AN IOT ARCHITECTURE FOR UBIQUITOUS CONTEXT-AWARE ASSESSMENTS

by

Salsabeel Yousef Shapsough

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

May 2017

Approval Signatures

We, the undersigned, approve the Master's Thesis of Salsabeel Yousef Shapsough

Thesis Title: An IoT Architecture for Ubiquitous Context-Aware Assessments.

Signature **Date of Signature**
(dd/mm/yyyy)

______________________________ ________________

Dr. Imran Zualkernan
Associate Professor, Department of Computer Science and Engineering
Thesis Advisor

______________________________ ________________

Dr. Rana Ahmed
Associate Professor, Department of Computer Science and Engineering
Thesis Committee Member

______________________________ ________________

Dr. Malick Ndiaye
Associate Professor, Department of Industrial Engineering
Thesis Committee Member

______________________________ ________________

Dr. Fadi Aloul
Head, Department of Computer Science and Engineering

______________________________ ________________

Dr. Mohamed El-Tarhuni
Associate Dean for Graduate Affairs and Research
College of Engineering

______________________________ ________________

Dr. Richard Schoephoerster
Dean, College of Engineering

______________________________ ________________

Dr. Khaled Assaleh
Interim Vice Provost for Research and Graduate Studies

Acknowledgement

I would like to thank my advisor Dr. Imran Zualkernan for providing knowledge, guidance, support, and motivation throughout my research stages. I'm deeply beholden for his great assistance, worthy discussion and suggestions.

I would like to thank Dr. Rana Ahmed and Dr. Malick Ndiaye for serving as examining committee and providing me with great guidance and significant input.

I would like to thank the professors of the Computer Engineering department who taught me the master level courses with mighty teaching methods and skills. I really appreciate their dignified advices and motivation.

I would also like to thank the Graduate Program and the College of Engineering at the American University of Sharjah for offering me the Graduate Teaching Assistantship which allowed me to continue my higher education in Computer Engineering.

Dedication

To my mother, my best friend and greatest company through it all, and to whom I owe the woman I am today and the one I aspire to be…

To my father, my teacher and friend, who taught me there is nothing I could not do, and who continues to believe so…

To my advisor, my motivator and mentor for the past six years, and the reason I am the academic and engineer that I am today…

Abstract

Ubiquitous learning environments aim to move students out of classrooms and into the real world where learners can engage in experiential and tangible learning. Ubiquitous assessment systems are one class of learning environments that enact student learning in the form of teacher, peer, and system-generated assessments that incorporate physical aspects of objects in outdoor locations. A key component of such systems is a wireless-enabled edge device augmented with various types of sensors to represent the state of physical objects and environments. Most such current systems are constructed using traditional Internet technologies which are not suited for this purpose and hence leading to cumbersome and complex designs. This thesis presents a novel generic technical architecture for such systems built around Internet of Things (IoT) computing platforms. The newly proposed architecture is designed to seamlessly incorporate various IoT communication protocols. A commonly used IoT edge device was used to implement four variants of the proposed architecture. The variants based on Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP), Message Queue Telemetry Transport (MQTT), and Extensible Messaging and Presence Protocol (XMPP) IoT protocols were evaluated using an experimental setup. Each implementation was evaluated in terms of power consumption, CPU utilization and RAM usage, as well as end-to-end latency and throughput in response to network disturbances. In addition, qualitative aspects of each implementation were analysed based on maximum message size, overhead, security, reliability, and ease of implementation and flexibility. While there were statistical differences in power consumption between the four implementations, the practical difference was negligible. CoAP proved to be the most efficient in terms of CPU and memory utilization but produced the lowest latency in lagged networks only. However, due to payload limitations and lack of reliability features, CoAP was considered ill-suited for such applications. Among the other three variants, MQTT and AMQP seem more appropriate in terms of qualitative features, although MQTT was more resource efficient in most technical aspects.

Search Terms: ***ubiquitous learning; internet of things; assessment systems; application level messaging protocols***

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

This chapter begins with a brief introduction into the concept of smart education and how it fits into the new wave of Internet of Things technologies. The next section presents the problem statement. Next, we present the thesis statement as well as the contribution. The last chapter provides an overview of the thesis organization.

1.1. Education in the Internet of Things Era

The domain of learning and education is experiencing a shift in the way information is presented to students. Many institutes are choosing to migrate from lecture-based lessons towards class environments that allow for three-way interaction; where students, instructors, and physical surroundings take part in setting the curriculum and controlling the rate and the form by which the knowledge is offered. This is encouraged by studies that claim that when provided with the opportunity for an open flow of information, students gain a bigger incentive to pay attention and participate in class activities. Furthermore, these interactions can provide constructive feedback for the instructors as they will be able to improve the course material year after year, guided by the students' interaction. The result is a new market for mobile applications that attempt to reshape and elevate the learning experience for both students and educators by incorporating modern computing technologies. The emergence of Internet of Things (IoT) and IoT-based technologies created a wave of innovation in various aspects of everyday life. Education is not an exception; as the unprecedented availability of cheap sensors, and ease of app development led to the outgrowth of ubiquitous learning environments [1]. Ubiquitous Learning –sometimes referred to as u-learning- was originally vaguely defined in literature as “learning anything, anywhere, anytime”. A u-learning application is designed to actualize two main objectives: giving students access to learning material through a platform that they are familiar with, and presenting the information at pace that is appropriate for them [2]. The recent advancements in internet computing technologies, however, have morphed the term into a wide umbrella under which fall various applications that incorporate sensing and communication technologies for a situated, context-aware learning environment. Using interactive elements such as RFID tags, QR-codes, and smart sensors, learning can take place outside classrooms and in authentic settings.

Over the years, dozens of environments and mobile applications have been developed to implement the concept of ubiquity in learning. In [3], Njoku analyzes 135 studies on ubiquitous learning environments conducted in the year 2015-106; retrieved from the seven top journals on education and learning technologies. Among the main research questions were: “which subjects do ubiquitous learning studies focus on the most?” and “at which academic level are studies more likely to participate in these studies?” Analyzing the 135 articles showed that most studies are aimed at undergraduate students enrolled in science courses; while the least considered subjects were law and mathematics, and the least common test group was postgraduate students. Furthermore, the analysis found a 100% increase in the number of publications ubiquitous learning studies and environments per year; compared to the number between the years 2010 and 2013 [4]; suggesting a steadily-growing interest in the field over the past decade.

1.2. Problem Statement

Due to the lack of guidelines into how such applications should be built, it is often up to the developer to decide on an architecture for their specific learning system. Moreover, the abundance of classic and IoT technologies have made it even more challenging for the developer to select the most suitable components. This calls for the introduction of a generic IoT-based architecture that is designed with the needs of these applications in mind. Systems of this class have requirements such as real-time communication, high throughput, robustness, security, and low power consumption. They, moreover, consist of several types of hardware-based and software-based components that work together in delivering the content to the students and instructors. The developer is then left to choose the most suitable option for each component, integrate different components together, and ensure compatibility. Several organizations have presented projects that are meant to integrate computing technologies into educational environments, and implement a few modern pedagogies. However, a generic paradigm for systems of this class is yet to be developed; and therefore, the integration of different parts of these systems is done manually for each application. This not only contributes to the complexity and time needed to deploy a new application, but also causes a lack of compatibility between different systems which may be functioning in the same domain but developed by different organizations.

1.3. Thesis Statement

In this thesis, we propose a unique IoT-based architecture for inquiry-based, ubiquitous learning platforms designed to meet the requirements of existing works and improve their performance. We focus on edge IoT devices as they are responsible for linking users at the edge with core services, and there performance therefore dictates the user experience, which is a vital aspect in educational platforms. We consider four protocols for dispensing information throughput the system; three of which are designed for IoT applications, while the forth protocol is a classic-internet, trusted messaging protocol. The rest of the system components, such as application servers and databases, are set as control elements for the four implementations, and chosen on thorough research and literature review. A number of experiments are conducted using four physical implementations in order to evaluate the performance of the protocols in terms of resource requirements and user experience.

1.4. Thesis Organization

This thesis is organized as follows: the first chapter provided a brief introduction into the domain of IoT-based ubiquitous learning platforms. The second chapter surveys a wide selection of ubiquitous learning platforms, and provides an analysis of their components based on the IoT architecture. We also survey the most popular standards used for assessment transfer and management. The third chapter builds an education-specific IoT paradigm based on the requirements, use cases, and scenarios extracted from existing ubiquitous learning platforms. The fourth chapter presents the experimental setup and elements used to evaluate the performance of edge nodes, focusing on the edge devices and communication. The fifth chapter presents a discussion of the quantitative assessment explained in chapter four, while the sixth chapter discusses the qualitative advantages and disadvantages of each implementation. Finally, conclusion and future work are presented in the eight chapter.

Chapter 2. Background and Literature Review

This chapter is organized as follows: the first section introduces ubiquitous learning and presents a number of case studies that investigate its effectiveness in different settings. The following sections survey studies and learning environments that implement variations of ubiquitous learning using different combinations of software and hardware elements. The next section in this chapter compares the technical components of the presented works mapped into the Internet of Things paradigm. Finally, the last section surveys the most popular assessment standards using in e-learning, m-learning, and IoT-based learning systems.

2.1. Ubiquitous Learning

Saito et al. presents a study on basic support for ubiquitous learning (BSUL) environments; which they define as mobile learning environments that implements four important principles: light weight and simplicity, adaptiveness and customization, support for various learning styles, and support for collaborative learning [5]. Such systems have often been implemented using internet-connected PDAs and personal computers in coordinance with existing Learning Management Systems (LMS); thus allowing students to seamlessly learning experience where they have access to any information they need, whenever they need it. To conduct the study, a ubiquitous learning software was developed implementing the activities seen as most important in classrooms; including: report submission and attendance taking, in addition to student feedback and assessment response. The system was used in multiple case studies, most notably a Language-learning outside the classroom with handhelds (LOCH) case study [6] where students enrolled in a Japanese language course were asked to walk around town and perform certain tasks, aided by their PDAs. For example, students were asked to engage in conversations with locals to acquire certain information such as prices of different products and different outlets' opening hours. Another type of tasks required them to conduct interviews with school faculty in Japanese, and use their PDA to take a photo of the interviewee and note down important information. The PDAs were connected to the internet; meaning that the instructor will be able to track students on a map and update them with new tasks and/or feedback. Similarly, the authors in [7] present the results of a study that investigates the effect of context-aware ubiquitous learning environments on the performance of English learners. In the study, students are divided into two groups; a control group, and an experimental group. Both groups were asked to perform 3 activities: practice listening and speaking during their free time, participate in activities during class time, and perform a story relay. The control group was provided with classical learning aid such as printed materials, audio CDs, and voice recorders. The experimental group, on the other hand, was provided with HELLO (Handheld English Language Learning Organization); an application that runs on their mobile phones and presents each of the tasks in the form of a game that they can play outdoors. Context-awareness is implemented using area-specific QR codes; which students user their phones to scan and download different learning material depending on their location. Chen et al. introduces a personalized context-aware ubiquitous learning system (PCULS) that helps students learn English by focusing on vocabulary [8]. By taking the learner's vocabulary level, learning time, and their location –acquired using wireless network positioning- the environment is able to adapt and customize the content in order to provide a better learning experience.

Aside from language courses, mobile-based learning applications have also been popular in history classes. Museums, for example, often resort to electronic trails in order to provide guests with an educational experience [9]. A museum trail is typically a paper, brochure, or pamphlet that guides guests through different sections, providing brief historical facts about each object. Electronic trails provide the same information but in different forms, including audio and video. Electronic trails also allow museum visitors to add feedback and reflections in the different forms. This is particularly useful for visitors who wish to access the information later on; to prepare a presentation about their visit, for example.

In [10], students from 10 primary schools learning about the history of Amsterdam were asked to walk around the city, observing various monuments and learning about historical characters and stories hidden within their walls. Each group of students was divided into a city team (CT) and a headquarter team (HQT). The CT students will then walk around the city and use UMTS/GPS phones to perform certain tasks and share new findings with the HQT students, while HQT students provided directions using a modern map and a medieval map of the city; with the end goal being a full narrative that connects various historical elements of the city. The project builds on the concept of “storification”, where students are more likely to understand and remember historical characters and events when they are woven into a narrative. Being in the same vicinity as said elements adds a dimension of reality, helping students draw a full picture of events that took place around the city. Similarly, [11] presents the design and evaluation of a u-learning application designed to ease the process of editing ubiquitous learning material. In their work, Chin et al. present a system that consists mainly of a mobile application and a main server. Students walk around an area where certain elements have been labeled with QR-codes. A student can scan a QR-code and download learning content that has been previously edited and uploaded by their teacher. Students are then asked to use this information to perform different tasks. The system was used in a study where first year college students learnt about the Taiwanese Cultural Heritage; in which QR-code were placed on different artifacts such as statues and old journals. Not only did students report a higher level of enjoyment compared to typical learning methods, but they were able to achieve higher scores. QR codes are a common component of ubiquitous learning systems because they provide a cheap way for students to learn in an authentic setting and interact with different elements of their environments. For example, [12] presents a game that combines the use of QR code with the concept of prompt-based learning. In this game, students studying 5th grade-level natural science are assigned unique locations in a garden which serves as a home for different kinds of plants. Students move around the garden based on rolls of dice, and earn points by completing tasks where they are asked to locate a certain plant and answer questions about its characteristics using material they download by scanning QR codes. Scanning the QR also serves as a way to locate the student and identify their context, which is then used to form the next task. Hung et al. describes a similar system in [13] under the title Context-aware Reflection Prompt System (CRPS), with the main difference being the prompt and response type. This is motivated by studies that claim students enjoy a better learning experience when prompts are in video format rather than text. Not only are videos believed to be more engaging, but they fit better into real-life narratives.

Similar technologies to QR codes have also been adopted in other u-learning systems; mainly Radio Frequency IDentificaiton (RFID) and Near Field Communication (NFC). Liu et al. presents an environment of ubiquitous learning with educational resources (EULER) designed to aid students study natural sciences by going to outdoor locations [14]. Students, for example, can learn about animals by walking around a zoo and accessing context-specific learning material at different zones. This is made possible by placing RFID tags at different areas and cages; where the student will use their devices to scan the RFID tag and download available information uploaded by their teacher. Groups of students then combine their findings into a report to be reviewed by their teacher. Another, similar system that uses QR codes and NFC objects to teach students about hardware components of a computer is presented in [15]. One thing that is emphasized in the latter but never mentioned by the previously is that many of these systems are what is now considered the simplest form of an Internet of Things application. RFID, QR codes and NFC are the some of the easiest and most common ways to turn everyday objects into "things". In [15], Gomez et al. show how a QR code/NFC-based ubiquitous learning system can fit into the classical IoT paradigm; which consists of three layers: things, communication technologies, and a client. In this case, QR codes and NFC objects are things, websockets is the communication protocol, and a web-based application allows users to interact with the system at the client level. This leads to the question of "how much can IoT technologies offer to the field of ubiquitous learning?"

With IoT-based technologies gaining momentum the recent decade or so, cheap wearable technologies became more readily available than ever; and subsequently found their way into u-learning. The usage of u-learning is not limited to literature-based courses; as demonstrated in [16] and [17]. Prete-a-apprendre [16] is a u-learning system that incorporates the concept of context-aware ubiquitous learning into primary school mathematics, but adds an element of physical activity to it. In this system, students are asked to come up with three true-or-false math questions each at the beginning of the game. After the questions have been collected, the game begins and a random question is assigned to one student each round. The assignment is done using wearable, zigbee-equipped Lilypad microcontrollers sewn into each student's shirt. The shirt also has two pressure sensor patches; one for "true", and another for "false". In each round, the shirt of the student that has been assigned a question lights up, and other students must try to catch them and answer the question using the pressure sensors. Hoodies and Barrels [17] is another, similar system where children play hide and seek based on mathematical constrains set either by their teacher or by themselves. The system also incorporates wearable technology that is used to send children rules for the type of place where they can hide; e.g. behind a barrel that is 1 m^2 in volume. The child has to formulate the rules themselves and then send it to their peers; who then use that rule to find them. In terms of mathematical knowledge, while Prete-a-apprendre focuses on theory and students doing mathematical operations on the fly, Hoodies and Barrels is more concerned with students' perception of concepts such as shapes and volumes. The latter trait is also often the main focus of u-learning systems developed for science courses such as biology and geology. OBSY [18] is an excellent example of systems that implement context-aware u-learning which emphasize the students' perception of and interaction with their physical surroundings. The developers make use of old PC tablets that were distributed to schools in rural Northern Thailand as a part of the One Tablet PC per Child policy. Each student is offered a Raspberry Pi-based sensor node equipped with a camera and a group on sensors that they can interact with via an application deployed on their tablet. The system aims to connect students to their environment by offering them the chance to associate their own experience with a space with a quantitative values acquired by sensors. Using the camera, students are also able to monitor elements in their environment –for example, the growth of a small plant- in order to add a realistic experiential value to is usually taught through textbooks. The student can move around campus, position their personal node (OBSY) anywhere they wish, and take sensor readings and footage to be used in answering different questions. The incorporation of smartphones' existing sensors has been investigated in [19]. In their work, Shapsough et al. present the design and evaluation of an assessment system where students use smartphones to access and answer assessments set by their teachers. The use of sensor data in this particular system differs from other discussed systems because the readings are not involved in the students' answers; but are relevant to understanding the context in the student was while answering the assessments. However, only few application-level modifications will be needed in order to change the system so that it allows students to access the sensor information and incorporate them in their answers.

2.2. Inquiry-based learning

Inquiry-based learning (IBL) refers to a student-centered educational approach where the majority of the class interaction is initiated by students. Students are encouraged to contribute to the material presented in the classroom by formulating their own hypothesizes, posing questions, and sharing findings and conclusions [20]. This is believed to help students develop analytical thinking skills, improve social skills, and better the students' understanding of the material [21]. It also encourages students to perceive the information presented in a critical manner and process it for any doubts, instead of accepting it as facts. Furthermore, when given the opportunity to construct and pose their own questions, students are put in a situation where they have full freedom –and motivation- to speak their minds and share ideas without the pressure or fear of standing out in class. They are also given the opportunity to consider minor details and use their imagination in order to come up with unique questions, which promotes metacognition. In many cases, a student's scores and quality of work were closely related to the type and amount of interaction that took place between them and their peers.

One of the first and most notable efforts to build an IBL environment is the PocketSchool Interactive Learning Ad-hoc Network (PSILAN); developed by the Stanford University School of Education in 2011 [22]. PSILAN allows students to come up with their own questions, combine them into a quiz, and share it with their peers. Students can attempt to solve quizzes posed by their peers, as well as rate them based on quality and usefulness. Questions are not limited to text, as a student can take a photo of a diagram from their textbook or an item in their environment, for example, and incorporate it into a question. Around three years later, SMILE was introduced. SMILE [23] (the Stanford Mobile Inquiry-based Learning Environment) is a mobile-based system that students to construct their own multiple-choice questions, answer ones proposed by their peers, rate each other's questions, and view question-related statistical data -such as most answered questions and student with most correct answers. The system offers extended functionalities for the teacher which allows them to control the flow of the activity, and view information about all the students involved. SMILE consists of two main components: the mobile application used by the students, and the management application used by the teacher. Different nodes in the system (mobile application entities and management application) have the option of communicating over either ad-hoc network or the internet. The application was tested on several test groups; the student groups came from different parts of the world, and belonged to different age groups. The tests showed that although the students expressed their enjoyment using the application, in order to completely utilize the inquiry-based aspect, the students need to be trained on how to ask questions. For example, one teacher asked her class of fifth graders to give questions that require recall only a low rating, and give a higher rating to questions that required applying analytical skills. Through continuous iterations of posing questions, and then answering questions posed by others, students were then able to improve not only the quality and complexity of their questions, but also their understanding of the material and their critical thinking process. Similarly, Pearson's Learning Catalytics [24] provides students with the opportunity to engage in peer-to-peer learning activities. Rather than being constricted to multiple-choice questions, however, Learning Catalytics also allows students to answer questions by plotting graphs, highlighting points on images, and filling in numerical values.

In [25], the authors present the Community of Inquiry (COI) framework in a form of multi-player game. The COI framework is based on three main concepts: constructive socialization, reflection, and practical inquiry. Video game players often employ these skills in multi-player games that require players to either challenge each other or work together to achieve certain goals and unlock new traits. Based on such, the authors present a Virtual COI (VCOI) environment where students learn about Applied Thai Traditional Medicine (ATTM) by engaging in a virtual reality game. The game environment consists of a virtual classroom where each student, in addition to the teacher, controls an avatar. Students are assigned tasks that require their avatars to venture out from the classroom and interact with avatars and objects in the game universe. Students are able to customize and control their avatar just as they will in a normal simulation game; thus providing an element of not only fun, but also familiarity, which is crucial in ubiquitous learning. By moving around the virtual universe in their avatar, they can perform actions on objects to obtain information, and then use that information to form hypothesis, pose and answer questions, or engage in academic discussions with their peers.

The same framework is also adopted in SimAULA, a learning environment designed to gamify the process of inquiry-based learning in science classrooms [26]. The game starts with a virtual classroom where each student, in addition to the teacher, controls an avatar. Students are assigned tasks that require their avatars to venture out from the classroom and look around for certain clues; for example. They are then asked to use various analysis tools offered by the environment to study the clues they have acquired, draw conclusions, and discuss them with their classmates.

In [27], chemical engineering students enrolled in a Control course where asked to participate in a game where they will apply what they learnt from the course in a real remote lab. The lab was equipped with a set of water columns controlled by remotely-accessible panels, and each team of four was assigned a water column that they were responsible for. In every round, two teams will go against each other where each tries to keep their water column in control, while sending the opponent's into oscillation.

2.3. Context-Aware Inquiry-Based Ubiquitous Learning

An excellent attempt to combine context-aware ubiquitous learning with inquiry-based learning is Archi-Pods [28]. In this system, students enrolled in an Architecture Principles university-level course are provided with web-based mobile applications that interact with sensor nodes distributed around campus. The nodes are equipped with various sensors including temperature, motion, and illumination. Students are asked to walk around campus, interact with sensor nodes to acquire sensor readings, and then incorporate this information into constructing or answering interesting questions about a particular space. In one task, for example, students are asked to locate an architectural element such as an arc. Once a students has located an arc, they can take a photo using their mobile phones and use it to answer the question. In another task, students are asked to find illumination level in a space, and relate it to how the space is being used. The system also allows students to rate questions, which in turn works as a mechanism for improving their question posing skills and elevating critical thinking.

WeSPOT (Working Environment with Social, Personal and Open Technologies for Inquiry Based Learning) [29], [30] is another example that implements this category of learning systems. It is an inquiry-based learning system based on the Personal Learning Environment (PLE) paradigm -a learner-centric environment which provides tools and services for students to design their own learning experience. WeSPOT aims to patch the gap between different inquiry tools and services, students' profiles, and the curricular con-text in order to create a cohesive learning environment tailored to the students' needs. The environment offers students a game-like experience that is meant to mimic the sort of interaction that takes place in social media. This, the developers believe, motivates the students to take advantage of the system's capabilities as they feel a sense of accomplishment earning points in form of badges as they reach certain milestones. Students can access the system through their mobile application that they can download on their smartphones and proceed to pose and answer questions that fall under tasks specified by the instructor. The system also supports a type of tasks where students are asked to collect visual and audio information from their environment through their smartphones, which incorporates the element of personal experience into the learning process. Google Glass Personal Inquiry Manager (Glass PIM or GPIM) [31] is a Glassware (Google Glass application) that was introduced in order to elevate the weSPOT experience. GPIM focuses on the experimental aspect of the environment as students can now incorporate images, audio, and video taken by their Google Glass unit into their inquiries. The incorporation of Google Glass is meant to provide smoother flow of information between the students' physical surroundings and the learning system, and makes it easier for the student to keep track of their peers' contributions while working to complete their tasks, instead of having to constantly shift their attention between the application and their surroundings.

2.4. Suggested Architectures

A more technical approach was proposed in [32], as they describe ubiquitous learning environment they chose to call Youubi. Driven by the need for a unified architecture for ubiquitous learning applications, they propose a component-oriented reference paradigm for learning environments. Youubi not only represents a fully-functioning ubiquitous learning application, but it also presents an environment that can be used to develop different applications following the ubiquitous learning approach. Software-wise, Youubi implements a client-server architecture, and consists of eight server-side components, and a client application through a service API. The application server is based on Java, and the communication between entities is done through HTTP requests, and the current mobile application is Android-based. To test the system, the developers of Youubi asked a group of students to use the application for a period of time, and compared the results with a control group. The results show that the overall level of engagement was almost the same between the control group and the Youubi group, which the developers attribute to different variables. However, the majority of the Youubi group found it interesting and enjoyable to read challenges posted by other students, and expressed interest in using the application in the future. The issue with Youubi, however, is that it is designed only to accommodate ubiquitous learning applications. The architecture specifies roles for various users and resources, but does not include features for real-time communication or class interactions, neither does it consider interactions with the physical world. For those reasons, this architecture is not suitable for building the class of applications we define.

Chen et al. propose the design of a subject-independent Ubiquitous Open-structured Neo-tech Edutainment (u-ONE) architecture which combines learning robots, sensing technology, mobile computing, and wireless networks [33] to support children's learning. The framework is designed based on the concepts of experiential and gamified learning, and can be used in instruction, collaborative learning, and adaptive self-learning. A total of five hardware component types are defined in an implementation of the system: a wireless network, an output element, a mobile computing element, perception elements like sensor and RFID, and a learning robot. The inclusion of all five elements is claimed to be essential, although the choice of product for each category and how they are integrated into the system is left to the developer. This is meant to provide flexibility in terms of customization and cost; as implementation may differ depending on the subject being taught, and –and more importantly- on the availability and affordability of different products. Each student is provided with one or more perception elements to help them interact with their surroundings, and a learning robot that serves as their access point to the system. On the other side of the classroom, the instructor is provided with a mobile computing device capable of collecting and processing student performance data, and any sort of output screen to display the results. For the most part, the framework mimics all of the systems discussed earlier; with the exception of the learning robots. In addition to adding a touch of fun into the learning environments, they can provide real-time one-on-one interaction with every student and real-time feedback; thus reducing the pressure on the instructor who will otherwise have to divide their time, attention, and energy among all students. Three scenarios have been suggested for how the system may be used. In case of an instruction-based class period, instruction material will be distributed to the learning robots, and students will have equal chances to participate during the lecture. The same learning robot can also be used to communicate with other students in a collaborative learning session, or provide adaptive learning material in the case of self-learning. Wireless communication between different stations is most crucial in the first and second scenarios, but is not needed for self-learning; as most learning material is pre-downloaded into the station at the beginning of the session.

2.5. Comparison of Existing Work

There are several published works that compared different e-learning, ubiquitous learning, and inquiry-based learning platforms. In addition to the ones previously discussed in [3] and [4], we cite the work done by Abdullah and Ward in [34], and Chang et al. in [35]. In both publications, the authors survey a number of e-learning and u-learning environments -107 environments in [34] and 22 in [35]- and compare them in terms of technical support, and case study dimensions. Such comparisons are useful to acquire a general picture of the technologies involved in u-learning environments. However, in order to bring the field of u-learning into the Internet of Things (IoT) paradigm, it will be more sensible to examine existing systems in the light of said paradigm.

As to comply with the characteristics of an IoT application, we first define the architecture we base our work on. Different authors, developers, and stakeholders present different versions of the IoT generic architecture; which often vary in the number of layers defined and the classification of technologies into the layers. This arises from the fact that people from different backgrounds are taking part in the IoT revolution [36]. A corporation, for example, will define a paradigm that more based on supply chain and business operations, while a telecommunication institute will focus more on the communication technologies and topologies. For our work, we choose the generic architecture derived from ones proposed in [36]–[40]. The architecture consists of five layers: perception, network, middleware, application, and business. The Perception Layer (PL), sometimes referred to as the Device Layer- is provides interfaces between the system and the physical world, mainly identifying objects and context, and acquiring data. Technologies that fall under this layer are sensors, QR codes, RFID, smart meters, and others. The next layer is the Network Layer (NL), which provides communication media between PL devices and upper layers in the architecture. The layer covers small-scale and wide-scale networks, and employs wireless communication technologies such as Wi-Fi, Bluetooth, and others. On top of the NL is the Middleware Layer (ML), which provides management services as well as databases, processing services, messaging brokers, and others. The layer is responsible for handling the heterogeneity of the devices in the IoT system and providing a seamless exchange of information. Following the ML is the Application Layer (AL) which provides interfaces between the users and the IoT system in the form of mobile applications, websites, reports, and others. The last layer is the Business Layer (BL) which provides services such as data management and business model management. The IoT architecture we present in Figure 2.1 is based on this generic architecture, excluding the business layer. This is to ensure an easy integration of the architecture with other IoT-based systems, and will be used to develop a system architecture for our class of applications.

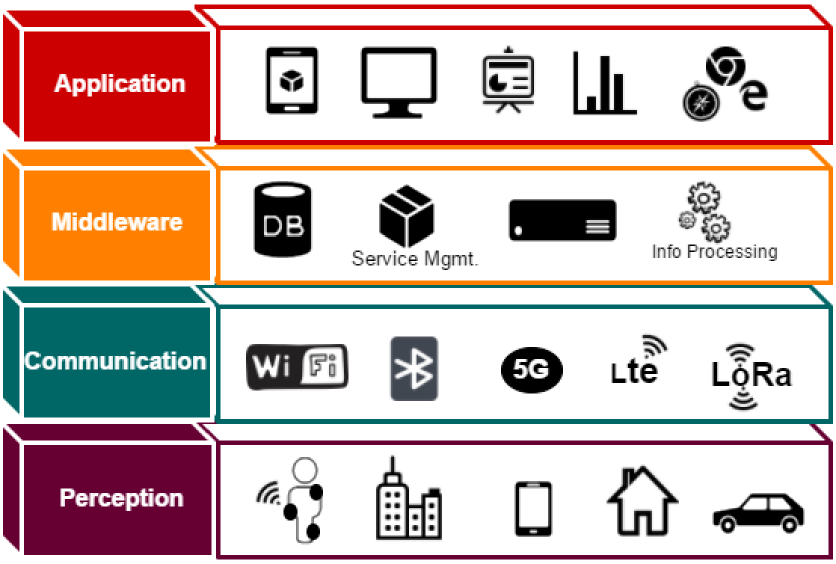


Figure 2.1: Internet of Things paradigm

Based on the previous IoT architecture, we present a comparison of the discussed ubiquitous and inquiry-based environments as they fit into IoT paradigm. All of the systems proposed in Table 2.1 and Table 2.2 are custom-made; the developer is given the responsibility of choosing the product/technology for each element, and then integrate them together. This is highly inefficient considering that most systems, in essence, have the same requirements and are designed to achieve the same goal. A unified architecture for this class of systems will massively reduce the design and implementation time required to produce a system instance. The systems in Table 2.3 attempt to implement said unified architecture. However, most of them are use technologies developed for classic internet applications to achieve what is already presumed to be an IoT application. IoT applications have special requirements [36], [41], [42] that more often than not cannot be supported by classic internet technologies; thus the exponential increase in IoT-specific hardware and software [38]. It is thus more reasonable to propose an architecture for learning systems which employs the most suitable of such technologies.

2.6. Standards for Assessment

In this subsection, we go over the most prominent assessment standards used by learning organization and assessment tools.

2.6.1. QTi. Question and Test Interoperability (QTi) [43], [44] is the most widely used open assessment standard for online assessments. The standard was developed by The IMS Global Learning Consortium in 2002, and offers an XML-based format for representing assessments and their corresponding results. The intention is to facilitate the exchange of assessment items between question repositories, test tools, and test delivery systems. The standard supports a wide variety of test items which include not only multiple choice questions, but also interactive test items which allow the students to select and move text and graphic objects. A JSON-ised version of QTi is also available, as QTILite [45].

2.6.2. MOODLE. Moodle [46], [47] is a free assessment platform designed to aid instructors in creating customized assessment environments. The platform allows instructors to organize and give assessments to students as well as keep track of their scores and progress. The tools defines two formats for assessments: Moodle XML, and GIFT.

2.6.3. Moodle XML. Moodle XML [48] defines a format for importing and exporting assessments within Moodle, but can be used with other assessment tools that support XML. The format supports various question types including short answer, multiple choice, true or false, matching, gap fill, calculated, and essay. Since XML is a popular messaging and assessment format, Moodle XML items can easily be exported and imported between various assessment tools.

2.6.4. Moodle XHTML. The third export format offered by Moodle is Moodle XHTML [35]. It is used for exporting bulks of questions as XHTML items, while the answers are often excluded. It was not developed for the purpose of exporting questions directly to students, but for sharing collections of questions between web applications for modification, and auto-generating quizzes.

Table 2.1: Science and technology systems

Project	Pedagogy	Software architecture	Architecture							Data type
			PL		NL	ML	AL			
			Edge device	Sensing element			Application-level communication	User Application	Data-base	
Natural science text [12]	CA-UL	Client-server	Smartphone	QR codes	WiFi	N/S	HTTP	Android-based app	N/S	Text
CRPS [13]	CA-UL	Client-server	Smartphone	QR codes	WiFi	N/S	HTTP	Android-based app	N/S	Text, Video
EULER [14]	CA-UL	Client-server	RFID reader, PDA	RFID, Camera	WiFi	N/S	HTTP	Web-based application	SQL	Text, images, audio, video
IoT-based UL [15]	CA-UL	Client-server	Smartphone	NFC, QR code	WiFi, 3G	N/S	WebSocket	Cross-platform app	N/S	Text, images, audio, video
OBSY [18]	CA-UL	Client-server	RPi	sensors, camera	WiFi	N/S	HTTP	Web-based application	-	Text, images
Prete-a-apprendre [16]	IBL	Peer-to-peer	Arduino	sensors	WiFi, ZigBee	ZUL	HTTP, ZigBee	Personal computer application	-	Text
Hoodies and Barrels [17]	CA-UL+ IBL	Peer-to-peer	Arduino, RFID reader	sensors, RFID	WiFi, ZigBee	ZUL	ZigBee	Personal computer application	-	Text
Control lab [26]	CA-UL + IBL	Client-server	Control panel	Remote lab equipment	WiFi, GPRS	N/S	SMS, HTTP	Web-based application	-	Text

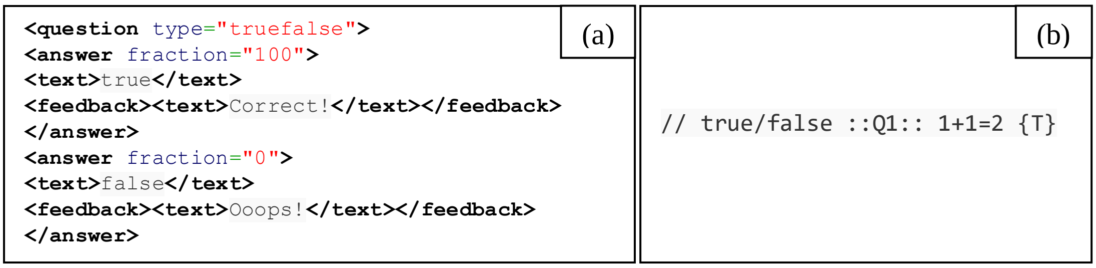
Table 2.2: Language and social science systems

Project	Pedagogy	Software architect-ture	Architecture							Data type
			PL		NL	ML	AL			
			Edge device	Sensing element			Application-level communication	User Application	Database	
LOCH [6]	CA-UL	SOA	Smartphone	GPS, camera, mic	PHS	N/S	HTTP	J2EE Web-based app	N/S	XML: Text, images, audio
HELLO [7]	CA-UL	Agent-based	Smartphone	QR code	WiFi/WCDMA	N/S	HTTP	Web-based app	SQL	Text, images, audio, video
PCULS [8]	CA-UL	Agent-based	Smartphone	Triangulation of WLAN signal	WiFi	N/S	SMS, HTTP	Web-based App	SQL	Text
Museum [9]	CA-UL	Client-server	Smartphone	Camera, mic	WiFi	N/S	HTTP	Web-based App	N/S	Text, images, audio, video
Amsterdam [10]	CA-UL	Peer-to-peer	Smartphone	GPS, camera, mic	UMTS	N/S	SMS, MMS	Web browser	-	Text, images, audio, video

Table 2.3: Subject-independent systems

Project	Pedagogy	Software architecture	Architecture							Data type
			PL		NL	ML	AL			
			Edge device	Sensing element			App-level communication	User App	Database	
QR-Lumps [11]	CA-UL	Client-server	Smart phone	QR codes	WiFi	N/S	HTTP	Android-based app, PC application	N/S	Text, images, audio, video
Context-aware assessment [28]	CA-UL	Pub/sub	Smart phone	Sensors, GPS	WiFi	Custom	MQTT	Cross-platform app	CouchDB	Text
PSILAN [22]	IBL	Client-server	-	-	WiFi	MobiSocial Junction framework	XMPP	Android-based Junction Activity Director	N/S	Text, images
SMILE [23]	IBL	Client-server	-	-	WiFi	MobiSocial Junction framework	XMPP	Web-based application	N/S	Text, images
Learning Catalytics [24]	IBL	Client-Server	-	-	WiFi	N/S	HTTP	Web-based application	N/S	Text, images
WeSpot [29]	CA-UL + IBL	Client-server	Smart phone	Camera, mic	WiFi	Elgg (PHP)	HTTP	Web-based application, Android App	MySQL	Text, images, audio, video
GPIM [31]	CA-UL + IBL	Client-server	Google Glass	sensors	WiFi	Elgg (PHP)	HTTP	Glassware	MySQL	Text, images, audio, video
Youubi [32]	CA-UL	SOA	Mobile, Smart Watch	GPS	Ethernet, WiFi	RESTEasy	HTTP	Web-based application	N/S	Text, images, audio, video
u-ONE [33]	CA-UL + IBL	Peer-to-peer	Learning robot, RFID, PDA, iPod, OLPC	Barcode, RFID, QR, E-Pen, Magnetic Card, Laser	BLE, Wi-Fi, ZigBee, GroupNet	N/S	Bluetooth, HTTP, ZigBee	PC application, mobile app	N/S	Text

2.6.5. GIFT. GIFT [48], on the other hand, defines a simpler, more user-friendly format for assessments that allows users to construct assessments using a text editor. The format makes the process of constructing, editing, and proofing questions into categories much more accessible for instructors with less-advanced technical skills. Furthermore, GIFT supports the same level of question type variation as Moodle XML, and Moodle provides conversion tools between Moodle XML and GIFT. This way, questions that were previously constructed in XML can later be viewed and edited as GIFT items. Moodle also provides a tool to quickly export and import questions between quizzes and question banks. In order to better understand the difference between the two formats, Figure 2.2 presents an example of a true or false question, formatted in Moodle XML and in GIFT



```
<question type="truefalse">
<answer fraction="100">
<text>true</text>
<feedback><text>Correct!</text></feedback>
</answer>
<answer fraction="0">
<text>false</text>
<feedback><text>Ooops!</text></feedback>
</answer>
```

```
// true/false ::Q1:: 1+1=2 {T}
```

Figure 2.2: Moodle True or False question format example: (a) Moodle XML (b) GIFT

2.6.6. BLACKBOARD. Blackboard Learn [49] is one of the most popular online course management environments. The environment is used to manage courses, assessments, and student record, and is popular especially among higher education institutes. As a part of the environment, Blackboard Learn defines its own assessment format that is fundamentally based on QTI for exporting and importing assessment items. While the format was originally developed to facilitate sharing of records and material between different institutes, the environment offers tools to export files to other learning environment so that Blackboard Learn can still be used as a supplement in institutes where it is not the primary course management environment.

2.7. Summary

This chapter presented a survey of studies and learning environments that aim to implement pedagogies like ubiquitous learning, inquiry-based learning, and context-aware learning. While some of the works were developed for specific subjects and settings, other were designed with the goal of providing a generic environment that can service multiple audiences. The most interesting part was that even though all of the presented works were implemented organically with no unified design in mind, they all share common key components and structural elements. By mapping these works into the IoT paradigm, it is possible to design a unified architecture using IoT technologies that can not only serve in the place of previous systems, but also improve upon their performance.

Chapter 3. Methodology

This chapter is divided as follows: the first section defines a set of non-functional requirements that were deduced from the works presented in Chapter 2. The requirements are then used to construct a system definition, which is presented in the second section. The second section also includes a use case diagram for the system, as well as functional scenarios. In the third section, the different layers of the system architectures are discussed one by one, along with a study of components and a comparison of available technology options.

3.1. Problem Formulation

The first step to defining an architecture that can suit and improve inquiry-based, ubiquitous IoT application is to identify the requirements and characteristics of such system. The requirements are defined by examining existing systems and extracting the most common and critical functionalities. Once requirements and use cases have been defined, a system definition is constructed based on the given information. The system definition comprises all front end and back end hardware and software, which is chosen by studying and comparing different technologies, and selecting the most suitable ones. In principle, each one of the presented applications implements two or more of the pedagogies discussed earlier. However, as criteria for evaluation, we define sets of requirements and use cases that we believe such a learning environment should support. We will also use these use cases and requirements as guideline for our learning environment architecture; and to identify the various technologies required to build such environments. First of all, the system requirements are defined as follows:

- Students should be able to interact with peers in real-time. A major requirement in inquiry-based and collaborative learning is the ability to engage in real-time communication with other students in the system. The communication types that can take place are one-to-one, and one-to-many communication.
- Students should be able to pose questions from a defined location. Once a student has formulated a question, they will proceed to pose it to particular nodes in a physical environment. If the question is general, it should be posted to all nodes. But if the question is related to a certain geographical location or

type of space, the question should only be posted to nodes that fall under the specified category. This will be resolved using a principle of "topic + context"; where every node is not only linked to certain topics in the course material, but also to the physical context it exists in.

- Teachers should be able to pose questions independently of their location. Unlike students, teacher need not be in a specific location in order to pose questions. A teacher should be able to pick pods to publish the question to form a list. The question should then be published to the selected pods.
- Students should be notified once a new question has been posted. Once a question is available at a pod that students have access to, students should receive a notification through their mobile application or similar means.
- Students should be able to retrieve a question from pods located in areas that relate to that question. After students receives a question through their device, they should walk to a physical node in case of a general question, and a specific node in case of a context-linked question- and retrieve the question. The communication between students and the node need not take place on the same communication network used for posting questions.
- The teacher should be able to dynamically assign and un-assign pods to multiple different categories. Each pod is assigned a group of different categories and topics that determine whether or not the student can access a certain question from that pod. This should be dynamic; i.e. the teacher should have the ability to assign individual or pools of pods to a topic, and remove individual or pools of pods from that topic.
- Students should be able to post answers. After retrieving a question, students should be able to attach an answer the question and send it to the node it is linked to.
- Teachers should be able to see answers posted by their students. Once a new attempt has been posted, interested teachers should be able to view the answer.
- Students should be able to integrate sensor readings into their questions. Sensors are the key component required to implement the experiential learning aspect. Various sensors will be used to obtain location and environment-related information, which will then be used within the class interaction. Sensors can generate analog or digital data of various sizes and formats, and the sensors

readings will need to be processed and reformatted before being sent to the application. The user should be able to retrieve real-time sensor data that can be used to answer the question, and also have the option to include the sensor data in a question or an answer.

- Students should be able to attach or link to media files. As a part of the experimental learning process, the students may need to include images, audio files, or video files in their questions and answers. The application should allow the user to either attach the media file or include a link to it.
- Students should be able to search through repository of questions. Whether to review old learning material, or reuse previous assessments in a new contribution, students should be able to search through a repository of previous questions and answers.
- The pod should function as a standalone unit in case of network problems. In case of network problems, the user should still be able to use the application to pose new questions, answer questions that have been published before network failure, and access old contributions. The user should also be able to access last saved data from any sensor.
- A new pod added to the system during operation should be able to start functioning right away. Adding a new pod to the system entails assigning topics to the pod, and replicating relevant databases so that students can begin to interact with it in little to no time.
- Pods, teachers, and students should be updated with all offline interactions once communication is reconciled. Once communication has been restored, the user should be notified and be able to view questions that have been proposed by other users during offline time. Questions posed from the user should also be published to all applicable nodes and users should be notified.
- Students and teachers should be able to access all functionalities of the system regardless of the type of device they own. Students and teachers should be able to use the application of various platforms, including mobile phones of most operating systems, tablets, and personal computers.
- Teachers should have able to use the system to send data that follows the assessment standard of their choice. Different institutions and instructors may choose to have the contributions follow different assessment standards that

differ in format. The user should be able to use the application just the same regardless of the assessment standard.

3.2. System Definition

Figure 3.1 shows the system diagram which satisfies the use cases for the proposed class of educational applications.

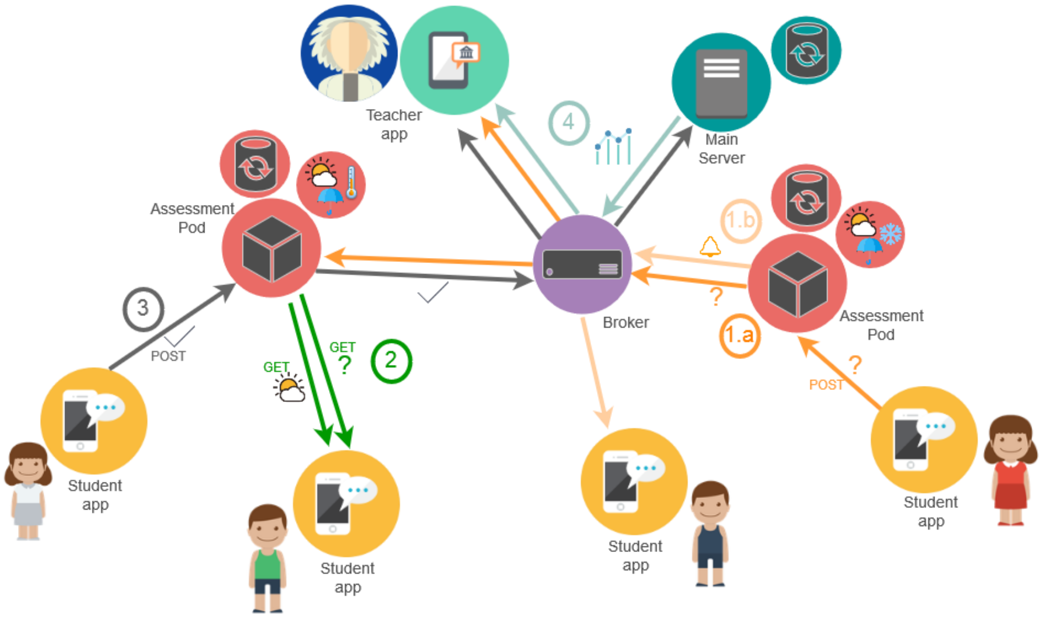


Figure 3.1: System Diagram

The system consists of four segments:

- Mobile application: The mobile app provides two different sets of functionalities for the teachers and the students. Students use the mobile app to communicate with the pods and receive notifications. The teacher app unlocks extra functionalities that allow them to monitor student activities
- Learning pod: Pods consist of a processing unit, a communication unit, and a group of sensors. They serve as the students' gateway into the network of interconnected pods, students, and teachers. Each pod host a webserver which students can POST questions and attempts to, and GET sensor readings and questions posted by other students. Once a new question/attempt has been posted, it is the pod's responsibility to publish it to other pods as well as notify students so that they can retrieve it from the same or a remote pod.

- Main server: the main server, is responsible for providing processing related to student performance analytics. All new attempts are immediately published to the main server, and depending on the system requirements, the main server can make use of analysis engines in order to provide teachers with different levels of analytics.
- Broker: the broker is responsible for delivering questions, notifications, and attempts. Whenever a student posts a question or an attempt, pods will send them as well as related notifications to the broker so that they can be published to related students, teachers, and other pods.

The system use case diagram is shown in Figure 3.2.

There are four main scenarios that take place in the system:

- In the first scenario (Figure 3.3), a student connected to the same wireless network as the Pod poses a new question. The new question initiates a series of event, which include the new question being published to other pods as well as the main server, and notification messages being published to students.
- In the second scenario (Figure 3.4), a student who received a question notification uses the question ID included in the notification to request the question body from a Pod.
- In the third scenario (Figure 3.5), a student requests sensor reading from a Pod. The student will then use the sensor information either in posing a question, or answering one posted by a peer.
- In the fourth scenario (Figure 3.6), the student POSTs an attempt to a question to a Pod. This scenario is similar to the first scenario, except no notifications are needed. Furthermore, if the type of question allows it, attempt feedback is provided to the student immediately and is independent from the rest of the process. This is possible for true-or-false, multiple-choice, and fill-in-the-blanks questions, but not short answer questions as an instructor has to be involved in the latter.

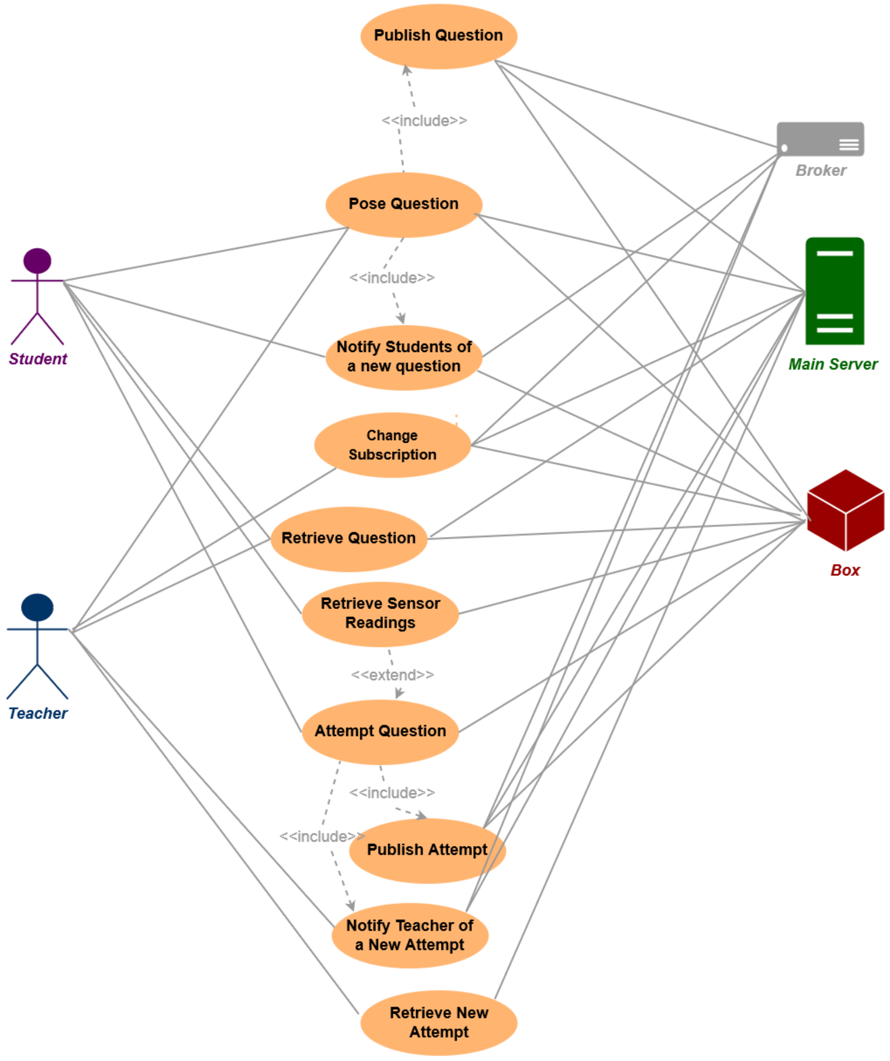


Figure 3.2: System use case diagram

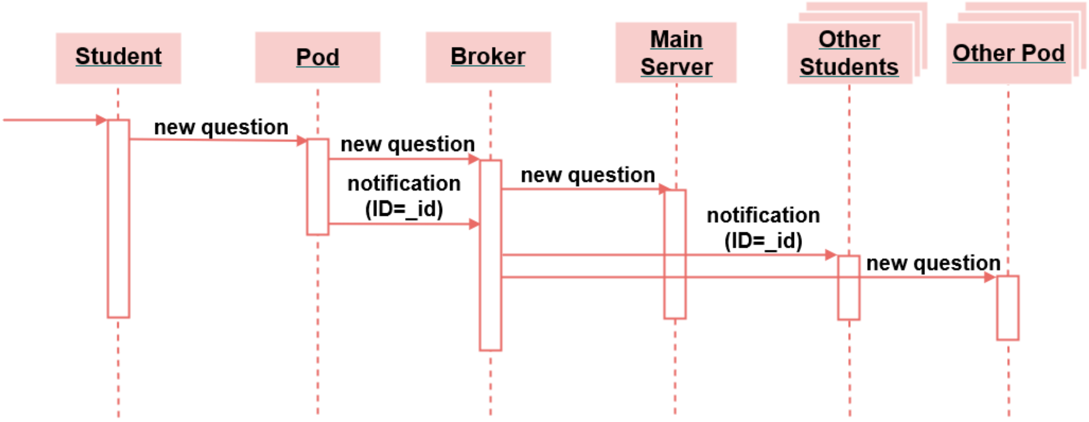


Figure 3.3: Scenario 1: post a question

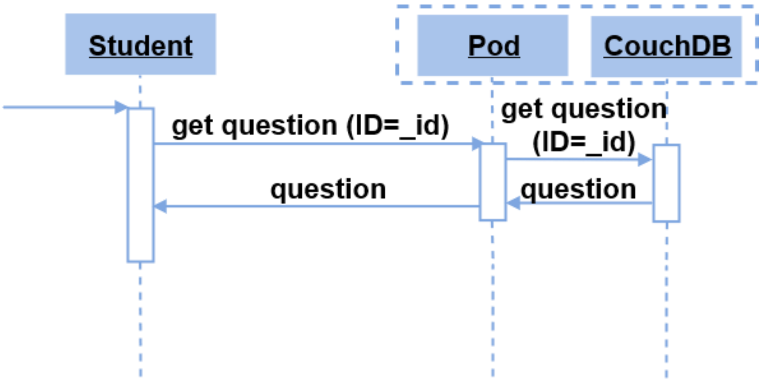


Figure 3.4: Scenario 2: request a question

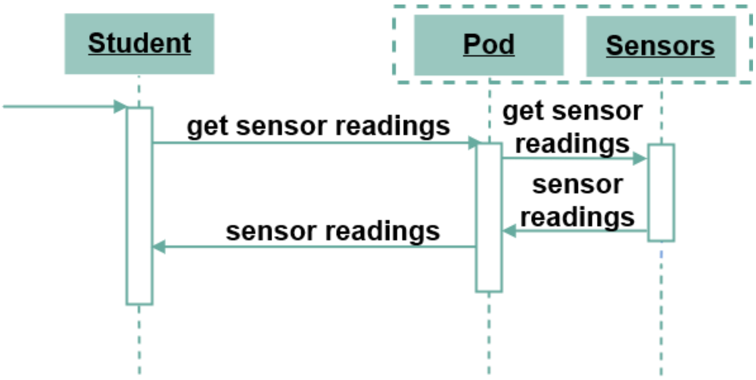


Figure 3.5: Scenario 3: request sensor readings

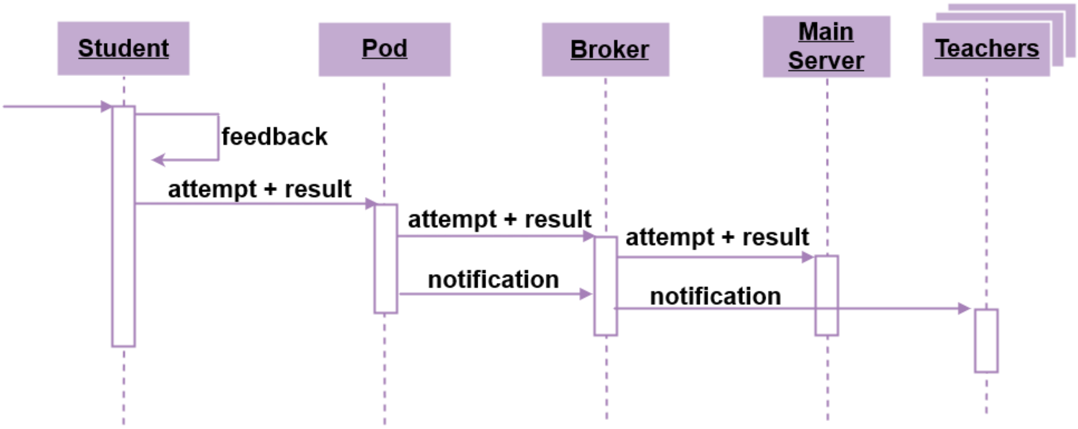


Figure 3.6: Scenario 4: post an attempt

3.3. System Architecture

The system is built based on the Internet of Things paradigm (shown previously in Figure 2.1) which consists of the Perception Layer, Communication Layer, Middleware Layer, and Application Layer.

3.3.1. Perception layer. The perception layer is represented by devices called information pods. Every information pod is made up of two segments. The first segment is responsible for sensing, and consists of an assortment of analog and digital sensors, and a processing unit. The sensors will vary from one system implementation to another depending on the curriculum. For example, an inquiry-based learning system implemented for an architecture course, such as the one presented in [28], will incorporate temperature, humidity, and illumination sensors, but will not require CO2 sensors. The sensors connect to a processing unit which reads sensor output data upon request, and interprets the data into intelligible information that can then be reported back to the requesting user. The second segment is responsible for connecting the information pod to the rest of the system and managing data. It acts as a messaging client that receives messages from student nodes requesting physical information, prompts the sensing segment for sensor readings, and replies back to the requesting node. The second segment also includes a database which stores sensor data along with other information such as date and time the reading was taken. In addition to storing contributions, the database functions as a cache memory for sensor readings as to save power from repetitive requests within a short period of time. We divide available options for the sensing or edge computing unit into three main categories: Single Board Computers (SBCs), specialized IoT edge System on Chip (SoC), and smartphones and watches.

SBCs refer to chips that contain of all functioning elements of a computer, including a CPU, memory, and Input/Output (I/O), on a single circuit board; making them highly popular in applications where lower cost and smaller space are favored over high performance. The price for an SBC can range from $9 to $180+, depending on the manufacturer. Some of the most popular low-cost ones include Raspberry Pi [50], BeagleBone [51], ODROID [52], Intel's Edison [53], and C.H.I.P [54]. One major advantage of using SBCs is that they are capable of performing the roles of both the computing and the sensing segment. On one hand, most SBCs support variations of Linux-based operating systems which are capable of running and hosting most applications that are compatible with normal Linux systems. On the other hand, SBCs expose I/O functionalities, and provide tools and libraries that handle reading and writing to sensing modules, and communication with other embedded devices over various protocols. In addition to lowering the cost of the node hardware, it allows for flexibility and customization in terms of which modules are to be added depending on the application. That being said, connecting the extra modules will require a certain level of technical ingenuity. This can be a disadvantage in certain settings like a primary school, for example, where a teacher setting up the nodes does not possess the appropriate technical skills.

The second option for an edge device will be to use a consumer electronics as Android-based phones or smart watches as the processing/communication unit. Devices under this category run either Android or Tizen [55]. Tizen was designed with the purpose of providing an open software platform to host various applications, including cross-platform apps. As for Android-based devices, they are all capable of running native and cross-platform apps, but the more recent ones also offer the possibility to install and run a chroot [56] version of Linux OSs such as Debian [57]. Therefore, with an Android-based or a Tizen-based device as the node "brain", it is possible to develop the node server application as a native/cross-platform application, or host it using environments that can run on Linux systems; with the help of chroots. Sensing-wise, smartphones and smart watches are often equipped with sensors like accelerometers, proximity sensors, microphones, and cameras that can easily be accessed by various mobile apps. However, limited sensor types are available, and therefore applications that requirement environmental readings such as temperature or humidity will require an expansion module. The expansion module be in the form of a low-cost microcontroller that communicates with the phone either wirelessly over WiFi and Bluetooth, over USB, or through the audio jack [58]–[61]. A comparison of the most popular devices in the SBC and consumer electronics markets is presented in Table 3.1 and Table 3.2.

The third option is System on Chip (IoT-SoC). By "IoT-SoC" we refer to Arduino-based SoCs which match two important criteria; the ability to write to and/or read from sensing elements, and an on-board wireless communication module. Many IoT-SoCs manufactures go further to provide cloud services, and software libraries that facilitate seamless communication with the cloud. IoT chips like Particle's Photon and Electron [62], Pinoccio [63], Adafruit's Feather Huzzah [64], and ESP8266 [65]. Within the context of the proposed architecture, an IoT SoC will be able to fill in for the sensing segment, but not the processing. Most IoT-SoCs are Arduino-based, and have limited processing capabilities that cannot support required tasks such as application and database hosting. They can, however, function as a medium between sensors and the processing segment. Not only will it reduce the sensor interfacing load on the processing/communication unit, but it can also be used to expand the number and type of sensors which can be incorporated into system. A comparison of IoT-based SOCs and similar low level devices is shown in Table 3.1 and Table 3.2.

Most of the devices in the first two categories of SBCs and consumer electronics- are capable of performing application hosting and communication, and sensing. This is especially true for SBCs as they are designed with the intention to support sensors, and various peripherals. On the other hand, while Android-based phones and smart watches are mostly capable of hosting applications, they pose a limitation on the types of sensors that can be incorporated into the system. Most of them have basic sensors such as temperature sensors and accelerometers on board, but do not natively support the integration of external sensors. IoT-SoCs complement the latter, by having limited support for application hosting, but supporting a wide range of analog and digital sensors and actuators. Accordingly, we can propose two possible general implementations. The first implementation consists of an SBC with an optional SoC or shield, but only if needed. For example, a device such as Raspberry Pi can be used as a standalone pod to hose an application, handle communication, and read from digital data. If the system requires analog sensors, however, then an ADC shield or an Arduino-based SoC can be added. The second implementation consists of an Android-based phone or a smart watch to function as the first segment, which communicates with an SoC over audio jack or USB to read sensor information.

Figure 3.7 provides a visual comparison between the edge nodes in terms of RAM size, CPU capabilities in Dhrystone Mega Instructions Per Second (DMIPS), and supported operating systems. Dhrystone is a benchmark program that measures the number of integer instructions a processor can perform in a single second. While the CPU and RAM affect the performance of the system, the type of operating system is vital to the pod's role in the system because it determines whether or not the pod is capable of hosting the application server and the database, as well as providing readings from sensors. The distribution of the edge devices in the figure shows the huge capabilities difference between SoCs, and more sophisticated devices. While the most powerful SoC is the Photon with a 128kB RAM and a processor capable of 150 DMIPS, the smallest SBC, which is the C.H.I.P Pro, is equipped with a 256MB, and is capable of 2000 DMIPS. The relative price is also indicated by the size of markers, where the cheapest is C.H.I.P Pro at a unit price of $16, and the most expensive is SmartWatch3, at $135.

Table 3.1: Summary of popular SBCs and smart devices -Part 1

	RPi 3.0 [66]	ODROID-C2 [67]	BeagleBone Black [68]	Edison [53]	CHIP PRO [69]	Samsung Artik 710 [70]	Samsung Artik 530 [71]
CPU	4× ARM Cortex-A53 1.2GHz	4x ARM Cortex-A5 1.5GHz	ARM Cortex-A8 1GHz	2x Intel Atom 500MHz	ARM Cortex-A8 1GHz	8x ARM Cortex-A53 at 1.4 GHz	4x ARM Cortex-A9 1.2GHz
GPU	Broadcom VideoCore IV	Mali-450 MP2	SGX530 3D, 20M Polygons/S		Mali-400	Mali T400	Mali-400 MP2
RAM	1GB LPDDR2	2GB DDR3 SDRAM	512MB DDR3L	1GB DDR3	256MB DDR3	1 GB DDR3	512MB DDR3
Network	10/100 Ethernet, 2.4GHz 802.11n, Bluetooth 4.1 LE	10/100/1000Mbps Ethernet, Infrared(IR) Receiver	10/100 Ethernet	Bluetooth 4.0, 2.4/5GHz 802.11 a/b/g/n	-	10/100/1000 Ethernet , 2.4/5GHz 802.11 a/b/g/n/ac, Bluetooth 4.1, BLE, FM Radio, Thread	10/100/1000 Ethernet, 2.4/5GHz 802.11 a/b/g/n/ac, Bluetooth 4.2, BLE, FM Radio, Thread
Storage	microSD	eMMC 5.0, MicroSD	4GB eMMC, MicroSD	4GM eMMC, microSD	512MB SLC NAND, microSD	4GB eMMC	4GB eMMC
Ports	HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, CSI, DSI	HDMI2.0, HDML, I2S, 4× USB 2.0, GPIO / UART / I2C / ADC	1x USB 2.0	-	-	HDMI, MIPI, PCM, USB 2.0, USB 3.0	HDMI, MIPI, PCM, USB 2.0, USB 3.0
Pins/ Sensors	40pin port (GPIO/ UART/ I2C/ SPI)	40pin port (GPIO / UART / I2C/ ADC/ I2S)	2x 46pin port (GPIO/ UART/ I2C/ CAN/ SPI/ PWM/ ADC)	40pin port (GPIO/ UART/ I2C/ SPI/ I2C/ USB/ PWM)	53pin port (GPIO/ UART/ SPI/ PWM/ USB)	(GPIO, I2C, I2S, SPI, UART, PWM, SDIO, USB 2.0, JTAG, ADC)	(GPIO, I2C, I2S, SPI, UART, PWM, SDIO, USB 2.0, JTAG, ADC)
OS	Windows, Android, Linux	Android, Linux	Android, Linux	Linux	Linux	Linux	Linux
Price	$35	$46	$55	$37.5	$16	$52.50	$59

Table 3.2: Summary of popular SBCs and smart devices - Part 2

	Samsung Gear S [72]	Sony SmartWatch 3 [73]	Samsung J5 [74]	Samsung Artik 1020 [75]	Endless Mini [76]	Arduino Yún [77]
CPU	2x ARM Cortex-A7 1.0GHz	4x ARM Cortex-A7 1.2GHz	4x ARM Cortex-A53 1.2GHz	4x ARM Cortex-15 1.5GHz + 4x ARM Cortext-A7 1.3GHz	4x ARM Cortex-A5 1.5GHz	MIPS32 24k 400MHz
GPU	Adreno 305	Adreno 305	Adreno306	Mali-T628 MP6	Mali-450	-
RAM	512 MB RAM	512MB RAM	1.5 GB RAM	2GB LPDDR3	1GB RAM	64MB
Network	2.4GHz 802.11 b/g/n, GSM, HSPA, Bluetooth 4.1	Bluetooth 4.0 LE, NFC	GSM, HSPA, LTE, Wi-Fi 802.11 b/g/n, Wi-Fi Direct, hotspot, Bluetooth 4.1, GPS, NFS, Radio	10/100/1000 Ethernet , 2.4/5GHz 802.11 a/b/g/n/ac, Bluetooth 4.1, BLE, FM Radio, Thread	10/100/1000 Ethernet	10/100 Ethernet, 802.11b/g/n
Storage	4GB eMMC	4GB eMMC	microSD	16GB eMMC, SD	24GB SSD	microSD, 16MB flash
Ports	microUSB	microUSB	3.5mm analogue audio, 1x microUSB	HDMI, MIPI, PCM, USB 2.0, USB 3.0	2x USB 2.0, HDMI, 3.5mm analogue audio	LED, ICSP header
Pins/ Sensors	GPS	Ambient light, Accelerometer , Compass, Gyro, GPS	-	51pin port (GPIO / ADC / UART / PCM / I2C / I2S / SPI / SDIO / JTAG)	-	20pins (GPIO/ UART/ PWM/ ADC/ DAC)
OS	Linux	Android	Android, Linux	Linux	Linux	Linux
Price	$115	$135	$97.5	$105	$79	$72

Table 3.3: Summary of popular IoT-SoCs

	Arduino Uno [78]	Photon [79]	Pinoccio [63]	Huzzah/ESP8266 [64], [65]	Feather M0 Bluefruit LE [80]
processor	ATmega328P 16MHz	ARM Cortex-M3 120MHz	ATmega128RFA1 16MHz	Tensilica Xtensa L106 160MHz	Cortext-M0+ 48MHz
RAM	2kB	128kB	16kB	36kB	32kB
Networking	-	802.11b/g/n, soft-AP	802.15.4	802.11 b/g/n, Wi-Fi Direct (P2P), soft-AP	BLE
Storage	32kB flash + 1kB EEPROM	1MB flash	128kB flash	4MB flash	256kB flash
Pins	14pins (GPIO/ PWM/ ADC/ UART/ SPI/ I2C)	24pins (GPIO/ PWM/ USB/ CAN/ SPI/ I2C/ I2S/ ADC/ DAC)	32pins (GPIO/ PWM/ ADC/ SPI/ I2C/ UART)	28pins (GPIO/ UART/ SPI/ I2C/ ADC/ USB)	32pins (GPIO/ UART/ SPI/ I2C/ PWM/ ADC/ DAC/ USB)
On-board peripherals	LED, ICSP header	RGB LED	Temperature, RGB LED	Red LED	Red LED
Price	$22	$19	$49	$17	$30

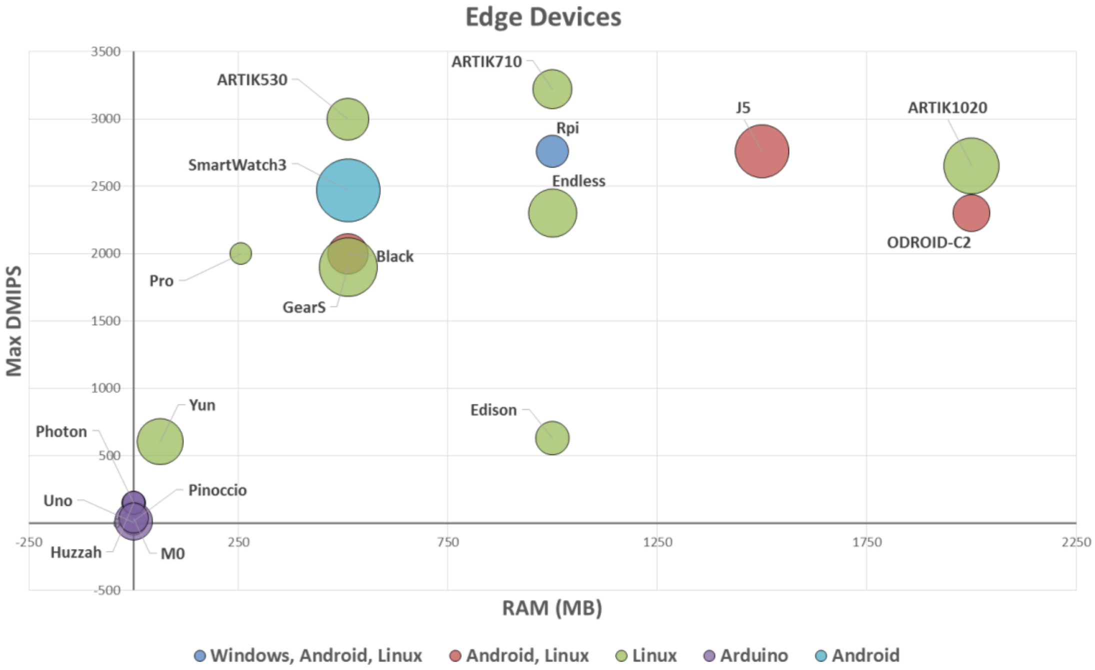


Figure 3.7: Edge nodes comparison

3.3.2. Communication layer. In terms of network scale, there are two types of networks over-which communication takes place: a local area network (LAN) for student-to-pod communication, and a wide area network (WAN) for pod-to-pods messaging. The importance of the student-to-pod being a LAN is to enable student clients to communicate with the server running on the pod locally. This functions as a security policy, as servers are not exposed to and accessible over the internet. LAN communication can also ensure that students are present at a certain location when communicating with pods instead of accessing it remotely; which is a major component of context-aware learning. On the other hand, a question or an attempt posted to one pod will most likely be destined for a group of remote pods. For example, a student may post a question that will be available for all students enrolled in Grade 5 mathematics in all schools that follow the same curriculum. Attempts posted by students should also be sent to the main server if the teacher or school require performance analysis data. Hence, pod-to-pods communication can only sufficiently take place over the internet. Internet access is also important for students to receive notifications when a relevant question has been posted, or significant system changes have taken place. In terms of communication medium, allowing students to walk around a location freely and connect to any pod entails that communication between students and pods takes place wirelessly. In addition, although the pods are not mobile, using wireless technologies to connect the pod to the internet reducing cabling cost and minimizes inconvenience caused by extending wires through the learning location. Based on the previous discussion, we define two possible network architectures, shown in Figure 3.8.

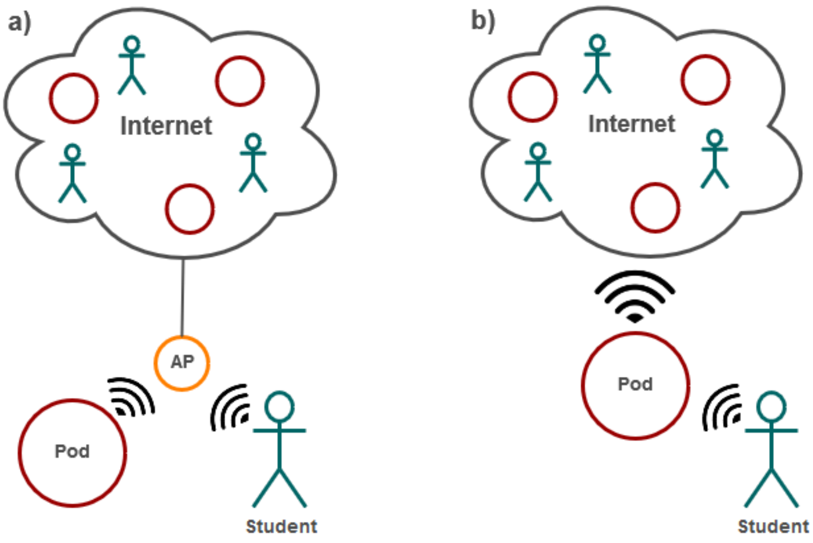


Figure 3.8: Pod network architecture: a) pod as a LAN client. b) pod as a hotspot host.

In the first architecture, a wireless access point provides communication between students and local pods, and internet access for both. In this architecture, students can request or post questions and attempts over the wireless LAN (WLAN), while messages meant for remote pods, servers, and students are routed to the WAN. Students in this case need only to connect to the WLAN once at the beginning of the session in order to communicate with any pod in the area, and receive real-time notifications. Contrarily, the second architecture sets the Pod up as an access point, where the pod uses one network interface to access the internet, while the second interface is used to deploy a WLAN. In this case, students need to connect to a specific pod's hotspot in order to communicate with the server running on it. At time when students are not connected to a pod, they can connect to a normal wireless access point, or to a 4G or 5G network. In both cases students can receive notifications as both normal access points and pods' hotspots are able to provide internet access. The second options has the advantage of ensuring a student is at the proper location in order to connect to the pod and answer context-specific questions. Furthermore, all pod application servers can be assigned the same IP address since the hotspots are mutually exclusive. This enables the student app to automatically connect to the proper server without the need to explicitly specify server information such as the IP address, or scan a QR code, for example. The downside, however, is that deploying the hotspot can cause additional power consumption.

3.3.3. Middleware layer. The middleware layer accommodates technologies that process, transport, and store information. This includes technologies such as servers, database software, communication protocols, cloud computing, and any other technologies that transform data acquired form edge devices into information presented at the application layer. The proposed architecture employs three types of middleware layer technologies: databases, application-level messaging, and backed application servers.

Questions, answers, student scores, and sensor readings must to be stored in a database for later access. Based on the period of time for which we will like to keep the information, we define two classes of database entries. The first type is information that should be stored indefinitely, and can be accessed at any time by any user. In [81] the student's ability to integrate previous knowledge to build new knowledge, and the student's ability to monitor their own progress in order to adjust their learning approach are underlined as two of the important learning principles. Allowing the student to access their own previous contributions and results allows to self-regulate their learning approach and adjust their learning approach so that any misconceptions or weak points will be tackled early on. Moreover, allowing the student to access their own and their peer's old contributions enables them to build upon previously acquired knowledge in order to get a coherent understanding of the curriculum. For these reasons, the system requires that every user is equipped with a scalable database that can standalone in case of a network failure, and synchronize with other databases when the network is up and running. On the student node's side, sensor readings need not be stored permanently. Since students require only the current sensor readings to integrate into questions and answers, the latest readings received from an information pod should replace the previous one. The readings also need not be replicated to other databases as sensor data should be acquired individually. Along the same lines, once an information node has received a sensing request, communicated with the sensing unit, and replied to the student, the information node should maintain the reading for a short period of time and use it to reply to other students instead of communicating with the sensing unit again.

This helps reduce the frequency of sensing and therefore saves power. For the choice of database we consider SQL-based MySQL [82] and Cassandra [83], and NoSQL-based databases CouchDB [84] and MongoDB [85]. In [28] we presented a comparison between the four databases, which can be seen in Table 3.4. The heterogeneity of data in terms of size, format, and lifecycle gives NoSQL databases an advantage; as they are document-based and therefore allow room for data flexibility. SQL database are more constrained in terms of the data structure and therefore the data will have to undergo reformatting in order to be inputted into the database. The second most important criterion is replication. As explained earlier, the application on each student's device will access a database that replicates with other databases; in order to stay updated with all contributions and also perform as a standalone unit in case of network failure. This requires master-to-master replication functionality, which allows individual users to perform insertion and update operations, and peer-to-peer data synchronization; which is available only in CouchDB and MySQL. The third criterion we consider is Map Reduce [86]. Map Reduce works in two steps: the map function filters documents according to a certain condition, and then the reduce function groups documents based on that condition. This enables parallel processing of large data, and boosts the performance of the system. Finally, since we aim for a JavaScript implementation all over the system as means to increase compatibility, we look for JavaScript support.

Table 3.4: Comparison of NoSQL and SQL databases

	CouchDB [84]	MongoDB [85]	Cassandra [83]	MySQL [82]
Database class	document store	document store	column store	relational DBMS
Data format	JSON	JSON	native, collection, user_defined, tuple, custom	numeric, string, date/time, JSON, spatial
Consistency	eventual	eventual	tunable	
Scalability	high	high	high	high
Supports JavaScript	yes	yes	yes	no
Map Reduce	yes	yes	yes	no
Replication	master-master master-slave	master-slave	master-master	master-master master-slave

The messaging middleware is responsible for handling the inquiry aspect, which is the core of this work. One of the most important requirements for the messaging protocol is support of large scale networks. Applications of this class -inquiry-based learning IoT- are designed with the ultimate goal of deployment on a global scale [23], where a large number of students, teachers, and information nodes will be sending and receiving questions, answers, and reports. This places huge pressure on the messaging broker as parameters such as latency and throughput become of great importance. One important factor that we take in mind is the communication architecture implemented by the protocol. This is because, as mentioned earlier, communications taking place in the system can be one-to-one or one-to-many. When a student proposes a question on a certain topic, the question should be broadcasted to all students enrolled in that class. On the other hand, a sensor data request made by a single student should be responded to by a unicast to that particular student. The second important factor is the data format supported by the protocol. The assessment standards considered here differ in question format, and while QTI and MoodleXML, for example, are based on XML, GIFT specifies a different format that is not necessarily supported by common communication protocols; in which case the conversion between different formats will be handled by the system. HTTP (HyperText Terminal Protocol) and XMPP (Extensible Messaging and Presence Protocol) are two of the most popular classic communication protocols, while MQTT (Message Queue Telemetry Transport), AMQP (Advanced Message Queuing Protocol), and CoAP (Constrained Application Protocol) are the most common for Wireless Smart Sensors (WSN) and IoT applications. Given that every node is equipped with computing device capable of communicating through either of the aforementioned protocols, we consider all of them for the messaging middleware.

During the early discussions of IoT technology, the choice to employ HTTP [87] as the communication protocol was at times dismissed for reasons that included large overhead, and high processing power and memory requirements which were not available for typical IoT constrained devices. However, a few researchers began investigating the performance of HTTP in an IoT as a new service-oriented, social media-based IoT paradigm emerged; which is now commonly referred to as the Web of Things (WoT) [42], [88]. HTTP facilitates the integration of web services and resources, and its employment in IoT applications is made possible by a cross-layer TCP/HTTP [89] optimization that, done properly, can run the protocol on small constrained devices. One issue that occurs right away is that in a system which requires a publish/subscribe communication mechanism, specifically between student nodes, HTTP implements a request/response architecture (Figure 3.9). One option, in this case, is to shift the responsibility of updating nodes with new questions to the nodes themselves. Nodes will be responsible for sending GET requests at a certain frequency in order to stay updated with the latest contributions. The communication between student nodes and information nodes, however, will be simple and direct. In terms of data format, HTTP supports HTML with XML and JSON.

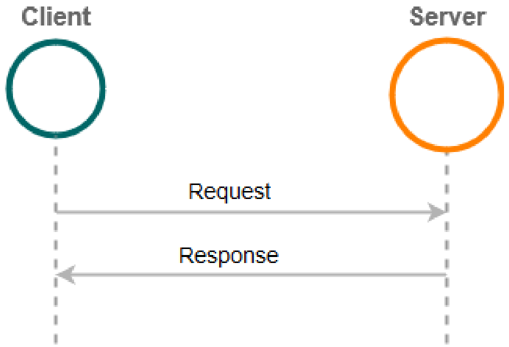


Figure 3.9: HTTP model

CoAP [90] is a messaging protocol specifically developed with the IoT constrained devices in mind. The protocol implements a Representational State Transfer (REST) architecture [91] which gives it an advantage over other protocols; as it enables embedded constrained devices to use web services; combining the benefits of HTTP and MQTT. One common misconception is that CoAP is “compressed HTTP”, which is not at all the case. CoAP offers the same features as HTTP at a lower cost by replacing the TCP layer with UDP, and compensating for the implicated loss of reliability by introducing a “message layer” [92] which handles packet sequencing and retransmission. One downside of it, such as the case with HTTP, is that it does not natively support the publish/subscribe architecture [93], but the architecture can be implemented using “observe” streams (shown in Figure 3.10). Another downside will be the CoAP packet size which is set to 1152 bytes per packet, 1024 of which are for the payload, which poses constrains on the amount of data, especially non-textual, that can be included in a message. In order to handle this issue, either data compression or data packet segmentation must be employed, both of which could significantly affect the performance of the system. In order to provide security that UDP lacks, CoAP implements Datagram Transport Layer Security (DTLS), which provides TCP-like privacy and integrity using handshakes and encryption [94].

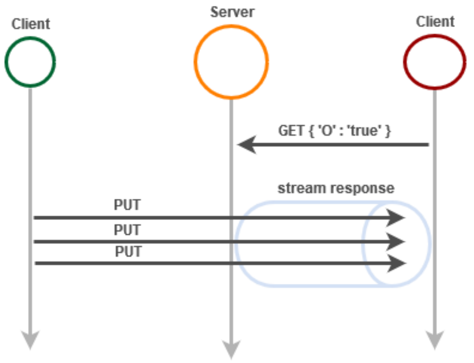


Figure 3.10: CoAP "Observe" model

XMPP [95] is an XML-based real time communication standard widely used by instant messaging applications. Despite its wide popularity among internet applications, it is similar to HTTP and often overlooked by IoT developers due to its complexity. XMPP was especially designed for web browsers and is often referred to a as a heavyweight protocol [96], [97]. Nonetheless, the protocol offers a superior performance, and its set of open XML technologies can be further-enhanced by protocol extensions (XEP) which offer features such as discovery, presence, and group chats. The latter serves as to adapt the protocol to diverse environments, which makes it a strong candidate for messaging in IoT environments with more complex and capable devices. Technically speaking, the standard follows a client-to-server-to-server-to-client architecture -where a message sent from one node is routed to its destination through a group of inter-communicating server- which is often regarded as a “logical” peer-to-peer architecture (Figure 3.11). On the other hand, using XEP the protocol can easily support a publish/subscribe architecture. This implies that it can support both of the communication types we define in an inquiry-based learning application.

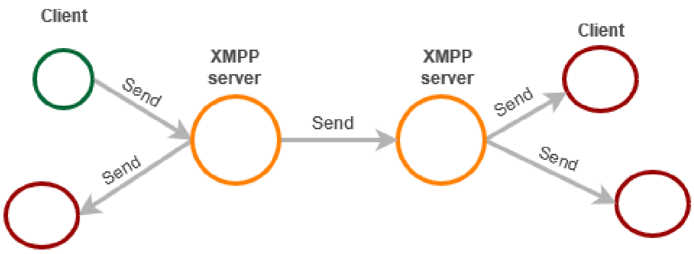


Figure 3.11: XMPP model

MQTT [37], [93], [98], [99] is a low-power, low-memory messaging protocol that has gained popularity in WSNs, and lately in mobile-based instant messaging applications [100]. It follows a data-centric approach, meaning that the messages a user receives are based not on their IP addresses, but on their characteristics and interests. The protocol originally implements a publish/subscribe architecture which enables easy one-to-many messaging from one publisher to multiple subscribers based on a “topic” tag. However, with some careful topic customization the protocol can implement a peer-to-peer architecture. The model for MQTT is shown in Figure 3.12.

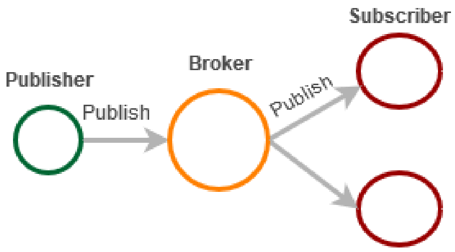


Figure 3.12: MQTT model

The format for MQTT message is not specified and is completely up to the developer to defines, which can serve as a pro and a con; as it gives the developer the freedom to define the format based on other components in the system, but increases the amount of implementation time and effort. In the TCP/IP stack, MQTT functions on top of the TCP layer. However, one option that is rising in popularity, especially among open-source projects such as HiveMQ [101] and Paho [102], is to implement MQTT on top of WebSockets. The upside of this option is that it makes it more compatible with other communication protocols, and easily integrate-able with HTML5 applications; as there are already libraries for MQTT over WebSocket. The downside of it, however, is that it sacrifices the low-weight benefit of the original implementation by causing more overhead.

AMQP [103]–[106] is a messaging protocol for message oriented middleware (MOM) originally designed for financial applications. The protocol provides distributed enterprise messaging with features including opaque, immutable content with no size limitation, reliability, and security. These features contributed to the adoption of AMQP for business-critical communication in a wide range of enterprise applications. Each AMQP message consist of pre-defined headers similar to ones defined by HTTP, and a message payload. The payload is a byte array, meaning it can be used to transport any message format as long as it is serializable. Figure 3.13 shows AMQP messaging model.

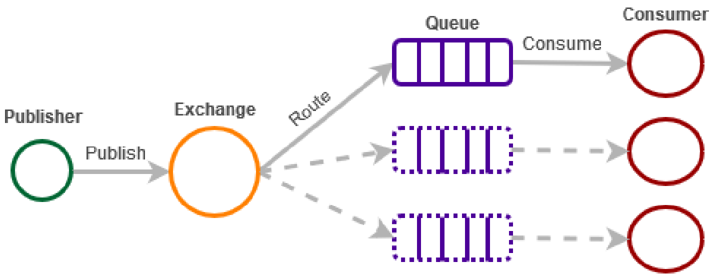


Figure 3.13: AMQP model

The choice of protocol often depends on factors such as data format, development language, security, and others. A technical comparison of the five protocols is presented in Table 3.5.

Table 3.5: Summary of the four messaging protocols

Protocol	HTTP	XMPP	MQTT	AMQP	CoAP
Native architecture	Req/Res	Pub/Sub, Req/Res	Pub/Sub	Pub/Sub	Req/Res, Pub/Sub using Observer
Message format	HTTP,XML, JSON	XML	Not defined	Not defined	XML, JSON, CBOR
Transport	TCP	TCP	TCP	TCP	UDP
Default payload limit	2MB for Apache, 2MB for IIS, adjustable	1MB for OpenFire, changeable	256MB	131kB, but segmentation is supported for larger messages	1280B for IPv4/IPv6, 127B for 6LoWPAN,
Header Size	8kB for Apache 26kB for IIS	3kB	2B	8B	4B
Security	TLS (HTTPS)	TLS (optional)	TLS/SSL (optional)	TLS, SASL, (optional)	DTLS (optional)
Constrained device support	No	No	MQTT-SN	Yes	Yes

In order to run the applications on all defined types of nodes, manage the databases, and facilitate system integration into the IoT world, an internet application server is needed. We consider three alternatives: Apache Tomcat, Apache + PHP, and Node.js+Express. All three servers are open source and are highly popular among web server developers. One important difference between them is the implementation language. Tomcat is Java-based, the second server uses PHP while the Node.js + express combination is JavaScript-based. A comparison between Node.js and Apache + PHP was presented in [107], while [108] compares Node.js against Java-based web servers. A brief comparison is presented in Table 3.6. Based on implementation language as well as some other criteria, we choose Node.js [109] as the web server. Node.js is an event-driven, non-blocking I/O server built on Chrome's V8 runtime engine. It was developed to solve the C10K problem, which refers to servers' inability to support more than 10,000 concurrent customers, and is efficient and lightweight, which makes it an ideal for running servers on mobile phones or microcomputers. The fact that is based on JavaScript provides the base for unifying the development language across the whole system [110], making it more convenient for developers. Along with Node.js, we are using Express [111], a web application framework which provides URL routing and the freedom of selecting the storage unit of choice, thus giving us freedom regarding the database. It supports HTTP protocols and may also work hand-in-hand with Socket.io. The only downside is that NodeJS web-based applications can crash with rapid traffic growth. NodeJS webservers can underperform in terms of serving static content like images and JS files, or when load-balancing between multiple servers. NGINX [112], an open-source high performance web-balancer and reverse proxy. NGINX is implemented based on a very similar architecture to NodeJS, and is thus easily integrated into a NodeJS application server environment.

Table 3.6: Comparison between alternative web servers

	Apache Tomcat	IIS + ASP.NET	Apache + PHP	Node.js + Express + nginx
Programming language	Java	.NET	PHP	JavaScript
Architecture	Thread-based	Thread-based	Thread-based	Event-driven non-blocking I/O using thread pools
Robustness	High	High	High	High
Scalability	High	High	Low	High
Embedded devices support	Low	No	Low	High

3.3.4. Application layer. The application layer serves as an interface between the users and the system. It integrates information and provides users with applications to monitor and interact with the system. One key requirement in the architecture we propose is ubiquity, the environment must be accessible on smartphones and/or tablets the students and instructors already own and are used to operating [113]. The first question we encounter at this stage is whether the application should be a native application or a web browser-based application. However, considering that have already defined the ability to function as a standalone unit as a system requirement, we immediately disregard the web-browser option. The second question is how the application should be developed, and we are offered two options: native, or cross-platform. Native applications are developed specifically for their respective platform, and while that enables them to take full advantage of the device hardware, once we introduce multiple platforms, the process of developing and maintaining an application for every platform becomes less efficient and more daunting [114]. Factors such as choosing the best SDK, framework stability, and guaranteeing the same user experience across all native applications play a great role in the development process as they directly affect the cost of development, updates, and marketing [115]. A more feasible option is to develop a cross-platform application which can be deployed all smart devices, with no more than a small amount of tweaking. Cross-platform development tools like Cordova [116] enable developers to write the main application in HTML5, and then deploy it on a variety of devices with the most popular ones being those manufactured by Apple, Samsung, LG, and HTC [117]. The downside, however, is that cross-platform applications cannot necessarily access all of the mobile's devices, which is why heavily-hardware-dependent applications may opt for native applications instead. Nevertheless, in the context of our application, we find that cross-platform development offers great advantages, with little to no sacrifices. For reference, a thorough comparison of the two approaches have been presented in [115] (seen in Table 3.7).

Table 3.7: Comparison of native app development vs. cross-platform app development

	Native	Cross-Platform
User experience quality	Excellent	Good
App quality	High	Medium to low
Potential User	Limited to particular platforms users	Covers users of all platforms
Development cost	High	Medium to low
Security	Excellent	Not as good
Ease of updating	Complex	Medium to complex
App extension	Yes	Yes

3.4. System Implementation

The final form of the system architecture and its components is shown in Figure 3.14. The major focus is on the edge device because it is responsible for interacting with and server end users, while also complying with restricting power and computation constraints. The pods will be distributed around the physical learning environment, often in locations where powering the pods using wires is infeasible. Remaining options will be to either power it using a portable power source; battery packs, rechargeable power banks, or solar chargers. Designing the pod for low power requirements reduces the cost of the power source, as well as the frequency at which it must be recharged or replaced -in the case of power banks and battery packs. As for the second constraint, which is computation, we refer to Figure 3.7. Among the devices compared earlier and shown in the figure, RPi seems to provide the best tradeoff between capabilities –

computation, operating system, sensor interfacing- and price. The edge devices takes in two types of networks, which we defined earlier as the edge network, and the core network. Edge network here refers to the student-to-pod communication, while the inner network refers to pod-to-pods communication, as stated in the Communication Layer section. In that sense, the Pod functions as the user's gateway to the system, and is required to support the type of communication dominating each network. The majority of communication between a pod and a student consists of the student requesting services. Whether the student is posting a question or an attempt, and retrieving a question or sensors readings, the type of communication between the student and the pod is natively one-to-one, request-response based. This is why, at the edge network, we choose to conduct communication over HTTP. This is not true for the core network, however. Each single question or attempt posted to a pod generates traffic that needs to be delivered to numerous students, teachers, and other pods. Furthermore, students, teachers, and pods do not know when a new question or attempt has been posted, but the traffic is rather generated due to the event of a new question or attempt. While client-server architectures can be customized to fit this communication, it will be highly inefficient. One-to-many, event-oriented communication is more naturally implemented using publish/subscribe architectures, where one or more clients subscribes to receive updates from one or more resources, and are automatically updated with new events by the broker. Thus, MQTT, AMQP, XMPP, and observer-based CoAP are better suited for the inner network.

3.5. Summary

This chapter presented the proposed system architecture which is believed to best meet the requirements. The sections first defined the system, then explored possible options for each element. Based on the comparisons in each section, a selection of technologies was chosen to perform all of the required roles. Because the Pod is responsible for connecting the system elements together, while also being limited by certain physical constraints, it will be the focus of the evaluation process. The most important among the Pods' tasks is the communication, as it transforms the system from a simple assessment application to a global network connecting students, instructors, and assessments. It is also believed to be the element that affects the Pods' performance the most, in terms of hardware resources, and user experience.

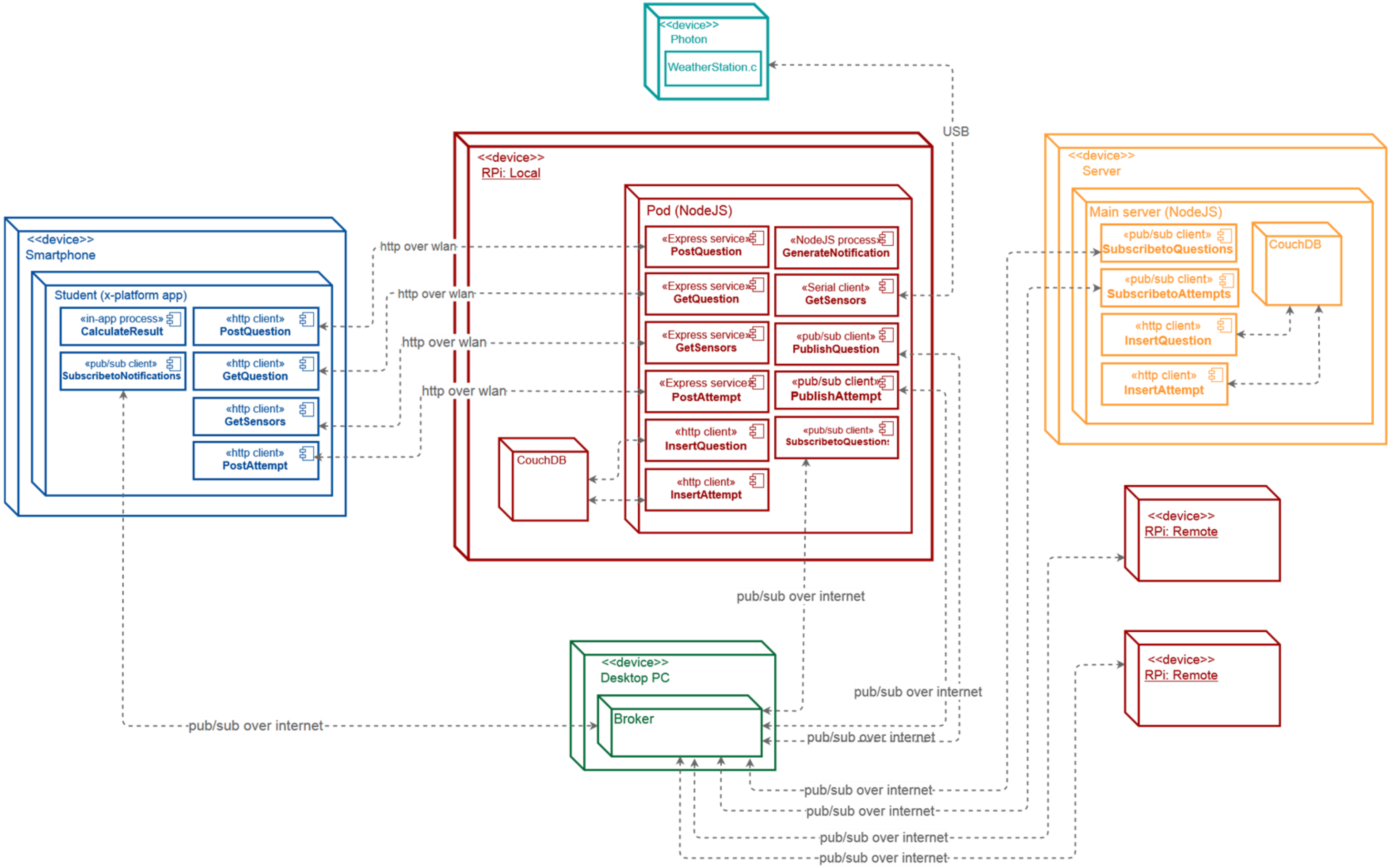


Figure 3.14: System deployment diagram

Chapter 4. Experimental Design

In this chapter, we discuss the evaluation setup used to compare the four implementations. Before any experiments are conducted, evaluation components must be described. The first component is the evaluation criteria. A set of criteria is determined by examining literature on ubiquitous inquiry-based systems evaluations, as well as studies that evaluate the protocols in other applications. Next, an evaluation scenario is decided upon. The evaluation can either be conducted by running all of the scenarios explained in Chapter 3. , or by focusing on scenario(s) where the edge device performance has the widest impact. While the first option provides an overall evaluation for the protocol's performance in the system, by focusing on the most demanding scenarios, it is possible to run more repetitions of the experiments and acquire performance statistics. The third component is the experimental data. Different IoT applications generate and circulate data in all sizes and formats. Thus, using data that best mimic the actual application data is fundamental in finding the protocol that best suits the specific type of application. Finally, input and output variables must be defined. Using appropriate tools to generate input variables and measure outputs is crucial to achieving an accurate, credible evaluation. While input variable are extracted from possible usage scenarios, output variables are based on evaluation criteria. The number, type, and range of input variables that influence the evaluation are defined are designed to mimic realistic situations, but also exaggerated values that stress test the system. The four components are then combined with software and hardware tools to design the experiments.

4.1. Evaluation Criteria

Based on the use cases presented earlier, and the evaluation of IoT applications presented in [93], [118], [119], we define a set of criteria for which the architecture will be evaluated. The criteria we evaluate for are: performance, reliability, latency, scalability, power efficiency, throughput, ease of implementation, and security. The experimental setups provides a realistic indicator of how the system will perform in a real life situation, and how much the system can be stressed but still achieve acceptable performance. The two most important advantages of physical implementation is that it makes it possible to measure hardware and network metrics, and get a real sense of how easy or difficult it is to implement the system using the technologies. Of the given criteria, the metrics calculated during the experiments are power/energy, performance in terms of CPU and memory utilization, and network latency and throughput.

Table 4.1: General criteria of evaluation

Criterion	Definition	Type
Performance	The system's capability of performing the required task, measured in terms of CPU and memory utilization	Quantitative
Latency	The time it takes a message to reach its destination	Quantitative
Throughput	The number of data bits that can correctly be communicated within a period of time	Quantitative
Scalability	The system's ability to handle a growing number of users, data, and traffic	Quantitative
Power	Power consumed by the device while using the system, especially important for embedded devices	Quantitative

4.2. Evaluation Scenario

The evaluation was done using the first scenario called "Post a Question", for multiple reasons. Since the purpose is to evaluate the four IoT protocols, the second and third are not necessary; as they depend on HTTP purely. The two scenarios that can be used are the first and last; which are Post a Question, and Post an Attempt. The two scenarios are technically similar in the sense that an HTTP request from the student client results in the pod publishing a message to other pods. However, realistic test data is available for the first scenario only. Furthermore, posting a question also requires more work on the pod's behalf to generate and publish a notification message, in addition to the original question message. The evaluation can be divided to two experimental setups; based on the independent variables. In the first setup, we test the effect changing the number of students using the system at a particular moment, and the frequency at which each student posts a question, on the pod's resource consumption. The second setup, on the other hand, assess the system's end-to-end latency in response to imperfect network conditions. In order to acquire enough data to perform statistical analysis and draw generalized conclusions, each test case in each experiment was replicated 15 times.

4.3. Experimental Data

Two question banks are available for the evaluation stage, used to simulate the "Post a Question" scenario. The first was retrieved from the Electronic Document

Management System (EDMS) database. The set consisted of a total of 2890 questions from Math, English, General Knowledge, Islamic Studies, Social Studies, and Urdu courses. The original questions format was XML. Because the system was implemented in Javascript, CouchDB accepts JSON only, and all of the protocols are capable of communicating serialized JSON, the questions were converted into JSON objects. Table 4.2 shows the question size statistics.

Table 4.2: EDMS question bank sample statistics

EDMS Question Size Statistics (Bytes)	
Mean	*1223.10*
Standard Error	*51.61*
Median	*486*
Mode	*232*
Standard Deviation	*2774.25*
Sample Variance	*7696441.75*
Kurtosis	*147.06*
Skewness	*9.26*
Range	*65749*
Minimum	*124*
Maximum	*65873*
Sum	*3534745*
Count	*2890*

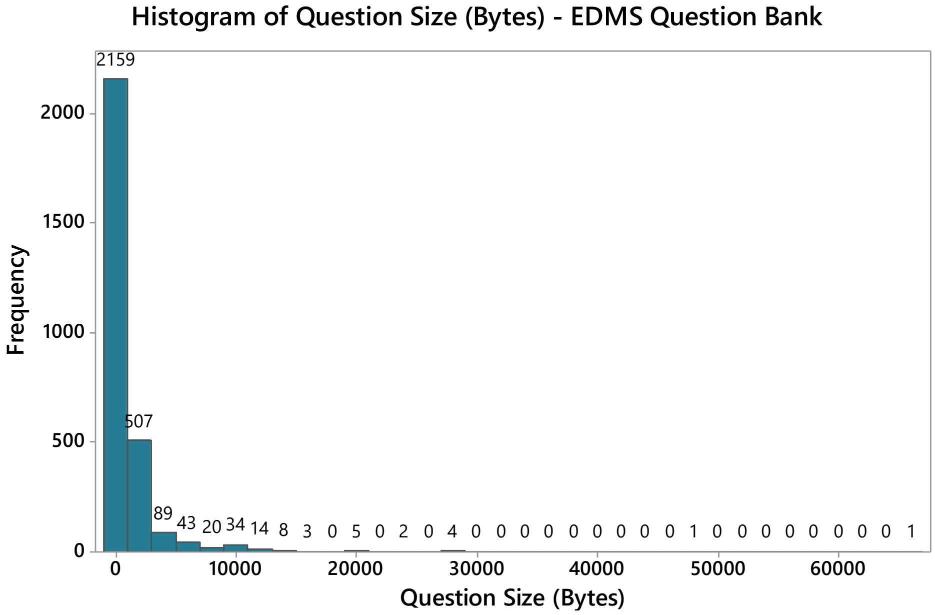


Figure 4.1: Question size distribution - EDMS set

The questions pool includes short answer, multiple choice, and true or false. The questions can also be either text-based, or include a serialized media object. Figure 4.2 shows an example of a Science multiple choice question. In such questions, the correct answer is included in the body of the question, but will not be displayed on the student app and will instead be used to provide automatic feedback without the need for the teacher’s interference. This is possible for multiple choice and true-or-false types, but not short answer questions.

```
// question: 435  name: DaysAndNights_0604052100201104
::DaysAndNights_0604052100201104::[html]<p>The occurrence of days
and nights is caused by the _______.<br></p>{
	~<p>Earth's spin around the Moon<br></p>
	=<p>Earth's spin on its axis<br></p>
	~<p>Earth's spin around the Sun<br></p>
	~<p>Earth's spin around Mars<br></p>}
```

Figure 4.2: EDMS multiple choice question

The question bank was used to assess the four implementations from quantitative and qualitative aspects. All of the questions from the EDMS set were pre-uploaded to a CouchDB server running on a PC. During the experiment, student client simulators request a random question from the database, then proceed to POST it to the edge device. The sizes of the question were well-supported by all protocols but CoAP. The maximum payload size supported by CoAP is 1280 Bytes. Since the payload includes not only the question, but also the student’s information, CoAP will not be able to send the majority of the available questions. However, for testing purposes, the experiments were designed so that if the random question combined with the student information exceeds the limit, it will be switched for a small question. The small question was defined as a constant variable in the student client simulator as additional time caused by re-requesting another question from the database. The values for each metric were chosen to range from real life situation values to mimic normal operation, all the way to exaggerated values that were aimed to stress test the system.

4.4. Experiment I

The objective of the first experiment is to assess the protocols in terms of resource usage including power consumption, CPU utilization, and RAM usage. The energy consumption becomes of crucial importance in the case of information nodes because not only will they be spread around outdoor locations, but they will also be constantly receiving and replying to requests and questions. Recharging them then becomes an issue as it requires the node to be taken down, and the more frequent the process is, the worse it is for the system performance. The most efficient solution to this issue will be to use solar batteries. In this case, measuring energy consumption is important for calculating the size of solar cell required to support the edge device's needs. Current consumption of the edge device during its operation with every protocol was measured using YoctoAmp [120], an isolated USB ammeter. Besides power consumption, the two crucial performance metrics are CPU and memory utilization. These metrics can be measured using nmon [121], a system performance benchmark tool for Linux-based systems. Nmon provides measurements of CPU utilization, memory usage, disk I/O reads/writes, and other metrics that can be used to compare the protocols and determine the most efficient one in terms of resources. The independent variables are: the number of students posting questions to the pod, and the frequency at which each student posts questions (Table 4.3). The dependent variables are shown in Table 4.4 while the control variables are shown in Table 4.5.

Table 4.3: Experiment I independent variables

Independent variable	Definition	Min	Step	Max
Question wait	The waiting time between two consecutive question post actions	1Q every 4min	2^x Q every min	1Q every 1min
Number of students	Number of student client processes	1	2^x	128

Table 4.4: Experiment I dependent variables

dependent variable	Definition	Unit
Power consumption	Power consumed calculated based on ammeter values	Watt-hour
CPU utilization	CPU% consumed by the NodeJS app server during the experiment	%
Page faults	The number of times at which the system fails to retrieve data from the RAM	Page fault/s

Table 4.5: Experiment I control variables

Control variable	Definition	Value
Channel bandwidth	Maximum bitrate supported by the edge device NIC	18.8 Mbps
Channel quality	Channel conditions such as lag and throttle	ideal
Server load	Number of other processes using server resources	N/A
Question bank	Questions are chosen randomly from a defined pool of questions	Table 4.2

During the experiment, each student client was simulated by a NodeJS process which pulls a random question from a questions repository stored in CouchDB, and then POSTs it via http to the Express server running on the Raspberry Pi. JXCore [122] was used to multithread student NodeJS processes, and simulate different situations by controlling the number of threads and the wait time between posting. A block diagram showing major components of Experiment I is shown in Figure 4.3.

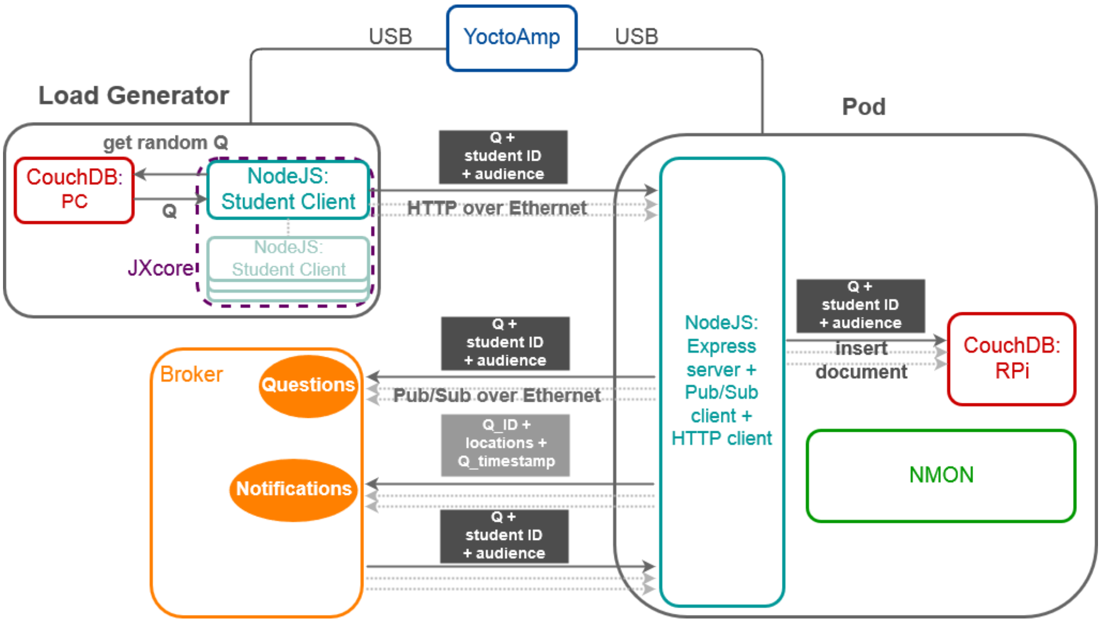


Figure 4.3: Experimental setup I

4.5. Experiment II

The objective of the second experiment (Figure 4.4) is to assess the performance of each protocol under imperfect network conditions. The primary metric is latency. We define latency as the time, in ms, required for a packet published from one pod to reach other pods using each of the IoT communication protocols. One way to calculate it is using timestamps, where "sent" and "received" timestamps are added to each packet depending on the moment of time it leaves the pod, and the moment it reaches another pod, respectively. By subtracting the two values it is then possible to get an estimate of the time period through-which the packet was able to travel through the network. The second network estimate is throughput in terms of bits/s, which can easily be extracted using Wireshark [123]. Clumsy [124] was used to control the network conditions. In order to measure a baseline channel bandwidth, Clumsy was activated to filter the packets without explicitly adding lag, throttling, etc. The effective bandwidth was then measuring using iperf [125], which is a tool for measuring the maximum possible bandwidth on IP networks. Measuring the bandwidth using iperf revealed that the original bandwidth we being our experiments with is actually around 20 Mbps, compared to the original value without Clumsy which was around 94 Mbps. The 20

Mbps bandwidth, which is in fact closer to the edge device's WiFi speed, and also the bandwidth for a 4G link. Since the tool does not offer a command line interface, automating different values was made possible using TestComplete [126], a GUI testing and automation tool. Every run consisted of 16 identical clients publishing a question every minute each, over the period of 10 minutes, and each run was repeated 10 times. The runs provided a total of 1600 questions with individual latency values that are used to identify the behavior of each protocol in each case. Independent variables are shown in Table 4.6, while the dependent variables are shown in Table 4.7, and control variables in Table 4.8.

Table 4.6: Experimental Setup II independent variables

Independent variable	Definition	Min	Step	Max
Lag	Hold the packets for a short period of time to emulate network lagging	100ms	100ms	1000ms
Throttle	Block traffic for a given time frame, then send Them in a single batch	5%	5	90%
Duplicate	Send cloned packets right after to the original one	5%	5	50%
Out of Order	Re-arrange the order of packets	5%	5	50%

Table 4.7: Experimental Setup II dependent variables

Dependent variable	Definition	Unit
Latency	Time needed for a question to travel from the publishing pod to other pods in the system	ms
Throughput	Number of bits transferred per second	bps

Table 4.8: Experimental Setup II control variables

Control variable	Definition	Value
Ethernet channel bandwidth	Maximum bitrate achievable by the Ethernet channel	94.3 Mbps
Clumsy Ethernet channel bandwidth	Maximum bitrate achievable by the Ethernet channel after Clumsy filtering has been activated, but no metric were explicitly set	20.9 Mbps
Server load	Number of other processes using server resources	N/A
Message size	Questions are chosen randomly from a defined pool of questions	Table 4.2
Publishing frequency	Number of questions posted by a given student client in a minute	1Q/min
Number of students	Number of student client processes connected at any given moment in time	16 students

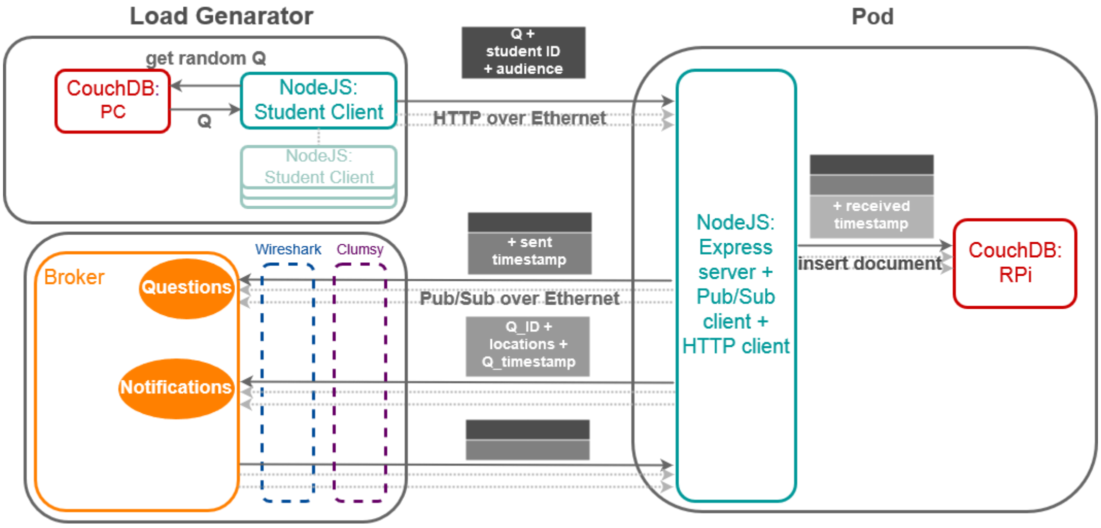


Figure 4.4: Experimental setup II

4.6. Evaluation Hardware

The edge device used to build the pod in the experimental setup was a Raspberry Pi 3.0. However, we can assume that its behavior will apply to any edge device of the same specifications. The specifications for the RPi 3.0 is shown in Table 4.9.

Table 4.9: Edge device specifications

RPi 3.0 [66]	
CPU	4× ARM Cortex-A53 at 1.2GHz
RAM	1 GB LPDDR2-900 SDRAM
Ethernet speed	94.3 Mbps
WiFi speed	20.9 Mbps
Storage	16GB microSD
OS	Raspbian Jessie with PIXEL

Chapter 5. Results and Discussion

This chapter consists of a discussion of quantitative results obtained from the two experimental setups discussed earlier. The first section discusses resource-related metrics including power consumption and CPU utilization. The second section then discusses user experience-related metrics including communication latency and throughput.

5.1. Experiment I

This section presents an analysis of the results acquired using experimental setup I, which focused on the resources consumed by each protocol. The resources include power consumption, CPU utilization, active memory, and network reads/writes.

5.1.1. Power consumption. The power consumed over 24 hours for each protocol is shown in Figure 5.1, and the values shown in the figure represent the average taken over all of the replications. Power consumption is expressed in Watt-hour (Wh). The power consumed over a certain period of time, is calculated as

$$P = V_{(V)}.I_{(A)}.t_{(h)} \tag{1}$$

For example, if the average device draws an average current of 300mA, given the average voltage is 5V, the power consumption for 24 hours will be 36Wh. The ammeter used in the setup provides current readings at the rate of value/second. Therefore, power consumption can be used by replacing the *I.t* value with the integration of the current value over the time period. In such case, equation (1)
can also be expressed as:

$$P_{day(Wh)} = V.\sum_{i=0}^{86,400} \mathrm{I}_{i(A)} \tag{2}$$

Finally, to calculate solar panel size in watts [127], simply divide the power consumption in a day by the number of hours in a day that the peak sun hours; i.e. the number of hours in a day that the panel will be exposed to peak sunshine.

$$Panel\ Size_{(W)} = \frac{P_{day(Wh)}}{t_{peak(h)}} \tag{3}$$

For example, the worst case scenario in terms of sun peak hours in the United Arab Emirates is 4.5 hours; during January. Given the previous power consumption value of 36Wh a day, the panel size required will be 8W. The results for the power consumption are shown in Figure 5.1. As expected, more power is consumed as the number of students and the frequency of posting questions increases. This is due to the increasing stress on the Pod to process, store, and publish more questions in a given minute. The baseline power consumption for the edge device in idle mode was measured to be 40.21 Wh. Among the four protocols, MQTT consumes the least amount of power, with the maximum value at the 128-1 point (128 students publishing 1 question per minute) amounting to 42.6 Wh. On the other hand, the highest amount of power consumed is around 43.8 Wh, by XMPP and CoAP. This is consistent with the behavior exhibited in [128] AMQP demonstrate a power consumption behavior of 42.2 Wh at the 128-1 point. Figure 5.2, on the other hand, shows the run with the maximum power consumption for each scenario. The maximum values are used to calculate the size of solar panel required to support even the worst-case scenario. The results from the maximum value runs coincide with the previous for all protocols except for CoAP. Data shows that CoAP experience a huge jump in power consumption in one of the 128-1 point runs. There are two likely causes for this behavior. The first possibility is that it could be the results of a large number of the 128 test clients posting large questions that exceed the payload limit. Another possible cause, however, is an application restart. The CoAP library for NodeJS is prone to crashing in severe conditions; in which case the edge device will automatically restart the NodeJS application, which may cause a peak in current draw. In order to put these values into real-life perspective, one can calculate the size of solar panel required to power each, in order to recognize if the Watt-hour value difference amounts to a physical cost difference. As mentioned in section 4.4, the worst-case scenario for peak hours in a day in the UAE is 4.5 hours, which happens during January. Panel size based on peak hours is then calculated in Table 5.1.

Table 5.1: Solar panel and battery sizing

	Baseline	AMQP	CoAP	MQTT	XMPP
Max power consumption (Wh)	40.22	43.56	44.40	42.71	44.13
Solar panel size (W)	8.94	9.68	9.87	9.49	9.81
Market price (solar panel + power bank [129]) ($)	$10	$10	$10	$10	$10

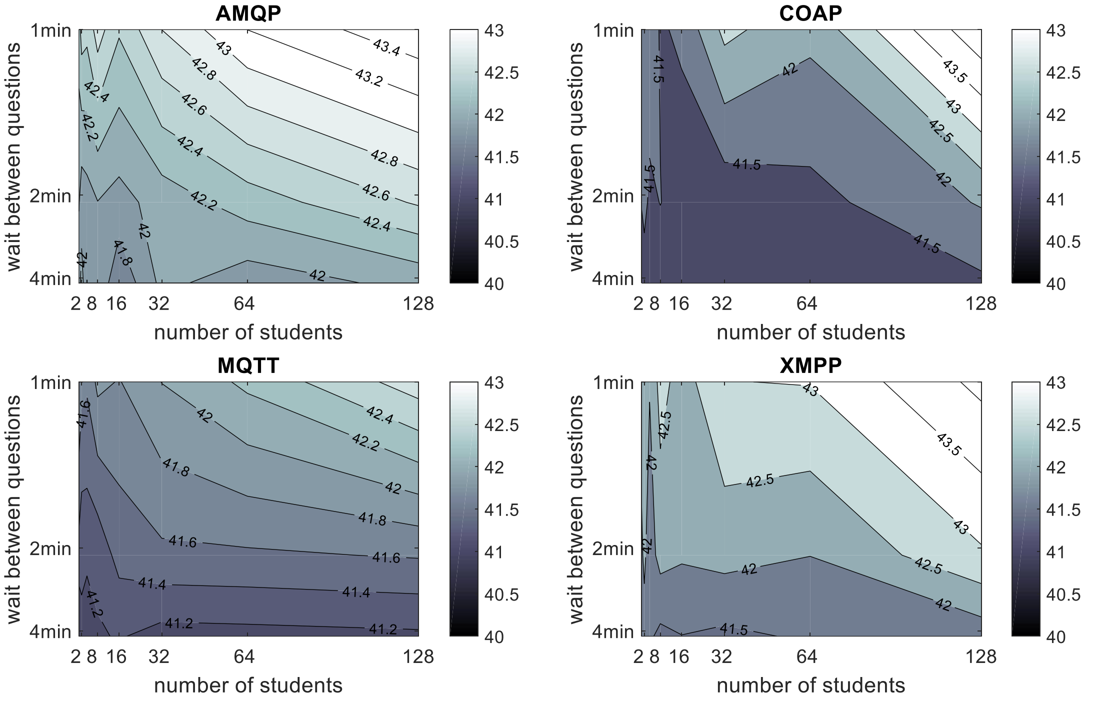


Figure 5.1: Power consumed by each protocol in Watt-hour

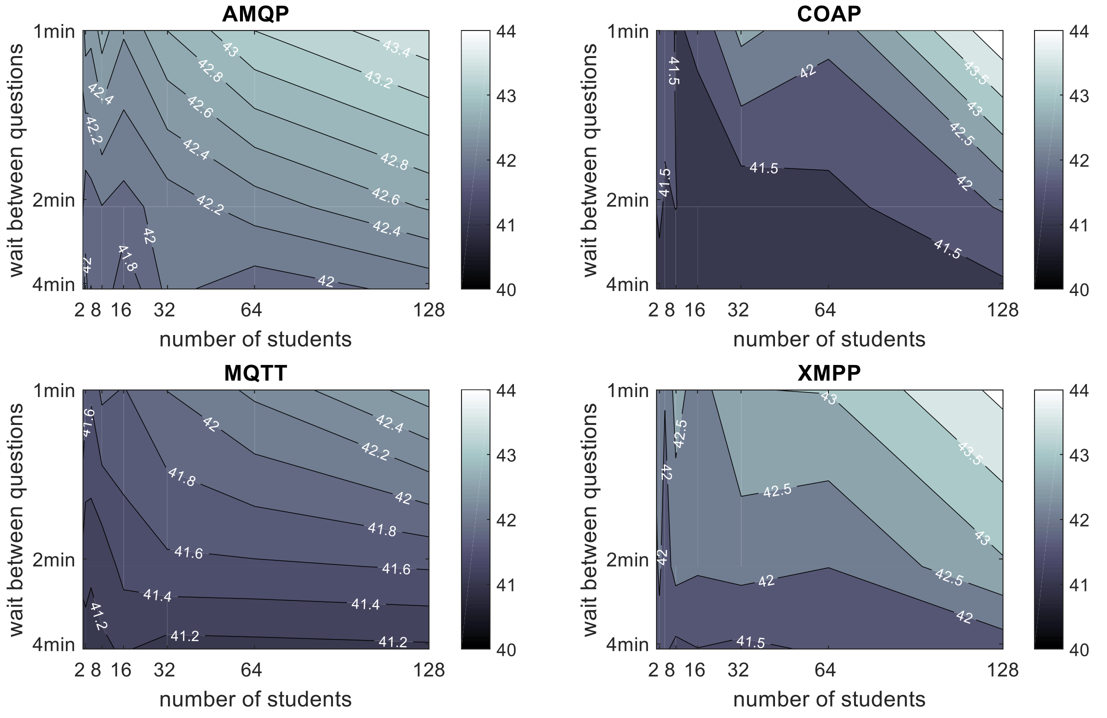


Figure 5.2: Maximum power consumed by each protocol in Watt-hour

Based on raw numbers, MQTT, on average, seems to be the most efficient protocol in terms of power consumption overall, while CoAP is more efficient at lower values. However, this changes once numbers are put into a real-world context. While MQTT consumes an average of around 1Wh less than the other protocols, the solar panel size required to power each of the protocols is the same. Therefore, in a real-life setting, the cost of powering the edge device is the same, and the difference in power consumption is not a strong enough justification.

5.1.2. CPU utilization. The average and maximum percentages of CPU consumed by user process, i.e. NodeJS application server running on the Pod, is shown in Figure 5.3 and Figure 5.4, respectively. The reason we are interested in both the average and the maximum value is because while the maximum CPU% utilization might cause an issue in busy environments, the difference in average value takes into account the CPU% required to keep the edge device's connection to the communication server alive at all times, even when idle. As expected, more processing power is required when increasing the number of students and the frequency of posting questions.

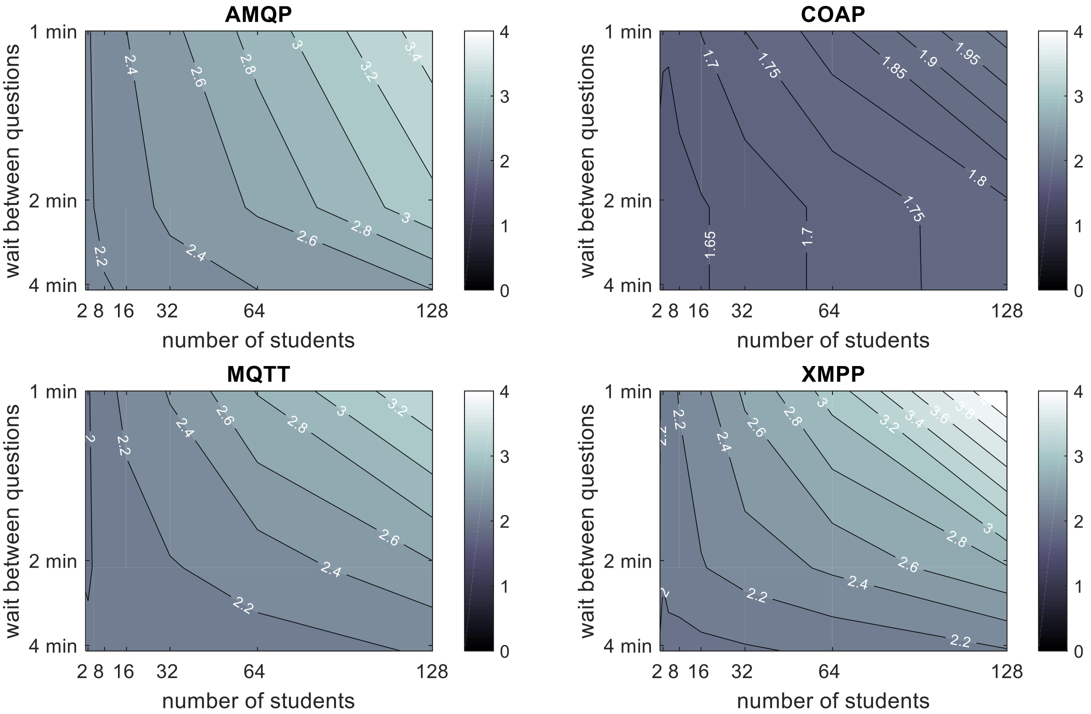


Figure 5.3: Average CPU utilization (%)

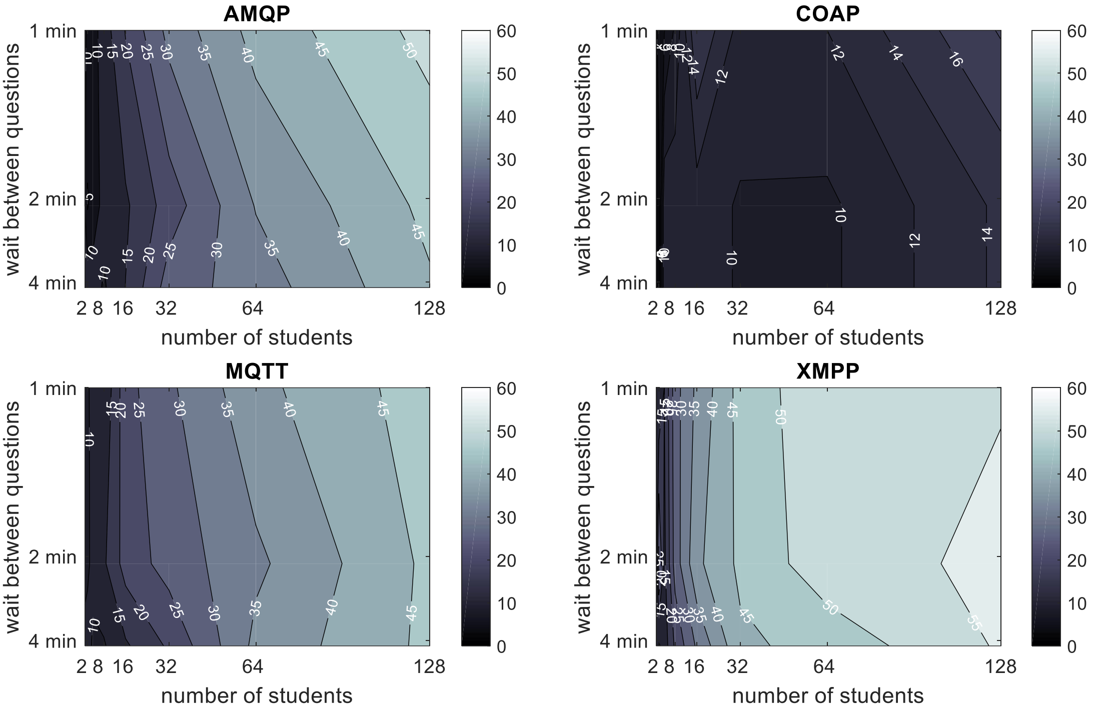


Figure 5.4: Maximum CPU utilization

Among the four protocols, CoAP consumes the least percentage of CPU on average. This is most likely due to the way CoAP is implemented, since it was designed especially to be used on constrained devices, and so requires less resources. The same behavior was found in [130]. It is also important here to remember than CoAP implements an observer-oriented architecture, while the other protocols are based on publish-subscribe. While both architectures can be used interchangeably to perform the messaging tasks required in this type of systems, the difference between them appears in terms of resources. Publish-subscribe protocols, like the three considered here, often implement a connection keep-alive mechanism by default, where even in idle state the client sends small packets to the server to ensure it is still connected. This is an important feature in environments where clients may suddenly disconnect, or simply no longer need updates on a certain topic. This allows the server to only publish packets to connected clients, and avoid flooding the network with unnecessary traffic. In CoAP, on the other hand, the implementation focuses on minimizing the load on the clients rather than the server; and so clients are able to save more resources by excluding the keep-alive mechanism, for the cost of the server possibly wasting resources on send messages to disconnected clients. This is acceptable because while the client has limited resources, the server station is usually not. The maximum value expresses the worst case scenario. While the CPU% is not what is consumed by the application at all times, it is important to consider it as a CPU overload could cause the edge device to crash, and disrupt the system. The highest values are exhibited by XMPP, which is mostly likely due to the data format, as opposed to other protocols which simply send the question message as is, the question in XMPP is embedded into a stanza. This requires more processing, and is most likely the reason behind the high CPU% utilization. MQTT and AMQP demonstrate a similar CPU utilization behavior, while CoAP peaks at less than 20%; which is around the third of XMPP's maximum utilization of around 60%. To further clarify, Figure 5.5 shows the CPU% utilization over a 5-min-long run for each protocol; with 128 student clients posting a question per minute, each.

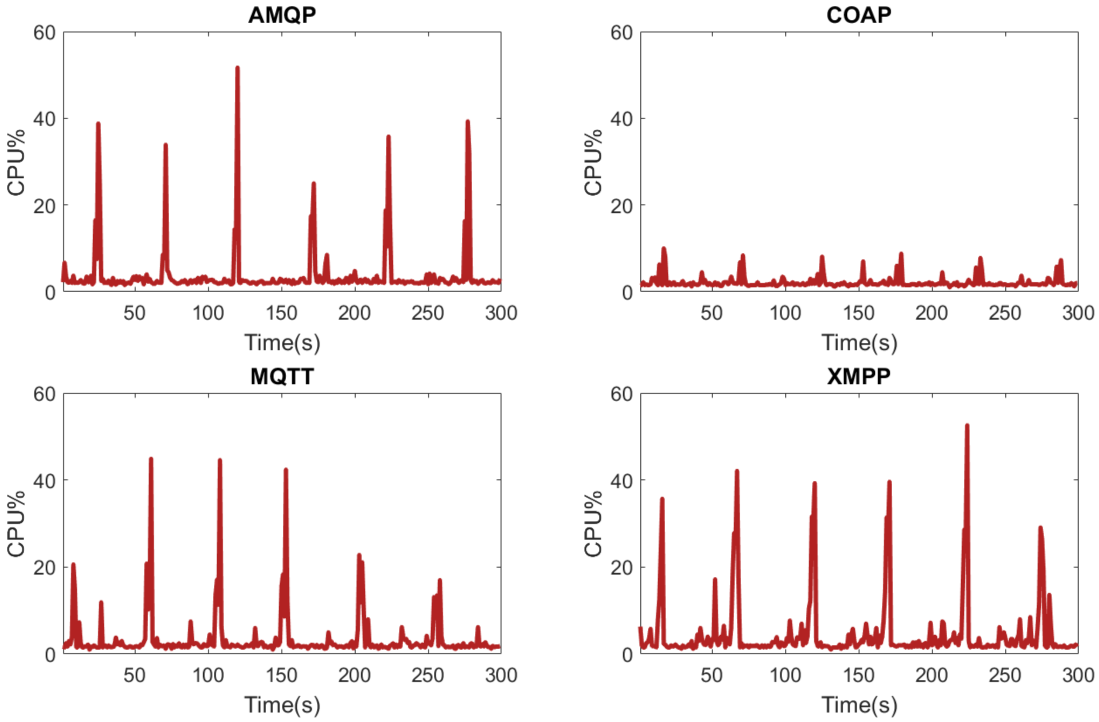


Figure 5.5: CPU% over 5 minutes for each protocol. students=128, frequency=1Q/min

Translated into a real-world context, given the peak CPU utilization at any point is around 60% of the available processing power, the edge device can be replaced with a cheaper Linux based system with lower processing power. Even more so if we consider CoAP only uses 20% of available CPU power, at max. However, the reduction in processing power should not go below the peak CPU utilization for a given protocol, if one is to avoid saturating the CPU and causing a drop in system performance. While this at first may indicate that CoAP is outperforming the other protocols in terms of resource saving, it is vital to remember that larger questions in CoAP are replaced with maximum allowable payload size, due to its payload limitations. In other terms, for a portion of the simulation time, CoAP is not doing as much work as the other protocols.

5.1.3. Page faults. There are numerous memory-related metrics that can be used to assess the usage of a Linux system. The aim from this experiment is to assess the performance of the RAM under each communication protocol, using page faults. In Linux systems, page faults happen when the system attempts to retrieve data from the RAM, is unable to find, which then forces it to look for the data in the disk. This takes place when the RAM is full and the system needs to accommodate new data, at which point older entries are moved to disk. IoT communication protocols are meant to support constrained device, which often have limited memory storage; especially in terms of RAM. It is therefore important that we investigate the performance of RAM and number of page faults as the system is increasingly being stressed out, under each protocol. The number of page faults in response to the number of student clients and question posting frequency is shown in Figure 5.6. CoAP exhibits the lowest number of page faults per second compared to the other three protocols, which translates into the best performance among the four.

There are two possible factors that may attributed to this behavior. The first is that CoAP is designed to occupy as little RAM as possible, in order to maintain its light weight. Naturally, the smaller portion occupied in RAM, the lower the number of pages faults, as explained earlier. The other reason could be related to the size of questions carried by CoAP, compared to the questions carried by other protocols. The latter will be explained in the next chapter. Among the other three protocols, MQTT –once again-outperforms XMPP and AMQP with an average page fault rate of around 250 at the worst point (128-1). AMQP is the third best, with an average of around 260 at 128-1, while XMPP exceeds it to score 270 page faults per second. Similarly to the CPU% discussion, a lower page faults implies less stress on the RAM, which in some cases allows for giving up the edge device for a smaller, less powerful edge device.

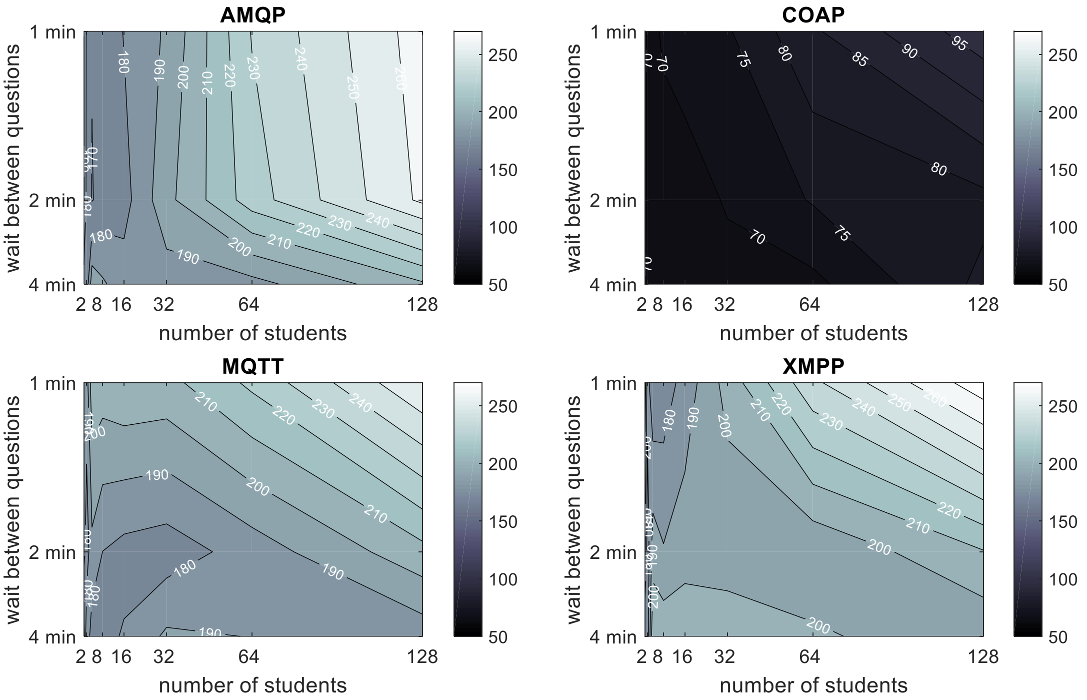


Figure 5.6: mean page faults per second

5.1.4. Summary. The results from the three resource-related experiments suggest that the protocols performance quality is in the following order: CoAP is best, followed by MQTT, then AMQP, and lastly XMPP. Based on numbers alone, CoAP may seem like the best option out of the three. However, CoAP suffers from a major functionality disadvantage, which may also be partially responsible for the performance difference between it and MQTT. CoAP has a payload size limit that prevents it from supporting the full extent of typical question sets. This is a vital factor to keep in mind when opting out for CoAP as means to employ a smaller edge device, as the cost occurs as usage limitation. In this case, MQTT and AMQP are quite valid alternatives, as they can function in a real setting, while giving leeway in terms of edge device capabilities; compared to XMPP.

Table 5.2 Summary of Experiment I results

	AMQP	CoAP	MQTT	XMPP
Power consumption	medium	medium	low	High
CPU%	medium	low	medium	High
Faults/second	high	low	high	high

5.2. Experiment II

This section presents an analysis of the relative performance of each protocol under imperfect network conditions. The performances will be assessed based on two variables which are the change in time required for a message to travel from a publisher to a subscriber under imperfect network conditions, and the network throughput in bits per second (bps).

5.2.1. Throughput. The throughput is defined as the number of bits that each protocol can carry, per second. Comparing pure UDP with pure TCP, UDP is theoretically usually faster as it does not go through reliability steps such as handshakes and packet acknowledgements. While TCP has to ensure the destination node is online, has successfully received the previous packets, and is capable of receiving more. On the other hand, UDP lacks the congestion control features that TCP has. While TCP uses buffers and attempts to send packets in the most efficient way, UDP simply pumps streams of data, which can possibly cause congestions and decrease the throughput of the link, as well as cause packet loss. While AMQP, MQTT, and XMPP follow the TCP specifications for the most part, CoAP is UDP-based but implements TCP-like features, which are expected to affect its performance. For this experiment, each protocol was allowed to send messages over a short period of time, over a relatively ideal network. It should be noted that Clumsy was actively filtering, and so the channel bandwidth was around 20 Mbps. The mean throughput and standard deviation for each protocol are shown in Table 5.3, while a visual representation is shown in Figure 5.7. The measured throughput values show that the highest throughput belongs to XMPP, at around 3.8 kbps. AMQP comes second with a little under 3.8 kbps, while MQTT is slightly slower at 3.4 kbps. The most interesting part, however, is CoAP. While the protocol is built on UDP, the results indicated that the throughput is much lower than the three other protocols, not exceeding 1.4 kbps at best. This is most likely the result of the additional TCP-like features on a UDP protocol. The same behavior was noticed in [131]. However, in this experiment, a third factor might be at play. CoAP was designed for constrained devices, and its size limitations compared to the typical sizes of questions causes it to be transporting at maximum, or close to maximum payload for a long period of time. The other protocols, on the other hand, are well-below their payload limit size. XMPP, for example, is most ideal for 8 kb data, which is way above the maximum size for questions. The throughput was also measured during the other experiments. The results for the throughput values at different network disturvance conditions can be found in Appendix C.

Table 5.3: Throughput mean and standard deviation values for each protocol

	Mean (bps)	Stdv. (bps)	Max (bps)	Min (bps)
AMQP	3781.504	713.8728	5465.393	2682.032
CoAP	1386.872	86.33655	1532.701	1221.923
MQTT	3436.573	719.4113	5133.64	2348.204
XMPP	3808.966	499.0668	4550.941	2975.024

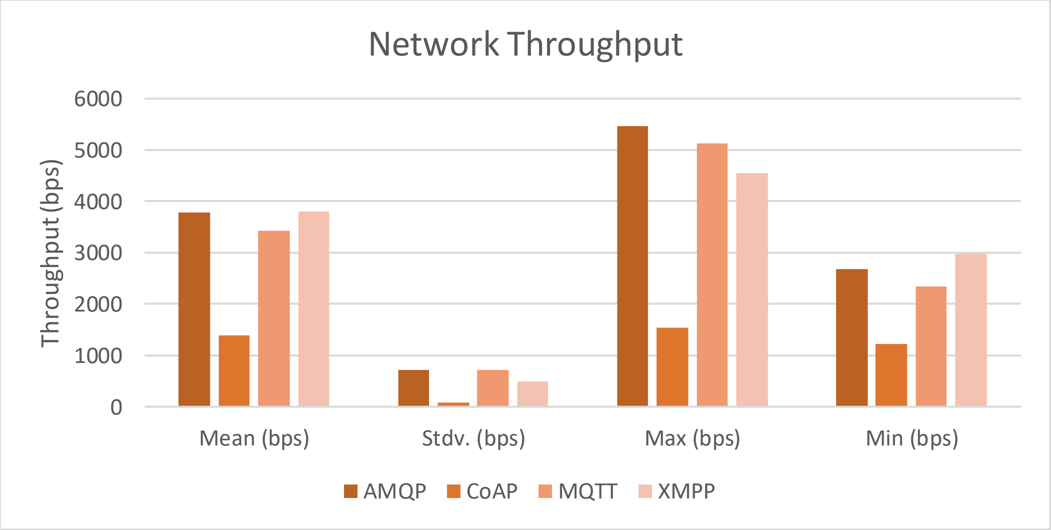


Figure 5.7: Network throughput for the four protocols

5.2.2. Latency response to duplicates. This set of experiments investigates the effect of duplicated packets arriving at the pod on the user experience. Duplicates often occur in noisy networks where congestions cause packets to either arrive late, not be properly acknowledged, or be completely dropped. In TCP protocols like MQTT, AMQP, and XMPP, either the source host or an intermediate node retransmits packets that are assumed to have been lost until an acknowledgement is received; as means of ensuring reliability. This, however, can cause several duplicates of the same packet to arrive at the receiving host. Sequence numbers are the most common technique used to ensure the host is able to distinguish duplicates from new data. Depending on the size of duplicate traffic however, this can increase the load on the receiving host. UDP protocols do not natively support reliable communication, and so while the sending host will not intentionally duplicate packets, duplicates generated by intermediate nodes will not be detected by the receiving host, and will therefore be processed as new data. Unlike typical UDP, however, CoAP implements a lightweight, TCP-like layer that handles packet sequencing and retransmission in order to add reliability to the system. The aim of this section is to compare the latency cost of high duplicates percentage across the four protocols. The mean latency values and standard deviation are shown in Figure 5.8 and Figure 5.9. The latency response for all protocols remains relatively low, even at 50% duplicates. Among the four, however, MQTT experiences the lowest latency, averaging at around third of XMPP latency, which is the highest. CoAP and AMQP, experience similar values. Looking at the standard deviation, CoAP, MQTT, and AMQP demonstrate a consistent behavior, while XMPP experiences high fluctuation. Overall, all four protocols demonstrate a constant latency behavior that is unaffected by duplicates percentage. The duplicates could have influenced the latency in two ways: by over-writing the “arrived” time causing the latency difference to increase, or by causing a congestion as more duplicate traffic travels through the channels. Since all protocols discard duplicates at the transport layer, and the bandwidth is evidently large enough to drown out the effect of excess traffic, the latency was unaffected.

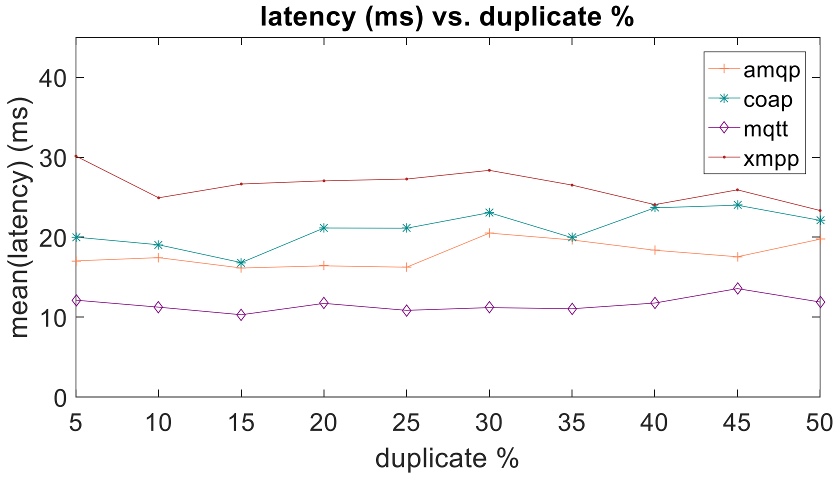


Figure 5.8: Mean latency under duplicated packets

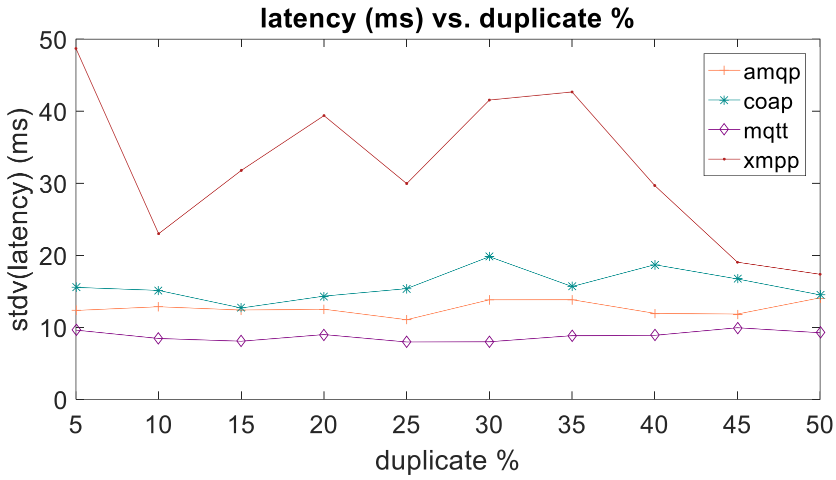


Figure 5.9: Standard deviation of latency values under duplicated packets

5.2.3. Latency response to lag. The goal was to examine the effect of increasing network lag on the increase in latency demonstrated by each protocol. In wireless networks, lag can occur due to various reasons such as bad weather conditions, physical obstacles, and badly-configured or legacy intermediate network devices. The values of network lag were varied from 100 ms all the way to 1000 ms, which is a relatively high value for networks. However, as noted in the previous chapter, the filtering process done by Clumsy adds lag to the network even when no explicit values have been entered. Therefore, the average effective bandwidth for each lag value produced by clumsy is shown in Table 5.4.

Table 5.4 Effective bandwidth after adding network lag via Clumsy

Lag	100	200	300	400	500	600	700	800	900	1000
Effective bandwidth (Kbits/s)	2,420	1,930	1,280	903	814	550	446	348	281	225

Figure 5.11 and Figure 5.11 shows the means and standard deviations of latency values at each network lag value. As expected, the latency increases with the increase in network lag. In the case of CoAP, this increase is perfectly linear with the latency amounting to double the network lag value. The latency is expected to be double because Clumsy adds the defined lag value to the link in both directions (client to broker, and broker to client), and so a network lag of 100 ms on each direction results of a latency of 200 ms; implying that no extra lag is introduced by the protocols itself. This, however, is not true for the other protocols. Two other distinct behaviors show in

the remaining protocols. XMPP starts with a higher latency, at around 1000 ms. However, it is less affected by the change in lag, and does not exceed 2000 ms, which is the same latency experienced by CoAP at the maximum lag value. AMQP and MQTT, on the other hand, start off with a small latency of around 100 ms latency at 100 lag. As the lag value increases, additional latency is added to the system until it reaches 3000 ms of latency at 1000 lag. In other words, both AMQP and MQTT experience latency equal to 1.5 times the added lag value in each direction.

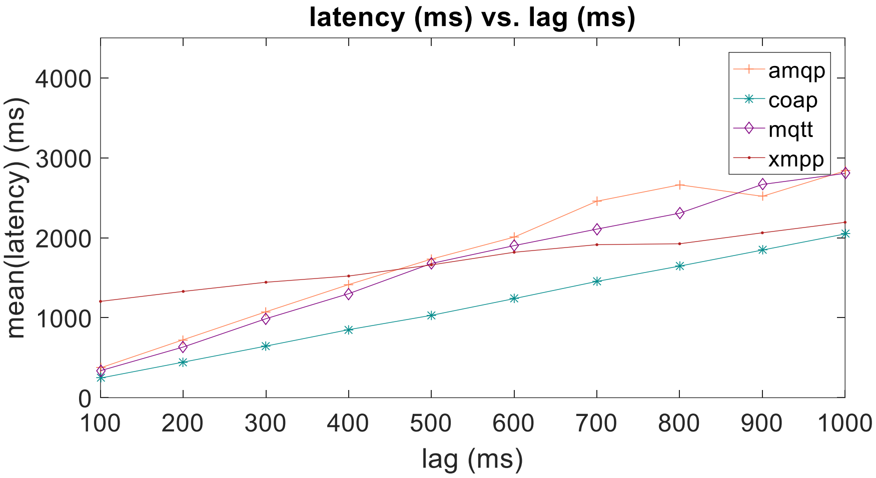


Figure 5.10: Mean latency (ms) demonstrated by each of the four pub/sub protocols vs. network lag (ms).

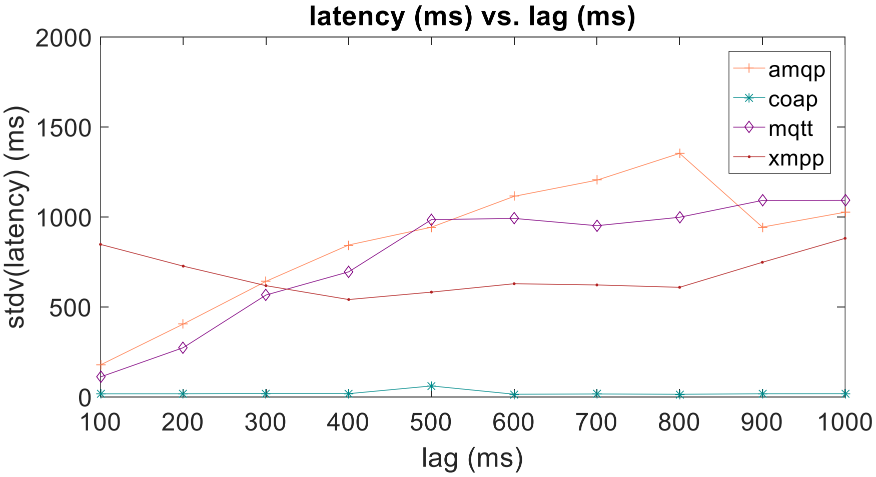


Figure 5.11: The standard deviation of the latency values vs. network lag (ms)

5.2.4. Latency response to out of order. This set of experiments investigates the performance of the protocols in networks where the packets may arrive out of order. This can occur in networks where different packet segments take multiple paths that differ in speed, resulting in some segments arriving at their destination before others.

This can also occur due to badly configured intermediate network devices, such as routers and access points that do not forward packets in a First-In/First-Out order [132], [133]. In the TCP/IP network stack, messages that arrive out of order cannot be delivered to the application layer until they have either been reassembled or retransmitted by at the TCP layer; using sequence numbers. This applies to MQTT, AMQP, and XMPP. UDP, on the other hand, does support sequence numbers, and out of order packets are therefore dropped. In order to avoid dropping packets, CoAP implements a lightweight reliability layer which handles retransmission [90]. By increasing the percentage of packets arriving out of order, the aim is to compare the cost of reassembling the packets observed by each protocol, in terms of added latency. The mean and standard deviation values are shown in Figure 5.12 and Figure 5.13. In terms of handling packets arriving out of order, MQTT and CoAP are able to keep the latency at a lower value compared to AMQP and XMPP. However, while MQTT, XMPP, and CoAP demonstrate a constant latency value regardless of the percentage of packets arriving out of order, the latency in AMQP shows a slight dependence on the percentage. This implies that among the four protocols, AMQP is the lease efficient in reassembling out of order packets, at it adds latency to the system.

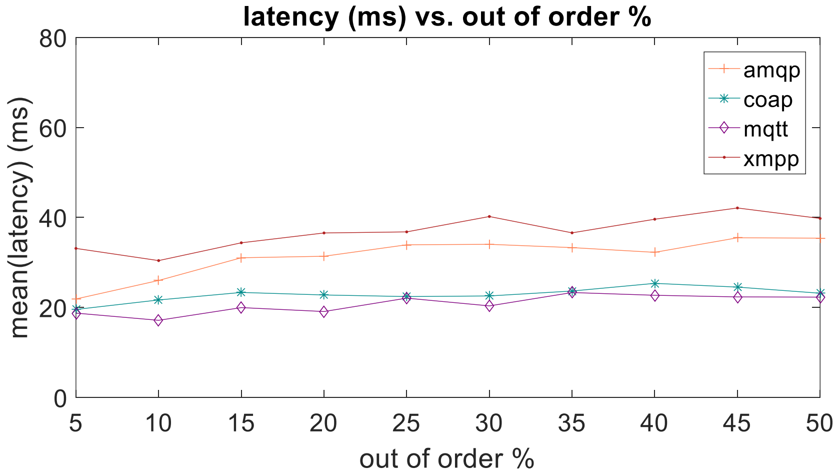


Figure 5.12: Mean latency (ms) vs. out of order %

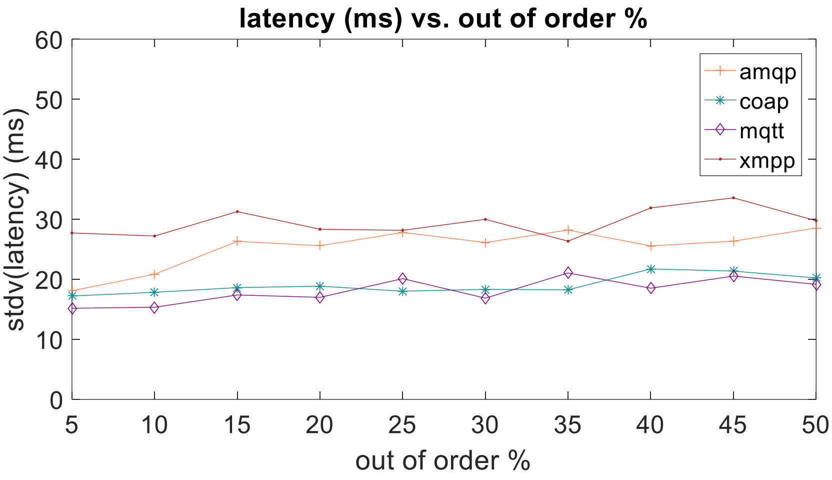


Figure 5.13: Standard deviation of latency (ms) vs. out of order %

5.2.5. Latency response to throttle. Changing the throttling percentage mimics a network bottleneck, where packets are held at a certain point for a period of time before being released, causing them to arrive in bulks. This can happen due to poorly-configured or ill-suited network devices. This can also happen if the pod goes offline for a period of time; as once it is back on, all of the questions that were published during the off period will be sent to the pod in bulks. The way throttling is being simulation packets in clumsy is by holding the packets for a certain period of time, and then releasing them at once. Not all packets are held back however. The percentage indicates the chance of each packet being throttled or not. If the packet is throttled, it is held back for 30 ms by default. The higher the chance percentage, the more latency we expect to see as the simulated bottleneck becomes tighter. The results are show in Figure 5.14 and Figure 5.15. All of the four protocols demonstrate a similar behavior in terms of latency increase. While messages experienced higher end-to-end latency as the throttling chance increases, the rate at which the latency increase is relatively small. However, once the system is stressed to a throttling percentage of 90%, while MQTT, AMQP, and XMPP remain consistent, CoAP latency values shoot up to an average of 2 seconds, compared to less than 150 ms for the other three. The same jump is exhibited in terms of standard deviation; meaning latency values are no longer predictable. As for the other protocols, MQTT is the most consistence. Even though it starts off with a latency that is 10 ms higher than AMQP and XMPP, the slope of the mean values is flat compared to the other protocols, and the standard deviation is almost fixed throughout different throttling chance percentages; i.e. the latency is highly predictable.

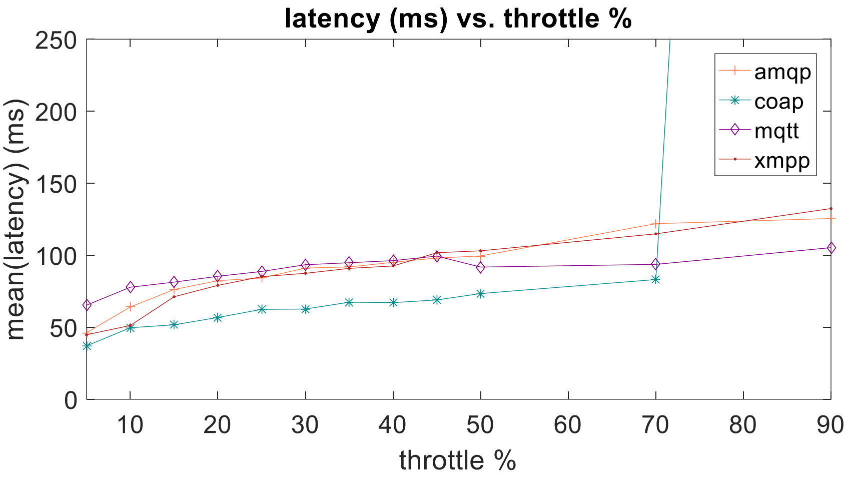


Figure 5.14: Mean latency response for throttling %

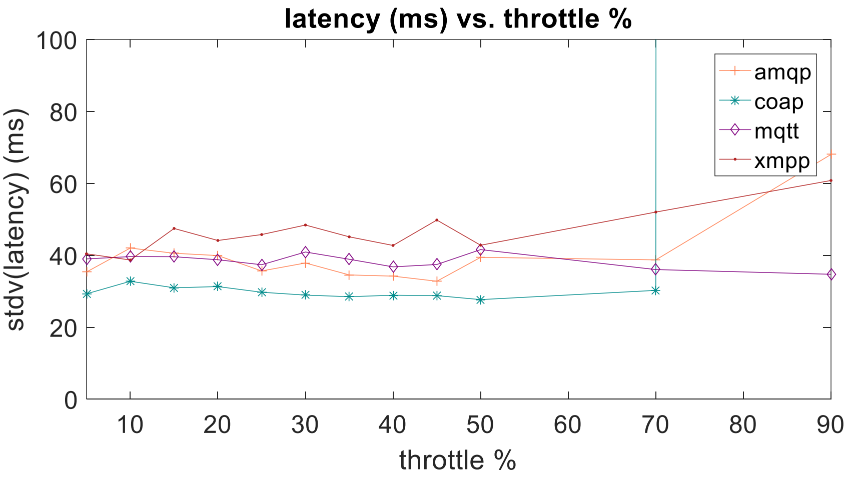


Figure 5.15: Standard deviation of latency response for throttling %

5.2.6. Summary. Table 5.5 shows the worst latency for each protocol, in response to each network issue. When comparing the protocols' performance under imperfect network conditions, CoAP demonstrated the best performance in terms of end-to-end latency. This is most likely due to UDP being the underlying transport layer, as it eliminates most mechanisms usually employed in TCP to ensure reliability such as handshakes and connection keep-alive. Instead, CoAP implements a minimal retransmission layer that can handle duplicated and out-of-order packets. The second best-performing protocol is MQTT, as it often demonstrated lower latency values than the other two; as low as CoAP in case of out of order packets. However, it is crucial to point out a major disadvantage of CoAP which in this case may have played a part in it out-performing the other protocols. The first is the very low payload size limits. While it was sufficient to carry a question of an average size or less, a big chunk of the available questions were replaced with smaller packets, which may have contributed to its high performance. Taking this into consideration, we can conclude that MQTT is in fact the best performing out of the four protocols. More details about the latency distributions are available in 0 and 8.

Table 5.5: Latency (ms) at the worst value of each network issue, per protocol

	Duplicates (50%)	Lag (1000ms)	Out of Order (50%)	Throttle (90%)
AMQP	19.76	2836.76	35.41	125.49
CoAP	22.11	2047.96	23.17	2135.20
MQTT	11.90	2805.22	22.31	105.27
XMPP	23.34	2193.44	39.78	132.41

When put in real-life situation perspective, however, the difference in performance is not of huge significant. While there is often a clear difference between the protocols, the latency values in response for the variable is in the range of tens of milliseconds, with the exception of lag. In a real life setting when students are posting questions or attempts, a latency of a litter under 3 seconds for 64 students publishing a question every minute can pass as acceptable latency.

5.3. Chapter Summary

This chapter investigates the effect of different scenarios and environment conditions on the protocols' performance. The first section investigates the difference then number of students using a pod at the same time, and the frequency at which they post questions, on the resource consumption of the Raspberry Pi when paired with each protocol. The experiments focused mainly on the power consumption, CPU utilization, and page fault rates. While the power consumption is required to calculate the size of solar panel needed to power the system, low CPU and memory utilization can that and edge device with less processing specifications can function as well as the edge device. The second section investigated the effect of network abnormalities such as high lag, throttling, packet duplicates, and out of order packets. The metric used to assess the performance was end-to-end delay, as it can directly affect the user experience. CoAP performed consistently better than three protocols, sometimes followed very closely by MQTT. This is most likely due to several factors, including the implementation being focused on constrained device support, UDP being the underlying transport layer, and the size limitation put on the question data that can be sent through CoAP specifically. Furthermore, the previous section focused on the technical implementation of each, although it touched upon qualitative aspects such as size limitation compared to real questions, and question queuing. This is not enough to assess how well the protocols will work for the particular defined class of systems. In order to relate the discussion to the specific context, the next chapter investigates issues that are more specific to the defined class of applications.

Chapter 6. Qualitative Evaluation

This chapter presents a qualitative evaluation of the four pub/sub protocols under study. The qualitative analysis assesses how suitable each protocol is for ubiquitous learning platforms based on their specifications and implementations. The first section considers the maximum payload size allowed in each protocol and compares them to the range of questions sizes available based on the two data sets discussed earlier. The second section, on the other hand, considers security needs for ubiquitous learning platforms, and examines security features offered by each protocol, as well as their costs.

6.1. Payload Size and Overhead

An important aspect to consider is the protocol's ability to support typical question sizes without the need for external segmentation by the developer. The majority of the protocols studied here either offer a high payload size limit, like MQTT, or are designed so that the payload size limit is set by and depends on the server or broker. This, however, is not true for CoAP. As mentioned in Table 3.5, the maximum payload size offered by CoAP is 1280 Bytes at best. To put this limit into perspective, simply refer to Table 4.2. Out of the 2890 questions available, 505 questions were too large for COAP payload limit. This amounts to about 17.5% of the typical questions, which cannot be supported by the protocol. A possible solution is to segment the packets at the front end, i.e. modify the pod application server so that if the question is too large, the app will automatically divide it down into adequately-size segments before sending them in multiple frames. The receiving ends must also be modified in to enable them to track segments and reconstruct large packets, which requires the use of segment IDs and other segmentation mechanisms. On the other hand, the other protocols, MQTT, AMQP, XMPP, and even HTTP at the edge side, do not face a size issue as the size limits, or lack of, are capable of supporting the maximum question sizes available.

The other size-related metric is overhead. Fixed overhead size dictates the percentage of the packet being transported that is irrelevant to the actual contents of the message. In constrained environments, overhead should be minimized as much as possible in order to save bandwidth and increase the efficiency of communication. This is why most IoT protocols aim to minimize overhead by including only the required information for the message to be delivered to its correct destination. In MQTT and AMQP, for example, information typically found in a classic communication protocol such as Content Type are spared by not specifying a message type at all, leaving the responsibility of decoding and understanding messages to the client side. In order to observe the actual difference the header size makes, the same message were published using each of the four protocols. The messages including the same questions, question IDs, and timestamps, and varied only in terms of extra information required by each protocol. The questions were chosen of sizes that relate to the mean, median, and first and second quartiles of each question set, as to represent the majority of the questions. The published messages from each protocol were captured through WireShark, to compare the length on wire, i.e. size of the frame at the datalink layer, with the original question size. The additional bytes added by each protocol are then divided over the total frame length in order to find the overhead percentage. The results for overhead experienced by questions at the mean, median, first quartile, and third quartile are shown in Table 6.1.

Table 6.1: Overhead % as measured by message size on wire

Protocol	Mean	%Overhead for Mean	Q1	%Overhead for Q1	Median	%Overhead for Median	Q3	%Overhead for Q3
Original message (Bytes)	1221		296		486		1005	
MQTT (Bytes)	1294	5.64%	369	19.78%	559	13.06%	1078	6.77%
CoAP (Bytes)	1301	6.15%	376	21.28%	566	14.13%	1085	7.37%
AMQP (Bytes)	1340	8.88%	415	28.67%	605	19.67%	1123	10.51%
XMPP (Bytes)	1442	15.33%	544	45.59%	730	33.42%	1233	18.49%

6.2. Security

Security is an issue rarely discussed in ubiquitous learning systems. This is most likely due to researchers often dedicating more efforts into the development and deployment of these systems, rather than securing them [134], which is common in novelty applications. IoT security in context ubiquitous learning applications focuses on two concepts: user authentication and authorization, and data integrity and confidentiality [135]. Authentication is concerned with verifying the identities of users. Once the user’s identity has been verified, they can be given authorization to access various resources and service in the system based on their identities. This is usually done using authentication techniques such as passwords, fingerprints, voice detection, and others. Integrity, on the other hand, is concerned with assuring that messages travel from end to end arrive at their destination un-tampered. Different encrypted techniques can be used to protect network traffic and prevent tampering. Encryption is also used to provide confidentiality; i.e. protecting the contents of a message from being accessed by an unauthorized user.

In relation to ubiquitous learning systems, network security plays a great part in maintaining a fair assessment and grading system [136]. Students earn grades by participating in posing and attempting assessments. It is therefore critical that students' identities are verified before they are able to participate and earn grades, and contributions such as questions and attempts. Equally critical is securing data flowing through the system such as feedback and results and protecting them from unauthorized access and tampering. Default and optional security features in each protocol are shown in Table 6.2.

Table 6.2: Optional security features of the various protocols

Protocol	Optional security feature	Purpose	Comments
AMQP	TLS[137]	Data security and authentication	may be an issue with low resource devices
	SASL[138]	Data security and authentication	may be an issue with low resource devices
	X509-certification[139]	Authentication	
CoAP	DTLS[140]	Data security and authentication	
	IPSec[141]	Data security and authentication	
	CoAP security options[142]	Data security and authentication	
MQTT	TLS	Data security and authentication	may be an issue with low resource devices
	TLS-PSK[143]	Data security and authentication	not widely supported
	X509-certification	Authentication	only available for MQTT over WebSockets
XMPP	TLS	Data security and authentication	may be an issue with low resource devices
	SASL	Data security and authentication	may be an issue with low resource devices

The default option for all protocols is to not use a security layer, although all offer optional layers. Transport Layer Security (TLS) is a common security layer among all protocols, including CoAP, which supports a lighter version defined as Datagram Transport Layer Security. In MQTT, however, TLS is found to add an overhead. This is true because unlike AMQP and XMPP, MQTT does not natively integrate TLS into its original implementation. It is important to note, however, that MQTT is also specifically designed for constrained devices, and so the relative significance of the overhead depends on the type of edge device. In other words, TLS in MQTT is costly in terms of computation and overhead for constrained devices where the number of bytes processed makes a huge difference. In non-constrained devices, however, TLS causes almost the same overhead to MQTT as it does to AMQP and XMPP. MQTT also supports Pre-Shared Key over TLS encryption, which is much lighter that TLS, but not as common. MQTT also supports server certificates, which help clients authenticate the identity of the broker in order to avoid server spoofing. While AMQP and XMPP do not support the latter two technologies, they do support Simple Authentication and Security Layer as an alternative to TLS. As for CoAP, supported third-party security layers are DTLS and IPSec. In order to decrease implementation and communication overhead, however, the protocol supports security options at the CoAP-layers that provide the same authentication and data security options as DTLS and IPSec, at a lower resource cost.

6.3. Question Queuing

The issue of question queuing is important in two situations: when a new pod is added to the system, and when a pod is cut off the network for a certain period of time. The introduction of a new pod to the system should be as seamless as possible in the aspect that previously-posted questions that will apply to a pod in that location and context should immediately be delivered to it. Pods can also go offline either because of technical issues, or because were to be moved to another location/context. Once the pod is reconnected, it should receive all of the previous contributions as well. As publish/subscribe protocols, AMQP, MQTT and XMPP support message queuing at the server/broker side, which means that republishing missed questions happens by default. With CoAP however, this is not true. The observe pattern in CoAP, while fully-functioning for the most part, is still at draft stage, therefore does not implement all of the basic functionalities of a publish/subscribe protocols, including queuing. This is one of the situations where the difference between an observe protocols and a pub/sub protocol is noticeable. Unlike pub/sub protocols, observe protocols are concerned with the current state of a certain resource, rather than the interactions of a certain client. This is why even in Ponte, only the last value published by a CoAP resource/client is preserved. Put into the context of our applications, this means that unless a custom server is developed for CoAP, only the last published question is delivered to a new/offline pod, which is simply impractical.

6.4. Topic Hierarchy

Messages in Publish/Subscribe protocols are routed based on topics. In learning systems, a topic can represent a grade level, subject, learning outcome, area, school, or classroom. Students subscribe to one or more of each type of topic depending on their enrollment. It is possible to implement different types of topic independently, i.e. create a topic for each grade level, subject, etc. However, these topics usually belong into a hierarchy, similar to the one shown in Figure 6.1.

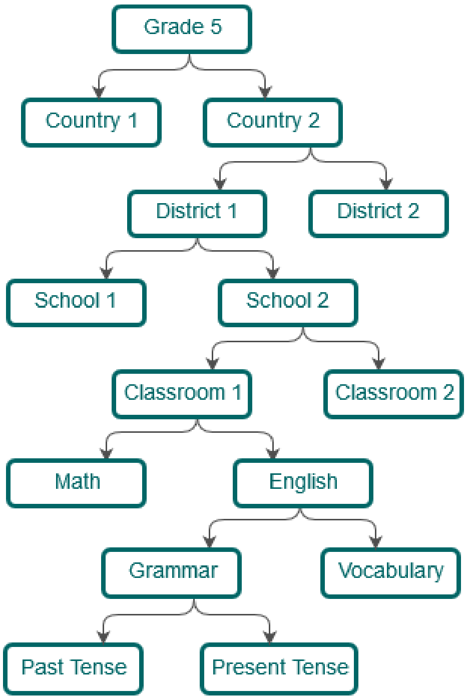


Figure 6.1: Simple topic hierarchy

The one presented is by no means a definite guide of how topic hierarchies should look like, but rather a simplified example. A hierarchy for a real-life, global-scale hierarchy will consist of thousands of topics, and can differ in how the topics are connected. The main idea, however, is the same. The communication protocol should be able to implement the hierarchy seamlessly. For example, messages posted to Grade 5 should be delivered to all students subscribed to all 5th grade students, anywhere in the world. School 2 students may post a general question targeted at their schoolmates. Alternatively, students in Country 2 may want to target a question at all students studying Past Tense in the country. Table 6.3 demonstrates how the topic field will look like in each case. This level of flexibly will reduce the number of broker topics. For example, a messaged posted to students in Country 2 studying Past Tense does not need to include addresses of all Past Tense topics under every classroom. This is usually referred to as the Wildcard topic [144], [145], where only certain fields of the topic address will be matched. Wildcards are possible, and supported in AMQP and MQTT because they offer multilayered topics, where topics can be arranged into a tree. In addition, AMQP offers the option of using regular expressions when matching topics. CoAP supports multilayered topics in the form of URIs, but not wildcards. XMPP, however, supports none, as each topic is represented as an independent node.

Table 6.3: A comparison of topic assignments for different scenarios

Protocol	Grade 5	Classroom 1 students	All students in Country 2 studying Past Tense
AMQP	Grade5.#	Grade5.Country2.District1.School2.#	Grade5.Country2.*.*.*.Past_Tense
CoAP	Grade5	Grade5/Country2/District1/ School2	Grade5/Country2/District1/Classroom1/English/Past_Tense Grade5/Country2/District2/Classroom3/English/Past_Tense Grade5/Country2/District2/Classroom4/English/Past_Tense Grade5/Country2/District3/Classroom7/English/Past_Tense …
MQTT	Grade5/#	Grade5/Country2/District1/ School2/#	Grade5/Country2/+/+/+/Past_Tense
XMPP	Grade5	Grade5_Country2_District1_ School2	Grade5_Country2_District1_Classroom1_English_Past_Tense Grade5_Country2_District2_Classroom3_English_Past_Tense Grade5_Country2_District2_Classroom4_English_Past_Tense Grade5_Country2_District3_Classroom7_English_Past_Tense …

6.5. Chapter Summary

This chapter investigated context-related qualities of each protocol. The comparison between the four protocols highlighted the shortcomings of CoAP and how they affect the user experience and the fairness of the system. The better choice, as concluded in this chapter is MQTT. MQTT offers a fixed, small overhead, supports very large questions, implements security features at a small overhead cost, and is natively well-suited for this type of applications. AMQP and XMPP, are not far behind MQTT, it simply comes down to overhead and topic hierarchy. A summary of the qualitative points is shown in Table 6.4. Each protocol is assigned a relative score out of 5, where 5/5 represents the best performance, while 0/5 represents the worst.

Table 6.4: Summary of qualitative evaluation

	Small Overhead	Secure	Reliability	Flexible Topics
AMQP	4/5	5/5	5/5	4/5
CoAP	2/5	5/5	0/5	2/5
MQTT	5/5	5/5	5/5	5/5
XMPP	3/5	5/5	5/5	0/5

Chapter 7. Conclusion, Limitations and Future Work

In this work, we proposed four implementations of an inquiry-based ubiquitous learning system, built using the same Internet of Things software and hardware technologies, combined with one of four IoT communication protocols. The protocols under study were XMPP, AMQP, MQTT, and CoAP. While XMPP, MQTT and AMQP employ TCP at the transport layer, CoAP uses UDP. A physical implementation was built for each option using a Raspberry Pi running NodeJS, CouchDB, and a client for each protocol. The implementations were evaluated for resources usage and network performance. Low CPU and RAM utilization implies that the edge device used for testing can be replaced with a cheaper device which is lower in memory and processing power, while maintaining the system's functionality. Power consumption, on the other hand, affects the size of the power source required to support the system. Of the four protocols, CoAP utilized the least amount of CPU and memory. The second best was MQTT, followed by AMQP and XMPP. In power consumption, however, MQTT proved to perform better than the other three protocols. However, when view from a real implementation perspective, the difference between the power consumption across the four protocols was negligible when viewed from a real life perspective. In terms of network performance, the metric used to assess the four protocols was end-to-end latency, as it relates directly to the user's experience. The system was developed for students and teachers to exchange assessments, and high latency will cause annoyance to the users and disrupt the learning experience. In terms of networks, MQTT demonstrated the lowest end-to-end latency in all cases, except for networks with high lag, where CoAP outperformed all three protocols. Additionally, the implementations were compared from qualitative aspects such as security, overhead, and ease of implementation. Overall, the protocols that performed the best in the experiments were MQTT and CoAP, followed closely by AMQP. XMPP on the other hand demonstrated relatively bad results in all experiments. However, it is important to keep in mind that CoAP was given an unfair advantage over the other protocols by excluding questions too that were too large for its payload size limit of 1280kB, which in itself is a major downside of the protocol. Furthermore, CoAP's performance comes at a reliability cost, as the protocol does not provide message queueing, which is crucial to ensure fairness in a system where pods are expected to go offline but not miss out on any assessments.

Neither MQTT nor AMQP suffers from the previous limitations, and because of how close their performance was, we can argue that either will be suitable for the class of IoT systems under study. While MQTT excels in quantitative experiments, AMQP offers security features, and both protocols can easily be customized for learning systems with hierarchies of topics, learning outcomes, and student levels.

There are some limitations that we recognize in this work, one of which is the choice of application server at the backend. NodeJS was chosen based on the literature review conducted of different application server technologies. However, it will be interesting to experiment with different servers to see if and how the system performance will be different. Another limitation is that the test data used was in a specific format and had the same characteristics. There several assessment formats currently being used by learning systems including known data formats such as JSON and XML, as well as proprietary formats. Different formats may affect the performance, especially with CoAP, where the message size was a major limitation. It will also be interesting to look into IoT-specific data formats such as the Open Data Format (O-DF) and the possibility of implementing assessments using them. Testing the system in a real-life environment with a busy network and real students will also produce more accurate evaluation results.

For future work, we suggest considering new network-level IoT protocols such as Thread and 6LoWPAN. These protocols were designed with the constraints of edge devices in mind, and are therefore likely to reduce resources consumption. It will also be interesting to test whether they are able to transport learning traffic efficiently, and how the user experience will be affected. Another aspect to look into is the link throughput. The one used in the evaluation was a 100MB Ethernet that’s been slowed down to 20MB to simulate the edge device’s max wireless communication speed, as well as the typical 4G bandwidth. We also plan to test the same system in constrained environments where the network can suffer from a combination of problems and limitations. Furthermore, we can address the limitations mentioned earlier by testing different application servers and databases, and converting different classic assessment formats to IoT data formats.

References

[1] G. J. Hwang, et al., “Criteria, Strategies and Research Issues of Context-Aware Ubiquitous Learning,” *J. Educ. Technol. Soc.*, vol. 11, no. 2, pp. 81–91, 2008.

[2] S. J. H. Yang, “Context Aware Ubiquitous Learning Environments for Peer-to-Peer Collaborative Learning,” *J. Educ. Technol. Soc.*, vol. 9, no. 1, pp. 188–201, 2006.

[3] M. G. C. Njoku, “Trend Analysis of Mobile and Ubiquitous Learning: 2014-2015,” *Int. J. Mob. Learn. Organ.*, vol. 10, no. 3, pp. 117–128, Jan. 2016.

[4] A. G. Gunay and I. Yakin, “The status of mobile and ubiquitous learning: A content review of the recent researches,” *Ubiquitous Learn. Int. J.*, vol. 6, no. 3, pp. 35–45, 2014.

[5] N. A. Saito, et al., “Supporting classroom activities with the BSUL environment,” in *Proceedings of IEEE International Conference on Wireless and Mobile Technologies in Education*, 2005.

[6] H. Ogata, et al., “Supporting Classroom Activities with the BSUL System,” *J. Educ. Technol. Soc.*, vol. 11, no. 1, pp. 1–16, 2008.

[7] T. Y. Liu and Y. L. Chu, “Using ubiquitous games in an English listening and speaking course: Impact on learning outcomes and motivation,” *Comput. Educ.*, vol. 55, no. 2, pp. 630–643, Sep. 2010.

[8] C. M. Chen and Y. L. Li, “Personalised context-aware ubiquitous learning system for supporting effective English vocabulary learning,” *Interact. Learn. Environ.*, vol. 18, no. 4, pp. 341–364, Dec. 2010.

[9] R. Reynolds, et al., “Web-based museum trails on PDAs for university-level design students: Design and evaluation,” *Comput. Educ.*, vol. 55, no. 3, pp. 994–1003, Nov. 2010.

[10] S. Akkerman, et al., “Storification in History education: A mobile game in and about medieval Amsterdam,” *Comput. Educ.*, vol. 52, no. 2, pp. 449–459, Feb. 2009.

[11] K. Y. Chin, et al., “Impact on Student Motivation by Using a QR-Based U-Learning Material Production System to Create Authentic Learning Experiences,” *IEEE Trans. Learn. Technol.*, vol. 8, no. 4, pp. 367–382, Oct. 2015.

[12] C. H. Chen and G. J. Hwang, “Improving Learning Achievements, Motivations and Flow with a Progressive Prompt-Based Mobile Gaming Approach,” in *Proceedings of 4th International Congress on Advanced Applied Informatics*, 2015, pp. 297–302.

[13] I. C. Hung, et al., “A context-aware video prompt approach to improving students’ in-field reflection levels,” *Comput. Educ.*, vol. 70, pp. 80–91, Jan. 2014.

[14] T. Y. Liu, et al., “Outdoor Natural Science Learning with an RFID-Supported Immersive Ubiquitous Learning Environment,” *J. Educ. Technol. Soc.*, vol. 12, no. 4, pp. 161–175, 2009.

[15] J. Gómez, et al., “Interaction System based on Internet of Things as Support for Education,” *Procedia Comput. Sci.*, vol. 21, pp. 132–139, 2013.

[16] I. A. Zualkernan, et al., “Prête-à-apprendre: Design and Implementation of a Wearable Assessment Tag Game for Children,” in *Proceedings of Fifth MIT International LINC Conference, Boston, Mass., USA*, 2010.

[17] I. Arroyo, et al., “Hoodies and barrels: Using a hide-and-seek ubiquitous game to teach mathematics,” in *Proceedings of International Conference on Advanced Learning Technologies*, 2011, pp. 295–299.

[18] P. Putjorn, et al., “Learning IoT Without the ‘I’- Educational Internet of Things in a Developing Context,” in *Proceedings of the 2015 Workshop on Do-it-yourself Networking: An Interdisciplinary Approach*, New York, USA, 2015, pp. 11–13.

[19] S. Shapsough, et al., “IoT technologies to enhance precision and response time of mobile-based educational assessments,” in *Proceedings of International Conference on Computational Science and Computational Intelligence*, Las Vegas, USA, 2016.

[20] O. C. Acosta, et al., “Content recommendation in an inquiry-based learning environment,” in *Proceedings of IEEE Frontiers in Education Conference*, 2014, pp. 1–6.

[21] J. L. Zafra-Gómez, et al., “Measuring the impact of inquiry-based learning on outcomes and student satisfaction,” *Assess. Eval. High. Educ.*, vol. 40, no. 8, pp. 1050–1069, 2015.

[22] S. Seol, et al., “Pocketschool interactive Learning Ad-hoc Network,” in *Proceeding of the International Conference on e-Education, Entertainment and e-Management*, 2011, pp. 70–75.

[23] E. Buckner and P. Kim, “Integrating technology and pedagogy for inquiry-based learning: The Stanford Mobile Inquiry-based Learning Environment (SMILE),” *Prospects*, vol. 44, no. 1, pp. 99–118, 2014.

[24] “Learning Catalytics^{TM}.” [Online]. Available: https://www.pearsonhighered.com/products-and-services/course-content-and-digital-resources/learning-applications/learning-catalytics.html. [Accessed: 03-May-2017].

[25] O. Chanprasitchai and J. Khlaisang, “Inquiry-Based Learning for a Virtual Learning Community to Enhance Problem-Solving Ability of Applied Thai Traditional Medicine Students,” *TOJET Turk. Online J. Educ. Technol. Adapazari*, vol. 15, no. 4, pp. 77-87, 2016.

[26] P. Lameras, et al., “Fostering Science Teachers’ Design for Inquiry-Based Learning by Using a Serious Game,” in *Proceedings of IEEE 14th International Conference on Advanced Learning Technologies*, 2014, pp. 222–226.

[27] I. A. Zualkernan, et al., “Using Problem Posing, Problem Solving for Game-Based Learning in Remote Labs,” in *Proceedings of IEEE 12th International Conference on Advanced Learning Technologies*, 2012, pp. 716–717.

[28] S. Shapsough, et al., “ARCHI-PODS: Ubiquitous Learning Technology to Teach Architectural Design Principles to Architecture Students,” *in Proceedings of 9th International Technology, Education and Development Conference*, 2015, pp. 4338–4347.

[29] A. Mikroyannidis, “weSPOT: A personal and social toolkit for inquiry-based learning,” *in Proceedings of International Conference on Web and Open Access to Learning*, 2014, pp. 1–4.

[30] M. A. Bedek, et al., “User-driven Development of an Inquiry-Based Learning Platform: Qualitative Formative Evaluations in weSPOT,” *Interact. Des. Archit.*, no. 23, pp. 122–139, 2015.

[31] A. Suarez, et al., “GPIM: Google Glassware for inquiry-based learning,” *Interact. Des. Archit.*, no. 24, pp. 100–110, 2015.

[32] B. de Sousa Monteiro, et al., “Youubi: Open software for ubiquitous learning,” *Comput. Hum. Behav.*, vol. 55, p. B, pp. 1145-1164, Feb. 2016.

[33] N. S. Chen, et al., “Developing Ubiquitous Learning System with Robots for Children’s Learning,” in *Proceedings of 3rd IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, 2010, pp. 61–68.

[34] F. Abdullah and R. Ward, “Developing a General Extended Technology Acceptance Model for E-Learning (GETAMEL) by analysing commonly used external factors,” *Comput. Hum. Behav.*, vol. 56, pp. 238–256, 2016.

[35] H.-Y. Chang *et al.*, “A review of features of technology-supported learning environments based on participants’ perceptions,” *Comput. Hum. Behav.*, vol. 53, pp. 223–237, 2015.

[36] L. Atzori, et al., “The Internet of Things: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[37] I. Mashal, et al., “Choices for interaction with things on Internet and underlying issues,” *Ad Hoc Netw.*, vol. 28, pp. 68–90, 2015.

[38] P. P. Ray, “A survey on Internet of Things architectures,” *J. King Saud Univ. - Comput. Inf. Sci.,* 2016.

[39] J. Guth, et al., “Comparison of IoT platform architectures: A field study based on a reference architecture,” *in Proceedings of 2016 Cloudification of the Internet of Things*, 2016, pp. 1–6.

[40] J. Gubbi, et al., “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[41] Z. Sheng, et al., “A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities,” *IEEE Wirel. Commun.*, vol. 20, no. 6, pp. 91–98, Dec. 2013.

[42] S. C. Mukhopadhyay and SpringerLink, "*Internet of Things: Challenges and Opportunities",* Springer International Publishing, 2014.

[43] “| IMS Global Learning Consortium.” [Online]. Available: http://www.imsglobal.org/question/index.html. [Accessed: 11-Apr-2017].

[44] B. Dolan, et al., “Next-Generation Assessment Interoperability Standards,” 2010.

[45] “| IMS Global Learning Consortium.” [Online]. Available: /question/qtiv1p2/imsqti_litev1p2.html. [Accessed: 11-Apr-2017].

[46] *Moodle - Open-source learning platform \textbar Moodle.org*. .

[47] Z. F. Muhsen, et al., “Moodle and e-learning Tools,” *Int. J. Mod. Educ. Comput. Sci.*, vol. 5, no. 6, pp. 1–8, 2013.

[48] “Moodle 3.0.” [Online]. Available: https://docs.moodle.org/. [Accessed: 24-Mar-2017].

[49] P. Bradford, et al., “The Blackboard Learning System,” *J. Educ. Technol. Syst.*, no. 35, pp. 301–314, 2007.

[50] "Raspberry Pi - Teach, Learn, and Make with Raspberry Pi" [Online]. Available: https://www.raspberrypi.org. [Accessed: 24-Mar-2017].

[51] “BeagleBoard.org - bone.” [Online]. Available: http://beagleboard.org/bone. [Accessed: 24-Mar-2017].

[52] “ODROID | Hardkernel.” [Online]. Available: http://www.hardkernel.com/main/main.php. [Accessed: 26-Mar-2017].

[53] “The Intel® Edison Module | IoT | Intel® Software.” [Online]. Available: https://software.intel.com/en-us/iot/hardware/edison. [Accessed: 25-Mar-2017].

[54] N. T. Co, “Get C.H.I.P. and C.H.I.P. Pro - The Smarter Way to Build Smart Things,” *Next Thing Co.* [Online]. Available: https://getchip.com/pages/chip. [Accessed: 25-Mar-2017].

[55] “Tizen | An open source, standards-based software platform for multiple device categories.” [Online]. Available: https://www.tizen.org/. [Accessed: 23-Apr-2017].

[56] “chroot - Debian Wiki.” [Online]. Available: https://wiki.debian.org/chroot. [Accessed: 24-Mar-2017].

[57] “ChrootOnAndroid - Debian Wiki.” [Online]. Available: https://wiki.debian.org/ChrootOnAndroid. [Accessed: 24-Mar-2017].

[58] A. Bari, et al., “Ultra-low cost vehicle data acquisition and transfer system from analog and digital sensors to audio channel of a phone,” *in Proceedings of 16th International Conference on Advanced Communication Technology*, 2014, pp. 698–704.

[59] D. Hong *et al.*, “Demo abstract: Continuous in-situ human wellness monitoring and feedback using sensors embedded in earphones,” *in Proceedings of 2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks*, 2012, pp. 159–160.

[60] P. A. Shinde, et al., “Real time vehicle monitoring and tracking system based on embedded Linux board and android application,” in *Proceedings of 2015 International Conference on Circuits, Power and Computing Technologies*, 2015, pp. 1–7.

[61] H. Jiang, et al., “An Audio Jack-Based Electrochemical Impedance Spectroscopy Sensor for Point-of-Care Diagnostics,” *IEEE Sens. J.*, vol. 17, no. 3, pp. 589–597, Feb. 2017.

[62] “Particle.” [Online]. Available: https://www.particle.io/. [Accessed: 21-Mar-2017].

[63] “Pinoccio,” *Crowd Supply*. [Online]. Available: https://www.crowdsupply.com/pinoccio/mesh-sensor-network. [Accessed: 21-Mar-2017].

[64] “Adafruit Feather HUZZAH with ESP8266 WiFi ID: 2821 - $16.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” [Online]. Available: https://www.adafruit.com/products/2821. [Accessed: 25-Mar-2017].

[65] “ESP8266EX Overview | Espressif Systems.” [Online]. Available: http://espressif.com/en/products/hardware/esp8266ex/overview. [Accessed: 25-Mar-2017].

[66] “Raspberry Pi 3 Model B,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 24-Mar-2017].

[67] “ODROID | Hardkernel.” [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php. [Accessed: 24-Mar-2017].

[68] “BeagleBoard.org - black.” [Online]. Available: https://beagleboard.org/black. [Accessed: 26-Mar-2017].

[69] “NextThingCo/CHIP_Pro-Hardware,” *GitHub*. [Online]. Available: https://github.com/NextThingCo/CHIP_Pro-Hardware. [Accessed: 26-Mar-2017].

[70] “Samsung ARTIK IoT Platform - ARTIK 710 IoT module.” [Online]. Available: https://www.artik.io/modules/artik-710/. [Accessed: 02-Apr-2017].

[71] “ARTIK 530 - ARTIK IoT Platform.” [Online]. Available: https://www.artik.io/modules/artik-530/. [Accessed: 02-Apr-2017].

[72] "Gear S Watch - GPS, 3G, Wi-Fi, Bluetooth v4.1 (Blue Black) - Samsung UK." [Online]. Available: http://www.samsung.com/uk/wearables/gear-s-watch-r750/SM-R7500ZKABTU/. [Accessed: 02-Apr-2017].

[73] "SmartWatch 3 SWR50 | Smartphone Watch - Sony Mobile (Global UK English)." [Online]. Available: https://www.sonymobile.com/global-en/products/smart-products/smartwatch-3-swr50/. [Accessed: 02-Apr-2017].

[74] "Samsung Galaxy J5 Quad Core 1.2GHz 5" Black," *Samsung uk*. [Online]. Available: http://www.samsung.com/uk/smartphones/galaxy-j5-j500fn/SM-J500FZKABTU/. [Accessed: 26-Mar-2017].

[75] "Samsung ARTIK IoT Platform - Samsung ARTIK 1020 IoT Module." [Online]. Available: https://www.artik.io/modules/artik-1020/. [Accessed: 02-Apr-2017].

[76] "Endlessm.com - Official Endless Store," *Endless Store*. [Online]. Available: https://endless-global.myshopify.com/. [Accessed: 28-Mar-2017].

[77] "Arduino - ArduinoBoardYun." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardYun. [Accessed: 26-Mar-2017].

[78] "Arduino - ArduinoBoardUno." [Online]. Available: https://www.arduino.cc/en/main/arduinoBoardUno. [Accessed: 26-Mar-2017].

[79] "Particle." [Online]. Available: https://docs.particle.io/datasheets/photon-datasheet/. [Accessed: 21-Mar-2017].

[80] "Adafruit Feather M0 Bluefruit LE ID: 2995 - $29.95 : Adafruit Industries, Unique & fun DIY electronics and kits." [Online]. Available: https://www.adafruit.com/product/2995. [Accessed: 26-Mar-2017].

[81] D. N. Aspin, et al., *Values Education and Lifelong Learning: Principles, Policies, Programmes*. Dordrecht: Springer, 2007.

[82] "MySQL." [Online]. Available: https://www.mysql.com/. [Accessed: 26-Mar-2017].

[83] "The Apache Cassandra Project." [Online]. Available: http://cassandra.apache.org/. [Accessed: 26-Mar-2017].

[84] "Apache CouchDB." [Online]. Available: http://couchdb.apache.org/.[Accessed: 18-Mar-2017].

[85] "MongoDB for GIANT Ideas." [Online]. Available: https://www.mongodb.org/. [Accessed: 18-Mar-2017].

[86] B. Holt, *Writing and Querying MapReduce Views in CouchDB*. O'Reilly Media, Inc., 2011.

[87] R. Fielding, Ed., J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014.

[88] D. Uckelmann, et al., *Architecting the Internet of Things*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[89] F. C. Delicato, et al., *Middleware solutions for the internet of things*. London: Springer, 2013.

[90] Z. Shelby, et al.,"The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014.

[91] M. Castro, et al., "Enabling end-to-end CoAP-based communications for the Web of Things," *J. Netw. Comput. Appl.*, vol. 59, pp. 230–236, Jan. 2016.

[92] C. Bormann, et al., "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Comput.*, vol. 16, no. 2, pp. 62–67, 2012.

[93] D. Thangavel, et al., "Performance evaluation of MQTT and CoAP via a common middleware," in *Proceedings of the IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2014, pp. 1–6.

[94] M. H. Elgazzar, "Perspectives on M2M protocols," in *Proceedings of 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems*, 2015, pp. 501–505.

[95] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011.

[96] S. Bendel, et al., "A service infrastructure for the Internet of Things based on XMPP," in *Proceedings of the 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2013, pp. 385–388.

[97] R. Klauck and M. Kirsche, "Chatty things - Making the Internet of Things readily usable for the masses with XMPP," in *Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2012, pp. 60–69.

[98] A. Banks and R. Gupta, "MQTT Version 3.1. 1," *OASIS Stand.*, 2014.

[99] U. Hunkeler, et al., "MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks," in *Proceedings of the 2008 3rd International Conference on*

Communication Systems Software and Middleware and Workshops, 2008, pp. 791–798.

[100] "Building Facebook Messenger" [Online]. Available: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/. [Accessed: 14-Feb-2017].

[101] “HiveMQ: Enterprise MQTT Broker.” [Online]. Available: http://www.hivemq.com/. [Accessed: 20-Mar-2017].

[102] “Paho.” [Online]. Available: https://eclipse.org/paho/. [Accessed: 20-Mar-2017].

[103] S. Aiyagari, et al., “Advanced Message Queuing Protocol (AMQP) Version 0-9-1,” OASIS Standard, November 2008.

[104] S. Appel, et al., “Towards benchmarking of AMQP,” in *Proceedings of the 4th ACM International Conference on distributed event-based systems*, 2010, pp. 99–100.

[105] J. E. Luzuriaga, et al., “A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks,” in *Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference*, 2015, pp. 931–936.

[106] R. Cohn, “A Comparison of AMQP and MQTT,” *StormMQ White Paper*, February 2012.

[107] K. Lei, et al., “Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js,” in *Proceedings of the IEEE 17th International Conference on Computational Science and Engineering*, 2014, pp. 661–668.

[108] L. Griffin, et al., “Scaling Instant Messaging communication services: A comparison of blocking and non-blocking techniques,” in *Proceedings of the IEEE Symposium on Computers and Communications*, 2011, pp. 550–557.

[109] “Node.js.” [Online]. Available: https://nodejs.org/. [Accessed: 20-Mar-2017].

[110] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, 2010.

[111] "Express - Node.js web application framework." [Online]. Available: https://expressjs.com/. [Accessed: 21-Mar-2017].

[112] "NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy," *NGINX*. [Online]. Available: https://www.nginx.com/. [Accessed: 13-Apr-2017].

[113] *PARCC: Leveraging Open Standards to Support Next Generation Online Assessments*. IMS Global Learning Consortium, 2015.

[114] S. Charkaoui, et al., "Cross-platform mobile development approaches," *Proceedings of 3rd IEEE International Colloquium on Information Science and Technology*, 2014, pp. 188–191.

[115] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," in *Proceedings of the 2013 9th International Wireless Communications and Mobile Computing Conference*, 2013, pp. 323–328.

[116] "Apache Cordova." [Online]. Available: https://cordova.apache.org/ [Accessed: 20-Mar-2017].

[117] J. Perchat, et al., "Common Framework: A Hybrid Approach to Integrate Cross-Platform Components In Mobile Application," *J. Comput. Sci.*, vol. 10, no. 11, pp. 2165–2181, 2014.

[118] M. A. Jazayeri, et al., "Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things," *Sensors*, vol. 15, no. 9, pp. 24343–24373, 2015.

[119] X. Che and S. Maag, "Testing protocols in Internet of Things by a formal passive technique," *Sci. China Inf. Sci.*, vol. 57, no. 3, pp. 1–13, 2014.

[120] "Yocto-Amp - Tiny isolated USB ammeter (AC/DC)." [Online]. Available: http://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-amp. [Accessed: 27-Mar-2017].

[121] "nmon for Linux | Main / HomePage." [Online]. Available: http://nmon.sourceforge.net/pmwiki.php. [Accessed: 27-Mar-2017].

[122] "JXcore," *GitHub*. [Online]. Available: https://github.com/jxcore. [Accessed: 27-Mar-2017].

[123] "Wireshark." [Online]. Available: https://www.wireshark.org/.

[124] "jagt/clumsy," *GitHub*. [Online]. Available: https://github.com/jagt/clumsy. [Accessed: 27-Mar-2017].

[125] "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool." [Online]. Available: https://iperf.fr/. [Accessed: 04-May-2017].

[126] "Automated Software Testing Made Simple | TestComplete." [Online]. Available: https://smartbear.com/product/testcomplete/overview/. [Accessed: 27-Mar-2017].

[127] "Calculating Your Solar Power Requirements." [Online]. Available: http://www.solartechnology.co.uk/support-centre/calculating-your-solar-requirments. [Accessed: 01-Apr-2017].

[128] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," in *Proceedings of 2013 International Conference on Computing, Networking and Communications*, 2013, pp. 334–340.

[129] "Solar Power Bank 10000mah Powerbank Portable Charger Extra External Battery For Mobile Phones" [Online]. Available: //www.alibaba.com/product-detail/Solar-power-bank-10000mah-powerbank-portable_60545891943.html. [Accessed: 02-May-2017].

[130] M. Collina, et al., "Internet of Things application layer protocol analysis over error and delay prone links," in *Proceedings of the 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop*, 2014, pp. 398–404.

[131] A. Talaminos-Barroso, et al., "A Machine-to-Machine protocol benchmark for eHealth applications – Use case: Respiratory rehabilitation," *Comput. Methods Programs Biomed.*, vol. 129, pp. 1–11, Jun. 2016.

[132] J. Perser, et al., "Packet Reordering Metrics." [Online]. Available: https://tools.ietf.org/html/rfc4737#section-1.1. [Accessed: 24-Apr-2017].

[133] A. Jayasumana, "Improved Packet Reordering Metrics." [Online]. Available: https://tools.ietf.org/html/rfc5236#ref-Pax97. [Accessed: 24-Apr-2017].

[134] C. Costinela-Luminiţa and C. Nicoleta-Magdalena, "E-learning Security Vulnerabilities," *Procedia - Soc. Behav. Sci.*, vol. 46, pp. 2297–2301, Jan. 2012.

[135] S. Shapsough, et al., "Smart grid cyber security: Challenges and solutions," in *Proceedings of the 2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*, 2015, pp. 170–175.

[136] F. D. S. Bahry, et al., "Conceptualizing Security Measures on Mobile Learning for Malaysian Higher Education Institutions," *Procedia - Soc. Behav. Sci.*, vol. 176, pp. 1083–1088, Feb. 2015.

[137] T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2." [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed: 19-Apr-2017].

[138] K. D. Zeilenga and A. Melnikov, "Simple Authentication and Security Layer (SASL)." [Online]. Available: https://tools.ietf.org/html/rfc4422. [Accessed: 19-Apr-2017].

[139] Y. Dzambasow, et al., "Internet X.509 Public Key Infrastructure: Certification Path Building." [Online]. Available: https://tools.ietf.org/html/rfc4158. [Accessed: 19-Apr-2017].

[140] N. Modadugu and E. Rescorla, "Datagram Transport Layer Security." [Online]. Available: https://tools.ietf.org/html/rfc4347. [Accessed: 19-Apr-2017].

[141] R. Atkinson and S. Kent, "IP Encapsulating Security Payload (ESP)." [Online]. Available: https://tools.ietf.org/html/draft-ietf-ipsec-esp-v2-05. [Accessed: 19-Apr-2017].

[142] A. Yegin and Z. Shelby, "CoAP Security Options." [Online]. Available: https://tools.ietf.org/html/draft-yegin-coap-security-options-00. [Accessed: 19-Apr-2017].

[143] H. Tschofenig and P. Eronen, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)." [Online]. Available: https://tools.ietf.org/html/rfc4279. [Accessed: 19-Apr-2017].

[144] "Part 4: RabbitMQ Exchanges, routing keys and bindings - CloudAMQP." [Online]. Available: https://www.cloudamqp.com/blog/2015-09-03-part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html. [Accessed: 30-Apr-2017].

[145] "MQTT Essentials Part 5: MQTT Topics & Best Practices." [Online]. Available: http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices. [Accessed: 30-Apr-2017].

Appendix A

This appendix presents the statistical distribution of latency values for each network disturbance experiments carried out during Experiment II. Each graph presents a comparison of the four implementations at a certain experimental value.

A.1. Out of Order

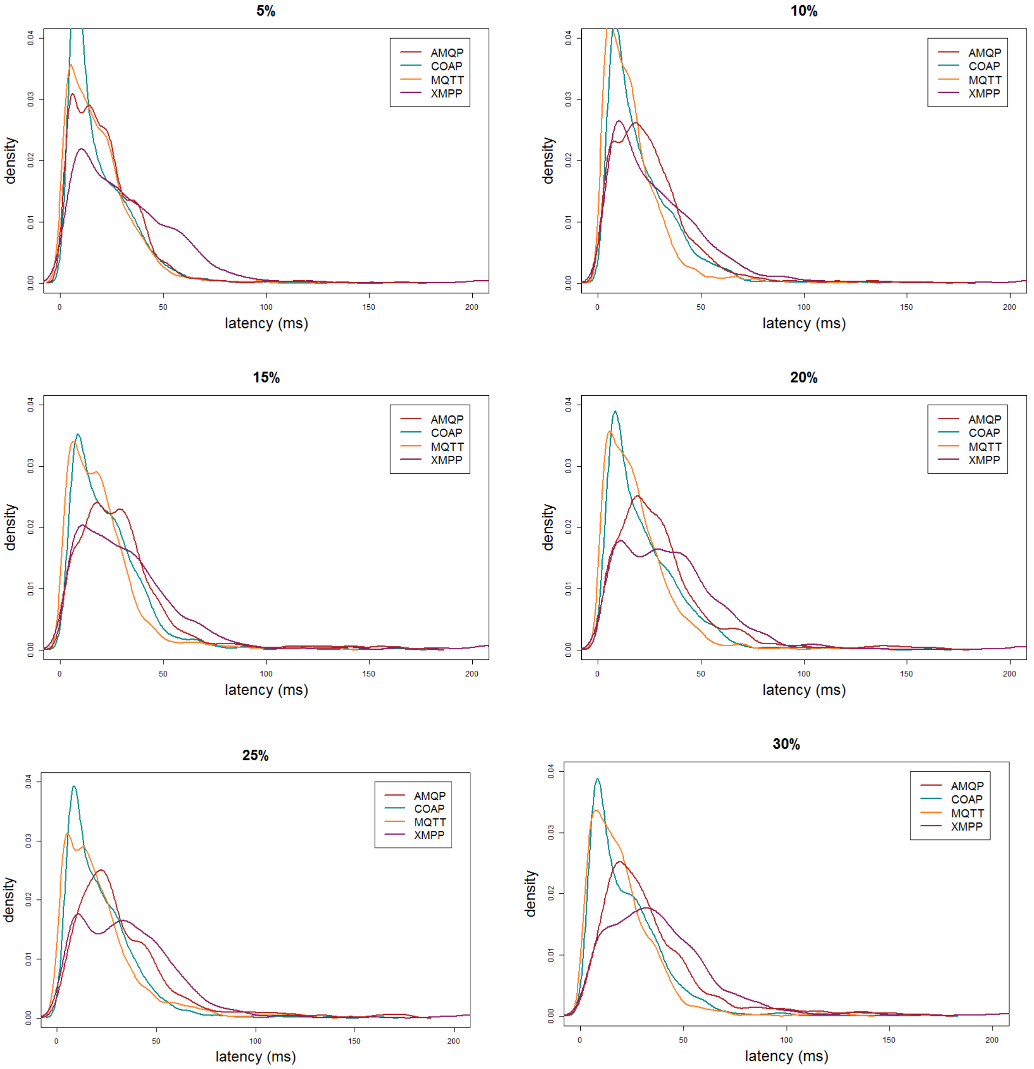


Figure A.1: Latency distribution as response to out of order packets (5% - 30%)

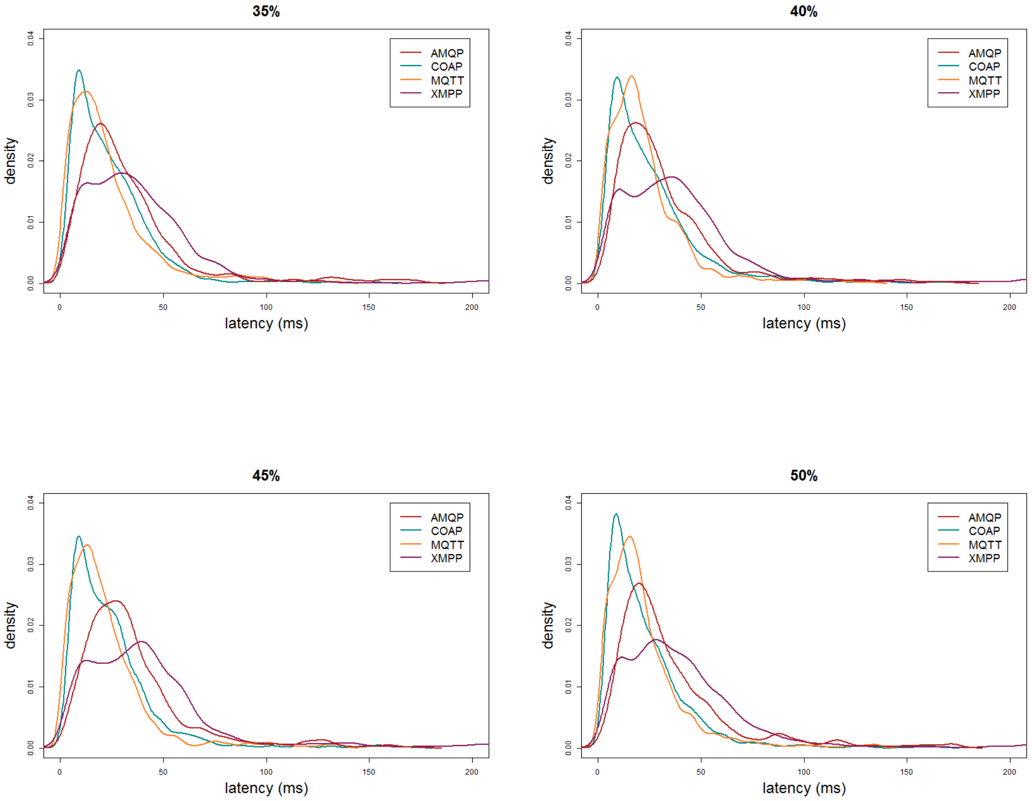


Figure A.2: Latency distribution as response to out of order packets (35% - 50%)

A.2. Lag

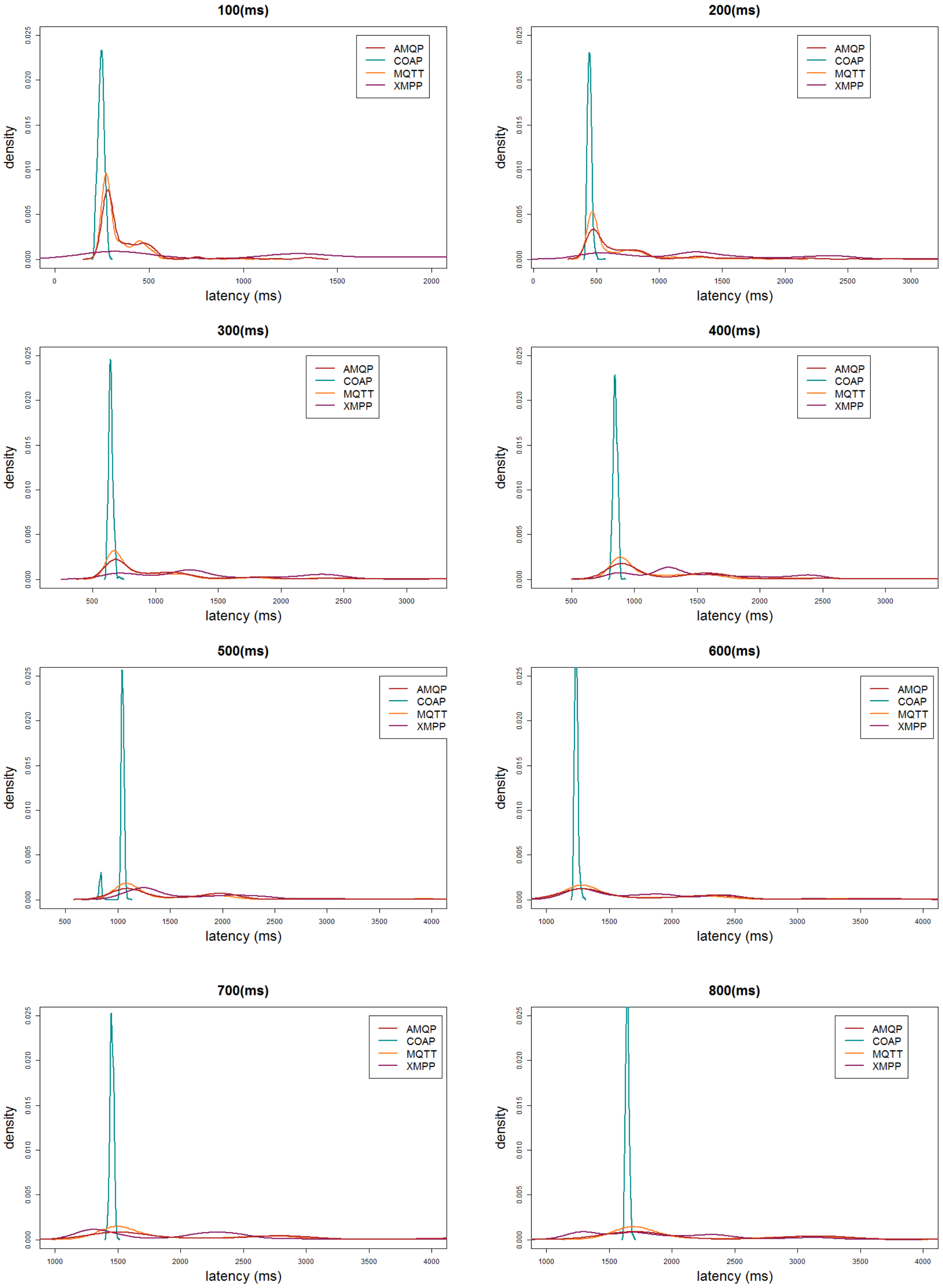


Figure A.3: Latency distribution as response to network lag (100ms – 800ms)

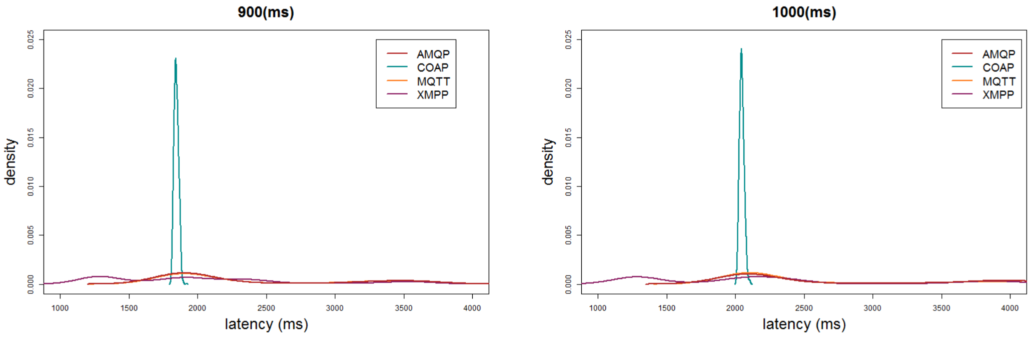


Figure A.4: Latency distribution as response to network lag (900ms – 1000ms)

A.3. Duplicates

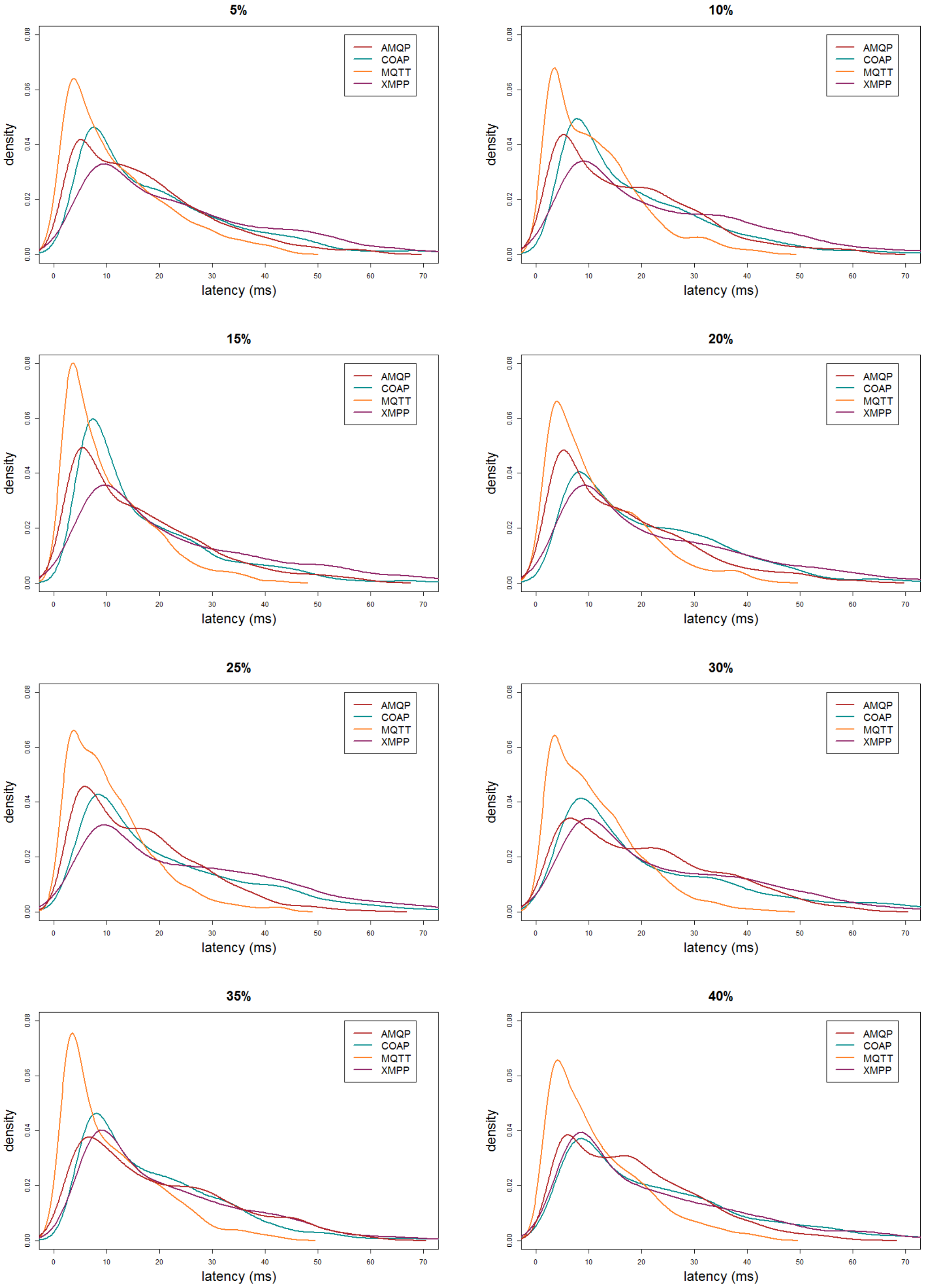


Figure A.5: Latency distribution as response to duplicated packets (5% – 40%)

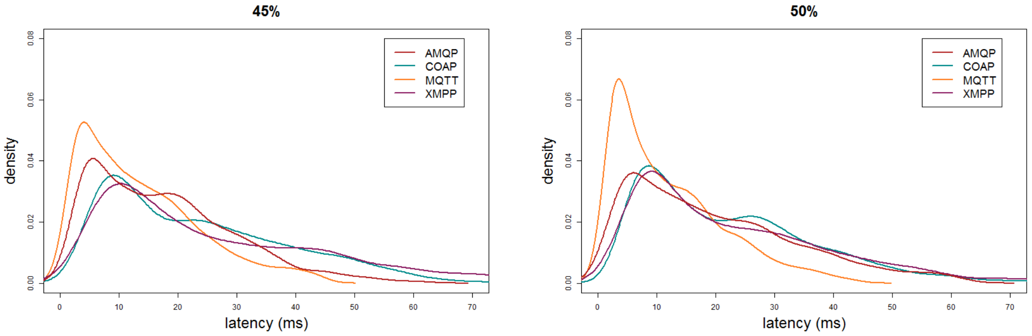


Figure A.6: Latency distribution as response to duplicated packets (45% – 50%)

A.4. Throttle

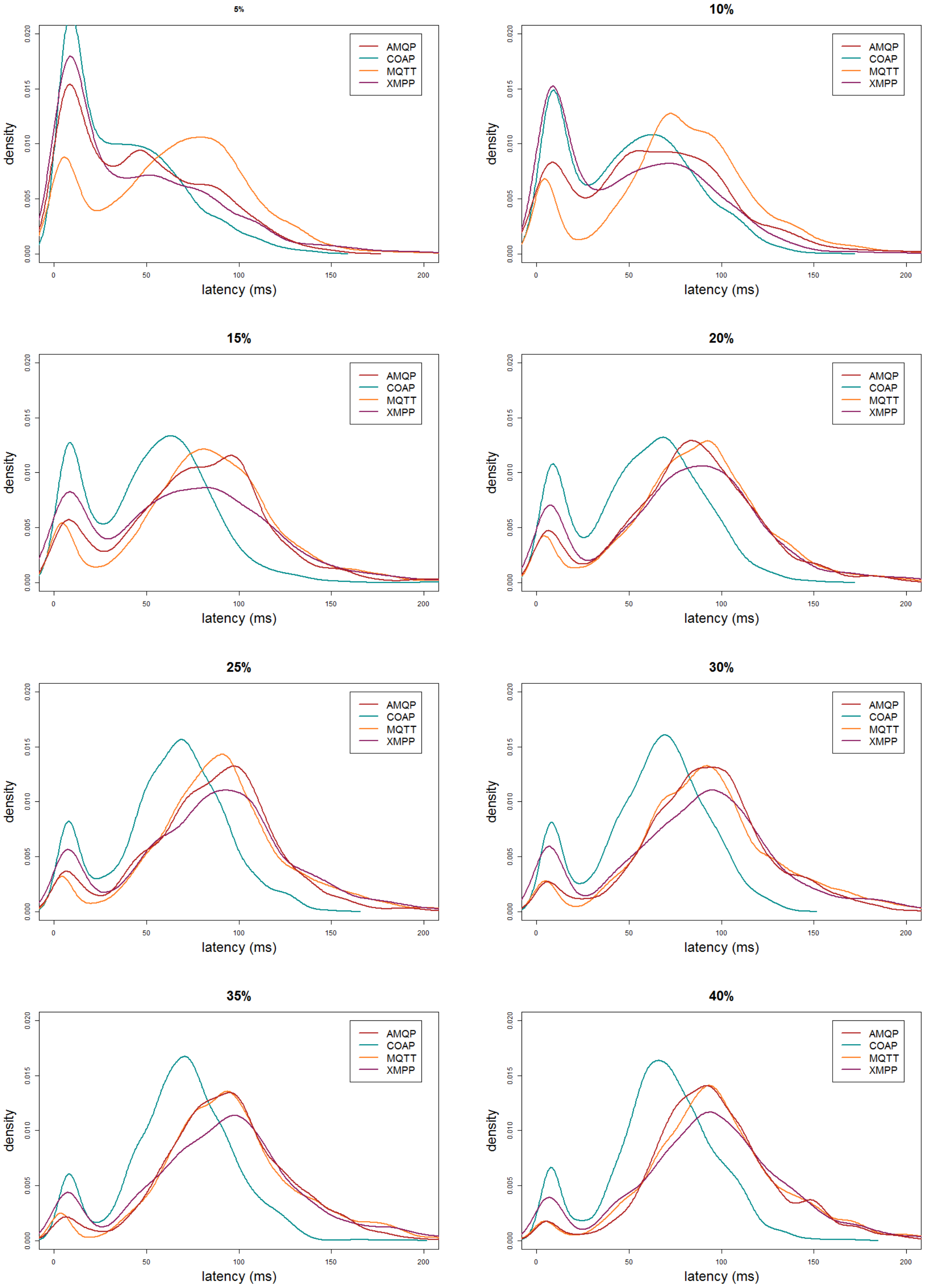


Figure A.7 Latency distribution as response to throttled packets (5% – 40%)

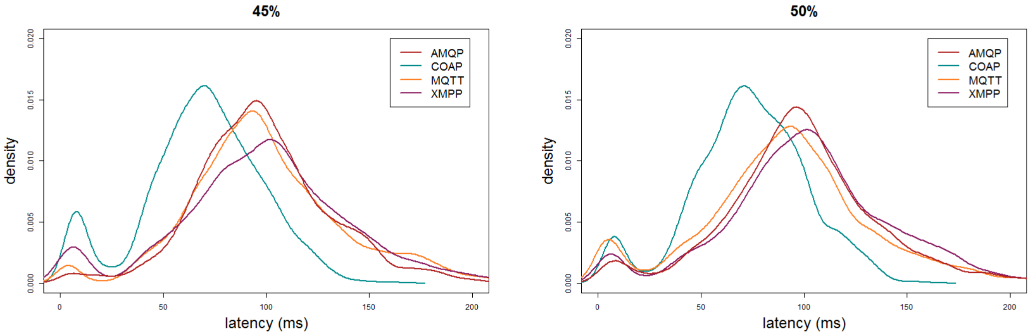


Figure A.8 Latency distribution as response to throttled packets (45% – 50%)

Appendix B

This appendix presents a comparison between the means for every two protocols, at each experimental value, for each network disturbance metric. Two-sample T-Test was carried out for delay values found in Experiment II for $p < 0.05$.

B.1. Out of Order

Table B.1. Comparing Delay vs. Out of Order % across protocols

Out of Order %	Implementation 1$_{mean}$	P-value	Implementation 2$_{mean}$
50	AMQP $_{35.4}$	0	CoAP $_{23.2}$
	AMQP	0	MQTT $_{22.2}$
	AMQP	0	XMPP $_{39.8}$
	CoAP	0.215	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > MQTT = CoAP			
45	AMQP $_{35.5}$	0	CoAP $_{24.5}$
	AMQP	0	MQTT $_{22.4}$
	AMQP	0	XMPP $_{42.1}$
	CoAP	0.003	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
40	AMQP $_{32.3}$	0	CoAP $_{25.4}$
	AMQP	0	MQTT $_{22.7}$
	AMQP	0	XMPP $_{39.6}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
35	AMQP $_{33.3}$	0	CoAP $_{23.7}$
	AMQP	0	MQTT $_{23.3}$
	AMQP	0.001	XMPP $_{36.6}$
	CoAP	0.634	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > MQTT = CoAP			
30	AMQP $_{34.0}$	0	CoAP $_{22.6}$
	AMQP	0	MQTT $_{20.4}$
	AMQP	0	XMPP $_{40.2}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
25	AMQP $_{33.9}$	0	CoAP $_{22.4}$
	AMQP	0	MQTT $_{22.1}$
	AMQP	0.005	XMPP $_{36.8}$
	CoAP	0.608	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > MQTT = CoAP			
20	AMQP $_{31.4}$	0	CoAP $_{22.8}$
	AMQP	0	MQTT $_{19.1}$
	AMQP	0	XMPP $_{36.6}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
15	AMQP $_{31.1}$	0	CoAP $_{23.3}$
	AMQP	0	MQTT $_{20.0}$
	AMQP	0.001	XMPP $_{34.4}$
	CoAP	0	MQTT

	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
10	AMQP $_{26.0}$	0	CoAP $_{21.7}$
	AMQP	0	MQTT $_{17.2}$
	AMQP	0	XMPP $_{30.4}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > CoAP > MQTT			
5	AMQP $_{21.9}$	0	CoAP $_{19.6}$
	AMQP	0	MQTT $_{18.8}$
	AMQP	0	XMPP $_{33.1}$
	CoAP	0.134	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > AMQP > MQTT > CoAP			

B.2. Lag

Table B.2. Comparing Delay vs. Out of Lag (ms) across protocols

Lag (ms)	Implementation 1_{mean}	P-value	Implementation 2_{mean}
1000	AMQP	0	CoAP
	AMQ	0.383	MQTT
	AMQ	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP = MQTT > XMPP > CoAP			
900	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
MQTT > AMQP > XMPP > CoAP			
800	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			
700	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			
600	AMQP	0	CoAP
	AMQP	0.003	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			
500	AMQP	0	CoAP
	AMQP	0.080	MQTT
	AMQP	0.015	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0.408	XMPP
XMPP = AMQP = MQTT > CoAP			
400	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			
300	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			
200	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP

AMQP > MQTT > XMPP > CoAP			
100	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
AMQP > MQTT > XMPP > CoAP			

B.3. Throttle

Table B.3. Comparing Delay vs. Throttling Probability across protocols

Throttle %	Implementation 1_{mean}	P-value	Implementation 2_{mean}
50	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0.009	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP>AMQP>MQTT>CoAP			
45	AMQP	0	CoAP
	AMQP	0.197	MQTT
	AMQP	0.009	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0.075	XMPP
XMPP>AMQP=MQTT>CoAP			
40	AMQP	0	CoAP
	AMQP	0.264	MQTT
	AMQP	0.047	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0.002	XMPP
AMQP=MQTT>XMPP>CoAP			
35	AMQP	0	CoAP
	AMQP	0.004	MQTT
	AMQP	0.488	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0.003	XMPP
MQTT>AMQP=XMPP>CoAP			
30	AMQP	0	CoAP
	AMQP	0.046	MQTT
	AMQP	0.011	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
MQTT>AMQP>XMPP>CoAP			
25	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0.491	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0.004	XMPP
MQTT>AMQP=XMPP>CoAP			
20	AMQP	0	CoAP
	AMQP	0.009	MQTT
	AMQP	0.02	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
MQTT>AMQP>XMPP>CoAP			
15	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0.001	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
MQTT>AMQP>XMPP>CoAP			
10	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0	XMPP
	CoAP	0	MQTT
	CoAP	0.182	XMPP
	MQTT	0	XMPP

MQTT>AMQP>XMPP=CoAP			
5	AMQP	0	CoAP
	AMQP	0	MQTT
	AMQP	0.27	XMPP
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
MQTT>AMQP=XMPP>CoAP			

B.4. Duplicates

Table B.4. Comparing Delay vs. Duplicates % across protocols

Duplicate %	Implementation 1$_{mean}$	P-value	Implementation 2$_{mean}$
50	AMQP $_{19.8}$	0	CoAP $_{22.1}$
	AMQP	0	MQTT $_{11.90}$
	AMQP	0	XMPP $_{23.3}$
	CoAP	0	MQTT
	CoAP	0.019	XMPP
	MQTT	0	XMPP
XMPP > CoAP > AMQP > MQTT			
45	AMQP $_{17.6}$	0	CoAP $_{24.0}$
	AMQP	0	MQTT $_{13.58}$
	AMQP	0	XMPP $_{25.9}$
	CoAP	0	MQTT
	CoAP	0.001	XMPP
	MQTT	0	XMPP
AMPP > CoAP > AMQP > MQTT			
40	AMQP $_{18.4}$	0	CoAP $_{23.7}$
	AMQP	0	MQTT $_{11.9}$
	AMQP	0	XMPP $_{24.1}$
	CoAP	0	MQTT
	CoAP	0.638	XMPP
	MQTT	0	XMPP
XMPP = CoAP > AMQP > MQTT			
35	AMQP 19.7	0.547	CoAP 19.9
	AMQP	0	MQTT 11.06
	AMQP	0	XMPP 26.5
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP = AMQP > MQTT			
30	AMQP $_{20.5}$	0	CoAP $_{23.1}$
	AMQP	0	MQTT $_{11.20}$
	AMQP	0	XMPP $_{28.40}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP > AMQP > MQTT			
25	AMQP $_{16.3}$	0	CoAP $_{21.1}$
	AMQP	0	MQTT $_{10.86}$
	AMQP	0	XMPP $_{27.3}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP > AMQP > MQTT			
20	AMQP $_{16.4}$	0	CoAP $_{21.2}$
	AMQP	0	MQTT $_{11.73}$
	AMQP	0	XMPP $_{27.1}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP > AMQP > MQTT			
15	AMQP $_{16.2}$	0.112	CoAP $_{16.8}$
	AMQP	0	MQTT $_{10.31}$
	AMQP	0	XMPP $_{26.7}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP = AMQP > MQTT			
10	AMQP $_{17.4}$	0.001	CoAP $_{19.1}$
	AMQP	0	MQTT $_{11.25}$
	AMQP	0	XMPP $_{24.9}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP

XMPP > CoAP > AMQP > MQTT			
5	AMQP $_{17.0}$	0	CoAP $_{20.0}$
	AMQP	0	MQTT $_{12.13}$
	AMQP	0	XMPP $_{30.2}$
	CoAP	0	MQTT
	CoAP	0	XMPP
	MQTT	0	XMPP
XMPP > CoAP > AMQP > MQTT			

Appendix C

This appendix presents average throughput values as measured by Wireshark, for every network disturbance metric, for Experiment II.

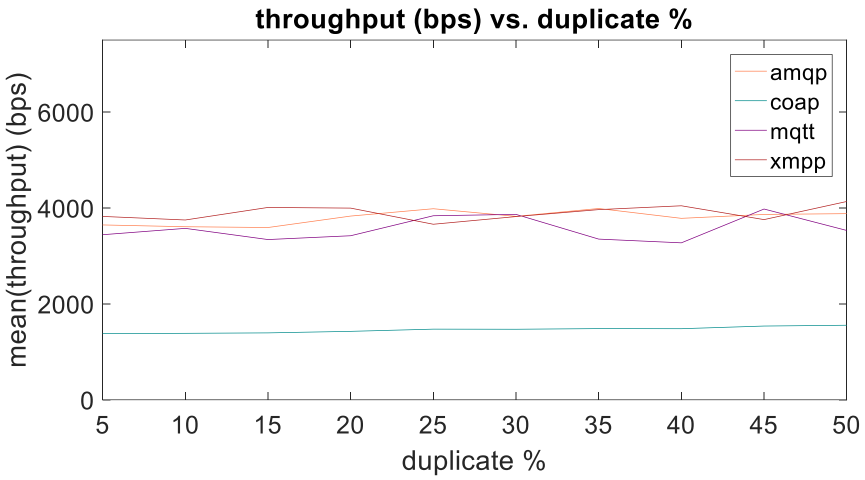


Figure C.1: Mean throughput vs. duplicates

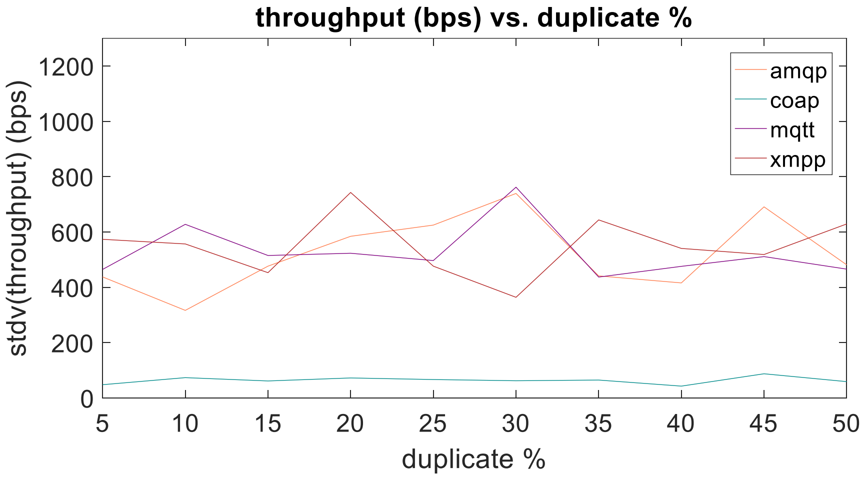


Figure C.2: Stdv. of throughput vs. duplicates

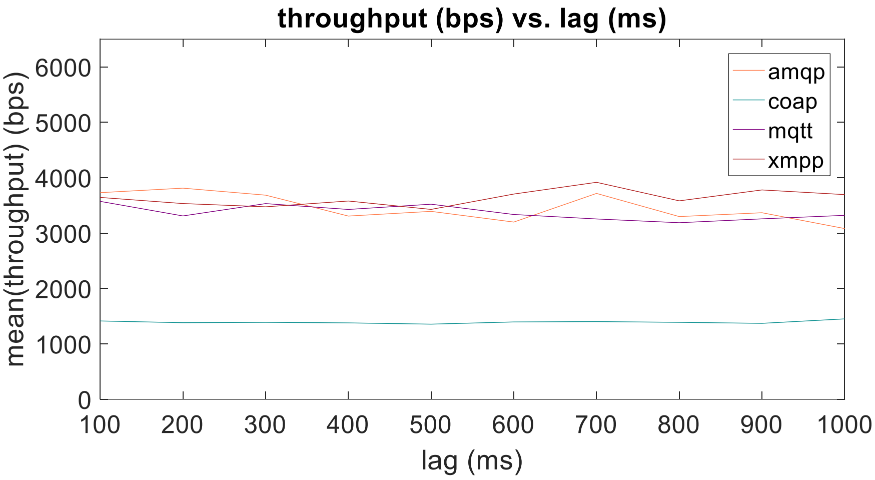


Figure C. 3: Mean throughput vs. lag

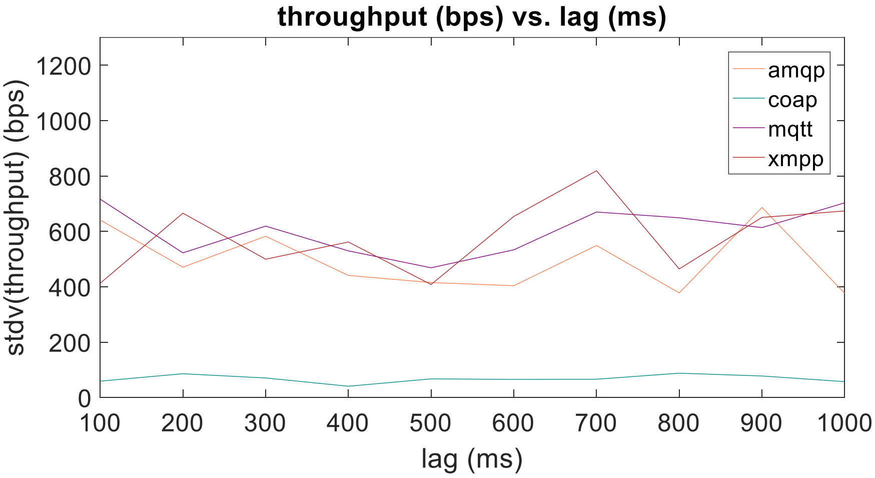


Figure C.4: Stdv. of throughput vs. lag

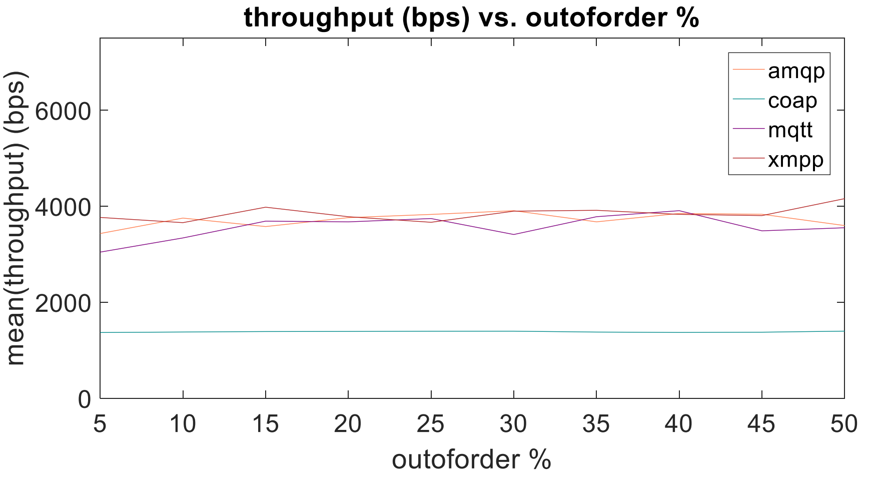


Figure C.5: Mean throughput vs. out of order

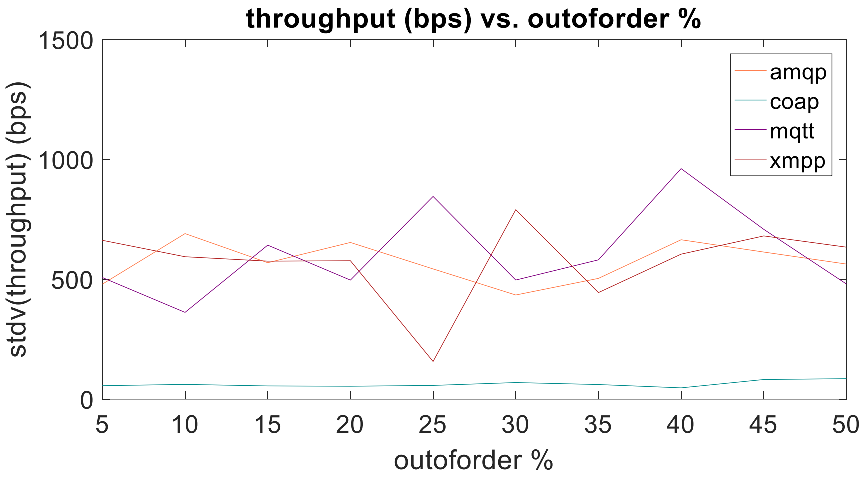


Figure C.6: Stdv. of throughput vs. out of order

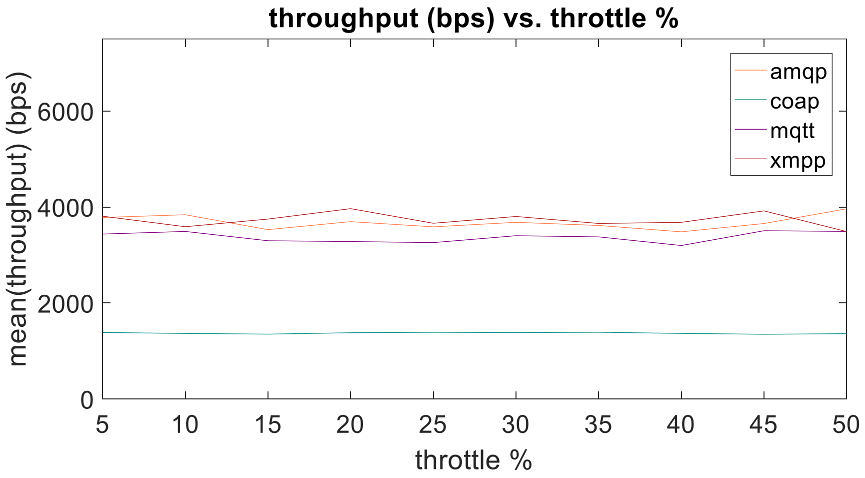


Figure C.7: Mean throughput vs. throttle

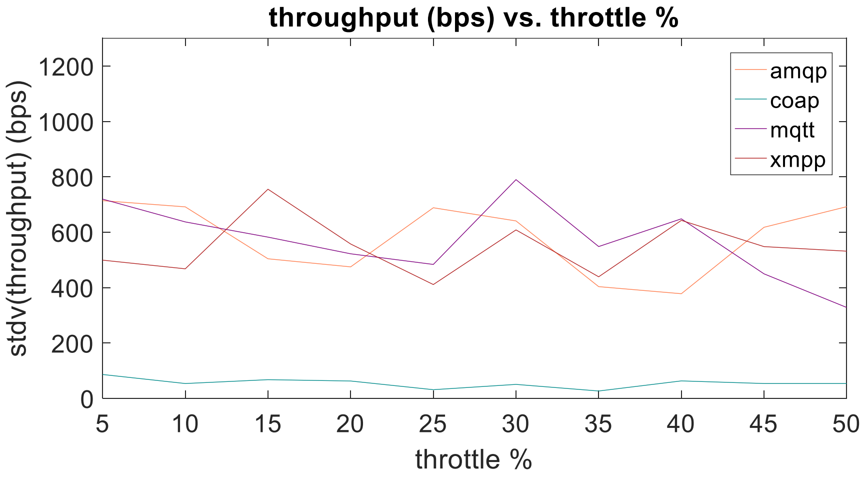


Figure C.8: Stdv. of throughput vs. throttle

Vita

Salsabeel Yousef Shapsough was born in 1993, in Amman, Jordan. She received her primary school education in Nezwa, Oman. She received her secondary and high school education in Dubai, UAE, and Ras Al Khaimah, UAE, respectively. She received her B.Sc. degree in Computer Engineering from the American University of Sharjah in 2014.

In February 2015, she joined the Computer Engineering master’s program in the American University of Sharjah as a graduate teaching assistant. During her master's study, she co-authored 6 papers which were presented in international conferences. Her research interests are Internet of Things, smart education, Big Data, and cybersecurity.