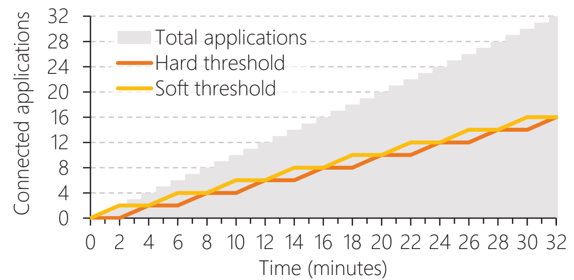
(a) S1



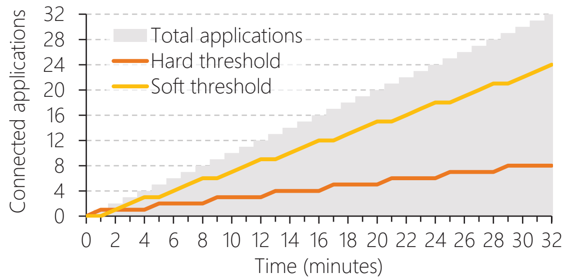
(b) S2



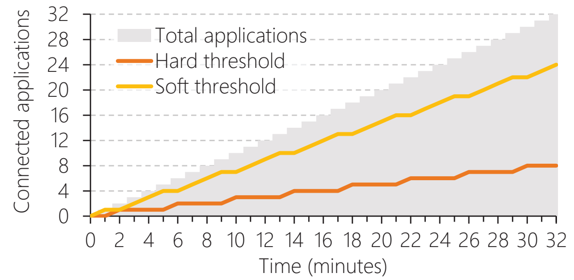
(c) S3



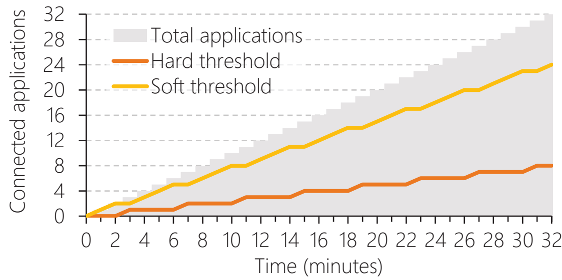
(d) S4



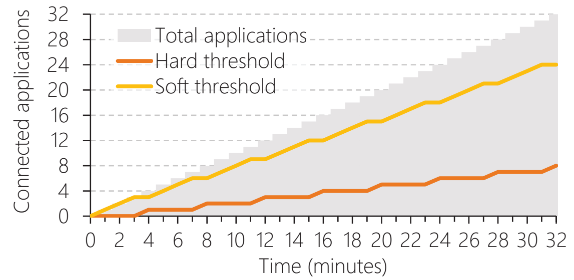
(e) S5



(f) S6



(g) S7



(h) S8

Source: elaborated by the author.

whole (100%) in the calculus of PA;

(iv) half of a unit (0.5) for the adaptation rate in order to produce a considerable increase in the weights $w_5()$ and $w_6()$.

Table 13: Prototype parameters.

Group	Parameter	Value
Monitoring	Collection interval	1 s
	Monitoring interval	6 s
	Cool-down interval	60 s
Service Orchestration	w_0	0.33
	w_1	0.33
	w_2	0.33
	w_3	0.1
	w_4	1.0
	α	0.5

Source: elaborated by the author.

5.5 Summary

This section presented all details related to the development and deployment of HealthStack. Also, it describes the evaluation modeling and the scenarios proposed to test the prototype. In particular, a complete prototype of the model was developed in C++. The deployment of the infrastructure was performed on two different sites. First, a production deployment was performed at a clinical partner that provided access to an actual hybrid operating room. Second, an experimental deployment was performed at a simulated operating room at the university. In the last case, three computer nodes were distributed in the environment interconnected by a Gigabit switch. One of the nodes run the database, the Core, and the publish-subscribe node. The other two run a Collector instance each to acquire data from an RTLS and an image sensor.

In the scope of workloads for evaluation, there is no workload characterization for applications focused on consuming data from medical middlewares in the literature to the best of our knowledge. Therefore, a synthetic application was developed to connect HealthStack and request data providing QoS parameters. Several evaluation scenarios were modeled by defining different scripts to run various application instances with different requirements. Additionally, some technical decisions are imposed on the implementation of the model prototype. That includes the definition of the middleware's parameters. This chapter also defined which parameters were employed in the experiments.

6 RESULTS

This chapter presents the evaluation of all scenarios previously described. Here, all the experiments comprise a 35-minute window of execution to encompass all 32 applications from the evaluation scenarios. To execute each scenario, the middleware components start at time zero, and, at the same time, the Manager starts to record metrics for evaluation. The recording ends when the execution time reaches minute 35. In the experimental scenarios' execution, the first application connection arrives after 60 s. Therefore, the first minute has no applications connected to the middleware. The goal is to analyze the effect in the measurements of the components at each new application connection. As there is one minute between new application arrivals, the total time involved in calculating the final results for each application is different. For instance, for application 1, this time is equal to 34 minutes and, for application 32, three minutes.

The following sections present the results from different points of view. Section 6.1 focuses on the results of the applications.. This section demonstrates the results of the QoS metrics for applications in each scenario. Following, Section 6.2 analyzes the middleware components' resource consumption for each scenario. This section shows the difference in resource requirements for each evaluation scenario. Next, Section 6.3 presents a timeline analysis of metrics from individual applications. Its main goal is to demonstrate the QoS Service Staking strategy's impact on the application's measurements. Based on the insights acquired by analyzing the results, Section 6.4 introduces a second phase of experiments with some modifications. Then, Section 6.5 discusses the main findings of the experiments. Finally, Section 6.6 summarizes the concepts from the current chapter.

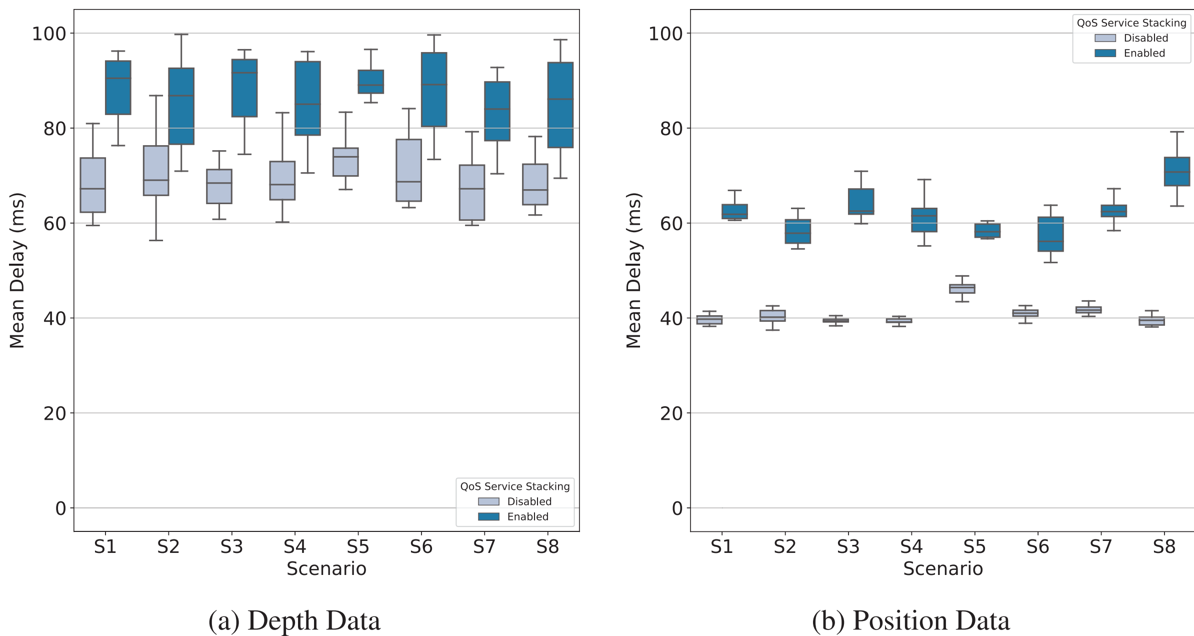
6.1 Applications' Delay and Jitter

This first section focuses on presenting the results from the application's perspective. In this context, the most important metrics are the delay and jitter experienced by the applications throughout their execution time. Note that their values are time units, more specifically milliseconds, and the lower these values are, the better. In order to summarize the results compactly, Figures 29 and 30 present box plots considering, respectively, the delay and jitter measurements of all observations for each application in each scenario. The data was first segmented by the type of sensor to verify differences in the results to produce the figure. Each box plot comprises 16 applications because the experiments include 16 applications requesting depth data and 16 requesting position data.

In the delay scope, Figure 29 shows that scenarios with QoS Service Stacking enabled achieved higher values than the scenarios without it for both data types. More specifically, considering depth data, the mean delay of applications is 86.6 ms for scenarios with QoS Service Stacking enabled, and 69.4 ms for scenarios with it enabled, representing an increase of

24.78%. In turn, considering position data, the mean delay of applications is 61.8 ms for scenarios with QoS Service Stacking enabled, and 40.8 ms for scenarios with it disabled, representing an increase of 51.47%. Even though the results are higher for QoS Service Stacking enabled scenarios, the delay level is still lower than the lower defined threshold of 100 ms, not impacting the application requirements. The diagrams (a) and (b) also show that position data achieves lower delay levels than depth data. That happens because depth data is considerably larger than position data, imposing more complexity to acquire, compress, and transmit.

Figure 29: Delay results considering the two different sensor types: (a) depth and (b) position. Each bar is calculated with the mean delay of each application.



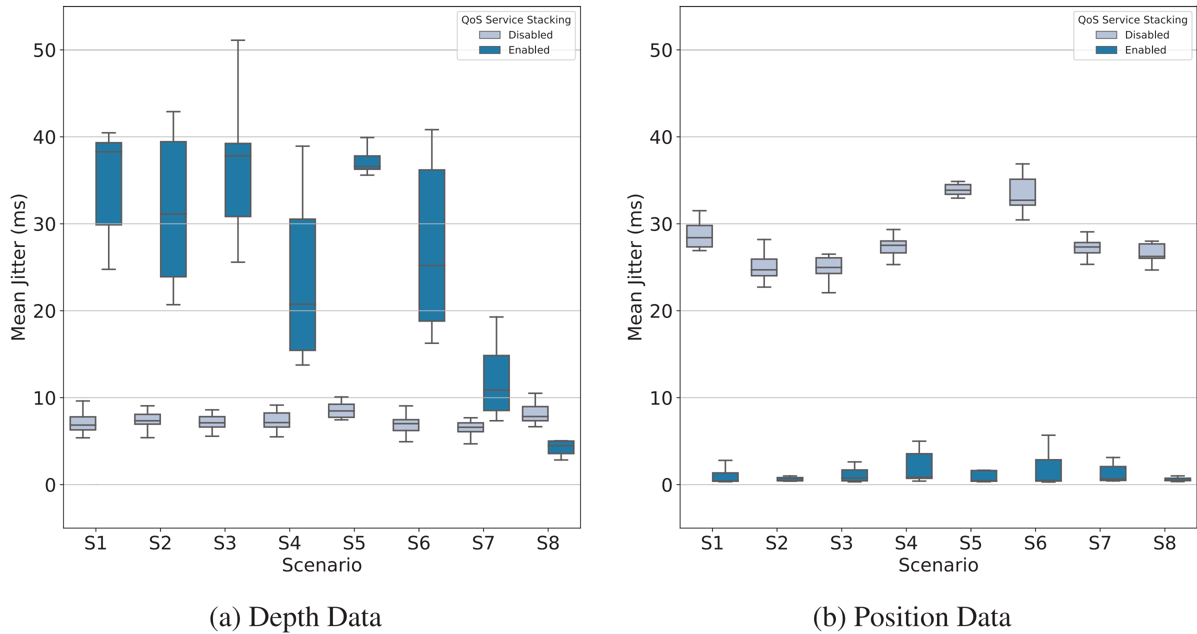
(a) Depth Data (b) Position Data

Source: elaborated by the author.

On the other hand, Figure 30 illustrates the measurements for jitter during data transmission. Each box plot is calculated in the same manner as for the delay metric. The figure leads to different conclusions than the previous one because now the data type has an impact. On the one hand, most scenarios in Figure 30a achieved lower values with QoS Service Stacking disabled. The only exception is the scenario S8'. By further analyzing data logs, in scenario S8' the Collector 1 ($node_0$) received only the Data Frequency Rate service, differently from all other scenarios with QoS Service Stacking enabled. It means that for all other scenarios, Collector 1 received the Data Compression service. This QoS service brought the drawback of inserting overhead in the data processing, which increased the time to deliver it. Averaging the results from applications that request depth data, the mean delay of applications is 26.3 ms with QoS Service Stacking enabled versus 7.5 ms with it disabled, representing an increase of 250%. On the other hand, considering position data, in Figure 30b, all scenarios with QoS Service Stacking enabled achieved lower results than scenarios with QoS Service Stacking disabled. The improvement is considerably expressive since it represents an improvement of 92.3%. More

specifically, the mean of jitter from scenarios with QoS Service Stacking disabled is 28.5 ms, while scenarios with QoS Service Stacking enabled it is 2.2 ms. The main reason is that the Sewio solution does not provide jitter guarantees, and it produces data in a non-steady way. Such improvement is possible due to the Data Frequency Rate service, which, when stacked, stabilizes the frequency of data arrivals at the applications side.

Figure 30: Jitter results considering the two different sensor types: (a) depth and (b) position. Each bar is calculated with the mean jitter of each application.



(a) Depth Data

(b) Position Data

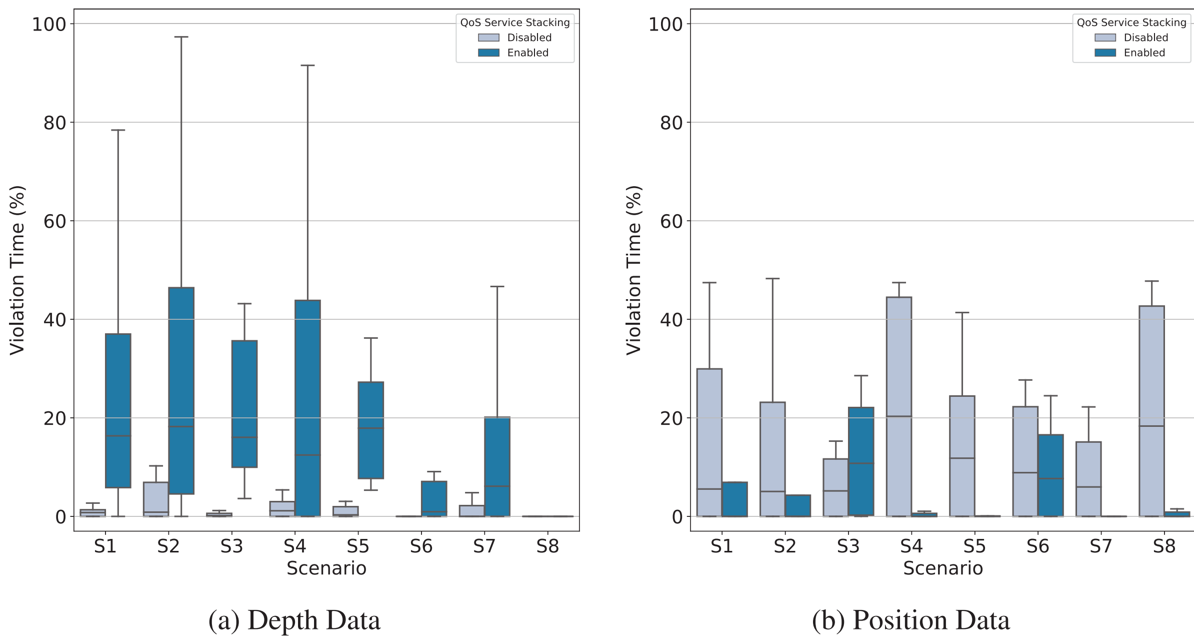
Source: elaborated by the author.

Besides delay and jitter themselves, it is also valuable to look at the applications' QoS violations over time. Therefore, the metric $violation_time$ counts the number of monitoring observations for each application in which the application had its QoS violated. Then, it computes the $violation_proportion$ against all monitoring observations. For instance, if a given application had its QoS violated in 25 observations of 100 total observations, its $violation_proportion$ is 25%. Figure 31 depicts the box plots of this proportion from all applications and scenarios, separating them into the two types of data. It is important to remember that, as each application connects at different times, the total number of observations is different. According to Figure 31a, the first four scenarios presented higher discrepancies than the last four. Besides, most scenarios with QoS Service Stacking enabled have their final results higher than scenarios with QoS Service Stacking disabled. The only exception is scenario S8, which achieved the same result, by 0.2%, as scenario S8. As said before, this is the only scenario in which the Manager did not stack the Data Compression service for Collector 1.

In contrast, Figure 31b shows that most scenarios with QoS Service Stacking enabled achieved lower violation levels, except for scenario S3'. Averaging the results, in 14% of the time, applications from scenarios with QoS Service Stacking enabled have their QoS violated.

On the other hand, in scenarios with QoS Service Stacking enabled, the applications had QoS violations 3.9% of the time, representing an improvement of 72.1%. The different data types' conflicting results demonstrate that the solution performs better for smaller size data types. A higher volume of data requires more computing and network resources, and, mainly, the Data Compression service introduces processing overhead leading to higher delay and jitter values. The next section will provide another point of view in which this particular QoS service may bring advantages.

Figure 31: Proportion of the number of monitoring observations in which the applications QoS was violated. At each observation, for each application, the $violation_time$ increments, and at the last observation, $violation_proportion$ divide results from dividing the $violation_time$ by the total number of observations. The bars represent this proportion of all 32 applications in each scenario.



(a) Depth Data (b) Position Data

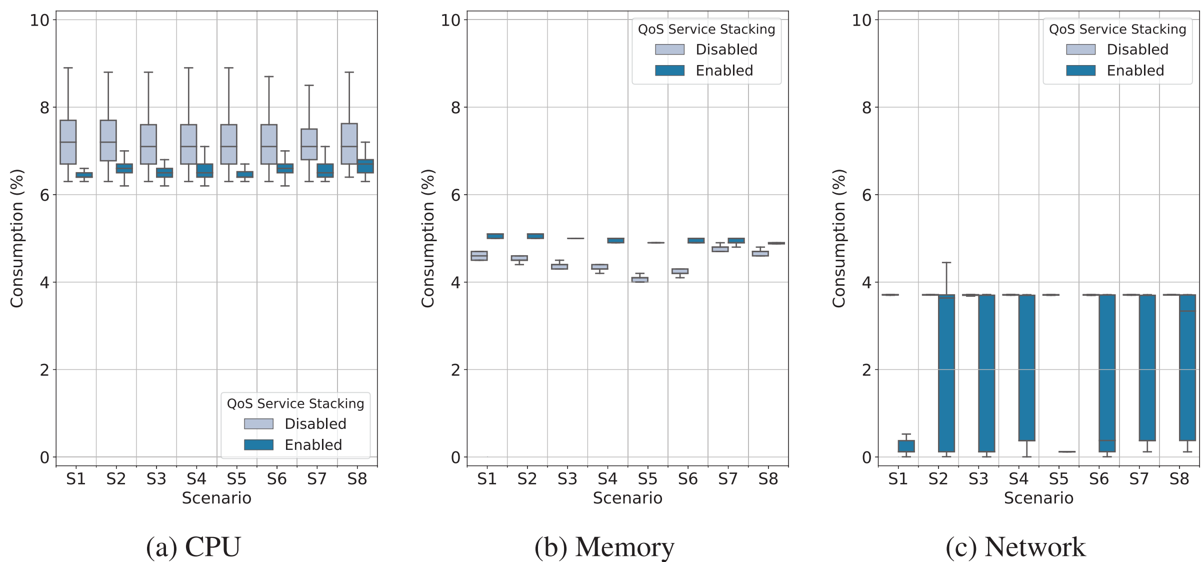
Source: elaborated by the author.

6.2 Resource Consumption

This section analyzes the results from the HealthStack components' perspective. As described in the methodology, the experiments comprise three computers which hosted three main component instances: (*i*) the Depth Collector instance 1 ($node_0$); (*ii*) the Position Collector instance 2 ($node_1$); and (*i*) the Core instance ($node_2$). Here, the analysis first examines the resource consumption profile of each host for different scenarios. Then, it analyzes the outcomes regarding the delay and jitter metrics specifically for each Collector instance. Starting with the Core, Figure 32 shows the mean of resource consumption of the $node_2$ for each scenario. Considering both CPU and network, the consumption is lower in scenarios with QoS Service

Stacking enabled. Averaging all scenarios' results, the CPU and network consumption are 8.3% and 55.5% lower when employing the QoS model strategy. The same is not valid for memory consumption. In this case, on average, the consumption was 25% higher in scenarios with QoS support enabled. By adjusting the amount of data through the Data Frequency Rate service and resizing the data with the Data Compression service, it is possible to decrease the amount of data transmitted and the CPU cycles needed. However, it comes with the cost of increased memory consumption. From all scenarios, the network consumption of scenarios S1' and S5' may draw the reader's attention since they are visibly lower than the other scenarios. That was possible because the Data Frequency Rate service was stacked to the Collector instance 1 at minute three for scenario S1' and minute one for scenario S5'. In contrast, for the remaining scenarios, the service was stacked after minute ten. As the Collector instance 1 produces depth data, which is larger than the position, when the Data Frequency Rate is adjusted, it decreases the amount of data transmitted over the network.

Figure 32: Resource consumption of the $node_2$ for all workload scenarios. The values correspond to the mean of all samples from the monitoring observations.



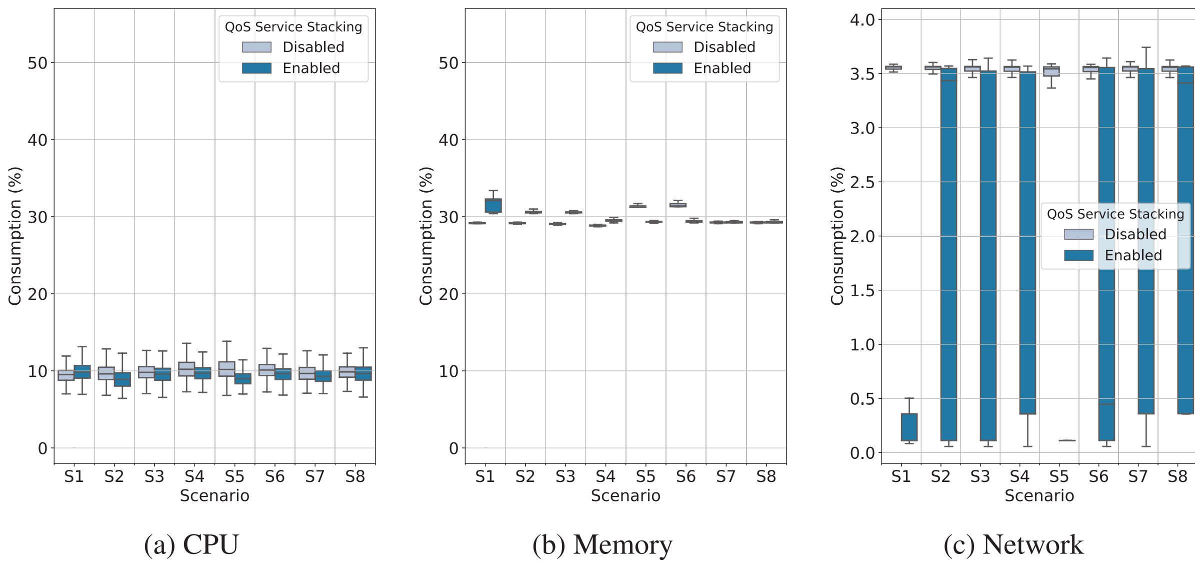
(a) CPU (b) Memory (c) Network

Source: elaborated by the author.

Regarding the Collector instances, Figures 33 and 34 depict the same information than Figure 32, but from the $node_0$ and $node_1$, respectively. Conclusions for the resource consumption of $node_0$ are very similar to those from $node_2$. The differences rely on the CPU consumption of S1' > S1, and the memory consumption of S5' < S5 and S6' < S6. Averaging the results, the variation of consumption from scenarios with QoS Service Stacking enabled related to QoS Service Stacking disabled is -3.2% for CPU, +0.8% for memory, and -61.8% for network. Regarding $node_1$, the measurements are more uniform in comparison to the $node_2$ and $node_0$, though they still resulted in lower CPU (-1.9%) and network (-30.6%) consumption, and higher memory consumption (+2.3%). The only significant difference between $node_0$ and

$node_1$ resource consumption is the network volume, which is explained by the type of data they transmit. The $node_0$ hosts the Collector instance 1 that transmits much higher volumes of data than Collector instance 2, hosted in $node_1$. Note that in the Figures 33 and 34, the ”Network Consumption“ axes have very different scales.

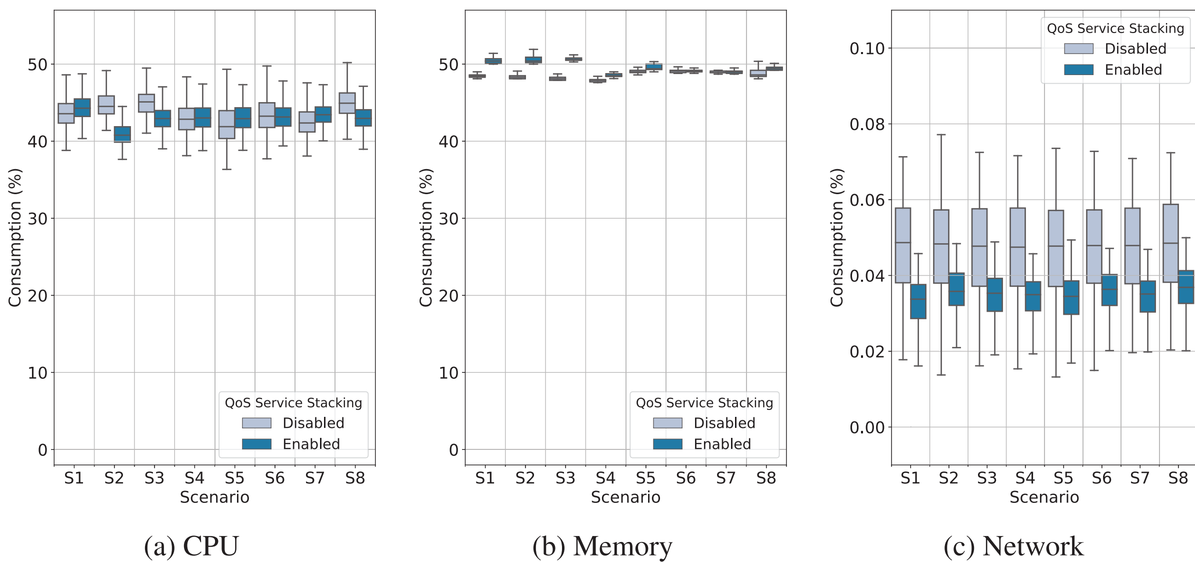
Figure 33: Resource consumption of the $node_0$, in which Collector instances 0 runs.



(a) CPU (b) Memory (c) Network

Source: elaborated by the author.

Figure 34: Resource consumption of the $node_1$, in which Collector instances 1 runs.



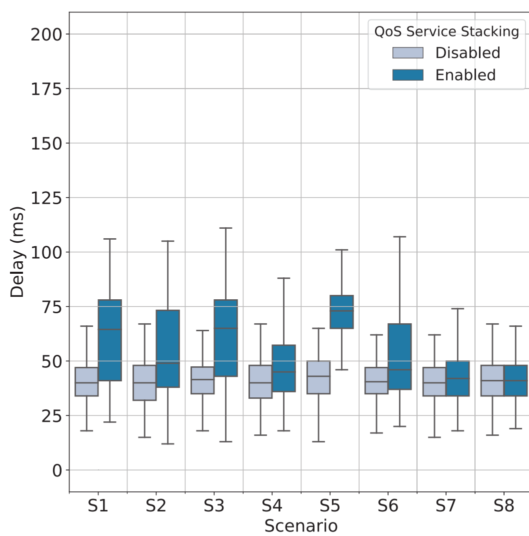
(a) CPU (b) Memory (c) Network

Source: elaborated by the author.

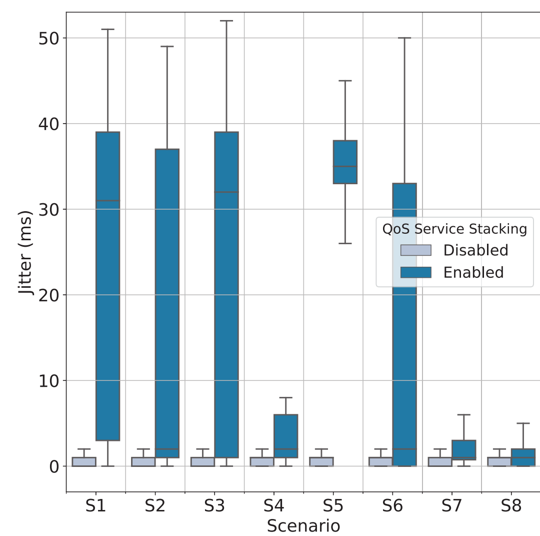
Finally, Figure 35 depicts the mean delay and jitter measurements from all monitoring observations of the Collector instances. The figures can be directly related to the results presented in the previous section. Although the results from scenarios with QoS Service Stacking enabled

resulted in higher values in Figures 35a, 35b, and 35c, they still fall below the hard thresholds, with exception of the jitter of scenario S5 from Collector 1 (Figure 35b) that falls between the soft and hard thresholds. In the case of the jitter of Collector 2 (Figure 35d), the results are even better since all scenarios with QoS Service Stacking enabled obtained lower values than scenarios with QoS Service Stacking disabled. Combined with the previous section results, these results demonstrate that the Collector instances performance dramatically influences the applications' final results. Although the $node_2$ centralizes the experiments' data acquisition, it is enough for deployments with few Collector instances. The current section results' also show that it is possible to improve resource consumption by employing different QoS services, resulting in energy consumption and cost savings.

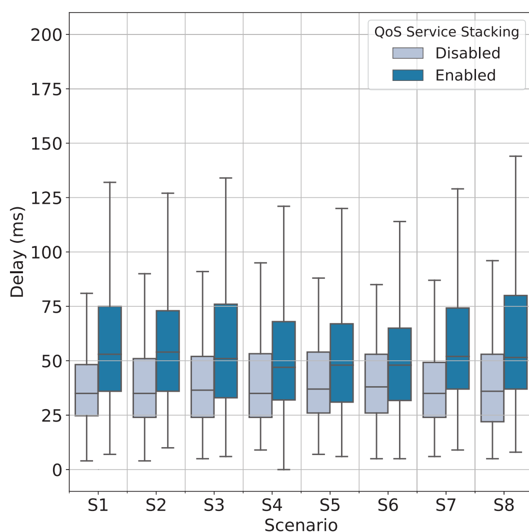
Figure 35: Delay and Jitter means of all observations from Collector instances 1 and 2.



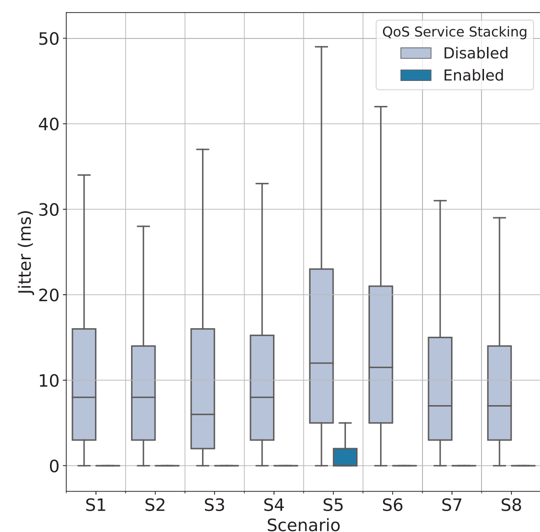
(a) Delay - Collector 1



(b) Jitter - Collector 1



(c) Delay - Collector 2



(d) Jitter - Collector 2

Source: elaborated by the author.

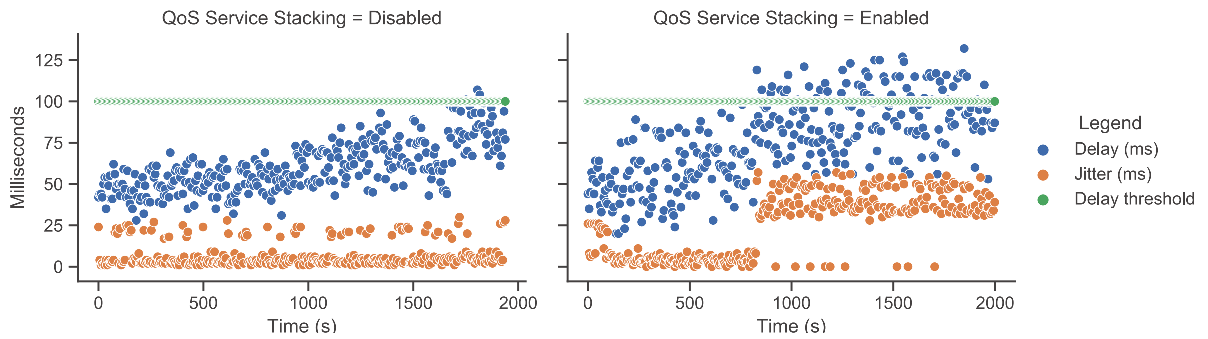
6.3 Applications' Delay and Jitter Time Series

The experiments consisted of eight 32-application scenarios with QoS Service Stacking enabled plus the same eight 32-application scenarios, with QoS Service Stacking disabled. It represents 512 ($32 \times 8 \times 2$) unique application datasets containing each application's measurements in each monitoring observation. That is much information that is not feasible to visualize together in a time series analysis. Additionally, the Core and Collectors' measurements 1 and 2 represent a total of 48 ($3 \times 8 \times 2$) unique datasets with many variables and resource measurements, which increases, even more, the complexity of showing them. Summing them up, it results in 560 different time-series to visualize the state of variables in each monitoring observation. To help the analysis, this section presents just a few time-series datasets to give a glimpse of the model's behavior.

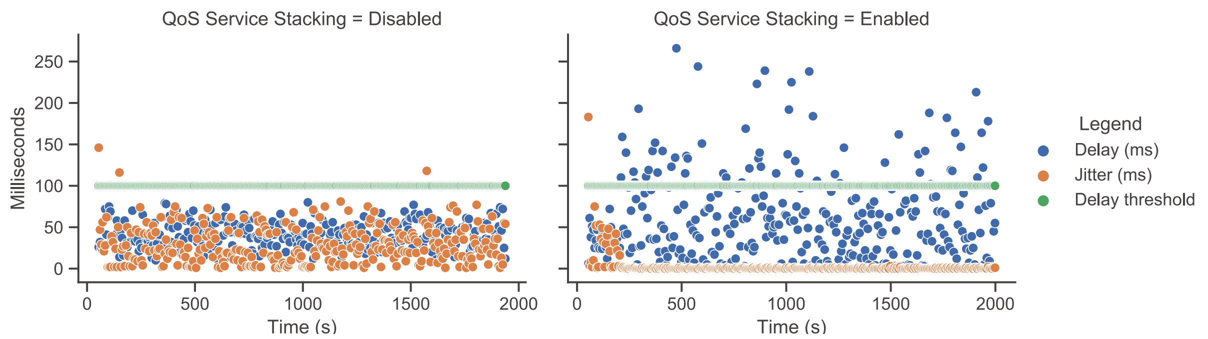
Figure 36 depicts a temporal series of the variables extracted from eight applications of scenarios S1 and S1' to make the comparison. The left diagram in each graph represents scenario S1, while the right scenario S1'. The graphs represent the variables delay, jitter, and the threshold over time from the first four applications. As demonstrated in Figure 27 from the evaluation methodology, the first and the third applications request depth data, while the second and fourth request position data. Comparing diagrams from the same scenario and applications that request the same data type are very similar since they require the same data type. For instance, the left column graphs (S1) demonstrate that the behavior on (a) and (b) are similar to the ones in (c) and (d), respectively. The diagrams demonstrate an increasing delay trend on applications that consume depth data, which is not present on applications that consume position data. This trend follows the increasing number of applications that connect the system every 60 s. The results show that this affects more time to deliver depth data than position data. It occurs because each new application that requests depth data requires the Core to send each frame to a new connection, and this imposes overhead on dispatching data for different applications. Also, it imposes the Core to increase its network consumption each time a new application arrives.

Let us look at the diagrams in the right column of the figure (S1'). Comparing the applications from S1' with their corresponding in S1, it is possible to note the differences resulting from QoS services' inclusion in the Collectors' stacks. In particular, graph (a) shows that the jitter stabilizes near zero right at the beginning, specifically at minute three (180 s). Nevertheless, after minute 15 (900 s), it increases and remains unstable until the end. Not only that, but also the delay variability is higher since minute three. The same occurs to the other applications; however, as applications 3 and 4 arrive after minute three, they were not connected when that early behavior occurred for applications 1 and 2. Looking specifically at applications 2 and 4, the graphs demonstrate that jitter stays near zero during the whole execution time, which explains the jitter results presented in Sections 6.1 and 6.2. For all cases, these changes in the measurement's behavior occur due to the stacking of services by HealthStack. The diagrams demonstrate different effects depending on the data type and service stacked. To better under-

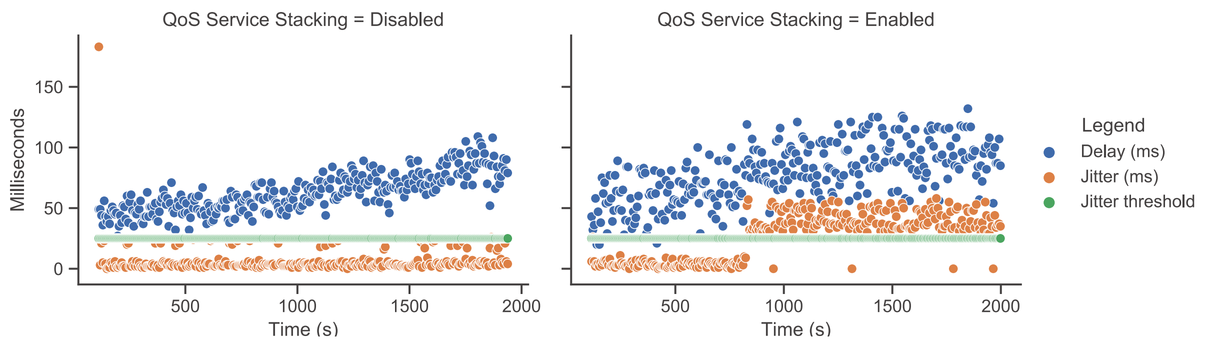
Figure 36: Measures of delay and jitter over time for applications 1, 2, 3, and 4 from scenarios S1 and S1'.



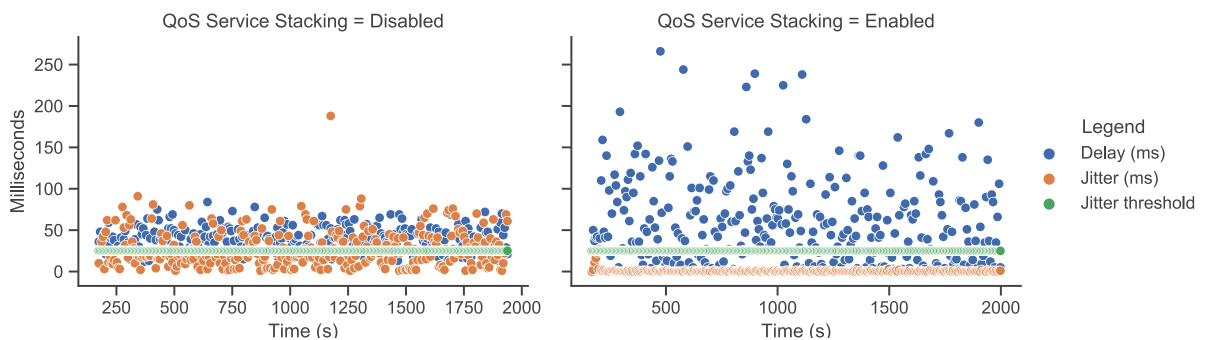
(a) Application 1 (Depth Data)



(b) Application 2 (Position Data)



(c) Application 3 (Depth Data)



(d) Application 4 (Position Data)

Source: elaborated by the author.

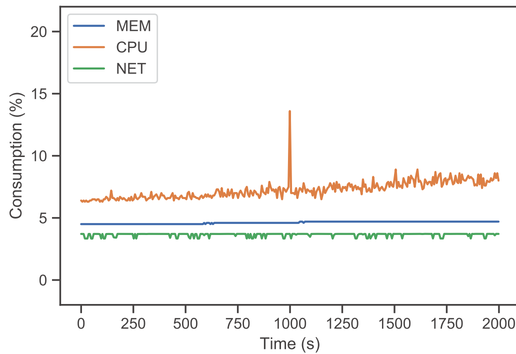
stand what caused such behavior in the applications, let us further explore the variables and QoS services when running these specific scenarios.

Figure 37 depicts the resource consumption and QoS services from the components in scenarios S1 and S1'. The left column presents the data from scenario S1, which do not demonstrate considerably visible variations. The only one is a slight increase in the Core CPU consumption due to the increasing number of applications. However, the right column illustrates interesting traces of the variables. The most crucial thing they expose is each QoS service's effect and when the Manager included them in the components QoS service stacks. Graph 37d shows the first QoS service stacking (Data Frequency Rate) for Collector 1 right after minute three (180 s). At this very moment, the network load dropped from 3.5% to 0.5%. This moment is the one during which the previous Figure 36a (right) shows a change in jitter behavior. The same occurs in Figure 37f but, in this case, right before minute five (300 s), which coincides with the jitter change in the previous Figure 36b (right).

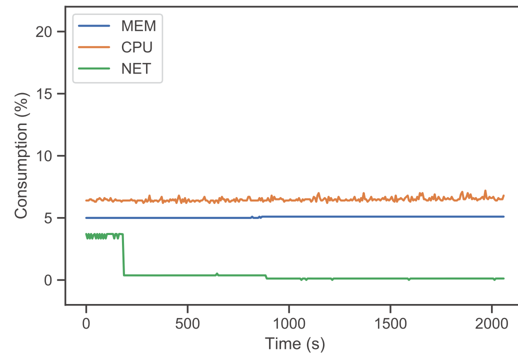
Another visible effect of the Data Frequency Rate QoS service is the decreasing network load for both Core and Collector 1, illustrated in Figures 37b and 37d. That happens because the sampling rate of the Collector instance 1 reduces directly affecting the amount of data transmitted from it to the Core. Following the analysis, let us jump to the instant in which the Manager stacked the second QoS service (Data Compression). In Figure 37d this instant is close to the minute 15 (900 s). The network load dropped from 0.5% to 0.1%; the memory load increased from 30.6% to 33.1% and the mean of CPU from 9.7% to 11.4%. The stacking of compression explains the increase in jitter and delay that can be seen in the previous Figures 36a (right) and 36c (right). The compression of data causes the system to delay the data transmission process and imposes more jitter variations. Accordingly, the same is experienced by the Collector instance 2 and applications 2 and 4 after minute five (300 s) as can be seen in Figure 37f, and the right graph of Figures 36b and 36d. In the third QoS service (Data Prioritization), as only one Collector instance can have this service, it was stacked only to Collector instance 2. However, the results visually do not change due to the inclusion of this service. The next figure (Figure 38) explains why this particular Collector received this last service, along with the details of the Service Orchestration process.

Finally, Figure 38 depicts the variables involved in the Manager decision-making process. More precisely, it presents the values for the weights $w_5()$ and $w_6()$, and also the PA for each Collector instance. Together with the previous two, this figure allows the complete process behavior's perception during the execution. The figure demonstrates the weights increasing over time due to the number of connections. Also, it illustrates the variability of delay and jitter measurements from the applications. The first violation occurs in the jitter of application 3 at minute three (180 s), which consumes data from the Collector instance 1. That increased $w_6()$, and at this moment, also considering the other variables, PA was higher for Collector instance 1. Thus, the algorithm selected the Collector instance 1 to stack the first QoS service. With new applications arriving and new violations, the algorithm followed computing PA, which

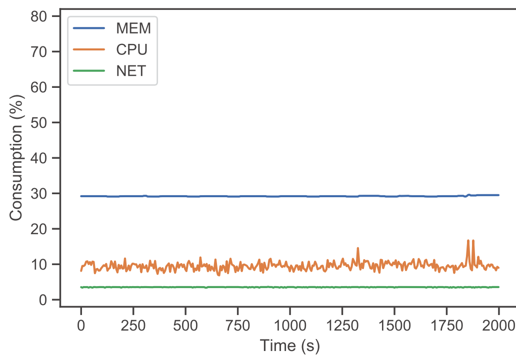
Figure 37: Resource consumption over time of Core and Collector instances from scenarios S1 and S1'.



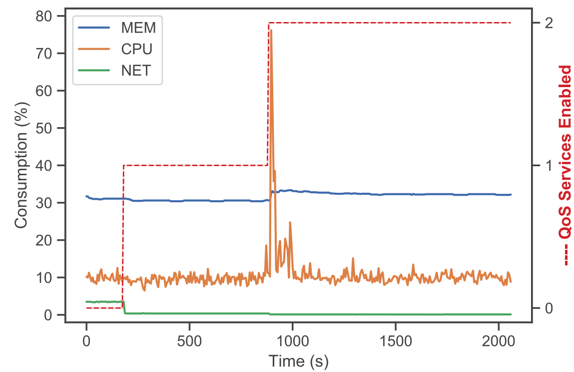
(a) S1 - Resources Core



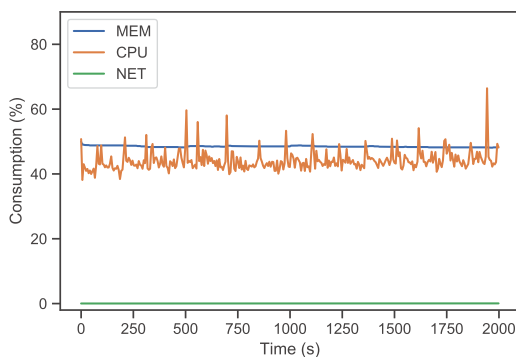
(b) S1' - Resources Core



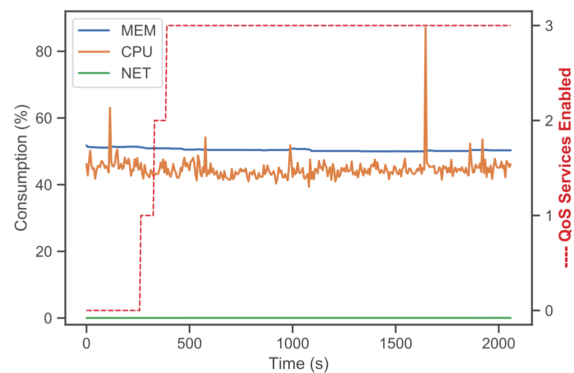
(c) S1 - Resources Collector 1



(d) S1' - Resources Collector 1



(e) S1 - Resources Collector 2



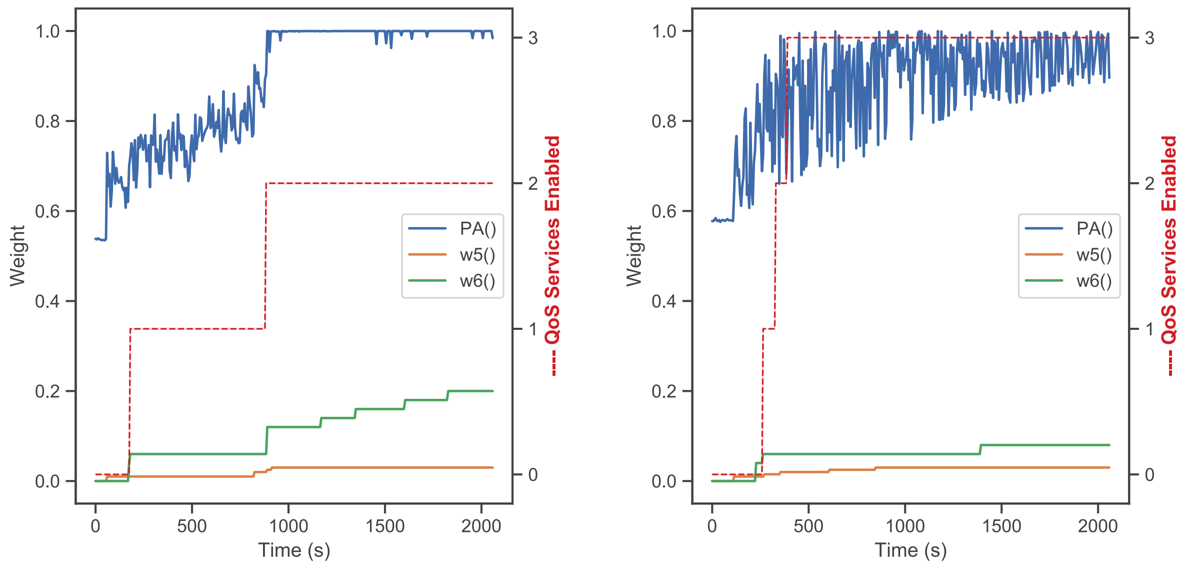
(f) S1' - Resources Collector 2

Source: elaborated by the author.

presents a high variability in its value similar to the applications' measurements consuming data from each Collector. Figure 38b shows that after minute five (300 s), the QoS service stack increased to the maximum until minute seven (420 s). This situation occurred because the QoS violations from applications consuming data from Collector instance 2 were higher than those from applications consuming data from Collector instance 1. For instance, Figure 36b (right) and 36d (right) are examples when compared to Figures 36a (right) and 36c (right).

However, more than these four applications impact the computing of PA for each Collector instance. Lastly, Figure 38 also demonstrates that after minute 15 (900 s), the PA from the Collector instance 1 was higher than instance 2; however, its QoS service stack remained with two services. That is because the last QoS service (Data Prioritization) was already stacked to Collector instance 2.

Figure 38: Looking at the Service Orchestration details. Weights $w5()$ and $w6()$, Potential of Adaptation $PA()$, and number of QoS services enabled over time of Collector instances 1 and 2 from scenarios S1 and S1'. The figures show the weights' variation, and the instant QoS services are stacked for each Collector instance. The QoS service sequence occurs as follow: 1st Data Frequency Rata, 2nd Data Compression, and 3rd Data Prioritization.



(a) S1 - Weights Collector 1

(b) S1 - Weights Collector 2

Source: elaborated by the author.

6.4 Enhancing the Experiments

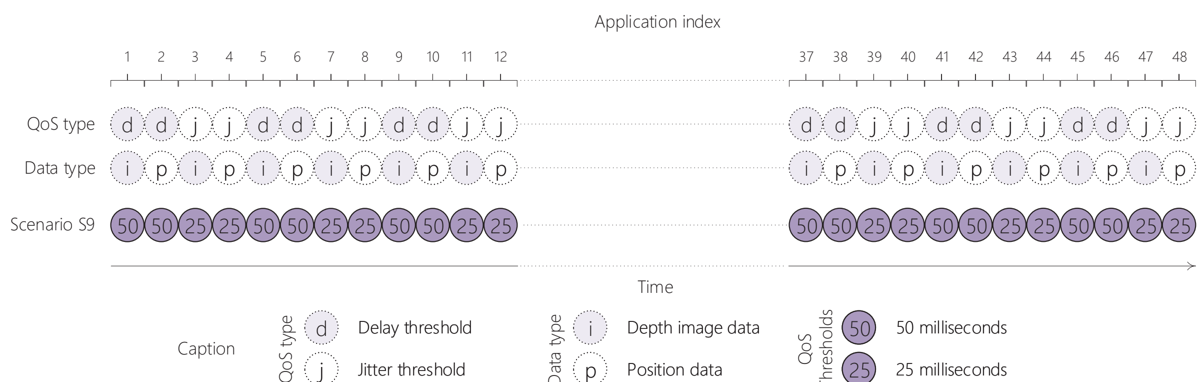
The previous results demonstrate that the middleware can reduce resource consumption, slightly penalizing the applications' metrics in the majority of the cases. Although causing an increase in the measurements, the values remained above the QoS thresholds of the applications. This section further evaluates the middleware. It presents additional experiments only employing the Data Prioritization and the Data Frequency Rate QoS services. That is, here, HealthStack does not employ the Data Compression QoS service. The main reason for that is that the last sections demonstrated that this particular service imposes a significant overhead on the system causing metrics to increase. Therefore, this second phase of experiments investigates the effect of not using this particular QoS service.

The evaluation methodology remained the same in terms of the prototype, infrastructure,

and parameters. The only change regards the evaluation scenario. Figure 39 depicts the new evaluation scenario in detail. Now, instead of 32 applications, the evaluation consists of 48 applications. The QoS type and data type sequence follow the same as in the previous scenarios. Each new application connection requests a different data type than the previous one. Specifically, Application 1 requests depth data, Application 2 position data, Application 3 depth data, and so on. Although the QoS type sequence is the same as before, the only difference is the threshold values. The applications define their target QoS threshold in a round-robin fashion: delay 50 ms, delay 50 ms, jitter 25 ms, and jitter 25 ms. In other words, Applications 1 and 2 define delay to 50 ms, Applications 3 and 4 determine jitter to 25 ms, and the sequence repeats for the next applications. The definition of these new values follows some studies on healthcare (MUKHOPADHYAY, 2017; NANDA; FERNANDES, 2007; MALINDI; KAHN, 2008; LEE et al., 2011).

The evaluation consisted of observing the execution of the middleware and the applications in ten minutes. The time zero represents the instant the HealthStack starts running. The middleware and the script to run the applications start at the same time. This particular script starts each application respecting the interval described above (one at every six seconds). In the beginning, the script sleeps for six seconds and then starts the first application at the time 00:06 (mm:ss). Consequently, the last application connects at the time 4:42 (m:ss). The applications and HealthStack keep running until minute ten to visualize more data when all applications are connected. As in the previous experiments, this application scenario executes under two different configurations to provide a vision of the benefits of employing the QoS Service Stacking strategy: (*i*) QoS Service Stacking enabled; and (*ii*) QoS Service Stacking disabled. The next subsections present the results.

Figure 39: The sequence of application connections in the new workload scenario. One application connects every six seconds requesting a Data type, a QoS type, and a QoS threshold. The application index refers to the sequence of the application connection to the middleware.

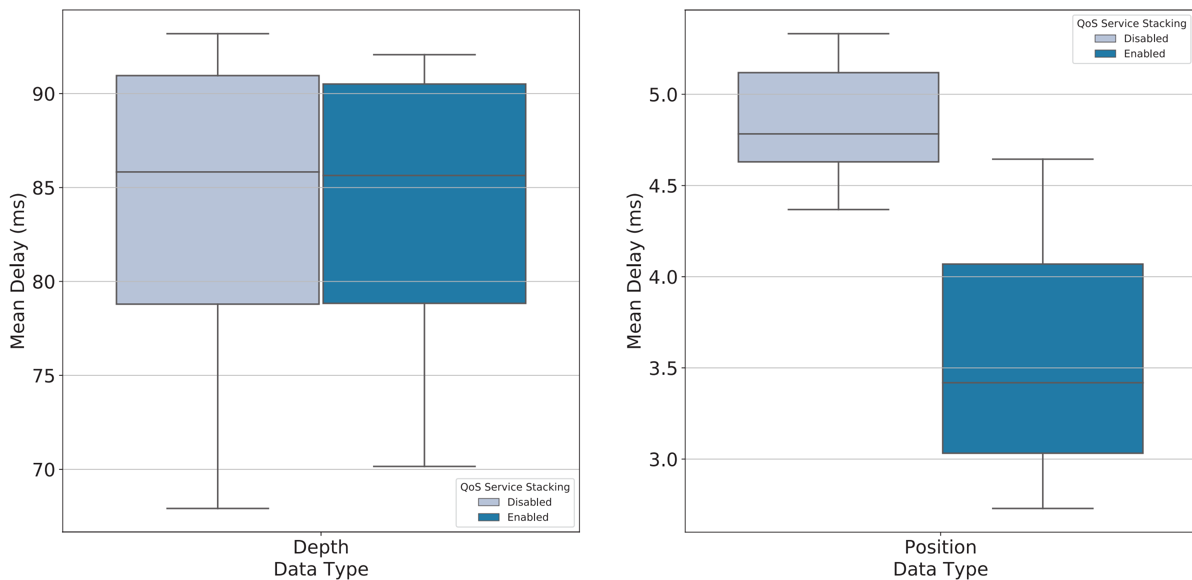


Source: elaborated by the author.

6.4.1 Applications' Average Delay and Jitter

This subsection presents the results for each application regarding the two target metrics: delay and jitter. With this evaluation, the reader looks at the delay and jitter of different applications brought by the QoS Service Stacking. First, let us focus on the delay metric. Figure 40 illustrates the distribution of the mean delay for each application considering the QoS Service Stacking enabled and disabled. As application request different data types, the figure segments the results according to the data type to show their relevance to the results.

Figure 40: Delay results for the new evaluation scenario considering the two different sensor types: depth and position. Each bar is calculated with the mean delay of each application.



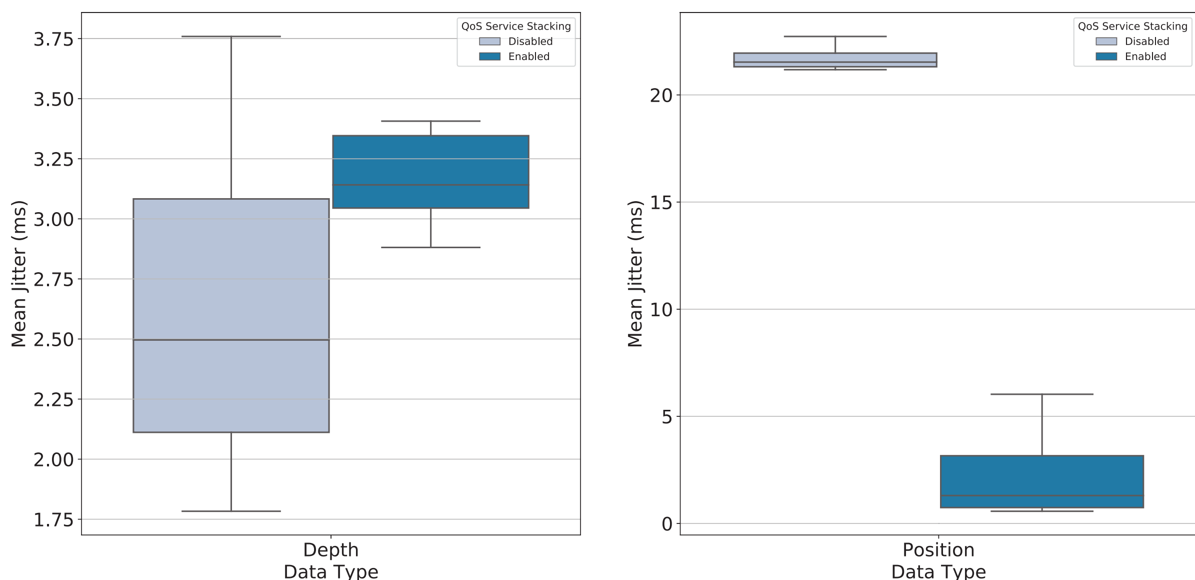
Source: elaborated by the author.

Most of the applications that consume depth data experienced improvements, but not all of them. From a total of 24, 18 applications (75%) presented improvements in their average delay. Computing the mean of the average delay of each of the 24 applications (that consume depth data) results in 84.2 ms when QoS Service Stacking is disabled and 84 ms (-0.23%) when enabled. On the other hand, applications that consume position data achieved more meaningful results (pay attention to the scale). Only one out of 24 applications achieved a higher average delay when enabling the QoS Service Stacking. The mean of the average delay for these applications when QoS Service Stacking is disabled results in 5 ms against 3.6 ms (-28%) of QoS Service Stacking enabled. Comparing both diagrams, the data size transmitted to applications from depth data is considerably higher than position data. Depth data requires more processing and transmission time; therefore, the changes represent a smaller percentile than the second set's improvements. Still, enabling the QoS Service Stacking strategy improves the results in most of the cases.

Now, let us change the focus to the jitter metric. Figure 41 depicts the distribution of the

mean jitter for each application considering the QoS Service Stacking enabled and disabled. Here, the variations are more expressive than the ones for the delay. However, different data types resulted in contrasting results. On the one hand, the left diagram demonstrates the results for applications that consume depth data. Only five from 24 applications had improvements in their jitter. Considering all 24 applications, the mean of their average jitter results in 2.6 ms with QoS Service Stacking disabled versus 3.3 ms (+26.9%) with QoS Service Stacking enabled. Although not demonstrating improvements, the resulting jitter is still low. It demonstrates that enabling the QoS Service Stacking results in better delay, not imposing higher jitter penalties, as a jitter lower than 25 ms is acceptable (NANDA; FERNANDES, 2007).

Figure 41: Jitter results for the new evaluation scenario considering the two different sensor types: depth and position. Each bar is calculated with the mean jitter of each application.



Source: elaborated by the author.

On the other hand, the right diagram shows that all applications that consume position data had considerable high improvements. Examining the individual results of all 24 applications that compose each box of this diagram, the mean of their average jitter results in 21.7 ms when disabling QoS Service Stacking versus 2.1 ms when enabling it. It represents a substantial decrease of 90.3%. The main reason for that is that the RTLS system does not provide tag position samples at a stable rate. It impacts directly on jitter. In this scenario, the Data Frequency Rate service plays an essential role because it stabilizes the frame rate to a lower value. It also stabilizes jitter since the Collector acquires samples at a lower rate. Comparing both Figures 40 and 41, it is possible to note that the data type turns to make difference in the results. Although the results demonstrate significant advantages for small-sized data types, they also demonstrate excellent results for large size data types. That is important in healthcare since much critical information is small size data types. For example, we have body temperature, ECG, and oxygen saturation level.

6.4.2 Resource Consumption

This subsection focuses on analyzing all three nodes' resource consumption from the infrastructure when running the new scenario. Table 14 organizes all results of CPU, memory, and network consumption for all nodes. The table presents the variation for each result when comparing the QoS Service Stacking enabled and disabled. Results demonstrate that the QoS Service Stacking decreases the needed resources to execute the application scenario for most of the cases. The only exceptions are the CPU consumption of $node_0$, memory consumption of $node_1$. For the remaining cases, the results are expressive. For instance, network consumption is dramatically lower when the QoS Service Stacking strategy is enabled. What causes that is the Data Frequency Rate QoS service because both Collector 1 ($node_0$) and Collector 2 ($node_1$) received this service during the execution. This QoS service decreases data transmission from both nodes to the $node_2$, causing a decrease for all nodes.

Table 14: Average resource consumption of each node CPU, memory, and network. Additionally, the last column represents the variation in resource consumption when enabling the QoS Service Stacking strategy.

Resource	Node	QoS Service Stacking		Variation
		Disabled	Enabled	
CPU	$node_0$	9.64%	10.21%	+5.91%
	$node_1$	2.03%	1.05%	-48.28%
	$node_2$	9.7%	6.92%	-28.66%
MEM	$node_0$	30.64%	29.09%	-5.06%
	$node_1$	19.93%	22.58%	+13.30%
	$node_2$	4.35%	4.35%	-0.02%
NET	$node_0$	3.53%	1.18%	-66.57%
	$node_1$	0.17%	0.15%	-11.76%
	$node_2$	3.66%	1.23%	-66.39%

Source: elaborated by the author.

The CPU results also demonstrate lower consumption with QoS Service Stacking enabled for $node_1$ and $node_2$ due to the Data Frequency Rate QoS service. The TCP/IP protocol requires extra processing cycles for each data transmission, directly impacting CPU consumption. With fewer data samples to transmit, the node spares cycles resulting in lower CPU consumption rates. Although the results for $node_0$ represent a higher consumption when the QoS Service Stacking is enabled, this represents an addition of only 0.57%. The results do not demonstrate significant improvements regarding memory consumption, but they still resulted in less memory consumption for two of three nodes. For $node_1$, the QoS Service Stacking resulted in more

memory consumption. However, it is not trivial to define the main reason for that. To better understand the profiles of resource consumption, the next figure is more helpful.

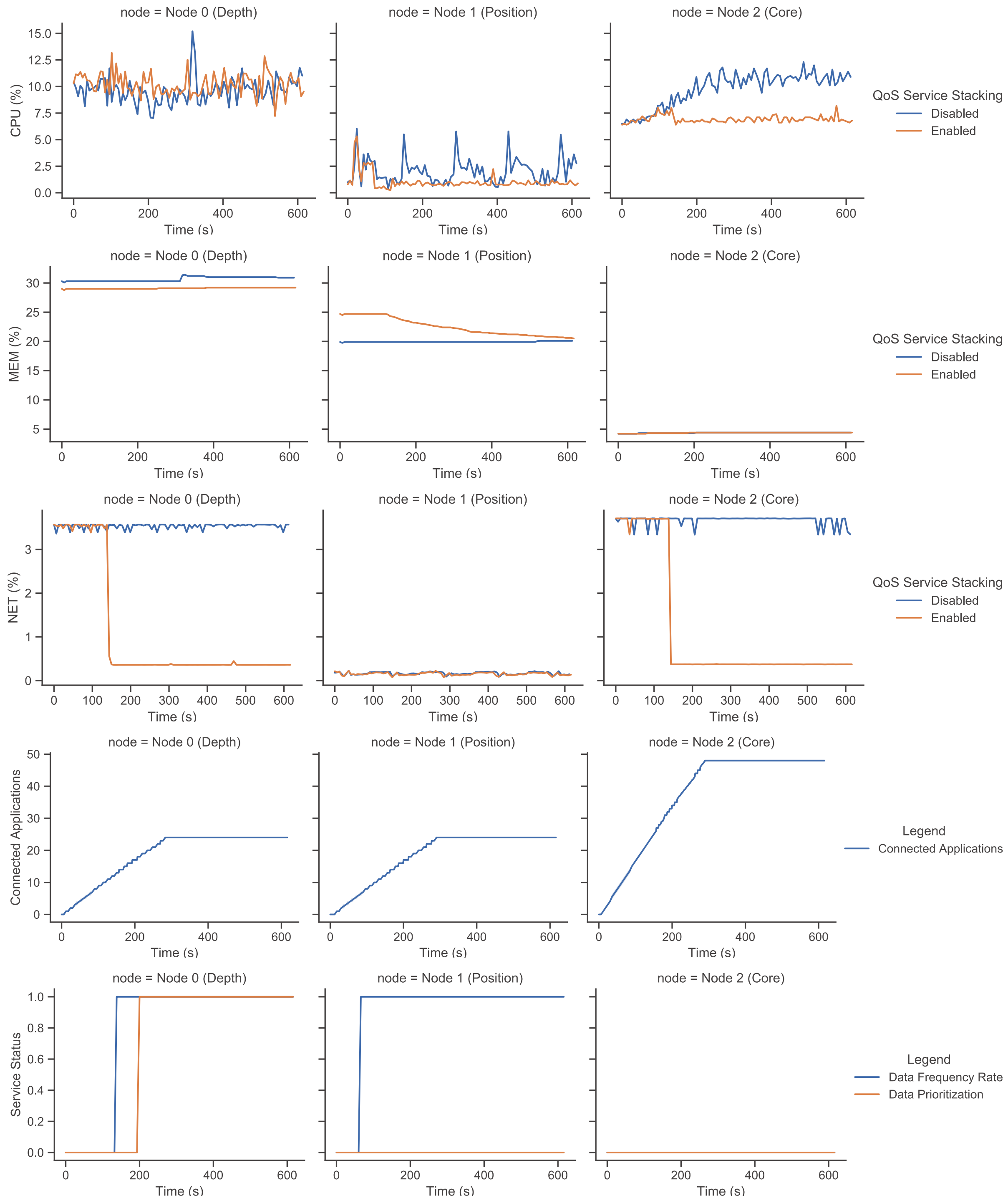
Figure 42 depicts each node's resource consumption over time in both scenarios with QoS Service Stacking enabled and disabled. The first three rows of the figure bring results from CPU, memory, and network consumption, in that order. To understand the variations in resource consumption that occur over time, the last two rows of the figure illustrate additional information on the number of connected applications and the distribution of services in the Collectors' stacks. It is important to note that, for $node_0$ and $node_1$, the fourth row demonstrates only how many user applications request their data to the Core, which runs in $node_2$. Besides, the last row demonstrates the services active for each node in a binary form. A value equal to zero in this particular diagram represents that the service is not enabled, while a value equal to one means that the service is enabled.

The first row is straightforward for $nodes_1$ and $node_2$ that results with the QoS Service Stacking enabled achieve lower values. The enabling of the Data Frequency Rate service stabilized the CPU consumption for both of them. On the other hand, the same is not valid for $node_0$. The CPU traces remain unstable all the time. This particular node runs the Microsoft Kinect API, which implements several computer vision techniques to extract depth information from the Kinect. This process produces too much overhead making the impact of the Data Frequency Rate negligible. The main reason for that is because this particular service works at the network level of the node. It only decreases network transmission by decreasing the Collector instance's network frame rate when it is enabled. It does not impact the Kinect device's primary depth data extraction process, which remains generating local information at its rate.

Network consumption has different behaviors for each node. First, $node_0$ demonstrates an increase in 1% of network consumption at 318 s. Second, $node_1$ shows very different results at the beginning and a decreasing trend when the QoS Service Stacking is enabled. When running the experiments, for both situations with QoS Service Stacking enabled and disabled, the same processes were running in each computer node. Thus, the starting point of network consumption should be the same. As that did not occur, some internal processes running in the operating system were consuming different amounts of memory in each execution. However, the results do not demonstrate a direct impact of HealthStack because it was running in both cases. The only difference is that when QoS Service Stacking is enabled, HealthStack stacks services to the Collector instance. Otherwise, it only monitors all parameters and does nothing when violations occur.

Finally, the most visible alteration in resource consumption regards the network in the third row. For both $node_0$ and $node_2$, there is a considerable drop in network consumption. That occurred because of the Data Frequency Rate QoS service stacked for Collector 1 ($node_0$) at 138 s of execution. This caused a decrease in the data transmission of $node_0$, which directly impacts the data reception of $node_2$ running the Core. For $node_1$, there is no visual alteration even though the same service is also stacked. That is because position data does not require

Figure 42: All nodes' resource consumption over time, including the number of connected applications in the middleware, and the services delivered for each node. In the last row, the y-axis represents whether a service is active (zero) or not (one).



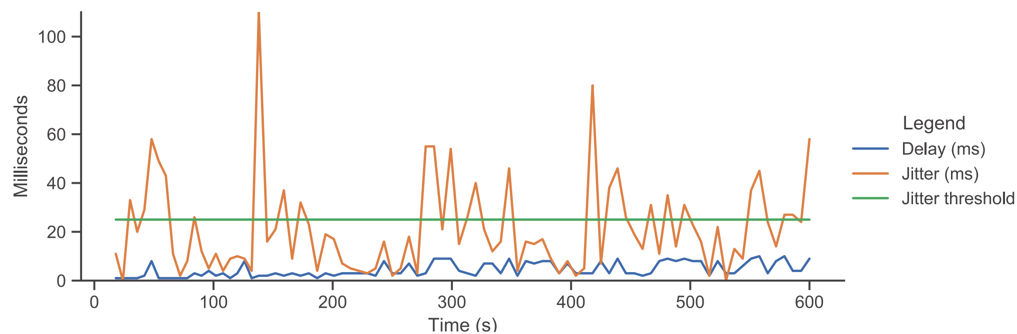
Source: elaborated by the author.

higher data transmission, and variations in the amount of its data are not significant.

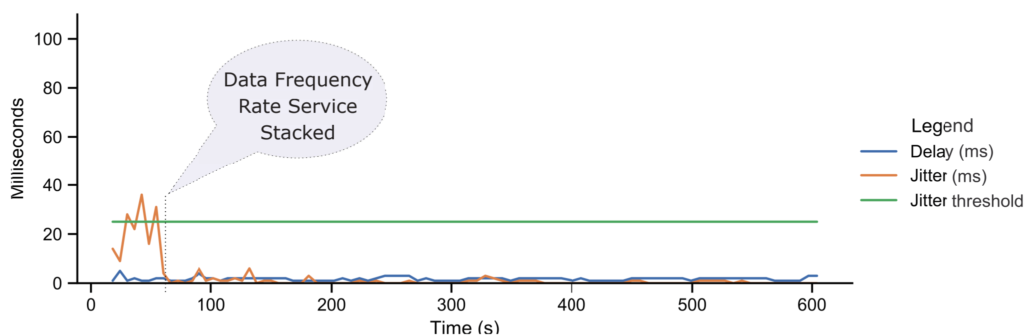
6.4.3 Applications' Measurements Time Series

This analysis selects data from one of the 48 applications to show the behavior of measurements over time and the QoS Service Stacking impact. Figure 43 depicts the measurements of delay and jitter over time for application 4. This application consumes position information and informs a jitter threshold of 25 ms. This figure of one particular application's behavior facilitates the understanding since plotting all data would confuse the reader. The figure shows the difference in performance for the application when enabling the QoS Service Stacking. Diagram (a) demonstrates that the application's jitter violates the application's threshold at several moments. These violations keep occurring during the whole execution.

Figure 43: Application ID 4 delay and jitter measurements over time with QoS Service Stacking disabled versus enabled. The goal is to visualize the impact on measurements when QoS services are enabled.



(a) QoS Service Stacking Disabled



(b) QoS Service Stacking Enabled

Source: elaborated by the author.

On the other hand, diagram (b) demonstrates a different performance. That is possible because, at time 68 s, HealthStack stacks the Data Frequency Rate service. Right after this moment, both metrics delay and jitter drop and keep stable until the execution. Looking specifically at the Jitter line, after violating the threshold in the first 60 s, the service activation solved the problem, and no more violations occurred. That demonstrates the power of the QoS Service Stacking strategy. Another important thing to keep in mind is that the system only starts to stack services after 60 s due to the cool-down period. HealthStack always starts in cool-down

to avoid taking premature actions.

6.5 Discussion

The evaluation of the model included three QoS services and several applications with different QoS requirements. The experiments demonstrate that each QoS service brings individual advantages and drawbacks. Therefore, the QoS services must be chosen wisely according to the main objective of the solution. For instance, adjusting the sampling rate of sensor nodes helps stabilize the jitter and reduce resource consumption. On the other hand, compressing data has distinct effects on the final results of applications and resource consumption because it introduces computation overhead at the sensor nodes. Thus, increasing the time to dispatch data over the network and simultaneously decreasing the amount of data to be transmitted.

Experiments demonstrate that, although the mean of all scenarios with service support increased by 34.7% in delay, the mean delay was 74.2 ms, which is lower than a hard QoS threshold of 100 ms. On the bright side, this can be useful in scenarios with network limitations despite the penalty delay. Besides the effects of the services, the experiments demonstrate that HealthStack can automatically perform the adjustments without any user actions. HealthStack Orchestration Service demonstrates to choose the right components to stack QoS services when QoS violations occur, resulting in improved resource consumption. All operations are transparent to the user, which may or may not define the application's QoS requirements. If the user does not inform QoS properties, HealthStack automatically assumes specific default values for each data type. Therefore, even if the user is not aware of the QoS model, the services will be available and running.

Comparing HealthStack with related work in the area, HealthStack demonstrates its strengths in the multi-service on-the-fly ability. That is, HealthStack employs multiple QoS strategies, and it activates them according to QoS violations. Therefore, HealthStack provides data to applications aware of their requirements. Only in case of applications have their QoS violated, HealthStack takes action to tackle the problems. Current strategies mostly employ specific QoS strategies directly to sensor data transmissions. They are tailored to specific environments, such as WBAN. Differently, HealthStack is agnostic of the type of sensor, allowing the integration of a wide variety of sensors.

6.5.1 Main Results

The first conclusions about HealthStack experiments led to an important question: considering the data types employed in the evaluation, do the experiments need to employ the Data Compression service? With this in mind, a new set of experiments was performed, not including this particular service. That demonstrated to be positive in the evaluation since the overhead this service imposes is significant for the data types and network setup where the experiments

were carried on. This second phase of experiments demonstrates, with more emphasis, the benefits of HealthStack. The QoS Service Stacking strategy demonstrated promising results, improving both delay and jitter from applications while decreasing computational resources' requirement to achieve that. In summary, HealthStack brings expressive improvements for low size data types, such as position data. Tables 15 and 16 highlight the main results from the two experiment phases side by side.

First, Table 15 shows that the QoS Service Stacking brought more improvements in average delay and jitter in the second phase. This phase does not include the Data Compression service, which did not bring good results in the first phase. Although each phase's scenarios are different, the table shows that the second phase achieved much better results. Unlike the first phase, three out of four values represent improvements from the application's perspective. From the resource consumption perspective, Table 16 shows that, in general, the second phase of experiments achieved much better results. Seven out of nine values represent improvements in the second phase, compared to six in the first, which is also positive. These results demonstrate that HealthStack can improve resource consumption in most cases.

Table 15: Variation in average delay and jitter of all applications when enabling the QoS Service Stacking. Improvements are highlighted in blue boxes.

Phase	Delay		Jitter	
	Depth	Position	Depth	Position
1st	24.78%	51.47%	250%	-92.3%
2nd	-0.23%	-28%	26.9%	-90.3%

Source: elaborated by the author.

Table 16: Variation in resource consumption of all applications when enabling the QoS Service Stacking. Improvements are highlighted in blue boxes.

Phase	*node*_0 (Depth)			*node*_1 (Position)			*node*_2 (Core)		
	CPU	MEM	NET	CPU	MEM	NET	CPU	MEM	NET
1st	-3.2%	0.8%	-61.8%	-1.9%	2.3%	-30.6%	-8.3%	25%	-56%
2nd	5.91%	-5.06%	-66.57%	-48.28%	13.30%	-11.76%	-28.66%	-0.02%	-66.39%

Source: elaborated by the author.

HealthStack achieved mixed results when looking at the colors of the values. As aforementioned, the first phase of experiments included the Data Compression service, which imposed extra time to process data. That directly impacted the applications' delay and jitter. The only exception was the jitter for position data that was improved even with this particular service. The RTLS system employed in the experiments does not generate position data at a steady frequency. When employing the Data Frequency service, HealthStack decreases the Position

Collector's FPS to 1 Hz, which allows the RTLS system to produce at least one new information at each second. That stabilizes the jitter resulting in the observed improvement. A similar jitter result is observed in the second phase of experiments for the same reason. On the other hand, this phase, now not employing the Data Compression service, achieved better results than the first one. Without the overhead of this particular service, HealthStack decreased both delay and jitter for applications. The only exception is jitter for depth data, which is still higher on average but much lower than the first phase.

6.5.2 Limitations

Looking at the HealthStack's limitations, the evaluation considered only two types of sensors. The experiments only employ position data and depth information. These sensors can represent a short range of sensor types, specifically sensors that generate just a few bytes or hundreds of kilobytes. It is essential to experiment HealthStack employing sensors that generate data with different patterns and much higher data volumes. In the latter case, the Data Compression service can be beneficial.

Besides the data types limitation, HealthStack considers many variables to be configured. The experiments did not focus on testing different parameters to assess their effect on the solution. The evaluation methodology defined a set of parameters that were fixed among all experiments and phases. Those parameters can change the behavior of the system and in which instant QoS services are stacked. Therefore, setting them is challenging and requires technical awareness from the technical team that is deploying HealthStack. An evaluation of different sets of parameters and their effects could make it easy to define them.

Also, from the four QoS services proposed, the evaluation did not cover the Resource Elasticity. As the middleware Core runs in a dedicated and robust server, it has plenty of computational resources to address the proposed evaluation scenarios. Although the Core centralized the data in our evaluation, it did not show high enough resource consumption to interfere with the results. The results demonstrate that the Core resources never achieved their limits, which would trigger the Service Orchestration to deploy the Resource Elasticity service. Besides running the Core in a robust server, the evaluation methodology employed two Collector instances, which is another limitation.

Hospital deployments can employ a large number of sensors, which would require more Collector instances. Therefore, experimenting HealthStack in a large-scale deployment could provide better insights for such environments. The HealthStack model comprises multiple instances of the Core, which can be configured to collect data from different Collector instances. However, the experiments only employed a single Core instance given the small scale of the environment. A large scale environment would also require more than one Core instance to balance the load from several Collector instances.

6.6 Summary

This section presented the results of the experimental tests of the model from different points of view: (*i*) average delay and jitter of applications; (*ii*) resource consumption; and (*iii*) time series analysis of the applications' measurements. Based on some conclusions obtained from the analysis of results, the second set of experiments was conducted with some changes compared to the first one. The first set of experiments demonstrated that the mean delay of all scenarios and data types with QoS Service Stacking enabled achieved higher results. More specifically, considering depth data, the mean delay of applications is 69.4 ms for scenarios with QoS Service Stacking disabled, and 86.6 ms for scenarios with it enabled, representing an increase of 24.78% when enabling the QoS Service Stacking. In turn, considering position data, the mean delay of applications is 40.8 ms for scenarios with QoS Service Stacking disabled, and 61.8 ms for scenarios with it enabled, representing an increase of 51.47%. Although the increase, the mean is still lower than the hard threshold for the delay, which is 100 ms.

On the jitter scope, averaging the results from applications that request depth data, the mean delay of applications is 26.3 ms with QoS Service Stacking enabled versus 7.5 ms with it disabled, representing an increase of 250%. Considering applications that request position data, the mean of scenarios with QoS Service Stacking disabled is 28.5 ms. In contrast, scenarios with QoS Service Stacking enabled it is 2.2 ms, demonstrating an improvement of 92.3%. The values demonstrate contrasting results depending on the data type. HealthStack demonstrates results in better performance for low size data types.

Regarding resource consumption, each node employed in the experiments has particular results. First, for $node_0$, QoS Service Stacking resulted in improvements of 3.2% in CPU, 61.8% in network, and overhead of 0.8% in memory consumption. Second, for $node_1$, QoS Service Stacking resulted in improvements of 1.9% in CPU, 30.6% in network, and overhead of 2.3% in memory consumption. Last, for $node_2$, QoS Service Stacking resulted in improvements of 8.3% in CPU, 55.5% in network, and overhead of 25% in memory consumption. These results demonstrate that HealthStack can improve resource consumption in the majority of cases. By further analyzing the first set of experiments, it was identified that the Data Compression service could avoid more expressive improvements. This particular service imposes overhead on processing data by the Collector instances. Thus, an additional set of experiments was performed with some changes, including removing the Data Compression service.

The second set of experiments changed the number of applications from 32 to 48 and the arrival interval from 60 s to 6 s. For this new scenario, the evaluation time changed from 35 minutes to ten minutes. The evaluation considered the same three points of view as the first set of experiments. Computing the mean of delay of each application that consumes depth data, it results in 84 ms when QoS Service Stacking is enabled and 84.2 ms when disabled, representing a decrease of 0.2%. Doing the same, but for applications that consume position data, it results in 3.6 ms when enabling QoS Service Stacking against 5 ms of QoS Service Stacking disabled,

demonstrating an improvement of 28%. Unlike the first set of experiments that employed the Data Compression service, now the QoS Service Stacking results in better delay performance for both data types.

Regarding the jitter metric, for applications that consume depth data, the mean of their jitter results in 3.3 ms with QoS Service Stacking enabled versus 2.6 ms with QoS Service Stacking disabled, representing an increase of 29.9%. For applications that consume position data, QoS Service Stacking enabled results in 2.1 ms versus 21.7 ms of QoS Service Stacking disabled, representing 90.3% of improvement. The results demonstrate that HealthStack provides considerable performance gains and does not impose high penalties in depth data since the mean jitter remains above the hard threshold of 25 ms.

Finally, analyzing the resource consumption results, each node has different results. First, for $node_0$, QoS Service Stacking resulted in improvements of 66.57% in network, 5.06% in memory, and overhead of 5.91% in CPU. Second, for $node_1$, QoS Service Stacking resulted in improvements of 48.28% in CPU, 11.76% in network, and overhead of 13.30% in memory consumption. Last, for $node_2$, QoS Service Stacking resulted in improvements of 28.66% in CPU, 66.39% in network, and 0.02% in memory consumption.

The experiments of the model comprised a roadmap that included two phases of experiments. The first phase included the Data Compression service. The results identified that this service imposed too much overhead in the data processing flow, not improving the results. Then, the second set of experiments without using this particular service demonstrated its impact on results. In particular, results from the second set of experiments were even better than the first set of results.

7 CONCLUSION

Modern hospitals advance towards the future in the path of the technological advances brought by IoT technology. Sensors will generate massive amounts of information regarding medical processes, patients, and medical staff. Nowadays, medical processes rely on reactive actions. Nurses and physicians act according to data from equipment that monitors patients. In the future, global analysis of many centralized data will allow the employment of artificial intelligence technology. Therefore, the correlation of data from different sources will favor the medical processes in a way never seen before. Data predictions will enable the medical staff to act more proactively through data analysis. More specifically, they will coordinate their actions according to forecasts of such systems.

However, this future relies on a distributed system with centralized data processing, which brings scalability problems. If the system loses performance due to too many sensors and applications, the medical processes' proactiveness would be compromised. Nowadays, many solutions focus on the QoS problems of systems in healthcare. Notwithstanding their contributions, they lack focus on the real-time property for high critical data in hospital environments and a more general system capable of acquiring data from multiple sensors. More precisely, they do not focus on proposing a system that connects many sensors to many users with a simple interface while offering QoS strategies according to the system load.

Given the background, Section 1.2 presented the following research question: *Which self-adaptative strategies a healthcare middleware, with dynamic sensor and application connections, needs to efficiently provide timed data for applications?* To answer that, this study proposed HealthStack, a QoS-aware middleware model for hospital settings that aims at delivering multiple sensor data for applications. The middleware monitors the delay and jitter of applications and takes actions in case thresholds are violated. HealthStack defines several components organized in a distributed system among several nodes. Besides the base components, the model comprises a QoS Manager module that enables automatic monitoring and QoS Service Stacking through a QoS model. The QoS Service Stacking strategy is a novel concept that decides when to enable QoS services for each system component based on measurements of the system and user requirements. The Manager monitors performance metrics periodically from both components and applications. The decision-making process analyzes all this information to decide which component is overloaded to activate QoS services to tackle the problem. Considering the proposed middleware and results achieved by this study, the following hypothesis presented in Section 1.2 could be verified: *An adapting-driven middleware for sensor-based healthcare environments, with dynamic sensors and applications connected to the system, can efficiently provide an acceptable quality of service for end-user applications.*

The current research development was part of a research project in partnership with a hospital in Brazil. It allowed the deployment of the prototype in an actual hybrid operating room from the hospital surgical block. Access to a critical environment, such as operating rooms,

bring valuable contributions to research development. The experience in such an environment enriches the study by providing real issues and concerns to the research development. Among them, focus on patient well-being is crucial when deploying any technology inside an operating room. Besides, installing equipment in the hospital requires the operating room to be unavailable for a while. That affects the operating room schedule, and for that reason, the deployment process should be as efficient as possible.

7.1 Lessons Learned

This research's scientific roadmap included designing and implementing the middleware in a real hybrid operating room. The development and installation were demonstrated to be complex but possible to achieve. This section discusses the aspects of the experience in accomplishing the study in a real hospital. It brings to discuss some lessons learned in many aspects, from building the partnership to performing experiments.

7.1.1 Lesson 1: Build a Strong Partnership

Bringing scientific research to real deployments in the healthcare scenario is, at the same time, challenging and essential. As a first step, establishing a partnership requires many meetings, including both hospital and university staff, to clarify the project's aspects. Once the project is approved, technical details should be well discussed between the hospital's technical team and the scientific team. That is important since different sectors of technology and engineering management the whole hospital infrastructure. Thanks to that, deploying hardware and software within the hospital settings must be previously discussed with both sectors. That spotlights the necessity of negotiating the best solutions to install equipment without causing problems to the current systems running in the hospital.

With all details solved, the installation process also requires people of different hospital sectors and even the scientific team to coordinate. The whole process demonstrates that forming a partnership and making it happen requires both commitment and proper communication of several persons. Although complicated, the process results in a partnership with outstanding outcomes for both sides. The ideas developed for the hospital can improve their services for patients, which also benefits from it. It is an excellent opportunity for researchers to grow and evolve new ideas with constant feedback from real settings.

7.1.2 Lesson 2: Consider the Complexity of the Environment in the System Design

Medical settings are highly critical environments that encompass complex processes. Such complexity includes a variety of equipment and medical roles performed by several persons at the same time. On top of that, the patients are the process's focus, and their well being is the

process' ultimate goal. Thanks to all that complexity, implementing new techniques requires awareness of multiple aspects. On the one hand, depending on the technology, it is crucial not to cause adverse effects on the current technologies already used in the hospitals. On the other hand, as medical processes involve legal issues, privacy concerns should also be considered. Thus, there is a limitation in what the researchers can do or not in the hospital facilities.

Considering all of that, this research study was as carefully as possible in deploying the infrastructure at the hospital. Hence, we opted to install our private network infrastructure with the hospital's biomedical engineering sector's help. This choice is due to two main reasons. First, as some sensors generate a considerable amount of data, this could impact the network and, consequently, other services already running in the hospital. Second, a private network only for the middleware data exchange prevents the transmission of raw data through the hospital network due to several privacy concerns. Besides, this also prevents network sniffers from capturing the middleware packets and corrupting them. As a bonus, it increases the experimental environment's control, enabling customization and improving the experiments' reliability.

7.1.3 Lesson 3: Tailor the Technology to the Target Environment

This topic defines what is necessary to bring such technology to modern hospitals. Two main steps compose it: deploy the architecture and design the applications. For the first step, as mentioned before, it is required the installation of some equipment dedicated to run the middleware architecture. The hardware capacity will depend on the type of sensors. More robust sensors, such as Microsoft Kinect v2, impose specific Collector nodes' specific requirements. In particular, it requires USB 3.0 interfaces and the Microsoft Kinect API so that the Collector can extract image frames from the Kinect. In this case, nodes must be compatible with the demands of computer vision techniques.

On the other hand, as Sewio RTLS, some lightweight sensors only require the node to run its VM or a single application. Some sensors even only require the compatibility of the node with some specific language to run their APIs. Besides the nodes, the network must be private. The technology and performance should suit the amount of data to be transmitted. All nodes running the Collector instances and the server that runs the Core should connect to this private network. The server requires at least two network interfaces: one for the private network and another to connect to the hospital network. Therefore, hospital applications can process data streams from the server in any hospital area, including remote locations.

Concerning the application's design, nowadays, there are already many well-known publish-subscribe platforms that provide easy to use API's for real-time data. Our middleware is designed to be compatible with such technologies. Thus, applications are only required to use our middleware wrapper, which adds QoS arguments to the subscription calls. The applications do not need to implement any additional code but only add their parameters. As the middleware has default delay and jitter limits, they will still be compatible if they do not inform them.

7.1.4 Lesson 4: Critically Analyze the Results to Find Solutions

One of the main problems when running a distributed system is known as a bottleneck. This problem occurs when a system provides its services for many clients through the same node or module. As the number of clients increases, the amount of work this particular node must perform also increases. However, due to hardware limitations, this node can experience overload situations. That is, the node cannot handle the amount of work, which affects its performance directly. By dividing the amount of work among several Collector instances, HealthStack removes this problem from the sensors. Each Collector serves only one client connection from the Core. Also, the QoS Service Stacking strategy brings benefits directly to the sensors, improving data collection performance.

Looking at the study's evaluation, HealthStack can still suffer from a bottleneck at the Core level. As the experiments comprised the middleware with only one core instance, they demonstrate the increase of delay according to client applications' growth. On the bright side, the services stacked at the sensors level helped to improve the results, even when the ascending delay occurred. Besides, HealthStack also comprises resource elasticity, although the experiments did not include it. The inclusion of an additional QoS service can deal directly with the increasing effect: admission control. This strategy is common in network QoS solutions. Its main benefit is to prevent too many connections in the system. Instead of accepting all connections, client applications can only connect while the system supports them. Therefore, the system can deliver a stable performance for applications that are already connected.

As a final point, the analysis of results must consider the medical point of view. Presenting results to the medical team is an important step. They can provide a different vision of the research team's kind of information from the results. Workshops and meetings with the medical partner are part of this process. In such moments, discussion raise possible customization to guide the next steps. It highlights the main aspects that the research team should address to achieve different outcomes.

7.2 Main Contributions

Envisioning the hospitals of the future, HealthStack provides strategies to guarantee performance for real-time applications that process data from medical processes and hardware optimization. More importantly, future real-time applications will require reliability in data flows from hospital processes to perform correctly. Many actuators will support it according to real-time data processing. Thus, the importance of guaranteeing the performance of these data flows will directly impact the quality of the services delivered to patients. This study's scope presented contributions to scientific, technical, and society on behalf of hospital services. The main contributions this study introduces are listed as follows:

(*i*) A middleware model for healthcare environments with automatic QoS support for real-

time data transmission;

(*ii*) QoS strategy based on artificial neurons to select middleware components with poor performance.

The experiments demonstrate that the strategy can improve the applications' experienced jitter mean by 92.3% and delay mean by 28% for position data samples. Also, it resulted in a reduction of network, memory, and CPU consumption by up to 66.4%, 5.06%, and 48.3%, respectively. The results are encouraging and demonstrate the importance of HealthStack. Besides the technical contributions, the solution offers a new level of reliability to time-critical applications directly impacting the patients' health. This research provides the improvement of medical services for patients. That also contributes to the hospital administration processes because they can access real-time data with QoS guarantees. The main challenges in the research roadmap were how to deploy such a system in a private hospital without impacting existing technologies. For that reason, the best option was to deploy a private communication network to avoid concurrence with hospital systems traffic. In a production environment, HealthStack requires the configuration of components individually. Once running, it is possible to change configurations remotely through the central core component that should run in a dedicated server, preferably.

7.3 Publications

This section lists all articles published and submitted to evaluation during the whole research period. These articles comprise results from this current research, which is part of a research project of the university, and results from other studies in the project's scope. The following three published articles contain the concepts presented in this study:

1. **Title**: Exploring Publish/Subscribe, Multilevel Cloud Elasticity, and Data Compression in Telemedicine
   Journal: Computer Methods and Programs in Biomedicine
   Impact Factor: 3.632
   DOI: http://dx.doi.org/10.1016/j.cmpb.2020.105403
   Citation: **RODRIGUES, V. F.**; PAIM, E. P.; KUNST, R.; ANTUNES, R. S.; COSTA, C. A. da; RIGHI, R. R.. *Exploring Publish/Subscribe, Multilevel Cloud Elasticity, and Data Compression in Telemedicine*. Computer Methods and Programs in Biomedicine, v. 191, p. 105403, 2020.

2. **Title**: On Providing Multi-Level Quality of Service for Operating Rooms of the Future
   Journal: SENSORS
   Impact Factor: 3.275
   DOI: https://doi.org/10.3390/s19102303

Citation: **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. da; ESKOFIER, B.; MAIER, A.. *On Providing Multi-Level Quality of Service for Operating Rooms of the Future*. SENSORS, v. 19, p. 2303-27, 2019.

3. **Title**: A Survey of Sensors in Healthcare Workflow Monitoring
   Journal: ACM Computing Surveys
   Impact Factor: 7.990
   DOI: https://doi.org/10.1145/3177852
   Citation: ANTUNES, R. S.; SEEWALD, L. A.; **RODRIGUES, V. F.**; COSTA, C. A. da; JUNIOR, L. G. S.; RIGHI, R. R.; MAIER, A.; ESKOFIER, B.; OLLENSCHLAGER, M.; NADERI, F.; FAHRIG, R.; BAUER, S.; KLEIN, S.; CAMPANATTI, G.. *A Survey of Sensors in Healthcare Workflow Monitoring*. ACM Computing Surveys, v. 51, p. 1-37, 2018.

Some other two articles that also contain the concepts of the current study are currently submitted for evaluation:

1. **Title**: Proactiveness, Global Analysis, and Artificial Intelligence: There Future Asks for QoS in Healthcare 4.0
   Journal: IEEE Technology and Society Magazine
   Impact Factor: 1.256
   DOI: -
   Citation: **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. DA C.; ANTUNES, R. S.. *Proactiveness, Global Analysis, and Artificial Intelligence: There Future Asks for QoS in Healthcare 4.0*. IEEE Technology and Society Magazine.

2. **Title**: HealthStack: Providing an IoT Middleware for Malleable QoS Service Stacking for Hospital 4.0 Operating Rooms
   Journal: ACM Transactions on Computing for Healthcare
   Impact Factor: -
   DOI: -
   Citation: **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. DA C.; ANTUNES, R. S.; BAZO, R.; REIS, E. S.; SEEWALD, L. A.; JUNIOR, L. G.; ESKOFIER, B.. *HealthStack: Providing an IoT Middleware for Malleable QoS Service Stacking for Hospital 4.0 Operating Rooms*. ACM Transactions on Computing for Healthcare.

Besides the articles related to this study, several other articles were produced in the project's scope and studies during the research period. The following list presents the articles already published:

1. **Title**: Use of Internet of Things With Data Prediction on Healthcare Environments
   Journal: International Journal of E-Health and Medical Communications (IJEHMC)

Impact Factor: 0.570
DOI: http://dx.doi.org/10.4018/ijehmc.2020040101
Citation: FISCHER, G. S.; RIGHI, R. R.; **RODRIGUES, V. F.**; COSTA, C. A. da. *Use of Internet of Things With Data Prediction on Healthcare Environments*. International Journal of E-Health and Medical Communications, v. 11, p. 1-19, 2020.

2. **Title**: Looking at Fog Computing for E-Health through the Lens of Deployment Challenges and Applications
   Journal: SENSORS
   Impact Factor: 3.275
   DOI: http://dx.doi.org/10.3390/s20092553
   Citation: VILELA, P. H.; RODRIGUES, J. J. P. C.; RIGHI, R. R.; KOZLOV, S.; **RODRIGUES, V. F.**. *Looking at Fog Computing for E-Health through the Lens of Deployment Challenges and Applications*. SENSORS, v. 20, p. 2553, 2020.

3. **Title**: Baptizo: A sensor fusion based model for tracking the identity of human poses
   Journal: Information Fusion
   Impact Factor: 13.669
   DOI: https://doi.org/10.1016/j.inffus.2020.03.011
   Citation: BAZO, R.; REIS, E.; SEEWALD, L. A.; **RODRIGUES, V. F.**; COSTA, C. A. da; JUNIOR, L. G.; ANTUNES, R. S.; RIGHI, R. R.; MAIER, A.; ESKOFIER, B.; FAHRIG, R.; HORZ, T.. *Baptizo: A sensor fusion based model for tracking the identity of human poses*. Information Fusion, v. 62, p. 1-13, 2020.

4. **Title**: Towards providing middleware-level proactive resource reorganisation for elastic HPC applications in the cloud
   Journal: International Journal of Grid and Utility Computing
   Impact Factor: 1.890
   DOI: https://doi.org/10.1504/IJGUC.2019.097220
   Citation: RIGHI, R. R.; **RODRIGUES, V. F.**; NARDIN, I. F.; COSTA, C. A.; ALVES, M. A. Z.; PILLON, M. A.. *Towards providing middleware-level proactive resource reorganisation for elastic HPC applications in the cloud*. International Journal of Grid and Utility Computing, v. 10, p. 76-92, 2019.

5. **Title**: Pipel: Exploiting Resource Reorganization to Optimize Performance of Pipeline-Structured Applications in the Cloud
   Journal: International Journal of Computational Systems Engineering
   Impact Factor: -
   DOI: https://doi.org/10.1504/IJCSYSE.2019.098414
   Citation: MEYER, V.; **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A.; GALANTE, G.; BOTH, C. B.. *Pipel: Exploiting Resource Reorganization to Optimize Performance*

of Pipeline-Structured Applications in the Cloud. International Journal of Computational Systems Engineering, v. 5, p. 1-14, 2019.

6. **Title**: ElBench: a microbenchmark to evaluate virtual machine and container strategies on executing elastic applications in the cloud
   Journal: International Journal of Computational Science and Engineering
   Impact Factor: 0.890
   DOI: https://dx.doi.org/10.1504/IJCSE.2020.106068
   Citation: RIGHI, R. R.; COSTA, C. A. da; YAMIN, A. C.; **RODRIGUES, V. F.**; BRAUNER, D.. *ElBench: a microbenchmark to evaluate virtual machine and container strategies on executing elastic applications in the cloud*. International Journal of Computational Science and Engineering, v. 1, p. 1-16, 2019.

7. **Title**: Elastic-RAN: An adaptable multi-level elasticity model for cloud radio access networks
   Journal: Computer Communications
   Impact Factor: 2.816
   DOI: https://doi.org/10.1016/j.comcom.2019.04.012
   Citation: RIGHI, R. R.; ANDRIOLI, L.; **RODRIGUES, V. F.**; COSTA, C. A. da; ALBERTI, A. M.; SINGH, D.. *Elastic-RAN: An adaptable multi-level elasticity model for cloud radio access networks*. Computer Communications, v. 3, p. 67-81, 2019.

8. **Title**: A systematic literature review of data forecast and internet of things on the e-health landscape
   Journal: International Journal of Computational Medicine and Healthcare
   Impact Factor: -
   DOI: https://doi.org/10.1504/IJCMH.2019.104359
   Citation: FISCHER, G. S.; RIGHI, R. R.; COSTA, C. A. da; **RODRIGUES, V. F.**. *A systematic literature review of data forecast and internet of things on the e-health landscape*. International Journal of Computational Medicine and Healthcare, v. 1, p. 34, 2019.

9. **Title**: Towards Characterizing Architecture and Performance in Blockchain: A Survey
   Journal: International Journal of Blockchains and Cryptocurrencies
   Impact Factor: -
   DOI: https://dx.doi.org/10.1504/IJBC.2020.109002
   Citation: FURTADO, F. R.; SILVA, J. V. S. e; CAPPELARI, M. J.; CASTILHOS, C. H. M.; **RODRIGUES, V. F.**; COSTA, C. A. da; RIGHI, R. R.. *Towards Characterizing Architecture and Performance in Blockchain: A Survey*. International Journal of Blockchains and Cryptocurrencies, v. 1, p. 1-33, 2019.

10. **Title**: MigPF: Towards on self-organizing process rescheduling of Bulk-Synchronous Parallel applications

Journal: Future Generation Computer Systems
Impact Factor: 6.125
DOI: https://doi.org/10.1016/j.future.2016.05.004
Citation: RIGHI, R. R.; GOMES, R. Q.; **RODRIGUES, V. F.**; COSTA, C. A. da; ALBERTI, A. M.; PILLA, L. L.; NAVAUX, P. O. A.. *MigPF: Towards on self-organizing process rescheduling of Bulk-Synchronous Parallel applications*. Future Generation Computer Systems, v. 78, p. 272-286, 2018.

11. **Title**: Toward analyzing mutual interference on infrared-enabled depth cameras
    Journal: Computer Vision and Image Understanding
    Impact Factor: 3.121
    DOI: https://doi.org/10.1016/j.cviu.2018.09.010
    Citation: SEEWALD, L. A.; **RODRIGUES, V. F.**; OLLENSCHLÄGER, M.; ANTUNES, R. S.; COSTA, C. A. da; RIGHI, R. R.; JUNIOR, L. G.; MAIER, A.; ESKOFIER, B.; FAHRIG, R.. *Toward analyzing mutual interference on infrared-enabled depth cameras*. Computer Vision and Image Understanding, v. 178, p. 1-15, 2018.

12. **Title**: Towards Combining Reactive and Proactive Cloud Elasticity on Running HPC Applications
    Journal: Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security
    Impact Factor: -
    DOI: https://doi.org/10.5220/0006761302610268
    Citation: **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. da; SINGH, D.; MUNOZ, V. M.; CHANG, V.. *Towards Combining Reactive and Proactive Cloud Elasticity on Running HPC Applications*. In: 3rd International Conference on Internet of Things, Big Data and Security, 2018, Funchal. Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security, p. 261, 2018.

13. **Title**: Combinando Elasticidade Reativa e Proativa para Aumentar o Desempenho de Aplicações HPC
    Journal: Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul
    Impact Factor: -
    DOI: -
    Citation: **RODRIGUES, V. F.**; RIGHI, R. R.. *Combinando Elasticidade Reativa e Proativa para Aumentar o Desempenho de Aplicações HPC*. In: XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS 2018), 2018, Porto Alegre. Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS 2018), 2018. v. 18.

14. **Title**: A lightweight plug-and-play elasticity service for self-organizing resource provisioning on parallel applications
    Journal: Future Generation Computer Systems
    Impact Factor: 6.125
    DOI: https://doi.org/10.1016/j.future.2017.02.023
    Citation: RIGHI, R. R.; **RODRIGUES, V. F.**; ROSTIROLLA, G.; COSTA, C. A. da; ROLOFF, E.; NAVAUX, P. O. A.. *A lightweight plug-and-play elasticity service for self-organizing resource provisioning on parallel applications*. Future Generation Computer Systems, v. 78, p. 176-190, 2018.

15. **Title**: On exploring proactive cloud elasticity for internet of things demands
    Journal: 2017 XLIII Latin American Computer Conference (CLEI)
    Impact Factor: -
    DOI: https://doi.org/10.1109/CLEI.2017.8226417
    Citation: **RODRIGUES, V. F.**; CORREA, E.; COSTA, C. A. da; RIGHI, R. R.. *On exploring proactive cloud elasticity for internet of things demands*. In: 2017 XLIII Latin American Computer Conference (CLEI), 2017, Córdoba. 2017 XLIII Latin American Computer Conference (CLEI), 2017. p. 1-65.

16. **Title**: Towards Enabling Live Thresholding as Utility to Manage Elastic Master-Slave Applications in the Cloud
    Journal: Journal of Grid Computing
    Impact Factor: 2.095
    DOI: https://doi.org/10.1007/s10723-017-9405-3
    Citation: **RODRIGUES, V. F.**; RIGHI, R. R.; ROSTIROLLA, G.; BARBOSA, J. L. V.; COSTA, C. A. da; ALBERTI, A. M.; CHANG, V.. *Towards Enabling Live Thresholding as Utility to Manage Elastic Master-Slave Applications in the Cloud*. Journal of Grid Computing, v. 11, p. 101-125, 2017.

17. **Title**: Brokel: Towards enabling multi-level cloud elasticity on publish/subscribe brokers
    Journal: International Journal of Distributed Sensor Networks
    Impact Factor: 1.151
    DOI: https://doi.org/10.1177/1550147717728863
    Citation: **RODRIGUES, V. F.**; WENDT, I. G.; RIGHI, R. R.; COSTA, C. A. da; BARBOSA, J. L. V.; ALBERTI, A. M.. *Brokel: Towards enabling multi-level cloud elasticity on publish/subscribe brokers*. International Journal of Distributed Sensor Networks, v. 13, p. 155014771772886, 2017.

18. **Title**: MMEliot: Um Modelo para Internet das Coisas Explorando a Elasticidade da Computação em Nuvem
    Journal: Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comu-

nicação
Impact Factor: -
DOI: http://dx.doi.org/10.5281/zenodo.877328
Citation: **RODRIGUES, V. F.**; YVES, T.; RIGHI, R. R.; COSTA, C. A.. *MMEliot: Um Modelo para Internet das Coisas Explorando a Elasticidade da Computação em Nuvem*. Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação, v. 1, p. 1, 2017.

19. **Title**: On providing on-the-fly resizing of the elasticity grain when executing HPC applications in the cloud
Journal: International Journal of Computational Science and Engineering
Impact Factor: 0.890
DOI: https://dx.doi.org/10.1504/IJCSE.2019.104432
Citation: RIGHI, R. R.; **RODRIGUES, V. F.**; CUNHA, L. F. S.; COSTA, C. A.. *On providing on-the-fly resizing of the elasticity grain when executing HPC applications in the cloud*. International Journal of Computational Science and Engineering, v. 1, p. 1-18, 2017.

Finally, the following articles are still in evaluation for publishing or already accepted for publication:

1. **Title**: ChainElastic: A Cloud Computing Elasticity Model for IoT-based Blockchain Applications
Journal: International Journal of Blockchains and Cryptocurrencies
Impact Factor: -
DOI: -
Citation: **RODRIGUES, V. F.**; SILVA, J. V. S. e; RIGHI, R. R.; COSTA, C. A. da; ROEHRS, A.; *ChainElastic: A Cloud Computing Elasticity Model for IoT-based Blockchain Applications*. International Journal of Blockchains and Cryptocurrencies. (ACCEPTED)

2. **Title**: A Survey on Single-View Multi-Person Pose Estimation
Journal: Elsevier Pattern Recognition
Impact Factor: 7.196
DOI: -
Citation: REIS, E. SOUZA DOS; SEEWALD, L. A.; ANTUNES, R. S.; BAZO, R.; **RODRIGUES, V. F.**; RIGHI, R. R.; COSTA, C. A. da; JUNIOR, L. G. SILVEIRA; ESKOFIER, B.; MAIER, A.; HORZ, T.; FAHRIG, R.. *A Survey on Single-View Multi-Person Pose Estimation*. Elsevier Pattern Recognition.

3. **Title**: A Survey About Real-Time Location Systems in Healthcare Environments
Journal: Journal of Medical Systems

Impact Factor: 3.058
DOI: -
Citation: BAZO, R.; COSTA, C. A. da; SEEWALD, L. A.; JUNIOR, L. G.; ANTUNES, R. S.; RIGHI, R. R.; **RODRIGUES, V. F.**. *A Survey About Real-Time Location Systems in Healthcare Environments*. Journal of Medical Systems.

7.4 Limitations and Future Work

HealthStack presents some limitations which might be explored in future researches. First, the QoS Manager chooses which component receives QoS services based only on their current status. It does not take into account past events, which would improve the decision-making process. Every time a QoS violation occurs, the Manager computes the weights and the Potential of Adaption for all components. To do that, the Manager considers as input the current measures from each application and middleware component. Pattern recognition and data forecasting strategies could improve this process providing a more proactive method. Second, the QoS Model does not assess the effect of QoS services on-the-fly. In other words, HealthStack does not keep track of the impact of delivering each type of QoS service. By having this information, every time the QoS Manages must deliver a new QoS service, it could verify past events to choose the most appropriate. For instance, if a specific QoS service always brings drawbacks to a specific data type, it would be useful to avoid employing it in similar situations.

A third limitation regards the specific QoS metrics HealthStack employs. The model employs only delay and jitter. Several other metrics could be considered as packet loss, for instance. However, the design decisions of HealthStack consider a reliable end-to-end network infrastructure, and for that reason, other metrics are not considered. Although covering two essential metrics, other options could provide more flexibility to the user when choosing their target threshold. Fourth, HealthStack takes a high number of human-defined parameters that can change the middleware's behavior. Some parameters only change the monitoring intervals and are easy to set. However, several parameters of the QoS Service Stacking strategy can change its behavior considerably. The current research proposed the values for each parameter that can be used to deploy HealthStack. Though, it is essential to keep in mind that HealthStack allows their modification. On the bright side, this allows the algorithm tunning for different scenarios.

In terms of evaluation, the QoS Manager only adds QoS services to the component stacks, never removing them. That happens because the experiments employ a batch of user applications that never disconnect the middleware. Therefore, components are always generating data for at least one application in the whole execution time. Besides that, the prototype covers a limited set of QoS services. Although HealthStack proposes four QoS services, the Resource Elasticity was not implemented in the experiments. This particular QoS service can bring different outcomes than the other ones because it allows resizing resources to execute the middleware components. However, it requires the infrastructure to offer hosts with a high capacity of re-

sources supporting vertical elasticity. Unfortunately, that is not always the case in hospitals and IoT systems. Considering the Healthstack limitations, the continuation of this thesis consists of studies in the following directions:

(*i*) Use of new heuristics in the QoS model to define the services for user applications and sensors;

(*ii*) Development of new services and metrics in the architecture modules;

(*iii*) Evaluation of HealthStack with other types of sensors;

(*iv*) Explore horizontal elasticity strategies for the service layer, proposing load balancing techniques.

Currently, the QoS model from the HealthStack QoS Manager defines a QoS Service Stacking strategy that takes several parameters. Future research can consider changing the strategy to select the components to receive QoS services. It would enable a comparison of different strategies and how they affect the middleware. There is a possibility to change this strategy by a different heuristic to select the services for user applications and sensors. For instance, it is possible to change it by fuzzy logic, strategy considering the QoS Manager's input variables. In the case of QoS parameters, it is possible to define new metrics and services in the model. It requires new metrics definitions and which components generate them. One example is to add the disc usage metric, which could be extracted from the operating system of each node running a HealthStack process. Additionally, adapting the QoS service according to historical data is also a promising strategy. Proactive decision-making is possible by analyzing past actions and their effect. This analysis could improve the process of defining the best actions in each overload situation.

Next, it is possible to analyze the QoS model parameters to identify their effect on different scenarios. Also, as the model comprises a limited number of services, future studies can investigate further services, including, for instance, admission control and multi-channel data transmission. Besides, the experiments of this research focused on only two types of sensor data sources. It is worthy to assess HealthStack against other types of sensors. Specifically, tags position and depth image are the raw data the Collectors extract from the sensors. An idea for future research would include the evaluation of the model considering a different set of data. It would be possible to acquire data directly from medical sensors, such as blood pressure and heartbeat information. Currently, camera RGB image frames are not valid in healthcare since medical environments have high privacy constraints, and patients' and medical staff's identification is sensitive information. Therefore, this type of data is not in the model, as in its evaluation.

Finally, HealthStack proposes as one of its services vertical elasticity. In future research, it is possible to explore the horizontal model of elasticity. In this regard, load balancing strategies

and new strategies for communication reconfiguration are the main challenges. A load balance process would be necessary to distribute user application connections among more than one instance of the PubSub Broker. Besides, replication of the Core process requires distribution of the Collector instances to avoid transmission and storing replicated data.

REFERENCES

ACETO, G.; PERSICO, V.; PESCAPé, A. Industry 4.0 and health: internet of things, big data, and cloud computing for healthcare 4.0. **Journal of Industrial Information Integration**, Amsterdam, Netherlands, v. 18, p. 100129, 2020.

ADAME, T. et al. Cuidats: an rfid-wsn hybrid monitoring system for smart health care environments. **Future Generation Computer Systems**, Amsterdam, Netherlands, v. 78, p. 602 – 615, 2018.

AFFERNI, P.; MERONE, M.; SODA, P. Hospital 4.0 and its innovation in methodologies and technologies. In: IEEE INTERNATIONAL SYMPOSIUM ON COMPUTER-BASED MEDICAL SYSTEMS (CBMS), 31., 2018, Karlstad, Sweden. **Proceedings...** IEEE, 2018. p. 333–338.

AGIRRE, A. et al. Qos management for dependable sensory environments. **Multimedia Tools and Applications**, Amsterdam, Netherlands, v. 75, n. 21, p. 13397–13419, Nov 2016.

AHMED, T.; LE MOULLEC, Y. A qos optimization approach in cognitive body area networks for healthcare applications. **Sensors**, Basel, Switzerland, v. 17, n. 4, 2017.

AL-TARAWNEH, L. A. Medical grade qos improvement using ieee802.11e wlan protocol. In: AMERICAN UNIVERSITY IN THE EMIRATES INTERNATIONAL RESEARCH CONFERENCE, 1., 2019, Dubai, UAE. **Proceedings...** Springer International Publishing, 2019. p. 229–235.

ALBAHRI, O. et al. Fault-tolerant mhealth framework in the context of iot-based real-time wearable health data sensors. **IEEE Access**, New York, NY, USA, v. 7, p. 50052–50080, 2019.

ALEMDAR, H.; ERSOY, C. Wireless sensor networks for healthcare: a survey. **Computer Networks**, Amsterdam, Netherlands, v. 54, n. 15, p. 2688–2710, 2010.

ALLIANCE, Z. **What is zigbee**. Available in: `http://www.zigbee.org/what-is-zigbee`. Accessed in: October 30th 2020.

ANTUNES, R. S. et al. A survey of sensors in healthcare workflow monitoring. **ACM Comput. Surv.**, New York, NY, USA, v. 51, n. 2, p. 42:1–42:37, Apr. 2018.

BAI, T. et al. An optimized protocol for qos and energy efficiency on wireless body area networks. **Peer-to-Peer Networking and Applications**, New York, NY, USA, v. 12, n. 2, p. 326–336, Mar 2019.

BAIG, M. M.; HOSSEINI, H. G.; LINDÉN, M. Machine learning-based clinical decision support system for early diagnosis from real-time physiological data. In: IEEE REGION 10 CONFERENCE (TENCON), 2016., 2016, Singapore, Singapore. **Proceedings...** IEEE, 2016. p. 2943–2946.

BANOUAR, Y. et al. Qos management mechanisms for enhanced living environments in iot. In: IFIP/IEEE SYMPOSIUM ON INTEGRATED NETWORK AND SERVICE MANAGEMENT (IM), 2017., 2017, Lisbon, Portugal. **Proceedings...** IEEE, 2017. p. 1155–1161.

GRZEGORZEK, M. et al. (Ed.). Real-time range imaging in health care: a survey. In: _____. **Time-of-flight and depth imaging. sensors, algorithms, and applications**: dagstuhl 2012 seminar on time-of-flight imaging and gcpr 2013 workshop on imaging new modalities. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 228–254.

BERNSTEIN, D. S. **Matrix mathematics**: theory, facts, and formulas (second edition). Princeton, New Jersey, USA: Princeton University Press, 2009. (Princeton reference).

BIOLCHINI, J. et al. **Systematic review in software engineering**. Rio de Janeiro, Brazil: System Engineering and Computer Science Department COPPE/UFRJ, 2005.

Bluetooth. **How it works**. Available in: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works. Accessed in: October 30th 2020.

Bluetooth. **Bluetooth low energy**. Available in: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/le-p2p. Accessed in: October 30th 2020.

BONDI, A. B. Characteristics of scalability and their impact on performance. In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, 2., 2000, New York, NY, USA. **Proceedings…** Association for Computing Machinery, 2000. p. 195–203. (WOSP '00).

BOULOS, M. N. K.; BERRY, G. Real-time locating systems (rtls) in healthcare: a condensed primer. **International journal of health geographics**, London, United Kingdom, v. 11, n. 1, p. 25, 2012.

BRADAI, N. et al. Qos architecture over wbans for remote vital signs monitoring applications. In: ANNUAL IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE (CCNC), 12., 2015, Las Vegas, NV, USA. **Proceedings…** IEEE, 2015. p. 1–6.

CELDRÁN, A. H. et al. Ice++: improving security, qos, and high availability of medical cyber-physical systems through mobile edge computing. In: IEEE INTERNATIONAL CONFERENCE ON E-HEALTH NETWORKING, APPLICATIONS AND SERVICES (HEALTHCOM), 20., 2018, Ostrava, Czech Republic. **Proceedings…** IEEE, 2018. p. 1–8.

CHEATLE, A. et al. Sensing (co)operations: articulation and compensation in the robotic operating room. **Proc. ACM Hum.-Comput. Interact.**, New York, NY, USA, v. 3, n. CSCW, Nov. 2019.

CHIU, D.; AGRAWAL, G. Evaluating caching and storage options on the amazon web services cloud. In: IEEE/ACM INTERNATIONAL CONFERENCE ON GRID COMPUTING, 11., 2010, Brussels, Belgium. **Proceedings…** IEEE, 2010. p. 17–24.

CHUA, C. L.; REN, H.; ZHANG, W. Towards a touchless master console for natural interactions in sterilized and cognitive robotic surgery environments. In: _____. **Robot intelligence technology and applications 2**: results from the 2nd international conference on robot intelligence technology and applications. New York, NY, USA: Springer International Publishing, 2014. p. 785–795.

COSTA, C. A. da et al. Internet of health things: toward intelligent vital signs monitoring in hospital wards. **Artificial Intelligence in Medicine**, Amsterdam, Netherlands, v. 89, p. 61 – 69, 2018.

DAS, S.; PERKINS, C.; ROYER, E. Ad hoc on demand distance vector (aodv) routing. **IETF RFC3561, July**, Santa Barbara, CA, USA, 2003.

DECIA, I. et al. Camacua: low cost real time risk alert and location system for healthcare environments. In: LATIN AMERICAN CONGRESS ON BIOMEDICAL ENGINEERING CLAIB, 7., 2017, Singapore. **Proceedings...** Springer Singapore, 2017. p. 90–93.

DJELOUAT, H. et al. Real-time ecg monitoring using compressive sensing on a heterogeneous multicore edge-device. **Microprocessors and Microsystems**, Amsterdam, Netherlands, v. 72, p. 102839, 2020.

ELLIS, C. A. Workflow technology. In: **Computer supported cooperative work**. Chichester, UK: John Wiley and Sons, 1999. p. 25–54.

ETSI. Etr003: network aspects (na); general aspects of quality of service (qos) and network performance (np). **ETSI Technical Report**, Valbonne, France, October 1994.

FANG, R. et al. Computational health informatics in the big data age: a survey. **ACM Computing Surveys**, New York, NY, USA, v. 49, n. 1, p. 12:1–12:36, 2016.

FARAHANI, S. **Zigbee wireless networks and transceivers**. Burlington, MA, USA: Newnes, 2008.

FEKI, M. A. et al. The internet of things: the next technological revolution. **Computer**, Piscataway, NJ, USA, v. 46, n. 2, p. 24–25, 2013.

THUEMMLER, C.; BAI, C. (Ed.). Surgery 4.0. In: _____. **Health 4.0**: how virtualization and big data are revolutionizing healthcare. Cham: Springer International Publishing, 2017. p. 91–107.

FOIX, S.; ALENYA, G.; TORRAS, C. Lock-in time-of-flight (tof) cameras: a survey. **IEEE Sensors Journal**, Piscataway, NJ, USA, v. 11, n. 9, p. 1917–1926, 2011.

FüRSATTEL, P. et al. A comparative error analysis of current time-of-flight sensors. **IEEE Transactions on Computational Imaging**, Piscataway, NJ, USA, v. 2, n. 1, p. 27–41, 2016.

GALANTE, G.; BONA, L. C. E. de. A survey on cloud computing elasticity. In: IEEE INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 5., 2012, Chicago, IL, USA. **Proceedings...** IEEE, 2012. p. 263–270.

GATOUILLAT, A.; BADR, Y.; MASSOT, B. Qos-driven self-adaptation for critical iot-based systems. In: SERVICE-ORIENTED COMPUTING – ICSOC 2017 WORKSHOPS, 2018, Cham. **Proceedings...** Springer International Publishing, 2018. p. 93–105.

GHANBARI, H. et al. Exploring alternative approaches to implement an elasticity policy. In: IEEE INTERNATIONAL CONFERENCE ON CLOUD COMPUTING (CLOUD), 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE, 2011. p. 716–723.

GIATRAKOS, N. et al. Interactive extreme-scale analytics: towards battling cancer. **IEEE Technology and Society Magazine**, Piscataway, NJ, USA, v. 38, n. 2, p. 54–61, 2019.

GOYAL, R. et al. An energy efficient qos supported optimized transmission rate technique in wbans. **Wireless Personal Communications**, Amsterdam, Netherlands, p. 1–26, 2020.

GUEZGUEZ, M. J.; REKHIS, S.; BOUDRIGA, N. A sensor cloud for the provision of secure and qos-aware healthcare services. **Arabian Journal for Science and Engineering**, Berlin, Germany, v. 43, n. 12, p. 7059–7082, Dec 2018.

HARTMANN, F.; SCHLAEFER, A. Feasibility of touch-less control of operating room lights. **International Journal of Computer Assisted Radiology and Surgery**, Berlin, Germany, v. 8, n. 2, p. 259–268, 2013.

HASSAN, M. M.; ALRUBAIAN, M.; ALAMRI, A. Effective qos-aware novel resource allocation model for body sensor-integrated cloud platform. In: INTERNATIONAL CONFERENCE ON ADVANCED COMMUNICATION TECHNOLOGY (ICACT), 18., 2016, Pyeongchang, South Korea. **Proceedings...** IEEE, 2016. p. 596–601.

HAZLEHURST, B. et al. How the icu follows orders: care delivery as a complex activity system. In: AMIA ANNUAL SYMPOSIUM PROCEEDINGS (AMIA), 2003. **Proceedings...** National Institute of Health, 2003. p. 284–288.

HERBST, N. R. et al. Self-adaptive workload classification and forecasting for proactive resource provisioning. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 4., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p. 187–198. (ICPE '13).

HU, L. et al. Design of qos-aware multi-level mac-layer for wireless body area network. **Journal of Medical Systems**, New York, NY, USA, v. 39, n. 12, p. 192, Oct 2015.

HUSSMANN, S.; HAGEBEUKER, B.; RINGBECK, T. **A performance review of 3d tof vision systems in comparison to stereo vision systems**. London, UK: INTECH Open Access Publisher, 2008.

HYNDMAN, R. J.; ATHANASOPOULOS, G. **Forecasting**: principles and practice. Monash University, Australia: OTexts, 2018.

IBRAHIM, A. A. et al. Weighted energy and qos based multi-hop transmission routing algorithm for wban. In: INTERNATIONAL ENGINEERING CONFERENCE "SUSTAINABLE TECHNOLOGY AND DEVELOPMENT" (IEC), 6., 2020, Erbil, Iraq. **Proceedings...** IEEE, 2020. p. 191–195.

IEEE Computer Society. **Ieee standard for information technology – local and metropolitan area networks – specific requirements – part 15.1a**: wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpan). Piscataway, New Jersey, USA: IEEE Computer Society, 2005.

IEEE Computer Society. **Ieee standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements – part 11**: wireless lan medium access control (mac) and physical layer (phy) specifications. Piscataway, New Jersey, USA: IEEE Computer Society, 2007.

IEEE Computer Society. **Ieee standard for low-rate wireless networks**. Piscataway, New Jersey, USA: IEEE Computer Society, 2016.

IMAI, S.; CHESTNA, T.; VARELA, C. A. Elastic scalable cloud computing using application-level migration. In: IEEE INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 5., 2012, Chicago, IL, USA. **Proceedings...** IEEE, 2012. p. 91–98.

IRANMANESH, S. A.; RIZI, F. Y. Qos provisioning for multiple co-existing body sensor networks. In: IRANIAN CONFERENCE ON ELECTRICAL ENGINEERING (ICEE), 2018, Mashhad, Iran. **Proceedings...** IEEE, 2018. p. 1595–1600.

ISLAM, S. R. et al. The internet of things for health care: a comprehensive survey. **IEEE Access**, New York, NY, USA, v. 3, p. 678–708, 2015.

ITUT. E. 800, definitions of terms related to quality of service. **International Telecommunication Union's Telecommunication Standardization Sector (ITU-T) Std**, Switzerland, 2008.

JACOB, A. K.; JACOB, L. Energy efficient mac for qos traffic in wireless body area network. **International Journal of Distributed Sensor Networks**, Thousand Oaks, CA, USA, v. 11, n. 2, p. 404182, 2015.

JAGADISH, H. V. et al. Big data and its technical challenges. **Communications of the ACM**, New York, NY, USA, v. 57, n. 7, p. 86–94, 2014.

KADKHODAMOHAMMADI, A. et al. Articulated clinician detection using 3d pictorial structures on rgb-d data. **Medical Image Analysis**, London, United Kingdom, v. 35, p. 215 – 224, 2017.

KHALIL, A.; MBAREK, N.; TOGNI, O. Iot service qos guarantee using qbaiot wireless access method. In: INTERNATIONAL CONFERENCE ON MOBILE, SECURE, AND PROGRAMMABLE NETWORKING, 2019, Cham. **Proceedings...** Springer International Publishing, 2019. p. 157–173.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing systematic literature reviews in software engineering**. Keele, Newcastle, United Kingdom: Keele University, 2007.

KOLAKOWSKI, J.; DJAJA-JOSKO, V.; KOLAKOWSKI, M. Uwb monitoring system for aal applications. **Sensors**, Basel, Switzerland, v. 17, n. 9, 2017.

KOPETZ, H. The real-time environment. In: ______. **Real-time systems**: design principles for distributed embedded applications. Boston, MA: Springer USA, 1997. p. 1–28.

KOPETZ, H.; OCHSENREITER, W. Clock synchronization in distributed real-time systems. **IEEE Transactions on Computers**, Washington, DC, USA, v. C-36, n. 8, p. 933–940, Aug 1987.

KURMOO, Y. et al. Real time monitoring of biofilm formation on coated medical devices for the reduction and interception of bacterial infections. **Biomater. Sci.**, London, United Kingdom, v. 8, p. 1464–1477, 2020.

LEE, C.-J.; JUNG, J.-Y.; LEE, J.-R. Bio-inspired distributed transmission power control considering qos fairness in wireless body area sensor networks. **Sensors**, Basel, Switzerland, v. 17, n. 10, 2017.

LEE, H. et al. Wireless lan with medical-grade qos for e-healthcare. **Journal of Communications and Networks**, Seoul, Republic of Korea, v. 13, n. 2, p. 149–159, 2011.

LEE, J. S.; SU, Y. W.; SHEN, C. C. A comparative study of wireless protocols: bluetooth, uwb, zigbee, and wi-fi. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY (IECON), 33., 2007, Taipei, Taiwan. **Proceedings...** IEEE, 2007. p. 46–51.

LIM, H. C. et al. Automated control in cloud computing: challenges and opportunities. In: WORKSHOP ON AUTOMATED CONTROL FOR DATACENTERS AND CLOUDS, 1., 2009, New York, NY, USA. **Proceedings...** ACM, 2009. p. 13–18. (ACDC '09).

LINDSEY, W. C.; SIMON, M. K. **Telecommunication systems engineering**. Chelmsford, MA, USA: Courier Corporation, 1991.

LIU, B.; YAN, Z.; CHEN, C. W. Medium access control for wireless body area networks with qos provisioning and energy efficient design. **IEEE Transactions on Mobile Computing**, Washington, DC, USA, v. 16, n. 2, p. 422–434, Feb 2017.

LIU, J. et al. Towards real-time indoor localization in wireless sensor networks. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY (ICCIT), 12., 2012, Chengdu, China. **Proceedings...** IEEE, 2012. p. 877–884.

LIU, Y. et al. Prominent coagulation disorder is closely related to inflammatory response and could be as a prognostic indicator for icu patients with covid-19. **Journal of Thrombosis and Thrombolysis**, Amsterdam, Netherlands, p. 1–8, 2020.

LIU, Z.; LIU, B.; CHEN, C. W. Transmission-rate-adaption assisted energy-efficient resource allocation with qos support in wbans. **IEEE Sensors Journal**, Piscataway, NJ, USA, v. 17, n. 17, p. 5767–5780, Sep. 2017.

LOGITRACK rtls. Available in: `https://logi-tag.com/`. Accessed in: October 30th 2020.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A review of auto-scaling techniques for elastic applications in cloud environments. **Journal of Grid Computing**, Berlin, Germany, v. 12, n. 4, p. 559–592, 2014.

MAATOUGUI, E.; BOUANAKA, C.; ZEGHIB, N. Self-adaptive architecture for ensuring qos contracts in cloud-based systems. In: INTERNATIONAL CONFERENCE ON MODEL AND DATA ENGINEERING, 2017, Cham. **Proceedings...** Springer International Publishing, 2017. p. 126–134.

MACK, C. A. How to write a good scientific paper: title, abstract, and keywords. **Journal of Micro/Nanolithography, MEMS, and MOEMS**, USA, v. 11, n. 2, p. 020101, 2012.

MAGALHÃES, A. M. M. d. et al. Processos de medicação, carga de trabalho e a segurança do paciente em unidades de internação. **Revista da Escola de Enfermagem da USP**, SP, Brazil, v. 49, p. 43 – 50, 12 2015.

MAINETTI, L.; PATRONO, L.; VILEI, A. Evolution of wireless sensor networks towards the internet of things: a survey. In: INTERNATIONAL CONFERENCE ON SOFTWARE, TELECOMMUNICATIONS AND COMPUTER NETWORKS (SOFTCOM), 19., 2011, Split, Croatia. **Proceedings...** IEEE, 2011. p. 1–6.

MALHOTRA, S. et al. Workflow modeling in critical care: piecing together your own puzzle. **Journal of Biomedical Informatics**, Amsterdam, Netherlands, v. 40, n. 2, p. 81–92, 2007.

MALINDI, P.; KAHN, M. T. Providing qos for ip-based rural telemedicine systems. In: INTERNATIONAL CONFERENCE ON BROADBAND COMMUNICATIONS, INFORMATION TECHNOLOGY BIOMEDICAL APPLICATIONS, 3., 2008, Gauteng, South Africa. **Proceedings...** IEEE, 2008. p. 499–504.

MILLER, F. P.; VANDOME, A. F.; MCBREWSTER, J. **Huffman coding**: computer science, algorithm, lossless data compression, variable- length code, david a. huffman, doctor of philosophy, massachusetts institute of technology. Orlando, FL, USA: Alpha Press, 2009.

MILLS, D. L. Internet time synchronization: the network time protocol. **IEEE Transactions on Communications**, New York, NY, USA, v. 39, n. 10, p. 1482–1493, Oct 1991.

MITROKOTSA, A.; DOULIGERIS, C. Integrated rfid and sensor networks: architectures and applications. In: _____. **Rfid and sensor networks**: architectures, protocols, security and integrations. Boca Raton, FL, USA: Auerbach Publications, CRC Press, Taylor & Francis Group, 2009. p. 511–535.

MOESLUND, T. B.; HILTON, A.; KRüGER, V. A survey of advances in vision-based human motion capture and analysis. **Elsevier Computer Vision and Image Understanding**, Amsterdam, Netherlands, v. 104, n. 2, p. 90–126, 2006.

WICKRAMASINGHE, N.; BODENDORF, F. (Ed.). Intelligent risk detection in health care: integrating social and technical factors to manage health outcomes. In: _____. **Delivering superior health and wellness management with iot and analytics**. Cham: Springer International Publishing, 2020. p. 225–257.

MONTENEGRO, G. et al. **Transmission of ipv6 packets over ieee 802.15.4 networks**. Worldwide: RFC Editor, 2007. (4944).

MORENO, I. S.; XU, J. Customer-aware resource overallocation to improve energy efficiency in realtime cloud computing data centers. In: IEEE INTERNATIONAL CONFERENCE ON SERVICE-ORIENTED COMPUTING AND APPLICATIONS (SOCA), 2011., 2011, Irvine, CA, USA. **Proceedings...** IEEE, 2011. p. 1–8.

MUKHOPADHYAY, A. Qos based telemedicine technologies for rural healthcare emergencies. In: IEEE GLOBAL HUMANITARIAN TECHNOLOGY CONFERENCE (GHTC), 2017., 2017, San Jose, CA, USA. **Proceedings...** IEEE, 2017. p. 1–7.

MUSTRA, M.; DELAC, K.; GRGIC, M. Overview of the dicom standard. In: INTERNATIONAL SYMPOSIUM ELMAR, 50., 2008, Zadar, Croatia. **Proceedings...** IEEE, 2008. v. 1, p. 39–44.

NANDA, P.; FERNANDES, R. C. Quality of service in telemedicine. In: INTERNATIONAL CONFERENCE ON THE DIGITAL SOCIETY (ICDS'07), 1., 2007, Guadeloupe, France. **Proceedings...** IEEE, 2007. p. 2–2.

NFC Forum. **Nfc technology**. Available in: http://nfc-forum.org/what-is-nfc/about-the-technology. Accessed in: October 30th 2020.

NIAZKHANI, Z. et al. The impact of computerized provider order entry systems on inpatient clinical workflow: a literature review. **Journal of the American Medical Informatics Association**, Oxford, United Kingdom, v. 16, n. 4, p. 539–549, 2009.

NIELSEN, M. A. **Neural networks and deep learning**. San Francisco, CA, USA: Determination Press, 2015. v. 2018.

NOLLERT, G.; WICH, S. Planning a cardiovascular hybrid operating room: the technical point of view. **The Heart Surgery Forum**, USA, v. 12, n. 3, p. 119–124, 2009.

NYCE, C.; CPCU, A. Predictive analytics white paper. **American Institute for CPCU. Insurance Institute of America**, Malvern, PA, USA, p. 9–10, 2007.

OODAN, A. et al. **Telecommunications quality of service management**: from legacy to emerging services. London, United Kingdom: Iet, 2003. n. 48.

OZTEMEL, E.; GURSEV, S. Literature review of industry 4.0 and related technologies. **Journal of Intelligent Manufacturing**, Amsterdam, Netherlands, v. 31, n. 1, p. 127–182, 2020.

PANDIT, S. et al. An energy-efficient multiconstrained qos aware mac protocol for body sensor networks. **Multimedia Tools and Applications**, Amsterdam, Netherlands, v. 74, n. 14, p. 5353–5374, Jul 2015.

PAULY, O. et al. Machine learning-based augmented reality for improved surgical scene understanding. **Computerized Medical Imaging and Graphics**, United Kingdom, v. 41, n. 1, p. 55–60, 2015.

PONCETTE, A.-S. et al. Improvements in patient monitoring in the intensive care unit: survey study. **J Med Internet Res**, Canada, v. 22, n. 6, p. e19091, Jun 2020.

POORANI et al. Qos guarantee cloud-based remote health monitoring system. **ASCI Journal of Management**, India, v. 46, p. 25 – 51, 2017.

PORCINO, D.; HIRT, W. Ultra-wideband radio technology: potential and challenges ahead. **IEEE Communications Magazine**, USA, v. 41, n. 7, p. 66–74, 2003.

PRAMANIK, I. et al. Smart health: big data enabled health paradigm within smart cities. **Expert Systems With Applications**, United Kingdom, v. 87, n. 1, p. 370–383, 2017.

PURI, T.; CHALLA, R. K.; SEHGAL, N. K. Energy efficient qos aware mac layer time slot allocation scheme for wbasn. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS (ICACCI), 2015., 2015, Kochi, India. **Proceedings...** IEEE, 2015. p. 966–972.

RAWAT, P. et al. Wireless sensor networks: a survey on recent developments and potential synergies. **The Journal of Supercomputing**, Amsterdam, Netherlands, v. 68, n. 1, p. 1–48, 2014.

RAZZAQUE, M. A. et al. Qos-aware error recovery in wireless body sensor networks using adaptive network coding. **Sensors**, Basel, Switzerland, v. 15, n. 1, p. 440–464, 2015.

REGOLINI, J. et al. A simple room localization method to find technology in a big trauma center. In: WORLD CONGRESS ON MEDICAL PHYSICS AND BIOMEDICAL ENGINEERING 2018, 2019, Singapore. **Proceedings...** Springer Singapore, 2019. p. 321–325.

ROJAS, E. et al. Process mining in healthcare: a literature review. **Journal of Biomedical Informatics**, Amsterdam, Netherlands, v. 61, n. 1, p. 224–236, 2016.

RUTLE, A. et al. A user-friendly tool for model checking healthcare workflows. **Procedia Computer Science**, Amsterdam, Netherlands, v. 21, p. 317–326, 2013.

SAMANTA, A.; LI, Y.; CHEN, S. Qos-aware heuristic scheduling with delay-constraint for wbsns. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC), 2018., 2018, Kansas City, MO, USA. **Proceedings...** IEEE, 2018. p. 1–7.

SAMANTA, A.; MISRA, S. Dynamic connectivity establishment and cooperative scheduling for qos-aware wireless body area networks. **IEEE Transactions on Mobile Computing**, Washington, DC, USA, v. 17, n. 12, p. 2775–2788, Dec 2018.

SÁNCHEZ, D.; TENTORI, M.; FAVELA, J. Activity recognition for the smart hospital. **IEEE Intelligent Systems**, USA, v. 23, n. 2, p. 50–57, 2008.

SCHARSTEIN, D.; SZELISKI, R. High-accuracy stereo depth maps using structured light. In: IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2003, Madison, WI, USA, USA. **Proceedings...** IEEE, 2003. p. 195–202.

SCHEUNEMANN, M. M. et al. Utilizing bluetooth low energy to recognize proximity, touch and humans. In: IEEE INTERNATIONAL SYMPOSIUM ON ROBOT AND HUMAN INTERACTIVE COMMUNICATION (RO-MAN), 25., 2016, New York, NY, USA. **Proceedings...** IEEE, 2016. p. 362–367.

SCHMALZ, C. et al. An endoscopic 3d scanner based on structured light. **Medical Image Analysis**, London, United Kingdom, v. 16, n. 5, p. 1063–1072, 2012.

SHARMA, A.; ADARKAR, H.; SENGUPTA, S. Managing qos through prioritization in web services. In: INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING WORKSHOPS, 2003. PROCEEDINGS., 4., 2003, Rome, Italy. **Proceedings...** IEEE, 2003. p. 140–148.

SISINNI, E. et al. Industrial internet of things: challenges, opportunities, and directions. **IEEE Transactions on Industrial Informatics**, USA, v. 14, n. 11, p. 4724–4734, 2018.

SKORIN-KAPOV, L.; MATIJASEVIC, M. Analysis of qos requirements for e-health services and mapping to evolved packet system qos classes. **International Journal of Telemedicine and Applications**, Egypt, v. 2010, 2010.

SODHRO, A. H. et al. Mobile edge computing based qos optimization in medical healthcare applications. **International Journal of Information Management**, United Kingdom, v. 45, p. 308 – 318, 2019.

SONG, H. et al. Artificial intelligence enabled internet of things: network architecture and spectrum access. **IEEE Computational Intelligence Magazine**, USA, v. 15, n. 1, p. 44–51, 2020.

STEINMETZ, R. Human perception of jitter and media synchronization. **IEEE Journal on Selected Areas in Communications**, USA, v. 14, n. 1, p. 61–72, 1996.

SZYMANSKI, T. H.; GILBERT, D. Provisioning mission-critical telerobotic control systems over internet backbone networks with essentially-perfect qos. **IEEE Journal on Selected Areas in Communications**, USA, v. 28, n. 5, p. 630–643, 2010.

TAN, B. et al. Wi-fi based passive human motion sensing for in-home healthcare applications. In: IEEE WORLD FORUM ON INTERNET OF THINGS (WF-IOT), 2., 2015, Milan, Italy. **Proceedings...** IEEE, 2015. p. 609–614.

TAN, Y.; VENKATESH, V.; GU, X. Resilient self-compressive monitoring for large-scale hosting infrastructures. **IEEE Transactions on Parallel and Distributed Systems**, USA, v. 24, n. 3, p. 576–586, 2013.

TANENBAUM, A. S.; VAN STEEN, M. **Distributed systems**: principles and paradigms. New Jersey, USA: Prentice-Hall, 2007.

THUEMMLER, C.; BAI, C. **Health 4.0**: how virtualization and big data are revolutionizing healthcare. 1st. ed. New york, NY, USA: Springer Publishing Company, Incorporated, 2018.

TSENG, H.-W.; WANG, Y.-B.; YANG, Y. An adaptive channel hopping and dynamic superframe selection scheme with qos considerations for emergency traffic transmission in ieee 802.15.6-based wireless body area networks. **IEEE Sensors Journal**, Piscataway, NJ, USA, v. 20, n. 7, p. 3914–3929, 2020.

TVETER, D. **The pattern recognition basis of artificial intelligence**. 1st. ed. USA: IEEE Press, 1997.

VADIVEL, R.; RAMKUMAR, J. Qos-enabled improved cuckoo search-inspired protocol (icsip) for iot-based healthcare applications. In: **Incorporating the internet of things in healthcare applications and wearable devices**. USA: IGI Global, 2020. p. 109–121.

VANKIPURAM, M. et al. Toward automated workflow analysis and visualization in clinical environments. **Elsevier Journal of Biomedical Informatics**, Amsterdam, Netherlands, v. 44, n. 3, p. 432–440, 2011.

MÖLLER, S.; RAAKE, A. (Ed.). Quality of service versus quality of experience. In: ______. **Quality of experience**: advanced concepts, applications and methods. Cham: Springer International Publishing, 2014. p. 85–96.

VENKATESH, K. et al. Qos improvisation of delay sensitive communication using sdn based multipath routing for medical applications. **Future Generation Computer Systems**, Amsterdam, Netherlands, v. 93, p. 256 – 265, 2019.

WAHEED, T. et al. Qos enhancement of aodv routing for mbans. **Wireless Personal Communications**, Amsterdam, Netherlands, p. 1–28, 2020.

WALDSPURGER, C. A.; WEIHL, W. E. Lottery scheduling: flexible proportional-share resource management. In: USENIX CONFERENCE ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 1., 1994, USA. **Proceedings...** USENIX Association, 1994. p. 1–es. (OSDI '94).

WANG, J.; SUN, Y.; JI, Y. Qos-based adaptive power control scheme for co-located wbans: a cooperative bargaining game theoretic perspective. **Wireless Networks**, Amsterdam, Netherlands, v. 24, n. 8, p. 3129–3139, Nov 2018.

WANG, Q. et al. Enable advanced qos-aware network slicing in 5g networks for slice-based media use cases. **IEEE Transactions on Broadcasting**, USA, v. 65, n. 2, p. 444–453, June 2019.

WANT, R. An introduction to rfid technology. **IEEE Pervasive Computing**, USA, v. 5, n. 1, p. 25–33, 2006.

WANT, R. Near field communication. **IEEE Pervasive Computing**, USA, v. 10, n. 3, p. 4–7, 2011.

WHEELER, A. Commercial applications of wireless sensor networks using zigbee. **IEEE Communications Magazine**, USA, v. 45, n. 4, p. 70–77, 2007.

WILLIAMS, A. M. et al. Artificial intelligence, physiological genomics, and precision medicine. **Physiological Genomics**, USA, v. 50, n. 4, p. 237–243, 2018. PMID: 29373082.

XIAO, J. et al. A survey on wireless indoor localization from the device perspective. **ACM Computing Surveys**, New York, NY, USA, v. 49, n. 2, p. 25:1–25:31, 2016.

ZHOU, T. et al. Multimodal physiological signals for workload prediction in robot-assisted surgery. **J. Hum.-Robot Interact.**, New York, NY, USA, v. 9, n. 2, Jan. 2020.

ZITTA, T. et al. Multi-channel access to improve qwl in health care services - infrastructure based qos ensurance in iot. In: INTERNATIONAL SYMPOSIUM ELMAR, 2018., 2018, Zadar, Croatia. **Proceedings...** IEEE, 2018. p. 7–10.

ZUHRA, F. T. et al. Miqos-rp: multi-constraint intra-ban, qos-aware routing protocol for wireless body sensor networks. **IEEE Access**, New York, NY, USA, v. 8, p. 99880–99888, 2020.