

AUS Repository

An IOT-Based Context-Aware Wearable Assessment Platform for Smart Watches

Item Type	Project
Authors	Al Solh, Mohamad Naeem
Download date	2026-04-16 10:44:00
Link to Item	http://hdl.handle.net/11073/9244

AN IOT-BASED CONTEXT-AWARE WEARABLE ASSESSMENT
PLATFORM FOR SMART WATCHES

by

Mohamad Naeem Al Solh

A Project 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

November 2017

Approval Signatures

We, the undersigned, approve the Master's Project of Mohamad Naeem Al Solh

Project Title: An IOT-Based Context-Aware Wearable Assessment Platform For Smart Watches

Signature

Date of Signature

(dd/mm/yyyy)

Dr. Imran Ahmed Zualkernan

Associate Professor, Department of Computer Science and Engineering
Thesis Advisor

Dr. Tarik Ozkul

Professor, Department of Computer Science and Engineering
Thesis Committee Member

Dr. Rana Ejaz Ahmed

Associate Professor, Department of Computer Science and Engineering
Thesis Committee Member

Dr. Fadi Aloul

Head, Department of Computer Science and Engineering

Dr. Ghaleb Hussein

Associate Dean for Graduate Affairs and Research
College of Engineering

Dr. Richard Schoephoerster

Dean, College of Engineering

Dr. Mohamed El-Tarhuni

Vice Provost for Graduate Studies

Acknowledgement

I would like to thank my advisor Dr. Imran Ahmed Zualkernan for providing his knowledge, guidance, support, inspiration, motivation and patience throughout all stages of my research. I'm deeply beholden to him for his valuable assistance, worthy discussion, insights and the new knowledge acquired from his guidance.

I would like to thank the professors in the Computer Science and Engineering Department who taught my master's level courses with their teaching methods and skills. I greatly appreciate their dignified advice and motivation.

At last but not least, I would like to thank my wife Mrs. Walaa for her unlimited patience and support throughout my master's program and work.

Dedication

*To my beloved family, my wife and daughters. Thank you for bringing joy to my life. I
dedicate this writing to you.*

Abstract

The purpose of this research is to build and evaluate an Internet of Things (IoT) architecture that utilizes the Message Queue Telemetry Transport (MQTT) and an end-to-end JavaScript stack with a NoSQL database to process real-time on-board sensor data on smart watches to implement context-aware ubiquitous learning applications. A prototype system was designed and implemented to serve curriculum-aligned, real-time assessments utilizing smartwatch sensors that represent a learner's environmental and bodily context. The system integrates a variety of on-board watch sensors such as a Global Positioning System (GPS), a Pedometer, Light Intensity, Ultra-violet Radiation, and a Heart Rate Monitor. An integrated smartwatch program uses JavaScript/HTML5. The JavaScript stack utilizes Node.JS/Express for the middleware and Angular 4 for the teacher administration portal. The system uses Google Classroom as the learning management system, PONTE as the MQTT broker, and CouchDB as the NoSQL database. The performance of the prototype was evaluated on real smart watches under various network conditions (Wi-Fi, 3G, EDGE, and 2G). The backend servers were also assessed for scalability. Without edge analytics, the average worst-case response time of telemetry submission and acknowledgement (160-second interval and about 95 KB sensor data) was an acceptable 4.5 seconds for the 2G Lossy Rural, and 356 milliseconds for Wi-Fi. Under normal conditions, watch CPU utilization was between 30-90% and never exceeded 98% in the worst case. Watch battery depleted on average 8.64% for a half-an-hour ubiquitous learning session. A typical quad-core laptop running broker, middleware, and database had an average CPU utilization rate of less than 6.25% and the worst case of 25% for serving eight physical watches simultaneously. The proposed IoT-based architecture for smartwatches seems to be feasible and scalable for context-aware ubiquitous learning applications. However, the system needs to undergo field-testing and further optimization using edge-analytics. Scalability to 100's or 1000's of watches should also be investigated because theoretically the architecture should scale.

Keywords: *Assessments, Crowdsourcing, Internet of Things, MQTT, Performance Test, QTI, Sensor Context, Smartwatch, Wearables*

Table of Contents

Abstract	6
List of Figures	10
List of Tables	15
List of Abbreviations	16
Chapter 1. Introduction	17
1.1. Overview	17
1.2. Project Objectives	17
1.3. Research Contribution	18
1.4. Project Report Organization	18
Chapter 2. Background and Literature Review.....	19
2.1. Incorporating Sensors in Assessments.....	19
2.2. Use of Wearables in Education.....	20
2.3. Sensor Context.....	22
2.4. Standards for IoT Sensor Data Definition	23
2.5. IoT Middleware Platforms	23
2.6. Educational Platforms Based on IoT and Wearables.....	25
2.7. Learning Management Systems	27
2.8. Crowdsourcing in Education.....	30
2.9. Crowdsourcing and Sensors.....	31
2.10. IMS/QTI.....	32
Chapter 3. Proposed System	36
3.1. High Level Design	36
3.2. Detailed Design.....	38
3.2.1. Message Broker	38
3.2.2. Question Bank.....	39
3.2.3. NoSQL Store.....	41
3.2.4. Teacher Portal	42
3.2.5. Middleware and the Context Processor.	42
3.2.6. Smartwatch/Client.....	44
3.3. Platform Use Case Analysis.....	46
3.3.1. GPS.	47
3.3.2. Pedometer.	47

3.3.3.	Light Intensity.....	47
3.3.4.	Ultra Violet (UV).....	48
3.3.5.	Pressure.....	48
3.3.6.	Heart Rate Monitor.....	48
3.3.7.	Other Suitable Use Cases.....	48
3.4.	Code Implementations.....	49
3.4.1.	Middleware Code-(1,3,4).....	50
3.4.2.	SGS2 Smartwatch Application Code (2,5,6,8,9).....	52
3.4.3.	The Context Processor Application Code (7).....	60
3.4.3.1	<i>MQTT code</i>	61
3.4.3.2	<i>Context processor code</i>	61
3.5.	Code Summary.....	65
3.5.1.	Tizen Smart Watch Application.....	65
3.5.2.	Middleware and Context Processor Application.....	66
3.5.3.	Teacher Portal Application.....	67
Chapter 4.	Experimental Setup.....	68
4.1.	Transactions Analysis.....	68
4.1.1.	Authentication (T0).....	69
4.1.2.	Telemetry Submission (T1).....	69
4.1.3.	Answer Posting (T2).....	70
4.2.	Platform Experimental Design.....	70
4.2.1.	Instrumentation.....	70
4.2.2.	Data Collection.....	74
4.2.2.1	<i>Response Time</i>	74
4.2.2.2	<i>Power, CPU and Memory</i>	75
4.2.2.3	<i>Bitrate</i>	76
4.3.	Experiments Definition.....	76
4.3.1.	OnDemand Mode Experiment.....	76
4.3.2.	Continuous Mode Experiment.....	77
4.3.3.	Multiple Watches Experiment.....	79
Chapter 5.	Results and Analysis.....	80
5.1.	Single Watch – OnDemand Mode Experimental Results.....	80
5.1.1.	Response Time.....	80

5.1.2.	CPU Usage, Bitrate, Power Drain and Memory Availability.....	81
5.2.	Single Watch – Continuous Mode Experiment Results.....	83
5.2.1.	Response Time.....	83
5.2.2.	CPU Usage.....	89
5.2.3.	Power Drain.....	100
5.2.4.	Memory Availability.....	101
5.2.5.	Bitrate.....	106
5.3.	Continuous Multiple Watches Experiment Results	106
5.3.1.	Server Load.....	107
5.3.2.	Response Time.....	109
5.3.3.	CPU Usage.....	113
5.3.4.	Memory Availability.....	115
5.4.	Summary of Results	115
Chapter 6.	Conclusion and Future Work	118
6.1.	Future Work.....	118
6.2.	Conclusion and Limitations	119
References	122
Appendix A:	Sample dynamic assessment objects.....	126
Appendix B:	Frequency distributions and other graphs related to response time	129
Appendix C:	Additional graphs for CPU usage over time	133
Appendix D:	Frequency distributions for CPU Max and %CPU-Second.....	134
Appendix E:	Additional memory variation graphs.....	138
Appendix F:	Additional descriptive statistics	141
Appendix G:	Additional graphs for sever load.....	150
Vita	152

List of Figures

Figure 2.1: MOSDEN architecture	24
Figure 2.2: M2Learn sample application screenshot	26
Figure 2.3: Prototype for learning Ohm’s Law	28
Figure 2.4: “Senseboard” kit	29
Figure 2.5: Sense application screenshot	29
Figure 2.6: QTIMaps screenshot	32
Figure 2.7: QuesTInSitu screenshot	33
Figure 2.8: Wonderland-QTI screenshot	34
Figure 3.1: Wearable School High Level Architecture	37
Figure 3.2: A fragment of the JSON format for an assessment item.....	40
Figure 3.3: Editing an Assessment Question in Google Classroom.....	41
Figure 3.4: Wearable device assignment in the Teacher Portal.....	43
Figure 3.5: Sample in-situ question shown on smartwatch. Screenshots from left to right, and top to bottom show the user interface cycle throughout the application.	45
Figure 3.6: Telemetry data model in the Wearable School Platform	46
Figure 3.7: Telemetry data model for the Wearable School Platform.....	49
Figure 3.8: oAuth code – Loading secret file	50
Figure 3.9: oAuth code – Creating oAuth client.....	51
Figure 3.10: Function to list courses and coursework from Google Classroom.....	51
Figure 3.11: MQTT message to HTTP GET/POST wrapper code.....	52
Figure 3.12: Adding “pagebeforeshow” event listener for any Tizen application page	53
Figure 3.13: MQTT client initialization code	53
Figure 3.14: MQTT topics subscription code	53
Figure 3.15: Sending the authentication request message	53
Figure 3.16: Receiving successful authentication message	54
Figure 3.17: Handling the post answer response	55
Figure 3.18: Handling the post answer response	55
Figure 3.19: Sensor collection mode selection handler	56
Figure 3.20: Sensor startup code.....	56
Figure 3.21: OnDemand sensor collection code	57

Figure 3.22: Sensor callback handling code	57
Figure 3.23: Telemetry submission code	58
Figure 3.24: Question content display code.....	58
Figure 3.25: Answers display code.....	59
Figure 3.26: Post answer code	60
Figure 3.27: Post answer code	60
Figure 3.28: Middleware MQTT connection startup.....	61
Figure 3.29: MQTT message handler	61
Figure 3.30: Authentication condition	61
Figure 3.31: Context Processor – First Part.....	62
Figure 3.32: Context Processor – Second Part.....	62
Figure 3.33: Context Processor – Third Part	63
Figure 3.34: Context Processor – Fourth Part.....	63
Figure 3.35: Context Processor – Fifth Part.....	64
Figure 3.36: Context Processor – Sixth Part.....	64
Figure 3.37: Context Processor – Seventh Part	65
Figure 4.1: Transaction sequence diagram	68
Figure 4.2: Test automator starting up the wearable school application automatically on the SGS2 watches.....	72
Figure 4.3: High-level diagram of automated testing environment.....	72
Figure 5.1: OnDemand average response time per network profile	81
Figure 5.2: Continuous mode line chart, 100% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval.....	83
Figure 5.3: Continuous mode line chart, 100% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval.....	84
Figure 5.4: Continuous mode bar chart, 0% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval.....	84
Figure 5.5: Continuous mode, 100% hit scenario average response time of T2.answerPosting per network profile and T1.telemetrySubmission interval.....	85
Figure 5.6: Average response time per network profile for each transaction type running in continuous mode in 80 seconds interval of T1.telemetrySubmission and 100% hit scenario	85

Figure 5.7: Box plot of response time for telemetry submission transaction for each interval and each network profile with 0% hit scenario	86
Figure 5.8: Box plot of response time for telemetry submission transaction for each interval and each network profile with 100% hit scenario	87
Figure 5.9: Box plot of response time for answer posting transaction for each interval and each network profile with 100% hit scenario	88
Figure 5.10: Average response time per network profile for T1.telemetrySubmission transaction running in continuous mode with 80-second interval of T1.telemetrySubmission and 0% hit scenario	89
Figure 5.11: Continuous mode CPU usage within the 30-minute episode run with Wi-Fi network profile and 0% hit scenario	90
Figure 5.12: Continuous mode CPU usage within the 30-minute episode run with Wi-Fi network profile and 100% hit scenario	90
Figure 5.13: Continuous mode CPU usage within the 30-minute episode run with 2G rural network profile and 0% hit scenario	91
Figure 5.14: Continuous mode CPU usage within the 30-minutes episode run with 2G rural network profile and 100% hit scenario	91
Figure 5.15: Box plot of CPU Max for telemetry submission transaction for each interval and each network profile with 100% hit scenario	92
Figure 5.16: Average bar chart of CPU Max for telemetry submission transaction for each interval and each network profile with 100% hit scenario	93
Figure 5.17: Box plot of CPU Max for telemetry submission transaction for each interval and each network profile with 0% hit scenario	94
Figure 5.18: Average CPU max bar chart for telemetry submission transaction for each interval and each network profile with 0% hit scenario	95
Figure 5.19: Box plot of %CPU-Seconds for telemetry submission transaction for each interval and each network profile with 0% hit scenario	96
Figure 5.20: Box plot of %CPU-Seconds for each interval and each network profile with 100% hit scenario	97
Figure 5.21: Average %CPU-Seconds bar chart for each interval of T1.telemetrySubmission and each network profile with 0% hit scenario	98
Figure 5.22: Average %CPU-Seconds bar chart for telemetry submission transaction for each interval and each network profile with 100% hit scenario	99
Figure 5.23: Continuous mode battery usage per network profile and T1 message frequency for 100% hit scenario	100
Figure 5.24: Continuous mode battery usage per network profile and T1 message frequency for 0% hit scenario	101
Figure 5.25: Continuous mode memory usage throughout the episode run per network profile	101

Figure 5.26: Continuous mode memory usage for each interval of T1.telemetrySubmission in 2G-Rural network with 0% hit scenario	102
Figure 5.27: Continuous mode memory usage for each interval of T1.telemetrySubmission in Wi-Fi network with 0% hit scenario	102
Figure 5.28: Continuous mode memory usage for each interval of T1.telemetrySubmission in Wi-Fi network with 100% hit scenario	103
Figure 5.29: Continuous mode memory variation for each interval of T1.telemetrySubmission in Wi-Fi network with 0% hit scenario	104
Figure 5.30: Continuous mode memory variation for each interval of T1.telemetrySubmission in Wi-Fi network with 100% hit scenario	105
Figure 5.31: Continuous mode network throughput	106
Figure 5.32: Server virtual machine specifications	107
Figure 5.33: Server Load for the baseline scenario without the context processor running	108
Figure 5.34: Average Server Load for the baseline scenario without the context processor running	108
Figure 5.35: Server Load for each network profile with 0% and 100% hit scenario and 8 watches	109
Figure 5.36: Bar chart Average CPU Load along number of watches for both 100% and 0% hit scenario for n watches (n = 1-8)	110
Figure 5.37: Average response time of T0.authentication in 100% and 0% hit scenario (n = 8)	111
Figure 5.38: Average response time of T1.telemetrySubmission in 100% and 0% hit scenario per network profile	112
Figure 5.39: Average response time of T2.answerPosting in 100% hit scenario	112
Figure 5.40: Average CPU max bar chart for each watch, each network profile and combination of 0% and 100% hit scenario	114
Figure 5.41: Average %CPU-Seconds bar chart for each watch, each network profile and combination of 0% and 100% hit scenario	115
Figure 5.42: Average available memory bar chart for each watch, each network profile and combination of 0% and 100% hit scenario	116
Figure A.1: Sample Dynamic Assessment Object: Pedometer Utilization	126
Figure A.2: Sample Dynamic Assessment Object: GPS Sensor Utilization	127
Figure A.3: Sample Dynamic Assessment Object: Light Sensor Utilization	128
Figure B.1: Frequency distribution of response time for telemetry submission transaction for each interval and each network profile with 0% hit scenario	129

Figure B.2: Frequency distribution of response time for telemetry submission transaction for each interval and each network profile with 100% hit scenario	130
Figure B.3: Frequency distribution of response time for answer posting transaction for each interval and each network profile with 100% hit scenario	131
Figure B.4: Continuous mode line chart, 0% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval.....	132
Figure C.1: Continuous mode CPU usage within the 30-minute episode run with Edge-Good network profile and 100% hit scenario	133
Figure C.2: Continuous mode CPU usage within the 30-minute episode run with Edge-Good network profile and 0% hit scenario	133
Figure D.1: Frequency distribution of CPU max for telemetry submission transaction for each interval and each network profile with 100% hit scenario.....	134
Figure D.2: Frequency distribution of CPU Max for telemetry submission transaction for each interval and each network profile with 0% hit scenario.....	135
Figure D.3: Frequency distribution of %CPU-Seconds for telemetry submission transaction for each interval and each network profile with 0% hit scenario	136
Figure D.4: Frequency distribution of %CPU-Seconds for each interval of T1.telemetrySubmission and each network profile with 100% hit scenario	137
Figure E.1: Continuous mode memory usage for each interval of T1.telemetrySubmission in Edge-Average network with 0% hit scenario	138
Figure E.2: Continuous mode memory usage for each interval of T1.telemetrySubmission in 3G-Good network with 0% hit scenario	138
Figure E.3: Continuous mode memory usage for each interval of T1.telemetrySubmission in 2G-Urban network with 100% hit scenario	139
Figure E.4: Continuous mode memory usage for each interval of T1.telemetrySubmission in Edge-Average network with 100% hit scenario	139
Figure E.5: Continuous mode memory usage for each interval of T1.telemetrySubmission in 3G-Good network with 100% hit scenario	140
Figure G.1: Server Load for each network profile with 0% and 100% hit scenario and 1 watch	150
Figure G.2: Server Load for each network profile with 0% and 100% hit scenario and 2 watches	150
Figure G.3: Server Load for each network profile with 0% and 100% hit scenario and 4 watches	151

List of Tables

Table 3.1: Description of JavaScript Libraries loaded in index.html.....	52
Table 3.2: Description and lines of code for the Tizen smart watch application source code files	66
Table 3.3: Description and lines of code for the middleware and context processor application source code files	66
Table 3.4: Description and lines of code for the teacher portal source code files	67
Table 4.1: Samsung Gear S2 Specifications	71
Table 4.2: Independent Variables.....	76
Table 4.3: Network Profiles	76
Table 4.4: Control Variables	77
Table 4.5: Dependent Variables	77
Table 4.6: Estimated T1.telemetrySubmission message size for each interval	78
Table 4.7: Episodes run for the multiple watches experiment	79
Table 5.1: OnDemand descriptive statistics for the T0.authentication transaction.....	81
Table 5.2: OnDemand descriptive statistics for the T0.telemetrySubmission transaction	82
Table 5.3: OnDemand descriptive statistics for the T2.answerPosting transaction	82
Table 5.4: Summarized continuous mode descriptive statistics for T0.authentication transaction	113
Table 5.5: Summarized continuous mode descriptive statistics for T1.telemetrySubmission	113
Table 5.6: Summarized continuous mode descriptive statistics for T2.answerPosting	113
Table 5.7: Continuous mode descriptive statistics for T2.answerPosting for each watch.....	114
Table F.1: Continuous descriptive statistics for the T1.telemetrySubmission transaction	141
Table F.2: Continuous descriptive statistics for the T2.answerPosting transaction..	143
Table F.3: Continuous descriptive statistics for max CPU results.....	144
Table F.4: Continuous descriptive statistics for %CPU-Seconds results.....	146
Table F.5: Continuous mode descriptive statistics for T1.telemetrySubmission for each watch.....	148

List of Abbreviations

IoT	Internet of Things
MQTT	Message Queue Telemetry Transport
SGS2	Samsung Gear S2
MEAN	MongoDB, ExpressJS, AngularJS, NodeJS
IMS-QTI	Instructional Management Systems Question and Test Interoperability

Chapter 1. Introduction

This chapter provides a short introduction to the Wearable School and the motivations for creating the platform. The contributions of the project to the educational assessment process are then presented, along with the platform's unique context-based operation. Finally, the general organization of the project is presented.

1.1. Overview

Internet technology has evolved dramatically over the last decade, during which the web has been transformed from a repository of static information to a platform for user contribution. Integration of ubiquitous connected devices linking the physical world to the web continues to evolve at a rapid pace, and the iconic term of Internet of Things (IoT) has been coined to describe the manner in which devices are contributing to the web through sensor readings and derived sensor context. This web evolution is beginning to play an important role in education through the introduction new techniques that bind the physical world surrounding students to their learning material. Students who perceive raw information without applying their knowledge through practice tend to quickly forget what they learn, but research has shown that providing students with the opportunity to experiment with the topics they are learning helps them to retain the information for longer periods, making the knowledge gained a more lasting experience through practice.

1.2. Project Objectives

To achieve the goal of binding physical context to educational material, it is necessary to enrich available online learning platforms by embedding sensor context retrieved from devices into the educational content offered in order to encourage students to interact with the surrounding environment while perceiving new information. This goal provides the focus of this report, which proposes a platform that incorporates wearable smartwatches into the online educational process, allowing for direct utilization of smartwatch sensor data in educational assessments. Students using this platform will be able to receive assessments specifically targeted to the context they are currently in, which will help them to better grasp material they are presented by relating it directly to features of the classroom where they interact with the physical world. Use of this type of educational platform will allow students to

continue to enhance their learning after class or during fieldtrips and can result in integrating the learning process into students' everyday living experience, so that they can engage with the material they are learning wherever they go and whenever time is available. More specifically, the style of assessments in the proposed platform will provide greater enhancement of students' knowledge and retention of concepts when compared to traditional assessments. Improved retention has been shown in earlier research, which will be presented in the literature review section.

1.3. Research Contribution

The contributions of this research can be summarized as follows:

- A new platform for embedding sensor context from wearable devices into educational assessments is proposed.
- The implementation of the platform is presented and feasible sample use cases are demonstrated.
- The efficiency and performance factors for the decisions guiding the design of the proposed platform are shown by performing experiments and analyzing the experimental results.

1.4. Project Report Organization

This report first surveys related research from different fields, including studies related to the Internet of Things (IoT), education, wearables and middleware platforms. Chapter 3 describes the solution approach and the architecture for the Wearable School educational platform as well as real-world use cases, including a range of sample use cases utilizing each currently available smartwatch sensor. Chapter 4 assesses the design decisions that were made for this architecture and the choice to use the Message Queue Telemetry Transport (MQTT) protocol throughout the platform to exchange messages. Experiments assessing the feasibility, performance and resource use of the platform are performed. The final chapter discusses the results of the experiments performed on the platform and evaluates platform performance under different network conditions that simulate real use with students roaming around and receiving questions, either continuously or on demand.

Chapter 2. Background and Literature Review

This section surveys a variety of research that supports the design of the proposed platform. The literature review is divided into ten topics, including incorporating sensors in assessments, the use of wearables in education, sensor context, standards for IoT data definition, IoT middleware platforms, educational platforms based on IoT and wearables, learning management systems, crowdsourcing in education, crowdsourcing and sensors, and Instructional Management Systems Question and Test Interoperability (IMS-QTI) specifications. A summary of relevant research findings pertaining to each topic is presented.

2.1. Incorporating Sensors in Assessments

The Wearable School platform architecture was described in a paper by Al-Solh & Zualkernan [1]. This paper describes the platform architecture in greater detail. The user interface for the platform described in the earlier paper did not follow the recommended Samsung design guidelines, and some of the decisions that had been left open were clarified during the course of this project. This research is a continuation of the previously described architecture and also includes a performance analysis providing clearer motivation for the design decisions applied to the Wearable School platform overall.

With respect to incorporating sensors into assessments, Zualkernan et al. [2] have introduced modifications to a wiki-based assessment platform in order to incorporate sensor readings into the wiki developed by Qutaifan & Zualkernan [3].

This assessment wiki was a modified version of Wikipedia designed to incorporate an editorial assessment workflow. The modifications involved multiple sensors connected to an Arduino board that sends the telemetry to a Raspberry Pi board. The readings are then transmitted through a message-oriented middleware called Openfire, which implements the XMPP protocol. The outcome of this implementation is that contributors to the assessment wiki can write questions that include sensor readings from the physical world, and users can answer questions that have dynamically changing answers based on the sensor readings. This implementation had a number of limitations where students had to log into the assessment wiki system to perform evaluations, as no other means of mobile-friendly

student user interface was available and no data analysis could be performed on their responses because the answers were not saved to a repository. The implementation was a proof of concept, and a similar concept is being implemented in this research with the addition of wearables.

2.2. Use of Wearables in Education

The introduction of wearables in education and their potential advantages have been discussed by Bower and Sturman [4]. The motivation for their research is the growth of wearable devices, of which 427 are now included in the Vandrico database [5]. These researchers have defined wearables as “[w]earable digital devices that incorporate wireless connectivity for the purposes of seamlessly accessing, interacting with and exchanging contextually relevant information.” [4, p.344], they surveyed 66 experienced educators knowledgeable in wearable technologies to extract qualities that they defined as the “affordances” – or potential advantages – of wearable technologies. The affordances observed were divided into three categories. The first category related to pedagogical uses, in which wearables provide a number of affordances, such as in-situ contextual information, which students can use to trigger events based on a location or situations similar to the scenarios presented in this paper. Wearables also allow users to record information, such as audio notes or event simulations in the case of wearable displays. Another pedagogical affordance is communication, which describes advantages such as students being able to send instant messages using smartwatches, medical students being able to access a first-person view of an operation through a head-mounted virtual reality display, or students receiving feedback while attending a lecture by answering questions sent via messages without interrupting the lecture. Additional communication affordances include students receiving in-situ guidance through feedback they receive while performing activities in class and distribution by annotation of notes over live scenes. Other pedagogical affordances exist that have been described in the research literature but were not collected during analysis of the survey output in this project such as gamification, in which students can be assigned competitive tasks to improve user engagement, like answering questions while competing for medals and ranks. Bower and Sturman’s second category, quality of education, included affordances such as increased efficiency through faster access to information, increased engagement and enhanced modes of non-physical presence. Their third category, logistical

affordances, included issues such as hands-free access, which enables users to perform activities and receive information at the same time. Another logistical affordance was freeing up space constraints so that students could perform their learning activities without the need to be at their desk. The paper also indicated a number of negative aspects associated with wearable technologies, such as cheating and privacy issues. Ultimately, however, it is up to the teaching community using these devices to realize and utilize the affordances they consider valuable for improving education.

In terms of applications of wearables in education, Labus et al. [6] surveyed different wearable technologies and their possible uses in Learning Management Systems (LMSs). This paper explored the use of smartwatches that allow educators to utilize time functionality to push reminders to students about important due dates. Smartglasses can also be used as a learning aid for students with disabilities. For instance, these devices can be used for displaying text for hearing impaired students to read or for zooming into content to aid students with poor eyesight. The paper also highlighted the educational uses of the “sixth sense wearable device” by Mistry and Maes [7], which consists of a projector, a camera and motion-sensing equipment to project content onto physical objects. This device also allows for live annotation during lessons.

Romero et al. [8] performed experiments using Massive Open Online Courses (MOOCs) in which students used an Android Wear application developed to push notifications to their smartwatches, which could be triggered by an announcement interface developed using Google course builder’s APIs. These APIs have evolved into Google Classroom, the learning platform used in our research. The researchers also used Google Cloud Messaging and Google Group technologies in their Android Wear application. A sample application developed in this research was tested on two student groups: one group had the application installed on their smartwatches while the other only received email notifications using conventional PCs or mobile phones. The results of the study showed that the proportion of students completing their assessments on time was significantly higher for students with smartwatches than for the students receiving emails only. A survey conducted at the end of the study showed

that students taking the online course who used the application developed to send push notifications to their smartwatches reported greater satisfaction.

2.3. Sensor Context

Multiple sensors are available in mobile devices and wearables. These sensors retrieve vast amounts of data that can be used to produce graphs and trends, but what about extracting information related to the physical world around students? Gellersen et al. [9] discussed multiple projects and were among the first to explore the idea of context-related sensor data. In general, context can be extracted from sensors to understand conditions in the physical world. One example of context realization discussed in the paper occurs when a person is wearing a smartwatch and the heart rate sensor is reporting average heart rate readings. This indicates the person is in stationary position. Another context example is the ability of an observer to deduce whether a person is walking, running or riding a bike based on heart rate readings and GPS data.

In their survey of context awareness, Abowd et al. [10] provided one of the best available definitions of context: “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.” Perera et al. [11] have categorized many schemes of context, including whether the context is considered primary or secondary. A primary context refers to reading the sensor data and extracting information directly from the sensor while a secondary context refers to a situation where the information extracted is used to derive another level of information, such as retrieving the id of an object from its RFID and deriving the characteristics of the object by reading from a database with respect to its ID. Another scheme is the context life cycle, which is divided into four phases: context acquisition, modeling, reasoning and dissemination. Acquisition can be triggered by software (pull) or sensor hardware (push). Both push and pull can occur either instantly with a specific decision, or in timed intervals. Also, acquisition can occur directly from the sensor, through a context server or through middleware. The sensors used during acquisition can be physical, virtual (e.g., Twitter status) or logical (e.g., an API representing thousands of sensors used for weather forecasting). In the modeling phase, context can be modeled in simple key-value

pairs, structured XML, relational tables, ontologies with specific domain scopes or non-relational JSON. The reasoning phase consists of three steps: data pre-processing (e.g., removing outliers), sensor data fusion to combine readings from multiple sensors, and context inference, which occurs through generating information from the data collected (e.g., deriving location names from GPS coordinates). Reasoning can be performed using different models. Reasoning can be based on if/else rules, which is the most popular method. Supervised learning such as decision trees, which are used in activity recognition. Unsupervised learning, which is used to detect abnormal events, and fuzzy logic, which is used to derive higher level information from very near states such as how the weather feels. The last phase is context distribution, which can be performed via request (specific queries that are initiated by the consumer of a service) or publish subscribe models (by users subscribing to a specific topic such as weather changes in London).

2.4. Standards for IoT Sensor Data Definition

Compton et al. [12] have summarized the capabilities of sensor networks. Sensors can be classified according to their function, output, or measurement method. Sensor networks can help locate sensors that perform specific measurements; aggregate data spatially, temporally, or according to its accuracy, infer domain knowledge from low-level data, or produce an event based on a specific condition. Based on their research, SensorML is a markup or modeling language that helps define these capabilities in a standard format, but there is a need for a technology, normally referred to as an ontology, to help extract these capabilities from the sensor hardware. These researchers have explored a number of ontologies, including one built by the W3C Semantic Sensor Networks Incubator Group (SSN-XG), whose goal to build a general ontology for sensors, which was later was called the Semantic Sensor Network (SSN) Ontology [13]. The SSN ontology covers most of the standards that were created by the Open Geospatial Consortium (OGC), the group that defined the SensorML standard.

2.5. IoT Middleware Platforms

Sensors are usually resource limited devices. Connecting sensors to the Internet can be expensive in terms of network infrastructure and power consumption, and there is a need for a mediator platform that connects all IoT sensors to the Internet

or one that will process sensor data before submitting its information to the requester over the Internet. Perera et al. [14] have proposed Mobile Sensor Data Processing Engine (MOSDEN), an open source IoT platform built on top of GSN middleware, unlike commercial products like Xively, a product initiated by Pachube then acquired by LogMeIn. MOSDEN is designed to be open source and not limited to proprietary hardware or software. The MOSDEN platform can query sensors in SQL-like queries such as “select * from wrapper”. MOSDEN architecture, shown in Figure 2.1, consists of sensors (S) connected to plugins (P) developed by the community to read specific sensors. In this system the plugin readings are processed by wrappers that convert the readings to virtual sensing devices. These virtual sensors are then managed using open source GSN middleware. MOSDEN uses the SSN ontology data model to store context information about the sensors. One of the main concepts that MOSDEN uses in its edge analytics is that whenever sending information requires more power than processing it, MOSDEN will first process the data before sending it over the network.

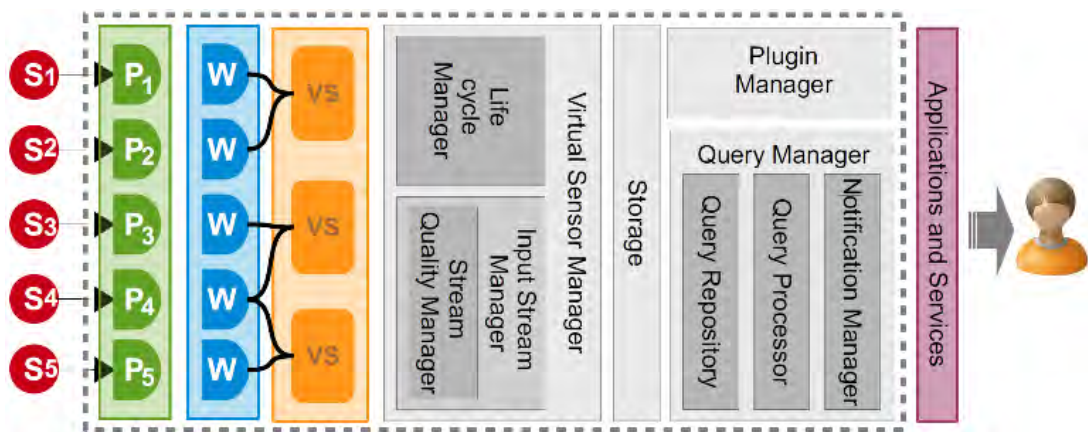


Figure 2.1: MOSDEN architecture [14]

One of the most important IoT messaging platforms used in the solution proposed in this paper is PONTE [15], which allows MQTT, COAP and HTTP communication in a single server instance running on top of Node.JS. Specific to MQTT communication, PONTE works on a publish subscribe pattern, where messages in MQTT can be filtered based on three criteria: subject-based filtering, which is mostly used to listen to messages based on a topic; content-based filtering, which filters messages based on a specific query inside the content (but to support this

feature, the content cannot be encrypted); and type-based filtering, in which messages can be subscribed based on the type of content (such as log of level “info”).

2.6. Educational Platforms Based on IoT and Wearables

Wang [16] has proposed a model for using IoT in English language instruction. In his model, both video and audio sensors can be utilized to monitor a learner’s speech in order to identify gaps in pronunciation and guide students to correct them. The video sensor monitors the shape of the student’s mouth while pronouncing English words, and the audio sensor monitors the frequency, tone, amplitude and speed of the student’s voice. These parameters can be used to compare students against fluent English speakers and to deduce the learner’s score.

Martin et al. [17] have proposed a framework for writing mobile learning applications. Their framework, M2Learn, utilizes well-established web 2.0 tools for blogging, wikis and e-portfolios, Moodle services, and sensors in mobile devices to construct a framework that provides APIs in which developers can build mobile applications while hiding the complexity behind the integration with backend services or devices. This allows developers to concentrate on the value of their application rather than duplicating integration efforts. Moreover, the framework enables developers to build more sophisticated applications. The M2Learn framework was developed with three goals in mind. The first was to provide an interface for eLearning platforms. The choice of platform for M2Learn was Moodle. The second goal was to be able to collect information about the learner’s e-portfolio from external platforms such as blogs. The third goal was to integrate different sensors such as GPS, RFID, and accelerometers transparently through the framework. The framework also supports content and service discovery based on spatial-temporal inputs and conforms to standards in eLearning, such as Learning Object Metadata (LOM) and IMS-QTI. To prove the framework’s feasibility, an experiment was performed in which students wrote a micro blogging application using the framework to submit short messages that appeared on mobile devices in a Twitter-like fashion. The framework provides APIs for location and profile services that add the student user tag and location to each post. M2Learn is also designed to collect the data entered and store it in Moodle in a contextualized manner so the teacher can use this data in a debriefing session. Figure 2.2 shows a screenshot of the sample application.



Figure 2.2: M2Learn sample application screenshot [17]

Hou et al. [18] have developed three iterations of a system called JAMIOLAS that helps students learn Japanese through mimicry and onomatopoeia (MIO) words. JAMIOLAS provides students with the means to learn MOI words through questions generated automatically based on the available context surrounding the student. The first iteration, JAMIOLAS 1.0, was developed using wearable sensors that are connected to a laptop. In response to the inconvenience of carrying the system all the time, JAMIOLAS 2.0 was developed utilizing wireless sensors but had limited support for representing words. The researchers have proposed JAMIOLAS 3.0, which has three functions to generate suitable assessments and materials to learn MOI words: weather information related to the student's position, which is read from real-time online weather information; playable media files; and free mode learning. In free mode learning, students can choose a word they wish to learn about and related examples or media will be presented to them. A teacher's interface is available to evaluate generated weather questions or manage words and their respective media/examples. The system was evaluated using a questionnaire to compare the performance of students who used the system to answer questions to students who

used only a dictionary. The result showed significant effectiveness of JAMIOLAS. Students who took part in the study and were invited to provide feedback for system improvements suggested that the system should also present an explanation of why an answer was correct in order to help students to better understand the words. The author proposed future work using Android smartphone sensors to generate more questions for MOI words. Sensors in Android available through the API include accelerometer, gyroscope, light, magnetic field, orientation, pressure, proximity, and temperature sensors. The authors also proposed adding biosensors to generate questions related to emotion.

2.7. Learning Management Systems

In schools and universities, it is becoming the norm to utilize Learning Management Systems (LMSs) and Course Management Systems (CMSs). A current disadvantage of CMSs and LMSs is that the learning institutes tend to keep their content closed. New concepts, such as the Merlot Project [19], have arisen to overcome this by introducing Learning Objects (LO) – instructional components that are reusable and scalable – and Open Educational Resources (OERs), such as MIT OpenCourseWare [20], which offer access to open course materials. Watson et al. [21] have proposed a new learning platform that they call a Personalized Integrated Educational System (PIES). PIES is a conceptual framework for creating educational software to support the learning process that attracts learners to enhance their skills based on their actual skill levels rather than grouping students by age. It combines the benefits of a CMS, LMS and a Personal Learning Environment (PLE) – a web 2.0 concept that revolves around user created materials. When compared to PLEs, the unique feature of PIES is that it supports management of the learning process by involving teachers and parents while maintaining personalized learning experiences for students. PIES has four primary functions: record keeping, planning, instruction, and assessments for student learning plans that demonstrate students have mastered specific skills. Assessments can be performed within PIES, or results can be input directly by the instructor. PIES is a promising concept that has yet to be practically implemented. In our research, we used the Google Classroom platform as the LMS due to its broad acceptance and widespread use in teaching institutes.

Vujovic and Maksimovic [22] have proposed a prototype for learning Ohm’s law, shown in Figure 2.3, that helps engineering students who study online and don’t have access to laboratories to perform experiments on a Raspberry Pi device. The Raspberry Pi device can be connected to a simple circuit that uses a variable power source. Teachers can control the current through the Raspberry Pi over the Internet, and students can read the current and voltage by connecting the circuit back to the Raspberry Pi. This is only one example of the type of exercises that can be performed; many others can be performed using the same system.

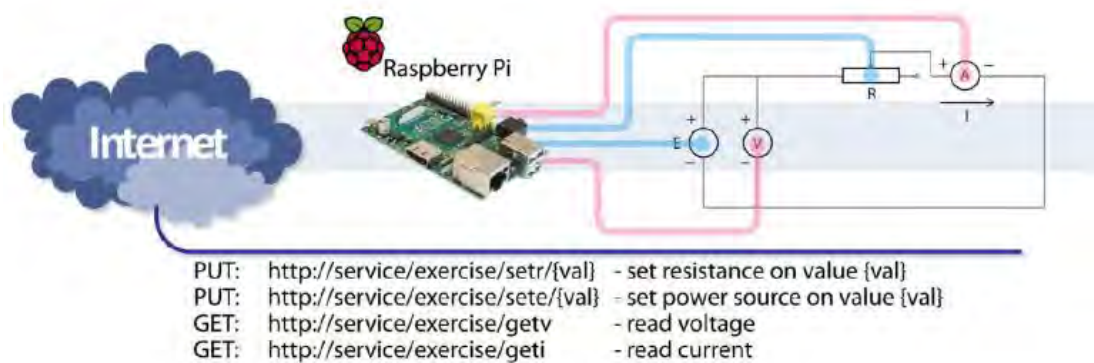


Figure 2.3: Prototype for learning Ohm’s Law [22]

Kortuem et al. have developed an online course called “My Digital Life” for The Open University [23]. These researchers have designed an easy-to-use, programmable custom board with plug and play sensors. Their goal is to make the course available to first-year undergraduate students. Students in the study received a “Senseboard” kit containing a thermistor, phototransistor, motion detector and a stepper motor, as shown in Figure 2.4. The students were able to program the device in a forked version of the scratch language, which is used to teach programming to students. The forked version was called “Sense” which is shown in Figure 2.5. The Sense language was empowered with features for embedded device programming. It allowed students to build blocks of if/else and control statements visually by dragging and dropping control statements, objects and variables into a script editor panel, which then showed the inputs and outputs graphically.



Figure 2.4: “Senseboard” kit [23]

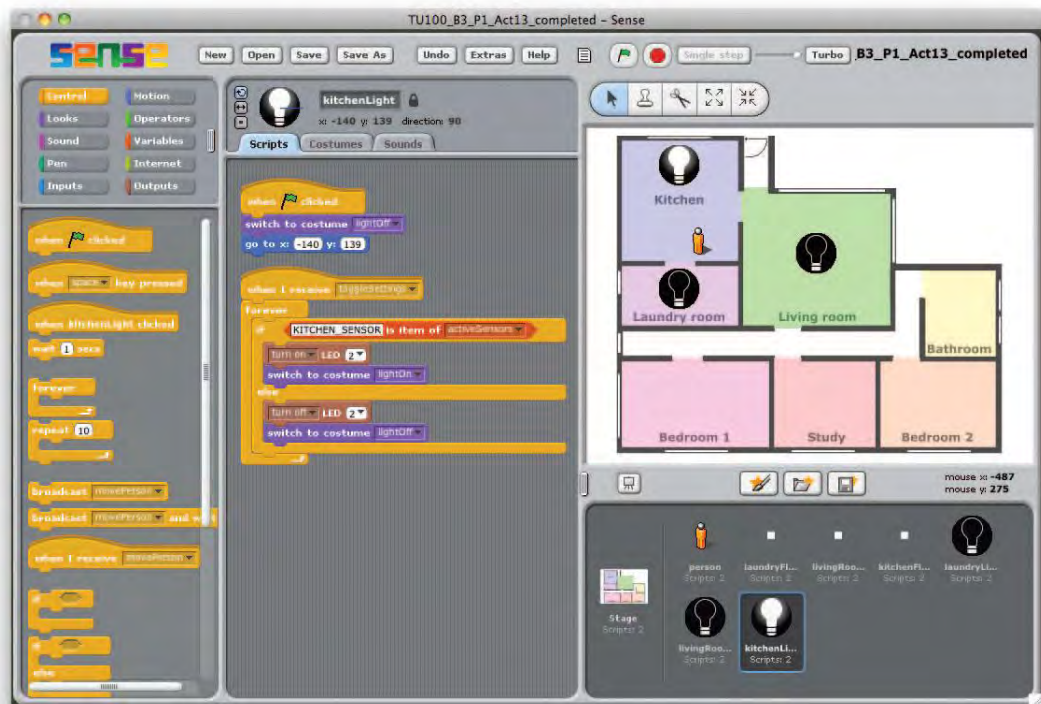


Figure 2.5: Sense application screenshot [23]

What made this platform unique is that it utilized a cloud infrastructure, allowing students to share inputs and outputs from sensors over the cloud.

2.8. Crowdsourcing in Education

Massive online open courses (MOOCs) have witnessed improvements in user adaptation. As of 2014, there were 16.8 million participants in courses offered by the top five MOOC providers. Porcello and Hsi [24] have discussed four ways of improving the quality of online educational resources through crowdsourcing: crowdsourcing metadata to classify online educational resources, defining criteria for quality of the online educational resources, relying on community input to rate content through rating and views of content, and interoperability of content by defining standards for content metadata to allow users to search across multiple content with their metadata. Developing these four aspects will certainly improve the quality of online educational content for users and allow them to choose the best resources for their needs. Porcello and His's findings can be used to improve the current research by extending the Wearable School platform to utilize rating mechanisms and define metadata over the assessments to increase their quality through crowdsourcing.

There are also other crowdsourcing possibilities that could be utilized in educational applications. Weld et al. [25] provide a number of examples, including grading papers using peer assessment, and analyzing which forum posts in Coursera's blog lead to better homework submissions when used as assigned reading. Another example is "translator", a mobile phone application that records students attempting to speak a foreign language so that tutors can give feedback on playback of the recordings. The well-known developer reference Stack Overflow provides virtual rewards for answers and comments related questions that drive the crowd to contribute more. The paper also discussed the challenge of having quality material that produces a flow of knowledge throughout the material instead of having assessments that contain disjointed questions. The paper also suggested that further research could include personalizing textbooks, videos, posts, and problems using machine learning techniques rather than manual tagging of data, and development of workflows for grading textual answers in assessments through crowdsourcing.

2.9. Crowdsourcing and Sensors

Guo et al. [26] have defined Mobile CrowdSensing as “a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices [and] aggregates and fuses the data in the cloud for crowd intelligence extraction and people-centric service delivery.” Mobile CrowdSensing is based on the concept of collecting knowledge extracted from sensor data in mobile devices, wearables, smart cars, and other smart devices. One of the key features of Mobile CrowdSensing is an evolution of participatory sensing that enables collection of sensor data through two methods. The first is classical explicit user participation. The second is the collection of data implicitly by reading sensor data from the devices. Another feature is collecting data through mobile social networks such as location-based social networks (e.g., Foursquare meetups), which can help researchers to understand the city and social behavior. The author suggested a framework for mobile crowdsensing, which consists of three layers. The first is a crowdsensing layer that collects data from sensors and social networks, which is also concerned with access control for data sharing. The crowdsensing layer utilizes a data transmission layer, which is concerned with the medium of sending data, whether it is infrastructure based (GSM,3G) or peer-to-peer (Bluetooth, Wi-Fi). The crowdsensing layer also utilizes a data collection infrastructure layer that provides anonymity services over the data collected for privacy reasons. The second layer is the crowd data processing layer, which uses machine learning techniques to extract knowledge from low-level data. Finally, the system’s application layer provides user interfaces and visualization to support services such as environmental monitoring and public safety. This paper also proposes utilizing hybrid human/machine processing throughout the layers to extract knowledge more effectively. One of the scenarios presented by the researchers is that a system could be used for the division and assignment of data contribution tasks to humans, such as location-based services in which humans can categorize a place as a café or restaurant. Guo et al.’s findings provide best practices that can be followed in the implementation of the crowdsourcing platform in cases where it is desirable to use crowdsourced sensor data in addition to data collected utilizing the sensors in the students’ smartwatches when performing assessments of practical applications beyond the assessment itself.

One example might be utilizing the GPS sensor data for studying human daily movement in a certain area for urban planning purposes.

2.10. IMS/QTI

As per the specifications of QTI 2.2 [27], the main element in the model is AssessmentTest, which consists of a number of testParts. Each testPart contains sections of tests containing assessmentItems, which are the questions for given tests used for processing the test and providing feedback. The two main elements are outcomeProcessing, which calculates the score and testFeedback, which provides feedback to the user for submitted answers. Santos et al. [28] have identified that the QTI 2.1 specification lacks important elements to make the tests more interactive. Their paper proposed an element called “interaction context”, which contains information about the visual representation of the test, as an addition to the assessmentItem body, as well as another element called “user Action”, which can support additional input methods for answering tests added to the response part of QTI specification. The paper also proposed that additional validation rules should be added to the response processing part of the QTI specification to support processing the answers to the interactive tests. One interesting scenario explored in this paper was QTIMaps, which utilizes the Google Maps APIs to present a map to the student so that the student can draw on the map to specify the answer to a question, such as marking the borders of a specific state. This technique can be used to enhance and reinforce memorization while the student is answering the test because the student is interacting with the map during the exercise, as shown in Figure 2.6.

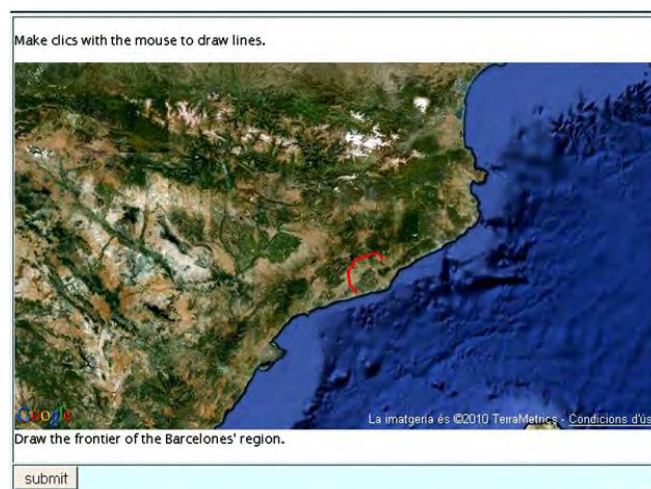


Figure 2.6: QTIMaps screenshot [28]

Another proposed scenario was QuesTInSitu, where questions can optionally have a context pre-condition for taking the test. For example, students might be required to have a mobile device with GPS support to answer the question and be in a specific location at the start of the test. An example test was discovering Barcelona, in which students had to follow a specific route and questions about the area were posed to the student at different locations. This technique has been specifically used in the current research. Figure 2.7 shows a student answering the question based on her location



Figure 2.7: QuesTInSitu screenshot [28]

Wonderland-QTI is a scenario in which the question representation can be presented in a virtual world and the student can answer questions based on his interactions with the actors inside it. This world can be a virtual museum, for example, in which the student answers questions about specific artifacts that he approaches, as shown in Figure 2.8.

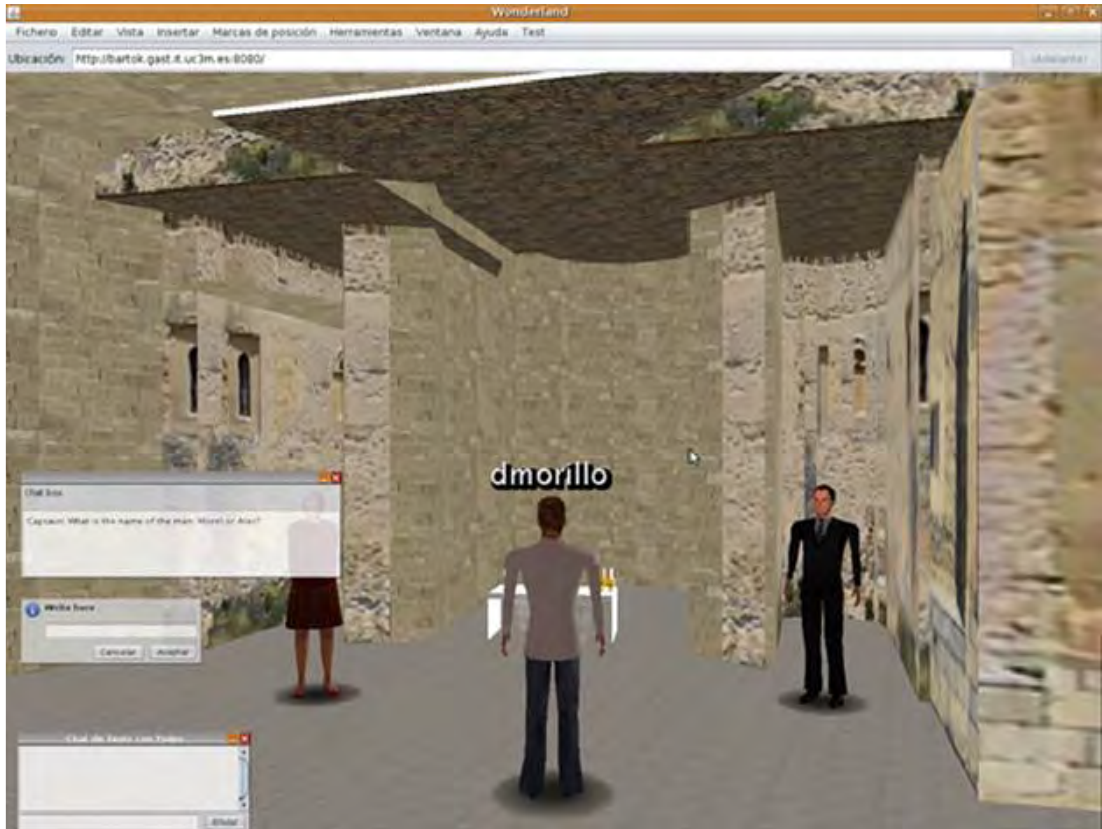


Figure 2.8: Wonderland-QTI screenshot [28]

K. Gramatova et al. [29] proposed a testing solution that can be integrated into eLearning environments. The eTesting solution consists of learning objects following the Sharable Content Object Reference Model (SCORM) standard that helps manage personalized learning resources. The testing solution also offers some educational games and provides the materials to be learned through interactive games. The core of the eTesting solution is the eTesting provisioning process that distributes assessments using the QTI 2.1 standard. The eTesting solution also makes use of intelligent agents that are able to suggest plans for tutors. The agents follow the Belief-Desire-Intent (BDI) architecture and agents are integrated with IoT sensors to determine the context surrounding students. These agents are designed using the JADEX engine, a reasoning engine that uses XML and Java to produce intelligent software. The software determines the assessments provided to students based on the data collected from SCORM and QTI test outcomes through a module called the Assessment Provisioning Module (APM). One of the problems discussed is the data exchange mechanism required to source data from external systems connecting to the eTesting solution. Their proposed solution was to expose the external systems with a REST

API for maximum compatibility. Utilizing QTI, the student is allowed multiple attempts to answer a question. Through the attempts the student can get hints from the feedback part provided by the QTI standard. The user interface component of the eTesting platform has different roles: one is for authors to create assessments, another is for learners to follow the eTesting process, and a general delivery user interface role provides results of the eTesting process.

Having surveyed literature relevant to the current research, the wearable assessment platform and a description of the proposed system architecture for building it on a high level will be presented in detail in the next chapter.

Chapter 3. Proposed System

In this section, the high level and detailed design of the proposed system is presented, also parts of the code implementations are described.

3.1. High Level Design

A detailed overview of the Wearable School platform implementation is presented in Figure 3.1. Following the flow of information through the system from right to left, a student wears a smartwatch that publishes telemetry data that includes readings from a variety of sensors to a message broker PONTE. On the receiving end, a middleware and context processing service running on a node.js instance is subscribed to all of the smartwatch telemetry data. The context processing service within the middleware then processes the data received from topic subscriptions and matches it with a suitable question from the list of questions loaded from the Google Classroom platform by matching the sensor data received with the questions that meet the requirements configured in the context eligibility criteria within the question JSON object.

The resulting in-situ matched questions are published back to the smartwatches through PONTE where the smartwatches are subscribed to topic ids that identify the watch that sent the telemetry data. The smartwatches receive the questions and display them to the student. The student can then select one of the matched questions, read the question on his smartwatch, then navigate to the answer screens and choose the correct answer. Answers are published to the middleware so that they can be saved in a nosql store, which is CouchDB [30]. In the proposed system, ideally the questions would be saved within Google Classroom itself, but Google Classroom APIs have a limitation that will be addressed in the future so that students will be able to submit their answers directly to it. The students can also request questions from their smartwatches in a continuous mode so that they are notified when a specific context is triggered over the smartwatch, such as a specific GPS location eligibility criteria being met.

The platform can also be extended to provide a custom assessment authoring portal where teachers or crowds feed in questions, and the portal is open to the crowd to feed and trigger questions based on a specific context or group of contexts.

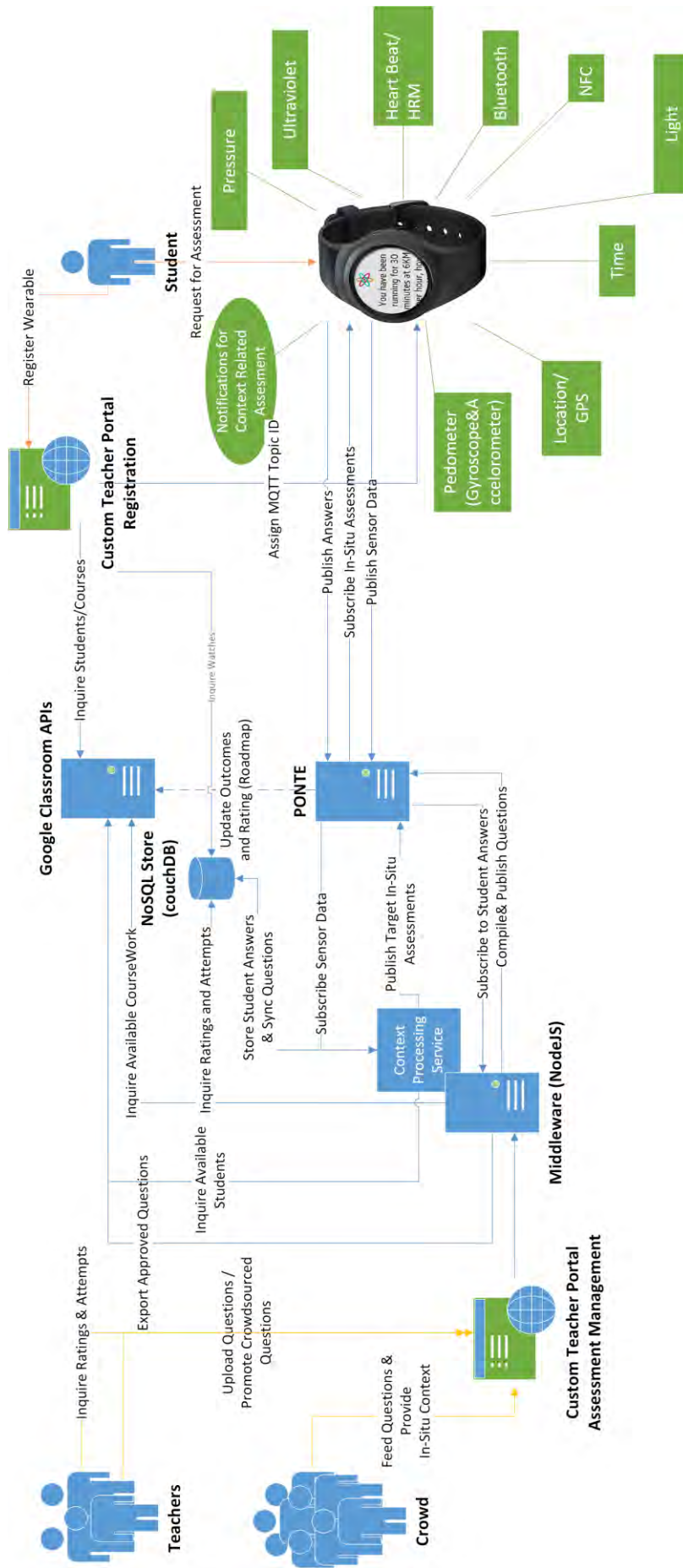


Figure 3.1: Wearable School High Level Architecture

Authors can choose which of the smartwatch sensors to incorporate with the questions and answers along with the context eligibility setup that triggers the question to be presented to the student. Once teachers have determined that the questions they are using have achieved an acceptable level of quality, there is an option to have them exported to the Google Classroom question bank through a middleware application running on nodejs.

3.2. Detailed Design

In this section, the design of the Wearable School will be presented in detail to demonstrate how each of the platform's components works. The design decisions for each component will also be explained.

3.2.1. Message Broker. A publish/subscribe mechanism was chosen for communication because it was better than synchronous request/response mechanisms (e.g., using Ajax requests) and because this type of mechanism allows for real-time events rather than polling to check the availability of new data from the context processor within fixed time intervals. One of the requirements for this system was a message broker that connected the smartwatch and the context processor to a publish/subscribe mechanism. A further requirement for the message broker was that it should have minimal impact on network bandwidth and power consumption, so MQTT was the only method used for communication throughout the platform. PONTE was found to be the best fit for our requirements since it has support for MQTT, COAP and HTTP. PONTE was used because the smartwatches in this study only support MQTT over web sockets that run on port 3000, and the context processor runs on native MQTT communication over port 1883. Using PONTE, each smartwatch can be registered with a topic that can be subscribed to, such as telemetry/smartwatch1.

To uniquely identify a smartwatch we chose the Tizen ID as the unique identifier. This identifier can be easily retrieved using the Samsung system APIs so that the resulting topic id will be telemetry/tizenId. MQTT supports different QOS levels: QOS Level 1 means the message should be sent at least once, QOS 0 means the message is sent at most once and QOS level 2 means the message is sent exactly once. Since the system exchanges telemetry data, level 0 was used because losing a message won't have a major effect on the systems and the message can be sent again

for telemetry with new data. For triggering assessments QOS level 2 is the best choice to ensure that the question was triggered to the student [31]

3.2.2. Question Bank. Because it was desirable to have a learning management system to manage student enrollment and assessment, the Google Classroom platform was chosen because many schools have adopted it and students are already familiar with the Google ecosystem (which includes Google Docs, Gmail, Google Calendar, etc.), a set of applications that can also be used on the Google Classroom platform. Google Classroom exposes a RESTful API that authenticates using OAuth 2.0 [32]. An application key was generated that can be used for both the context processor and the teacher portal. The API allowed CRUD operations to be performed on coursework that contains the assessments, but as mentioned earlier, the interface still has a limitation in that it does not allow users to programmatically submit answers. The Wearable School platform can be extended to use Google platform APIs to upload assessments straight from the teacher's portal. Google APIs are used only to retrieve the available assessments from the context processor at the moment. APIs are also used to list the available courses that a student is participating in so that the student will only receive questions related to his courses once the student's smartphone has been registered through the teacher portal. The Google Classroom UI shown in Figure 3.3 is also being used to create questions and view student enrollments. The question description field was used to store the JSON object that represents the question similar to IMS QTI 2.2 format [27] encoded in JSON. The question object contains the question body and a context eligibility function that matches a question with a specific in-situ context to make the question eligible to be sent to the watch, and the answers are evaluated in real time with dynamic content using the smartwatch telemetry data. This dynamic question and answer mechanism was inspired by the previously mentioned work on IoT Assessments WIKI [2].

Figure 3.2 shows an excerpt of the question JSON object that is stored on the Google Classroom platform.

```

{
  "courseWork": [
    {
      "courseId": "1017124722",
      "id": "5953763443",
      "title": "Time to Eiffel Tower",
      "description": {
        "assessmentItem": {
          .....
          "context": {
            "sensorNames": [ "GPS" ],
            "eligibility": "((<GPS.longitude> > -1000) && (<GPS.longitude> < 1000))"
          }
        },
        "responseDeclaration": {
          .....
          "correctResponse": "<var0>/500.toFixed(2)"
        },
        .....
        "itemBody": {
          "p": "if you are in a plane flying at the speed of 500 Km per hour and the Eiffel
tower is currently <var0> Km away from you.",
          "variables": [
            "getDistanceFromLonLatInKm(<GPS.longitude>,<GPS.latitude>,2.2922926,48.8583736
)"
          ],
          "choiceInteraction": {
            .....
            "prompt": "how long will it take to reach there in hours time?"
          }
        },
        .....
        "multipleChoiceQuestion": {
          "choices": [
            "<var0>/500.toFixed(2)",
            "<var0>/25.toFixed(2)",
            "<var0>/750.toFixed(2)",
            "<var0>/100.toFixed(2)"
          ]
        },
        "assigneeMode": "ALL_STUDENTS",
        "creatorUserId": "104076855707702137295"
      }
    }
  ]
}

```

Figure 3.2: A fragment of the JSON format for an assessment item

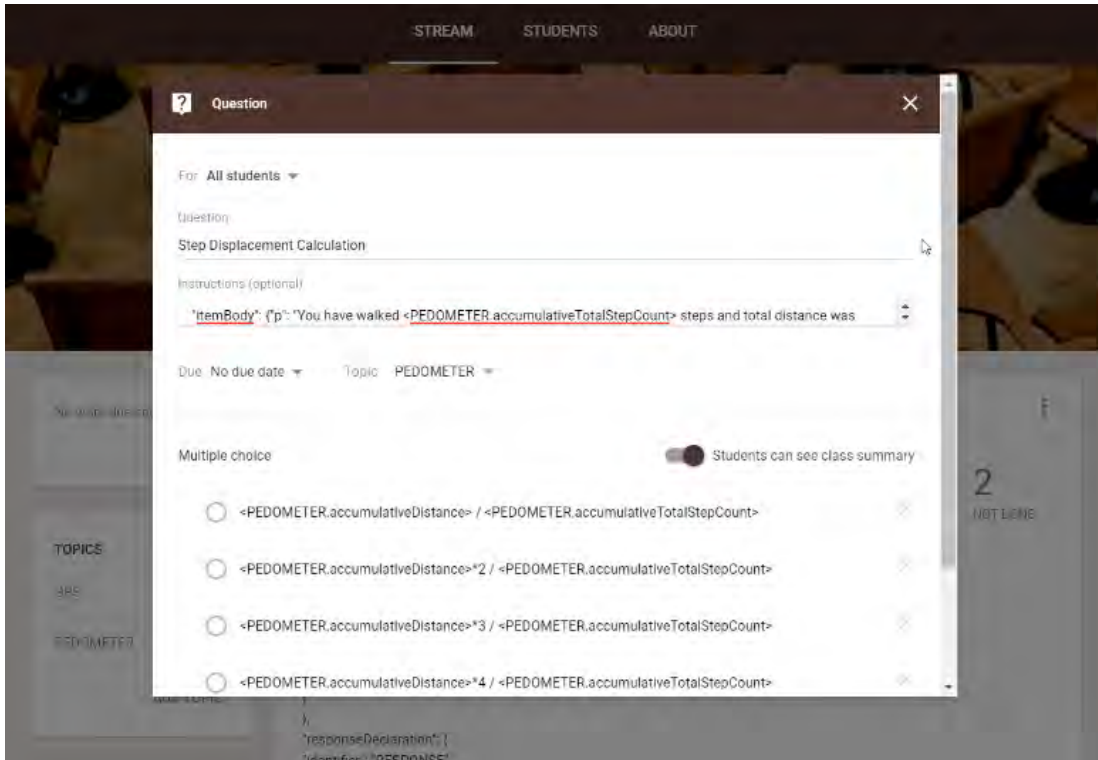


Figure 3.3: Editing an Assessment Question in Google Classroom

3.2.3. NoSQL Store. There was a need to store the watch-student relationship, answers submitted by the students (in order to overcome the Google platform limitation mentioned earlier), and some log data (which will be described in further detail in the performance analysis section).

Relational data is relatively slow and there is no requirement to preserve a transaction state in the data store, so any suitable NoSQL store was an option. Many solutions were available, such as MongoDB, but not all of the features available in MongoDB were necessary and it was desirable to have a seamless API that offered CRUD operations by exposing a REST API to the datastore. Since MongoDB does not have a rest API out of the box and requires third party components to expose it, CouchDB was found to best fit the requirements. It is also possible to use CouchDB to easily access its data through node.js applications using nano javascript library [33].

3.2.4. Teacher Portal. Since node.js code was used in the platform's context processor, Angular 4 [34] code was found to be suitable for use in developing the frontend portal for teachers, which is part of the MongoDB, ExpressJS, AngularJS, NodeJS (MEAN) architecture widely in use at the current time. The teacher portal currently developed was used to assign watches that were in "Pending Registration" status to a specific student so that all his interactions were registered against his student id in the CouchDB store. Shown in Figure 3.4, once the teacher tried to open the portal he was prompted to sign into his Google Classroom account. Luckily, a Google Classroom instance was already available at the university which is being used for testing purposes, so the same account used by teachers or students on campus could be authenticated in Google Classroom through oAuth 2.0. Once authenticated the teacher can choose a course to which he is currently assigned then choose a student he wants to register a particular smartwatch to.

This teacher portal can also be extended to create a question authoring portal and can be made publicly available so that the crowd can feed in creative questions and map them to specific situational contexts. While questions are being written, the crowd has the option to select specific sensors from the smartwatch and to incorporate the output values of these sensors in the question or answer body. Teachers can access the question authoring portal and have the option to select and publish questions in the Google Classroom question bank so that the context processing service can have more questions to query and match with the sensor values received from smartwatches. Another extension of this portal is that statistics from the answered questions can be produced and presented to teachers by querying the records saved in CouchDB

3.2.5. Middleware and the Context Processor. The middleware and context processor are both running in one instance of a node.js application. Using node.js helped to speed development time, as the code was written in JavaScript, which is similar to the smartwatch and teacher portal code. The processor first queries all the available questions in Google Classroom, then queries the updated questions every 5 minutes.

When the teacher portal is extended, this can be enhanced for authoring questions so that the context processor can be updated to read the new questions once they have been published to Google Classroom. The context service retrieves all

questions in all courses to build its cache and matches each question's context eligibility to the received sensor context. The context service then publishes the matched assessments to PONTE so that the smartwatches subscribed to the assessment topic id receive a notification.

The screenshot shows the Teacher Portal interface. At the top right, there is a green notification box that says "Watch Assigned". Below the header, there is a "Hello" message and "Login" and "Logout" buttons. The "Courses" section displays a table with two rows: one for "Languages" (id: 5921342162) and one for "WearableClassroom" (id: 1017124722). Below that, the "Students in Selected Course" section shows a table with two rows of student information. The "Watches" section contains a table with five columns: tizen id, student id, student eMail, and student name. It lists five watch assignments for two different students.

id	name
5921342162	Languages
1017124722	WearableClassroom

id	name	eMail
109710075556516364709	Mohamed ElSolh - Test Student 2	b00050762_st2@aus.edu
101385603138609705291	Mohamed ElSolh - Test Student 1	b00050762_st1@aus.edu

tizen id	student id	student eMail	student name
3B17oVj1PoUotIFvsk2NOIhrQA=	109710075556516364709	b00050762_st2@aus.edu	Mohamed ElSolh - Test Student 2
CHvQnTuxtJbABKb6ISL1S4UmwA=	101385603138609705291	b00050762_st1@aus.edu	Mohamed ElSolh - Test Student 1
JeFJ4jhTm4SLWMLvOXD9pKsOTQA=	101385603138609705291	b00050762_st1@aus.edu	Mohamed ElSolh - Test Student 1
O3yHeUyvwVHE0C2V6zqTUZIFQwA=	101385603138609705291	b00050762_st1@aus.edu	Mohamed ElSolh - Test Student 1
VGJzo39EcaQ5MgpyoXFRNDJFAA=	101385603138609705291	b00050762_st1@aus.edu	Mohamed ElSolh - Test Student 1

Figure 3.4: Wearable device assignment in the Teacher Portal

The middleware that accompanies the context processor code helps to propagate any GET or POST operation to CouchDB. Propagating the GET or POST through the middleware instead of the smartwatch itself makes the communication mechanism more efficient. If the smart watch performs these GET and POST operations to CouchDB, these operations will be exhaustive in terms of processing power for two reasons.

First, the HTTP request requires the client (smartwatch) to establish a new connection every time a POST or GET is invoked. Websockets, however, use the already established connection to send messages.

Second, the HTTP causes the client to wait for a response from the server while the websockets, in contrast, are used to send the message asynchronously so that the response arrives through a subscribed topic. Synchronous requests increase CPU usage and therefore increase the battery consumption in the watch. In addition, establishing new connections adds further overhead in terms of greater power consumption through increased CPU usage.

3.2.6. Smartwatch/Client. The smartwatch chosen for the platform is the Samsung Gear S2 (SGS2). Android smartwatches were also initially tested. A TizenOS based watch, however, was eventually selected as the ideal watch for the use cases. This is because, up until recently, the Android wearable platform has only allowed for standalone operation with a limited number of models that support Android wear 2.0. The Samsung S2 Smartwatch, in contrast, allows for standalone applications, which facilitates the construction of independent applications to enrich students' knowledge without requiring them to use a paired smartphone or tablet while performing assessments. Another reason for the choice is that the TizenOS allowed for HTML5, CSS3 and ES5 Javascript applications, making it a robust development environment that allows for reuse of any browser library in the smartwatch. In fact, the MQTT PAHO javascript library [35] was used to allow the smartwatch to initiate MQTT communication over websockets. In addition, the SGS2 Smartwatch allows easy input so that students can select correct answers by rotating a bezel around the watch instead of selecting radio buttons by touch, something that is quite cumbersome on a small screen. Because it supports HTML5, the SGS2 allows questions to be displayed in an intuitive interface, like the one shown in Figure 3.5, which simplifies the UI design. Samsung offers design templates and the TAU library with out-of-the-box components, which can be used on a watch that already has an intuitive design.

When the student requests assessment questions by opening the application, the smartwatch collects different sensor data from its built-in sensors, such as the status and speed of student movement captured by the accelerometer and gyroscope, as well as the student's heart rate and unprocessed telemetry data (location and NFC tag readings), which will be explained in detail in the next section. This telemetry data is published through the PONTE message broker. Reading sensor data directly from the smartwatch provides an advantage over reading sensor data from a smartphone. Because the student wears the watch all of the time, the gyroscope and accelerometer readings are more accurate. In addition, the GPS location is more accurate, since the smartphone may be set down and may be farther away from the student. When the context processor wants to publish questions to a specific smartwatch, each smartwatch will have its topic based on a dynamic TizenID initiated from the smartwatch. The sensor data published from the smartwatch helps the context

processor trigger in-situ questions to the watch, so it is not necessary to know which student sent the telemetry data until the student submits answers to the questions. This helps to anonymize the telemetry data and makes this platform safer in terms of privacy.

The smartwatch application has two modes for receiving in-situ questions. The first is sending telemetry data on demand to receive questions. The second, called the “Continuous” mode, sends sensor data continuously rather than on demand. This creates a considerable power drain on the smartwatch. The user is also warned about this drawback, although the sensor readings are being efficiently sent only when the sensor data collected is changing. Callback methods for sensor reading changes are available in Samsung APIs to support this feature. Further explanation of the possible use cases that can be performed on the smartwatch will be provided in the next section.



Figure 3.5: Sample in-situ question shown on smartwatch. Screenshots from left to right, and top to bottom show the user interface cycle throughout the application.

3.3. Platform Use Case Analysis

On the platform, from a usability perspective, the wearable device targeted for use is the Samsung Gear, which can have up to 9 sensors, including GPS, a heart-rate monitor, and light, pressure, magnetic, proximity, NFC, ultraviolet and gyroscope sensors. The Samsung Gear also offers ready-to-use derived values such as linear acceleration, gravity and pedometer readings that result from calculations of values taken from these sensors. Feasible use cases for selected sensor/derived values that can be used within the platform will be further explained. The object in Figure 3.6 is an aggregated sensor output taken from a virtual Gear S2, which shows all possible values that can be used within the platform. The object schema follows the sensor network object notation representation [36] and takes into account that the physical version of S2 does not have all of the sensors (e.g., none of the S2 models have the ultra violet sensor).

```
{
  "sensorName": "PEDOMETER",
  "value": {
    "accumulativeDistance": 2.2,
    "speed": 12,
    "accumulativeRunStepCount": 2,
    "accumulativeCalorie": 0.15,
    "walkingFrequency": 3,
    "accumulativeWalkStepCount": 0,
    "stepStatus": "RUNNING",
    "accumulativeTotalStepCount": 2,
    "stepCountDifferences": [
      {
        "timestamp": 1507570874,
        "stepCountDifference": 0
      }
    ]
  }
},
{
  "sensorName": "LIGHT",
  "value": {
    "lightLevel": 65535
  }
},
{
  "sensorName": "PRESSURE",
  "value": {
    "pressure": 759.995392
  }
},
{
  "sensorName": "ULTRAVIOLET",
  "value": {
    "ultravioletLevel": 6.1
  }
},
{
  "sensorName": "HRM",
  "value": {
    "heartRate": 100,
    "rRInterval": 600
  }
},
{
  "sensorName": "GPS",
  "value": {
    "speed": 0,
    "timestamp": 1507570874,
    "altitude": 0,
    "longitude": 127.053902,
    "latitude": 37.259042,
    "errorRange": 0
  }
}
}
```

Figure 3.6: Telemetry data model in the Wearable School Platform

3.3.1. GPS. This sensor is only available in the 3g/4g variant of SGS2 and is available on all models of the recent Gear S3. The most important values derived from GPS are longitude and latitude. A function was built that can be used dynamically within the assessments to calculate the distance between two points. A suitable use case is targeting an assessment to be triggered at specific coordinates to teach the student simple laws of physics in order to calculate distance, speed and time. As an example, the triggered question could ask the student how many hours remain to reach the destination of the Tower of Pisa if you are running at 5km per hour. Many other questions can be derived and targeted for different conditions, such as targeting a question based on speed rather than coordinates in order to trigger a question about the fuel economy for a moving vehicle.

3.3.2. Pedometer. This value is derived from both accelerometer and gyroscope readings. Samsung already offers an API to generate comprehensive calculated values such as the ones shown in Figure 3.6. A possible question for this value might be asking how many steps were taken to walk 500 meters if a step is 2 meters. Different questions can be formulated to calculate total distance, total steps or the length of a step. This type of theme for the questions can help students strengthen their mathematical skills and encourage them to move as these questions can be targeted for a specific number of steps or a particular state of movement using the Samsung API for activity recognition [37].

3.3.3. Light Intensity. This value expresses light intensity in the lux unit of measure. A possible question might be to ask the student how well a room he has entered is lit based on the current value of light intensity and whether the main light source is the sun or fluorescent lights. The question can be targeted for a specific light intensity value. This helps the student learn light intensity measurement values. Also, if the teacher is experienced with concepts about photosynthesis or farming techniques, he may target applicable questions for these subjects with the use of light intensity values.

3.3.4. Ultra Violet (UV). This value is generated by smartwatches for UV sensor ranges from 0-15. Questions can be triggered based on the value indicated by the smartwatch, and a sample question can be sent to ask the student whether he is currently in need of sunscreen or not. This teaches the student about the safe range of UV light and can act as a reminder to the student to take precautions in the event that UV values are dangerously high.

3.3.5. Pressure. This value is measured in hectopascal (hPa) units. A possible question that might be asked of the student is whether he is above or below sea level. This type of question can also help the student strengthen his knowledge of physics in terms of pressure and its value representation in hPa. The question can be triggered based on the output hPa value taken from the sensor.

3.3.6. Heart Rate Monitor. The value in beats per minute is taken from the smartwatch at specific intervals and questions can be triggered for a specific range of heart rate values. Possible questions that may be posed include asking the student the predicted age of a person if the maximum heart rate is the highest value produced by the watch while that person is running. Alternatively, the student might also be asked what the current percentage of the heart rate reading proportional to the max possible heart rate reading is if the person in question is 20 years old. This can help the student learn about health-related subjects and also encourages the student to be active and perform activities in order to receive more questions.

3.3.7. Other Suitable Use Cases. These use cases are not developed on the current platform, but a variety of extensions could be made to the platform in the future. The smartwatch contains an NFC sensor, so NFC tags could be placed on the animal description boards at different places in the zoo so that students can tap their smartwatches or smartphones to answer assessment questions at specific locations. In this type of scenario, when students are on a field trip, they can open the app, and when the desired context of a specific GPS location in the zoo is triggered, the student will be notified about an assessment question and asked to tap his smartwatch on the appropriate board to answer it.

Alternatively, the context eligibility criteria itself could be tapping on the information stand of an animal that is a mammal so that a specific question about

mammals is sent to the student. Also, as part of the question content, the students could tap their smartwatches on the information stand to hear information about the animal or view multimedia content related to the animal stand they have tapped.

Another use case involves questions that relate to interactions with the physical world to enhance the ability of students to learn new languages. For example, using the GPS location and a weather API (since the Gear S2 lacks a dedicated temperature sensor), students might be asked to determine what the weather feels like and asked questions in French to describe the current weather conditions.

3.4. Code Implementations

This section describes important parts of the code implementation in order to provide ease of reproducibility for any researcher who wishes to replicate any part of this work. The important aspects of the platform are the MQTT connectivity and the middleware-related code supporting the MQTT connection, the sensor context matching, and the assessment formulation. In addition, the teacher portal and logging code and oAuth 2.0 authentication code will also be described in brief.

Referring to the sequence diagram in Figure 3.7, the explanation will be divided into three parts: the middleware code (which consists of steps 3 and 4), the watch code (which contains steps 2,5,6,8 and 9) and the context processor code (step 7).

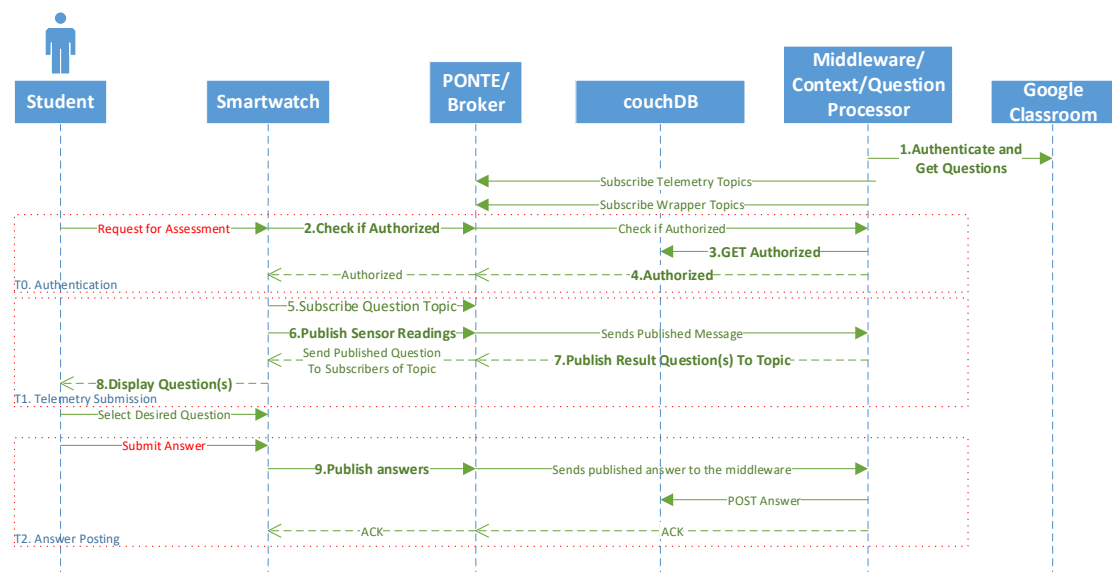


Figure 3.7: Telemetry data model for the Wearable School Platform

3.4.1. Middleware Code-(1,3,4). Referring to step one in the sequence diagram, the middleware authenticates to the Google Classroom platform using Google “auth” nodejs library and retrieves the list of courses, the coursework associated with each and the students in each class. Figure 3.8 shows the code for the authentication step.

```
var googleAuth = require('google-auth-library');

fs.readFile('/home/alsoh/code/wearable-school-workspace/wearable-school-
server/client_secret.json', function processClientSecrets(err, content) {
  if (err) {
    console.log('Error loading client secret file: ' + err);
    return;
  }
  authorize(JSON.parse(content), listCourses);
});
```

Figure 3.8: oAuth code – Loading secret file

The Google “auth” library is loaded and the client secret file containing the application level key is read for content to be processed in the “authorize” function. The listCourses function is also passed and called as the callback function if authentication is successful.

The key file is created by registering the server to Google Classroom only once, so most of the time the key is already be available. In [38], the Google Classroom APIs offer more detailed documentation of the remaining code that has been omitted from this explanation.

In Figure 3.9, the “authorize” function is called and initiates a new oAuth client with the credentials that were read from the secret file. The server will check if a previously created token exists. If not, the getNewToken function is called to create and store a token for the server to use directly on the next startup. This token is used for server level authentication with the Google platform on every GET request. Assuming the token already exists, the listCourses function will be called with the required authenticated client that contains a valid token.

Once the ListCourses function is called, Google Classroom REST interface is called to list all courses as shown in Figure 3.10. For each course, the coursework (assessments) is then cached by the middleware, along with the list of students in the course, by calling the functions listCourseWorks and listStudents by calling the Google APIs classroom.courses.courseWork.list and classroom.courses.students.list.

```

function authorize(credentials, callback) {
  var clientSecret = credentials.installed.client_secret;
  var clientId = credentials.installed.client_id;
  var redirectUrl = credentials.installed.redirect_uris[0];
  var auth = new googleAuth();
  var oauth2Client = new auth.OAuth2(clientId, clientSecret,
  redirectUrl);
  fs.readFile(TOKEN_PATH, function(err, token) {
    if (err) {
      getNewToken(oauth2Client, callback);
    } else {
      oauth2Client.credentials = JSON.parse(token);
      callback(oauth2Client);
    }
  });
}

```

Figure 3.9: oAuth code – Creating oAuth client

Now the courses, students and assessments are cached so the context processor can check for the student through the “studentid”, which was assigned using the teacher portal through which the eligible assessments are available.

The teacher portal also uses oAuth to authenticate the teacher using the Angular 4 custom library “angular-oauth2-oidc“

```

function ListCourses(auth){
  var classroom = google.classroom('v1');
  classroom.courses.list({
    auth: auth
  }, function(err, response) {
    if (err) {
      return;
    }
    try {
      var courses = response.courses;
      if (courses.length > 0) {
        tempCourses = response.courses;
        for (var i = 0; i < courses.length; i++) {
          var course = courses[i];
          listCourseWorks(auth, course.id, i);
          listStudents(auth, course.id, i);
        }
      }
    }
  }
}

```

Figure 3.10: Function to list courses and coursework from Google Classroom

Referring to steps three and four in the sequence diagram, the code in Figure 3.11 is executed to propagate any GET or POST message as a request to CouchDB.

The JSON message received (which will be explained in next section) is read and formulated into a GET or POST depending on payload.method, then a callback is registered to publish the response back to the watch with the same topic id received

but with “wrapper” replaced by “response” so that the watch can receive the message. Subscribing to the response topic id will also be explained in the next section.

```

payload = JSON.parse(message.toString());
var post_options = {
  host: payload.host,
  port: payload.port,
  path: payload.path,
  method: payload.method,
  headers: {
    'Authorization': "Basic " + btoa('admin' + ":" + '*****'),
    'Content-Type': 'application/json'
  }
};
var post_req = http.request(post_options, function(res) {
  res.setEncoding('utf8');
  res.on('data', function(chunk) {
    client.publish(topic.replace("wrapper", "response"), 'Response: ' +
  chunk);
  });
});
post_req.write(JSON.stringify(payload.data));
post_req.end();

```

Figure 3.11: MQTT message to HTTP GET/POST wrapper code

3.4.2. SGS2 Smartwatch Application Code (2,5,6,8,9). The Samsung Wearable School application was built to run on the Tizen OS. Tizen offers a familiar IDE based on Eclipse called Tizen Studio. The application code can be written in native C-based code or in HTML5, which was our choice. The application first navigates to the index.html page, each page loads appropriate JavaScript libraries and then, when the page operations end, the next page can be called with a special page navigation function from Samsung’s TAU library “tau.changePage('/page.html');”. The following JavaScript libraries in Table 3.1 are loaded in the index page and next to each library the purpose is described for it.

Table 3.1: Description of JavaScript Libraries loaded in index.html

Library	Description
log4javascript.js	JavaScript logger library similar to log4j or log4net
watchresourceslog.js	Custom library to log CPU, Memory and battery usage
tau.min.js	Samsung TAU library for UI related operations
mqttws31.js	MQTT JavaScript client based on PAHO
mqttoperations.js	Custom library written to handle MQTT messages and subscriptions
index.js	The starting page code that initializes a session id

It is important to note that for the navigation of every page, the JavaScript code that needs to be executed has to be wrapped with “pagebeforeshow” event shown in Figure 3.12, such as the screen turning on before the tizen id is shown on the screen so that the student can ask his teacher to register his smartwatch.

```

var page = document.getElementById( "main" );
page.addEventListener("pagebeforeshow", function() {
    tizen.power.turnScreenOn();
    localStorage.setItem("sessionId", sessionId);
    var tizenId =
    tizen.systeminfo.getCapability("http://tizen.org/system/tizenid");
    document.getElementById("tizenId").innerText = tizenId;
});

```

Figure 3.12: Adding “pagebeforeshow” event listener for any Tizen application page

After the index.js code is executed, mqttoperations.js gets loaded which contains the MQTT client initiation operation and subscription to appropriate topics shown in Figure 3.13.

```

client = new Paho.MQTT.Client('192.168.0.110', Number(3000), sessionId);
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
client.connect({onSuccess:onConnect,onFailure:doFail});

```

Figure 3.13: MQTT client initialization code

Referring to step number five in the sequence diagram, the code in Figure 3.13 connects to the PONTE server through MQTT over websockets, then registers call backs for message arrival. If the connection is successful, it will call the onConnect function to execute code in Figure 3.14 which subscribes it to the required topics

```

client.subscribe("assessments/" + encodedWatchId);
client.subscribe("response/" + encodedWatchId + "/"#");

```

Figure 3.14: MQTT topics subscription code

Referring to Figure 3.15, encodedWatchId is the tizenId but it is encoded in URI format to avoid illegal characters in the MQTT topic

```

message = new
Paho.MQTT.Message(JSON.stringify({responseTimeLog:{sessionId:sessionId,txnId:t
xnId,txType:'authentication',records:[{txTime:n,endPoint:'watch'}]},operation:
"GET",data:{"_id":watchId},url:'http://192.168.0.110:5984/watches/' +
encodedWatchId}));
message.destinationName = "wrapper/" + encodedWatchId + "/isWatchRegistered";
client.send(message);

```

Figure 3.15: Sending the authentication request message

Referring to step number two in the sequence diagram, the code in Figure 3.15 sends an authentication message request to the middleware. Instead of calling CouchDB directly, the middleware calls CouchDB by reading the wrapped content, which was shown in Figure 3.11, after the middleware receives the response from CouchDB. The middleware will propagate the response to the watch and as shown in Figure 3.16. The watch will receive an `isWatchRegistered` message from the response topic, then the watch will check if there is an authenticated student assigned to the `tizenId` of the current watch running the Wearable School application by checking if there is a `studentId` in the response then storing all authentication related information in the HTML5 local store for use across the application.

```

function onMessageArrived(message) {
    if(message.destinationName.indexOf('isWatchRegistered') > -1) {
        payload = JSON.parse(message.payloadString);
        if(payload.studentId != null){
            try{
                localStorage.setItem("studentId", payload.studentId);
                localStorage.setItem("studentEmail", payload.studentEmail);
                localStorage.setItem("telemetrySendInterval",
                    payload.telemetrySendInterval);
                localStorage.setItem("timeoutSelectMode",
                    payload.timeoutSelectMode);
                localStorage.setItem("timeoutConfirmQuestion",
                    payload.timeoutConfirmQuestion);
                localStorage.setItem("timeoutSubmitAnswer",
                    payload.timeoutSubmitAnswer);
                localStorage.setItem("episodeId", payload.episodeId);
                localStorage.setItem("automated", payload.automated);
                localStorage.setItem("mode", payload.mode);
                tau.changePage('/pages/radio/radioQuestionsMode.html');
            }
        }
    }
}

```

Figure 3.16: Receiving successful authentication message

The code in Figure 3.17 handles what needs to be done once the answer is posted, so if the Continuous mode is selected, it will go back to sensor collection page “authorizedV2”; otherwise it will exit the application. There is also code for the automated mode to handle restarting the application with the operation “`window.location.href = (/index.html);`” if the onDemand mode is selected but this causes a memory leak on the Tizen OS which will be described later.

The last part of the code shown in Figure 3.18: Handling the post answer responseFigure 3.18 handles the assessments topic id so that it loads the question selection page if the question count is greater than one; otherwise it loads the question

content page directly if the question count is exactly one. In the last case, if there are no questions found in the message, a message is shown to the user announcing that there no eligible questions have been found.

```
else if(message.destinationName.indexOf('postAnswer') > -1){

    if(localStorage.getItem("automated") == "true"){
        localStorage.setItem("logger1",JSON.stringify(logData));
        if(localStorage.getItem("continous") != "true"){
            window.location.href = ('/index.html');}
        else {
            tau.changePage('/authorizedV2.html');
        }
    } else {
        if(localStorage.getItem("continous") == "true"){
            tau.changePage('/authorizedV2.html');
        }
        else{
            window.tizen.application.getCurrentApplication().exit();
        }
    }
}
```

Figure 3.17: Handling the post answer response

```
else if(message.destinationName.indexOf('assessments') > -1){
    var questions = JSON.parse(message.payloadString);
    if(questions.length > 1){
        localStorage.setItem("questions", message.payloadString);
        tau.changePage('/pages/radio/radioQuestions.html');
    }
    else if(questions.length == 1){
        localStorage.setItem("question",JSON.stringify(questions[0]));
        tau.changePage('/pages/QuestionContent.html');
    }
    else if(questions.length == 0){
        document.getElementById("smwatchId").innerText = "No Eligibile
        Questions Found!";
        if(localStorage.getItem("continous") != "true"){
            if(localStorage.getItem("automated") == "true"){
                window.location.href = ('/index.html');
            }
            else{setTimeout(function(){
                window.tizen.application.getCurrentApplication().exit();}, 1000);}
            }else{
                tau.changePage('/pages/noContentFound.html');
            }
        }
    }
}
```

Figure 3.18: Handling the post answer response

The code for the mode selection `radioQuestionsMode.js` will now be described. The code in Figure 3.19 stores the user's selection, then navigates to the sensor collection page. Now we describe the code that runs when the sensor collection starts `sensorsdatacollection.js`.

The code in Figure 3.20 the starts the data collection for each of the six supported sensor types.

```

element.addEventListener('click', function() {
    if(selectedMode == "onDemand"){
        localStorage.setItem("continous", "false");
    }
    else {
        localStorage.setItem("continous", "true");
    }
    tau.changePage('/authorizedV2.html');
}, false);

```

Figure 3.19: Sensor collection mode selection handler

```

telemetrySendInterval = localStorage.getItem("telemetrySendInterval");
sensorsData = [];
sensingActive = true;
var tizenElement = document.getElementById("smwatchId");
tizenElement.innerText = "Reading Sensor Data...";
try{
    navigator.geolocation.getCurrentPosition(showMap);
} catch (err) {}
try{
    tizen.humanactivitymonitor.start('GPS', onChangeGPS,
onerrorGPS, option);
} catch (err) {}
tizen.humanactivitymonitor.start("HRM", onChangeHRM);
tizen.humanactivitymonitor.setAccumulativePedometerListener(onchangedCB);
lightSensor.start(LSonsuccessCB);
var pressureCapability =
tizen.systeminfo.getCapability("http://tizen.org/feature/sensor.barometer");
if (pressureCapability === true) {
    pressureSensor = tizen.sensorservice.getDefaultSensor("PRESSURE");
    pressureSensor.start(PRSonsuccessCB);
}
var ultravioletCapability =
tizen.systeminfo.getCapability("http://tizen.org/feature/sensor.ultraviolet");
if (ultravioletCapability === true) {
    ultravioletSensor = tizen.sensorservice.getDefaultSensor("ULTRAVIOLET");
    ultravioletSensor.start(UVSonsuccessCB);
}

```

Figure 3.20: Sensor startup code

The code in Figure 3.21 is designed for the OnDemand mode. Basically, it allows reading of all sensor values and providing a timeout of 10 seconds: if 10 seconds have elapsed and there has been no reading from any of the sensors it will proceed to send whatever sensor readings it currently has. is a sample function for heart rate data collection when the heart rate sensor is initialized and a change event is received

The code in Figure 3.22 represents a sample callback from a sensor. The code initializes the sensor data object with an identifier equal to “HRM” then embeds whatever data is contained in the sensor JSON object. The JSON object sensorData is then stringified and pushed to the sensorsData array to be sent when the timeout is

met or for OnDemand and all sensors have already been read. is the code for sending data over MQTT.

```
if(localStorage.getItem("continous") != "true"){
    refreshIntervalId = setInterval(checkIfCompleted, 1000);
}
else{
    setTimeout(sendMQTTMessage, telemetrySendInterval);
}
function checkIfCompleted(){
    console.log("checking " + remainingSensorCalls);
    remainingSensorCallsTimeout = remainingSensorCallsTimeout - 1;
    if ((remainingSensorCalls < 1 || remainingSensorCallsTimeout < 1)) {
        console.log("all sensors done");
        clearInterval(refreshIntervalId);
        sendMQTTMessage();
    }
}
```

Figure 3.21: OnDemand sensor collection code

```
function HRMSonsuccessCB(hrmInfo) {
    try{
        var sensorData = { "sensorName":"HRM", "value":hrmInfo };
        sensorsData.push(sensorData);
        console.log(JSON.stringify(sensorsData));
        remainingSensorCalls = remainingSensorCalls - 1;
    }
    catch(err) {
        console.log(err.message);
    }
}
```

Figure 3.22: Sensor callback handling code

Referring to step six in the sequence diagram, the code in Figure 3.23 first prompts the user data id being sent. Once the message is sent, it adds a delay of 3 seconds then shows the user a message indicating that the watch is collecting data again. Note that the 3-second delay is added asynchronously and does not affect the total interval time for T1.telemetrySubmission (this transaction will be described in the next chapter). The delay is for UI purposes only.

Referring to step eight in the sequence diagram, the code that runs when question content is received, which is available in questionContent.js will now be described. It is assumed that only one eligible question was received in Figure 3.24.

```

function sendMQTTMessage(){
    if(sensingActive){
        document.getElementById("smwatchId").innerText = "Sending
        Sensor Data...";
        message = new
        Paho.MQTT.Message(JSON.stringify({studentId:localStorage.getItem("studentId"),sensorsData:sensorsData}));
        message.destinationName = "telemetry/" + encodedWatchId;
        client.send(message);
        sensorsData = [];
        setTimeout(function(){
            document.getElementById("smwatchId").innerText = "Reading
            Sensor Data..."; }, 3000);
        }
    }
}

```

Figure 3.23: Telemetry submission code

```

tizen.power.turnScreenOn();
var app = tizen.application.getCurrentApplication();
tizen.application.launch(app.appInfo.id, function() {}, function()
{});
timeoutConfirmQuestion =
localStorage.getItem("timeoutConfirmQuestion");
var questionObject = JSON.parse(localStorage["question"]);
var old_element = document.getElementById("calibration-btn");
var new_element = old_element.cloneNode(true);
old_element.parentNode.replaceChild(new_element, old_element);
document.getElementById('menu-title').innerText =
questionObject.title;
document.getElementById('menu-detail').innerText =
questionObject.description.itemBody.p;
document.getElementById('menu-prompt').innerText =
questionObject.description.itemBody.choiceInteraction.prompt;
var element = document.getElementById('calibration-btn');
element.addEventListener('click', function() {
    tau.changePage('/pages/radio/radio.html');
}, false);
if(localStorage.getItem("automated") == "true"){
    setTimeout(function () {
        document.getElementById('calibration-btn').click();
    }, timeoutConfirmQuestion);
}
}

```

Figure 3.24: Question content display code

The code first turns on the screen, then focuses the application because Tizen automatically brings the watch to the home screen if screen timeout time has been met. The question object is then loaded from the HTML5 local store, where it was stored earlier by the code that handles the MQTT messages received. The question title and content are then set for the question to be displayed to the user. There is an additional section of code to check if the automated flag is true. Then the user

directly presses the answer button to go the answer selection page. The code for this, which is available in radio.js in Figure 3.25, will be described next.

Then code in Figure 3.25 first turns the screen on, then retrieves the answer object, creates radio buttons on the screen for each answer and adds an event handler for the submit button, so that when the button is clicked, code in Figure 3.26 is run.

```
tizen.power.turnScreenOn();
timeoutSubmitAnswer = localStorage.getItem("timeoutSubmitAnswer");
var questionObject = null;
selectedAnswer = "";
timesPageLoaded = timesPageLoaded + 1;
submitAnswerClicked = false;
document.getElementById('answersRadio').innerHTML = "";
var element = document.getElementById('okButton1');
element.addEventListener('click', function() {
    if(submitAnswerClicked == false){
        submitAnswer(selectedAnswer);
        submitAnswerClicked = true;
    }
}, false);
questionObject = JSON.parse(localStorage["question"]);
var myStringArray = questionObject.multipleChoiceQuestion.choices;
var arrayLength = myStringArray.length;
for (var i = 0; i < arrayLength; i++) {
    document.getElementById('answersRadio').innerHTML += '<li
class="li-has-radio" id="' + i + 'radioanswer"><label>' +
myStringArray[i] + '<input type="radio" name="radio-sample"
/></label></li>';
}
for (var i = 0; i < arrayLength; i++) {
document.getElementById(i + "radioanswer").addEventListener('click',
function() {
    selectedAnswer=this.innerText;
}, false);
}
```

Figure 3.25: Answers display code

Referring to step nine in the sequence diagram, the code in Figure 3.26 first prepares the MQTT message for posting the answer through the middleware instead of posting it directly to CouchDB. It includes the student information, the answer the student has selected, the correct answer and the assessment outcome.

Finally, the code in Figure 3.27 is the portion of the code that is then run for the automated mode to select a random answer so that the application can start another test point by navigating back to the sensor collection page once the answer posting confirmation message is received from the middleware.

```

function submitAnswer(selectedAnswer) {

    var d = new Date();
    var n = d.getTime();
    var assessmentOutcome;
    if(selectedAnswer ==
questionObject.description.responseDeclaration.correctResponse) {
    assessmentOutcome = "pass";
    }
    else{
    assessmentOutcome = "fail";}
    var answerData = {
        "episodeId": localStorage.getItem("episodeId"),
        "answerTime": n,
        "questionId": questionObject.id,
        "questionTitle": questionObject.title,
        "questionDetail": questionObject.description.itemBody.p,
        "questionPrompt":
questionObject.description.itemBody.choiceInteraction.prompt,
        "selectedAnswer": selectedAnswer,
        "correctAnswer":
questionObject.description.responseDeclaration.correctResponse,
        "assessmentOutcome": assessmentOutcome,
        "answeredWatchId": watchId,
        "answeredStudentId": localStorage.getItem("studentId"),
        "answeredStudentEmail": localStorage.getItem("studentEmail")
    };
    message = new
Paho.MQTT.Message(JSON.stringify({method:"POST",data:answerData,host:
"192.168.0.110",port:"5984",path:"/wearable",sessionId:localStorage.
getItem("sessionId")}));
    message.destinationName = "wrapper/" + encodedWatchId + "/postAnswer";
    log.info({logType:'txLog',txnType:'0.postAnswerStart',
endPoint:'watch'});
    client.send(message);}

```

Figure 3.26: Post answer code

```

if(localStorage.getItem("automated") == "true"){

    setTimeout(function () {
    document.getElementById( Math.floor((Math.random() * arrayLength)) +
'radioanswer').click();
    document.getElementById('okButton1').click();
    }, timeoutSubmitAnswer);
}

```

Figure 3.27: Post answer code

3.4.3. The Context Processor Application Code (7). The context processor runs under node.js and the code is contained in file server.js. server.js has two roles: it functions both as middleware and as a context processor.

There are many operations being run in server.js to connect to the Google Classroom platform and retrieve assessment questions every 5 minutes. The primary

focus in describing the code is the context processor and the MQTT connection. The code for these is available in the following sections.

3.4.3.1 MQTT code. Figure 3.28 shows the code for the MQTT connection startup and subscription to topics.

```
var client = mqtt.connect('mqtt://192.168.0.110:1883');
client.on('connect', function () {
  client.subscribe('telemetry/#');
  client.subscribe('wrapper/#');
})
```

Figure 3.28: Middleware MQTT connection startup

A handler is registered for any new message received in Figure 3.29.

```
client.on('message', function (topic, message) {
```

Figure 3.29: MQTT message handler

Conditions will then be based on the topic received to execute the code related to the topic shown in Figure 3.30.

```
if(topic.indexOf('isWatchRegistered') > -1) {
```

Figure 3.30: Authentication condition

3.4.3.2 Context processor code. The code in Figure 3.31 is executed when a telemetry message is received from a smartwatch. It is assumed that the courses and coursework objects are already filled with Google Classroom content. Therefore, the retrieval of these objects will not be described.

The code in Figure 3.31 has an enumerator containing all sensor types that needs to be checked for context matching the assessments. The second line parses the JSON object received that contains the sensor data into a “telemetry” variable.

The code loops through every sensor type from the enumerator, then queries the telemetry object to determine whether it contains any of the defined sensors in the enumerator. The code then loops through each property contained in the sensor type found.

```

var sensors = ["HRM", "GPS", "LIGHT", "PEDOMETER", "PRESSURE",
"ULTRAVIOLET"];
if (allCourses != null) {
  console.log(message.toString());
  telemetry = JSON.parse(message);
  aggregatedTelemetry = [];
  var sum = 0;
  for (var i = 0; i < sensors.length; i++) {
    queryTelemetry = jquery(['*sensorName=' + sensors[i] + '].value',
{
  data: telemetry.sensorsData
}).value;
  for (var j = 1; j < queryTelemetry.length; j++) {
    for (var property in queryTelemetry[0]) {

```

Figure 3.31: Context Processor – First Part

As show in Figure 3.32, for each of these properties the code will find the maximum value available with the exception of GPS, in which case it checks for the invalid value 200 and ignores it. This is required because some sensors, like the heart rate monitor, may at times return invalid readings like -3 and it is necessary to have the maximum sensor reading during the interval that the watch was collecting data in. After finding all the maximum values, the code pushes the filtered values to a variable called aggregatedTelemetry.

```

    if (queryTelemetry[0].hasOwnProperty(property)) {
      if (!isNaN(queryTelemetry[0][property])) {
        /*queryTelemetry[0][property] =
queryTelemetry[0][property] + queryTelemetry[j][property];*/
        if (queryTelemetry[0][property] <
queryTelemetry[j][property]) {
          if (property == "longitude") {
            if (queryTelemetry[j][property] !=
200) {
              queryTelemetry[0][property] =
queryTelemetry[j][property];
            }
          }
          else {
            queryTelemetry[0][property] =
queryTelemetry[j][property];
.....

```

Figure 3.32: Context Processor – Second Part

The code in Figure 3.33 loops through all of the course work and checks if the student who had the watch has a role to get the coursework in each of the classes available on Google Classroom platform.

Then the code filters the eligible coursework and adds it to a variable called “courseWork”.

```

        if (queryTelemetry.length > 0) {
            aggregatedTelemetry.push({"sensorName": sensors[i],
"value": queryTelemetry[0]});
        }
    }
    var courseWork = [];
    for (var iCourses = 0; iCourses < allCourses.length; iCourses++) {
        for (var iStudents = 0; iStudents <
allCourses[iCourses].students.length; iStudents++) {
            if (allCourses[iCourses].students[iStudents].userId ==
telemetry.studentId) {
                for (var iCourseWork = 0; iCourseWork <
allCourses[iCourses].courseWork.length; iCourseWork++) {
                    courseWork.push(allCourses[iCourses].courseWork[iCourseWork]);
                }
                break;
            }
        }
    }
}

```

Figure 3.33: Context Processor – Third Part

The code in Figure 3.34 loops through the coursework for each class the student is authorized for and checks the dynamic assessment object to determine which sensors are to be matched for context. The jQuery library is used to speed up the search in the JSON object.

```

if (courseWork.length != 0) {
    //start of handling new question format
    for (var i = 0; i < courseWork.length; i++) {
        try {
            var eligible = false;
            qtiContent = JSON.parse(courseWork[i].description);
            requiredSensors =
qtiContent["assessmentItem"]["context"]["sensorNames"];
            pedoTelemetry = jQuery('[sensorName=' +
requiredSensors[0] + '].value', {
                data: aggregatedTelemetry
            }).value;
        }
    }
}

```

Figure 3.34: Context Processor – Fourth Part

The code in Figure 3.35 loops through all of the properties of the targeted sensor type and injects the sensor attribute values into the question object and the answers object. It also evaluates any helper functions into variables to be placed within the question, such as the helper function for finding the distance between 2 coordinates.

The code in Figure 3.36 executes the JSON object part that states the requirements for the question eligible to be sent to the smartwatch, such as having a

specific longitude and latitude value. The condition is written dynamically in the JSON object of the assessment item that we have explored earlier.

```

        for (var attributename in pedoTelemetry) {
            console.log(attributename + ": " +
                pedoTelemetry[attributename]);
            qtiContent =
                JSON.parse(JSON.stringify(qtiContent).replaceAll('<' + requiredSensors[0] + '.' +
                    + attributename + '>', pedoTelemetry[attributename]));
            courseWork[i].multipleChoiceQuestion.choices =
                JSON.parse(JSON.stringify(courseWork[i].multipleChoiceQuestion.choices).replaceAll('<' + requiredSensors[0] + '.' + attributename + '>',
                    pedoTelemetry[attributename]));
        }
        //process variables
        if (qtiContent.itemBody.variables != null) {
            for (var j = 0; j < qtiContent.itemBody.variables.length;
                j++) {
                qtiContent.itemBody.variables[j] =
                    eval(qtiContent.itemBody.variables[j]);
                console.log(qtiContent.itemBody.variables[j]);
            }
            for (var j = 0; j < qtiContent.itemBody.variables.length;
                j++) {
                //itemBody.p .itemBody.p
                qtiContent =
                    JSON.parse(JSON.stringify(qtiContent).replaceAll('<var' + j + '>',
                        qtiContent.itemBody.variables[j]));
                for (var k = 0; k <
                    courseWork[i].multipleChoiceQuestion.choices.length; k++) {
                    courseWork[i].multipleChoiceQuestion.choices[k] =
                        courseWork[i].multipleChoiceQuestion.choices[k].replaceAll('<var' + j + '>',
                            qtiContent.itemBody.variables[j]);
                }
            }
        }
    }
}

```

Figure 3.35: Context Processor – Fifth Part

```

        console.log("eligibility - " +
            qtiContent.assessmentItem);
        eligible =
            eval(qtiContent.assessmentItem.context.eligibility);
        qtiContent.assessmentItem.context.eligibility =
            eligible;

        qtiContent.responseDeclaration.correctResponse =
            eval(qtiContent.responseDeclaration.correctResponse);

        courseWork[i].description = qtiContent;
    }
}

```

Figure 3.36: Context Processor – Sixth Part

Referring to step seven in the sequence diagram, the code in Figure 3.37 injects the evaluated values for either variables taken from the helper functions or the sensor parameters themselves into the dynamic answers object. Then the ready filtered questions are published to PONTE to the exact tizenID that sent the message containing the telemetry data processed.

```

        if (eligible) {
            for (var j = 0; j <
courseWork[i].multipleChoiceQuestion.choices.length; j++) {

courseWork[i].multipleChoiceQuestion.choices[j] =
eval(courseWork[i].multipleChoiceQuestion.choices[j]);
            }
            console.log(JSON.stringify(courseWork));
        } catch (err) {
            // handle the error safely
            console.log(err);
        }
    }
    filteredCourseWork = [];
    for (var i = 0; i < courseWork.length; i++) {
        try {
            if
(courseWork[i].description.assessmentItem.context.eligibility) {
                filteredCourseWork.push(courseWork[i]);
            }
        } catch (err) {
            // handle the error safely
            console.log(err);
        }
    }
    client.publish(topic.replace("telemetry", "assessments"),
JSON.stringify(filteredCourseWork));
}
}

```

Figure 3.37: Context Processor – Seventh Part

3.5. Code Summary

In the previous section, the code for the smartwatch Tizen application and the server node.js application containing the middleware and context processor was explained in detail. To provide an overall design summary for the wearable school platform, this section will present the code summary and a description of each component. Note that the dependency libraries are not included in the lines of code calculations.

3.5.1. Tizen Smart Watch Application. The smartwatch application was written in HTML5 and JavaScript. Samsung has offered APIs and JavaScript libraries to help create applications with the same level of control as native applications using their compiler in Tizen Studio. Table 3.2 shows a summary of each code file and its role or purpose.

Table 3.2: Description and lines of code for the Tizen smart watch application source code files

Source File	Source Code Lines	Description
authorizedV2.html	27	The page called when the application is authorized; it loads the sensorsdatacollection.js file
index.html	40	The first page that loads in the application, which loads the mqttoperations.js and watchresourceslog.js files
noContentFound.html	31	Page shown when no eligible assessments have been found; it loads the noContentFound.js file
QuestionContent.html	34	Page showing the question content; it loads the questionContent.js file
radio.html	28	Page that shows the answers selections; it load the radio.js file
radioQuestions.html	35	Page that shows the questions selections in case more than one question was found to be eligible; it loads the radioQuestionsMode.js file
radioQuestionsMode.html	41	Page that shows the option to select either OnDemand or Continuous mode; it loads the radioQuestionsMode.js file
index.js	11	The starting page code that initializes a session id
mqttoperations.js	110	Custom library written to handle MQTT messages and subscriptions
noContentFound.js	13	Code that handles showing the message for no eligible assessments then navigates back to the sensor collection page or quits the application in the OnDemand mode
questionContent.js	28	Code that handles showing the assessment question content and navigating to the answers page
radio.js	70	Code that handles showing the assessments answers and selection
radioQuestions.js	39	Code that handles showing the assessment questions selection for the case where more than one assessment was eligible and also handles navigating to the question content page
radioQuestionsMode.js	38	Code for handling the selection of the Continuous or OnDemand modes and navigation to the sensor collection page
sensorsdatacollection.js	256	Code for collecting sensor data and submitting the MQTT message for the telemetrySubmission transaction
watchresourceslog.js	96	Custom library to log CPU, memory and battery usage
config.xml	29	XML configuration for the application-required privileges from Tizen OS
Total:	926	

3.5.2. Middleware and Context Processor Application. The context processor and middleware was written in JavaScript to run on the node.js framework. Both the middleware and the context processor are managed in one application instance contained in the server.js file (details are available in Table 3.3). Other JavaScript files are used for converting the CouchDB document database to XLSX and for managing the server.js instance to be suited for an automated test (also explained in Table 3.3).

Table 3.3: Description and lines of code for the middleware and context processor application source code files

Source File	Source Code Lines	Description
couchdb2xlsx.js	88	Code that converts CouchDB document database to XLSX format to output transaction and resource data for analysis of results shown in Chapter 5
pcapjson2xlsx.js	33	Code that can extract MQTT messages sent over port 1883 from Wireshark exported JSON file
server.js	424	The context processor and middleware application code
testAutomator.js	22	Code for reading episode data from Excel and running an automated instance of server.js
wireshark.js	2	Code that calls the pcap2csv [39] library to output the bitrate data from a Wireshark PCAP file
Total	569	

3.5.3. Teacher Portal Application. The teacher portal application was built using the Angular 4 framework. The portal uses the bootstrap library to ease the UI design of web components and the PrimeNG [40] library for its mature data table UI component in the teacher portal pages. The role of this portal is to authenticate a teacher to the Google Classroom platform and enable that teacher to assign eligible students enrolled in the teacher’s class to smartwatches. A summary of the code is available in Table 3.4

Table 3.4: Description and lines of code for the teacher portal source code files

Source File	Source Code Lines	Description
app.component.html	6	The main HTML component that hosts the child components gsignin and students
gsignin.component.html	12	The page containing a greeting for the logged-in teacher, as well as login and logout buttons
index.html	14	Root index file for Angular 4 containing application title and icon
students.component.html	24	Component containing the list of students and list of watches PrimeNG table
.angular-cli.json	68	Configuration file for defining the CSS style to be utilized, which is bootstrap; also contains the Angular application startup settings
app.component.ts	22	Root component that contains the OAuth angular-oauth2-oidc library initialization code
app.module.ts	32	Root module that contains the module dependencies such as PrimeNG and angular-oauth2-oidc
gsignin.component.ts	27	Contains the handlers for login and logout events
main.ts	8	Contains code to inject the bootstrap module into the application
students.component.ts	89	Contains code for handling student and watch data tables
Total:	302	

Chapter 4. Experimental Setup

This chapter describes the experimental method followed to test the design decisions taken in the wearable school platform, first an analysis of the transactions running in the platform is performed, and then explanation will continue for the experiments.

4.1. Transactions Analysis

Figure 4.1 shows a diagram of the transactions and data flow that occurs within the platform. In this sequence diagram there are no exceptions or failed results expected while performing the telemetry and that the appropriate sensor values that retrieve eligible questions are assumed to be available. Three main transactions are marked with the red dotted lines: t0, t1, and t2. Each of these transactions will be explained in detail, bearing in mind that all the communication is happening in MQTT between all endpoints, except for the smartwatch which is using MQTT over web socket from the watch to PONTE message broker and back to the watch.

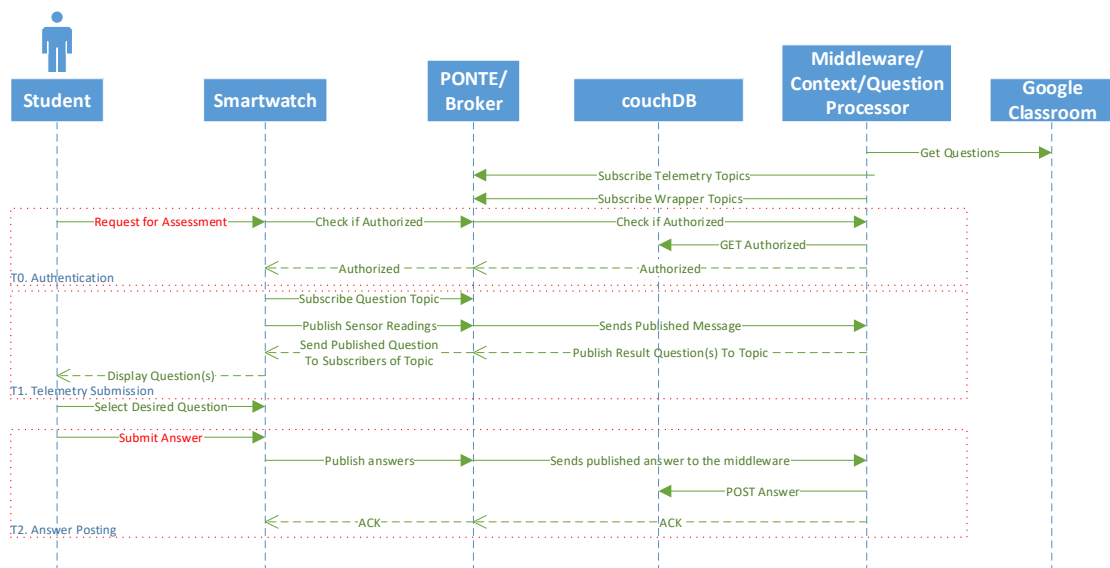


Figure 4.1: Transaction sequence diagram

4.1.1. Authentication (T0). In the transaction marked in the sequence diagram as “T0”, the smartwatch sends its TizenId to the middleware through the PONTE message broker using the topic id “wrapper/studentId/isWatchRegistered”. The TizenId is denoted as “studentId” in the topic url. The message content contains a wrapped get request in JSON format in which the middleware will communicate to CouchDB to find out whether or not the watch identifier contained in the JSON content of the message is actually registered to a student. The middleware can then respond, indicating success or failure by publishing to the topic “response/studentId/isWatchRegistered” and letting the watch proceed to the authenticated page to begin collecting sensor data.

4.1.2. Telemetry Submission (T1). In the transaction marked in the sequence diagram as “T1”, the watch collects sensor data from the six sensors: GPS, HRM, pedometer, light, pressure, and UV. The watch finds the sensors data that is both available and supported by the watch then publishes it to the topic id “telemetry/studentId”. The middleware processes the telemetry data and checks all the questions retrieved from Google Classroom to determine whether there are any eligible questions for the telemetry data collected. The middleware then formulates the question(s) object and sends it back to the watch by publishing to the topic id “assessments/studentId”. The watch then displays eligible questions for the user to choose.

The mode previously described is called “OnDemand”. There is another supported mode – the “Continuous” mode – that continues collecting and sending sensor values for a configurable interval and displays a question to the student as soon as one of the sensor values collected matches the context condition of the available questions posted to Google Classroom. Note also that the sensor collection mode in continuous mode is different in that it continuously reads and stores the sensor reading changes in an array then sends it once the interval has been reached.

4.1.3. Answer Posting (T2). In the transaction marked in the sequence diagram as “T2”, the student selects an answer and submits it where the watch publishes the answer to the wrapper topic id “wrapper/studentId/postAnswer”, which contains a wrapped POST request in JSON format so the middleware will post the message content to CouchDB and communicate the result to the watch by publishing to the topic id “response/studentId/postAnswer” so that the watch can close the application.

4.2. Platform Experimental Design

After reviewing the main use cases and their transactions, a performance test based on the three transactions will be presented. It is necessary to present a use case that does not require human intervention during the performance test because it is desirable to keep all of the watches being tested in a single location without humans wearing them during the test. The best option was found to be the light intensity use case. This use case was chosen for two reasons: light can be easily manipulated by changing the sources that emit light onto the watch. In addition, all Gear S2 watch models available during the experiment had built-in light sensors, unlike the GPS sensors, which, as explained earlier, are exclusive to the 4g/3g models. Only one of the 4g watches was available. The next section describes the instrumentation used in the test.

4.2.1. Instrumentation. As mentioned earlier, MQTT was used for all communication conducted through the watch, based on previous research Shapsough [39], which compared MQTT, COAP, and HTTP communication for IoT devices. In contrast to Shapsough’s research, this report measures how efficient the platform is for wearables using MQTT for communication under different network conditions. Efficiency is measured in terms of response time, power consumption, and CPU and memory usage at the watch end.

To support the design decisions, a test environment was prepared that included 8 smartwatches, all running an automated version of the Wearable School application by controlling a flag in the application, either by changing configuration parameters on CouchDB or by reading from a test input sheet and passing process arguments to the middleware. All of the watches were SGS2 models, specifically the Gear S2

Classic variant (Model R732). The watch is a relatively capable machine, considering its relatively small size. The watch’s specifications are presented in Table 4.1.

Table 4.1: Samsung Gear S2 Specifications [41]

Specification Type	Specification Details
CPU	Exynos 3250 Dual-core 1.0 GHz Cortex-A7
ROM	4 GB
RAM	512 MB
Wi-Fi	Wi-Fi 802.11 b/g/n
Battery	Li-Ion 250 mAh

The rest of the test environment is shown in Figure 4.3. The test environment was fully automated with a test automation instance running on top of nodejs that controlled the run of Wireshark by starting and stopping the tshark process so that tshark could monitor all packets flowing through the middleware node.js application, PONTE and the watch. The operating system used to host the automated environment was Ubuntu 16.4, which runs as a virtual machine (VM). This VM also runs the PONTE message broker instance. The host Ubuntu VM was connected to a router through an Ethernet cable. This is achievable in VMWare by bridging the Ethernet network of the host machine to the VM. The CouchDB NOSQL database also runs in the same VM, where the CouchDB instance saves all the watch logs described earlier.

The SGS2 smartwatches were being controlled remotely through Samsung’s SDB commands to start the application automatically on the targeted watches every episode using a test automator, which was written to read from an input excel file that contains each test episode’s definition. The episode definition contains the number of watches, the targeted network condition, 100% or 0% hit scenario, episode number and T1.telemetrySubmission transaction wait interval which will be explained later in this chapter, then the test automator initializes a server instance of the middleware and context processor with the targeted variables read from the test input sheet and start up the watches targeted in each episode. Figure 4.2 shows the automated test running on the watches.



Figure 4.2: Test automator starting up the wearable school application automatically on the SGS2 watches.

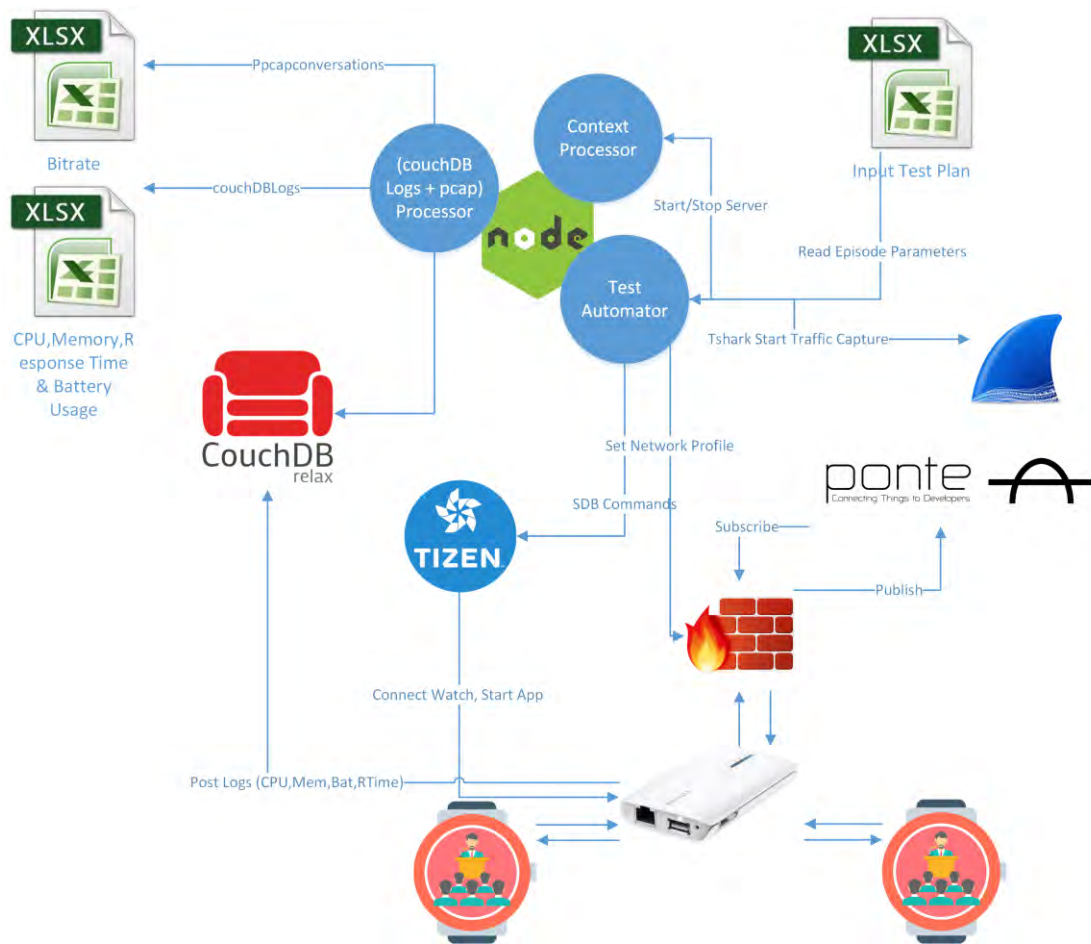


Figure 4.3: High-level diagram of automated testing environment

While initiating the performance analysis for each test episode, after the test run was finished on a certain network condition, the output Wireshark PCAP file was processed through a pcap2csv library created in Shapsough's research [39]. The CouchDB json documents were all processed through a custom node.js application written to produce an Excel file that combines all required data in different worksheets in order to perform statistical analysis on the data output in tabular/relational format. The exceljs library [42] was used to produce the Excel file output, which is well documented and provides a familiar API for those who understand the Excel language if we compare it with the xlsx library [43], but xlsx library offers the advantage of being able to run in browsers, something that was unnecessary in the use case.

To shape the network traffic tcconfig [44], the network shaper application was used. This application wraps tc and IPTables to limit bandwidth, produce delay, packet reorder, and packet loss to simulate different scenarios that involve poor network conditions. It is also automated through the test automator, and different network profiles can be predefined and chosen during the test run, depending on the test input sheet and which episode is being run in the sheet. Many other alternative solutions to limit network traffic were attempted. Clumsy has some limitations with respect to limiting the network bandwidth and it also only runs on Windows. However, it was required that the limiter run on Linux rather than on the PONTE broker instance. Cellular-Data-Network-Simulator [45] runs on top of OpenWRT, but there wasn't a compatible router with OpenWRT V14 available to run the application properly: ipfw was found to be obsolete and only runs on the FreeBSD version of Linux. All new distributions of Linux use IPTables and TC instead. Use of augmented traffic control (ATC) [46] was also attempted. However, it was difficult to use because it was designed for smartphones and its use depends on running the configuration page on the smartphone client side. With wearables, it is not possible to run the configuration page and select the network profile inside it. Therefore, tcconfig was the best alternative solution because it uses TC and IPTables and offers an easy-to-use command line to limit the network traffic. It was therefore easy to automate. The next section describes how the data was collected.

4.2.2. Data Collection. To produce data results that could be statistically analyzed, the logs were designed to be granular enough to aid analysis of the tests being performed. The following sections describe how logs were collected to provide data on response time, power, CPU and memory. The method of bitrate measurement is also described.

4.2.2.1 Response Time. To measure response time, the transaction time was logged together with details contained in the message being sent from the smartwatch to PONTE through to the middleware and all the way back to the smartwatch. Each endpoint appends the time a message is published or received from a subscribed topic. All of the endpoints were able to synchronize the system time with the worldwide NTP server except the watch, which synchronizes the time from the connected phone if available. Alternatively, the time can be set manually if the phone is disconnected. The phone synchronizes the time with the GSM network, which is approximately 31 milliseconds ahead of worldwide NTP time. This value varies from day to day, and in a single day the difference was observed to be around 1.03 seconds. The ClockSync Android application [47] was used to determine the exact difference in milliseconds. Therefore, to overcome the time difference, the time had to be set manually in the Android phone, the installed time phone server application [48] and the installed time sync application on the Windows server machine [49] to connect to the Android phone time through SNTP. After successfully syncing the time between the watch, the Windows machine and the Linux machine, tests were conducted by running the application a couple of times. Unfortunately, time sync in Tizen OS was affected by running the applications on it: the time was off by around 200 milliseconds following multiple application runs. Therefore, it was found that round-trip time of each transaction could only be tested on the watch itself since it was the most important endpoint in question and because the server machines and their network can be scaled in many ways while the watch is limited to the network conditions surrounding it. With the smartwatches, network conditions cannot be scaled or guaranteed because the students can be in any location while using the smartwatch application.

Therefore, the time was only measured at the watch end by making the watch log its transactions start and end times through a custom logger written to overcome another issue in the Tizen OS, specifically that the applications running in TizenOs

cannot be restarted without refreshing the entire page once it is running. A robust js library called log4javascript was initially used to log messages [50]. This library had a built-in option for an Ajax appender that queues log messages for a specified time or a specified number of logs. However, when the entire application page is refreshed, the queue that is recorded in the log4javascript library is lost.

To preserve the queue, attempts were made to save the log4javascript logger object to the HTML5 local storage. However a new obstacle was encountered in that the logger object contains cyclic objects and cannot be stringified so that they can be saved in local storage. Rather than finding a workaround for this issue, a simple logger was written to append messages to an array that could be saved and reloaded at each application restart during the test.

The queue size is set to 100 which results in sending the log data to the CouchDB server approximately every 80 seconds, depending on the network conditions. The better the characteristics of the network, the greater the number of transaction logs that can be generated within the same time period compared to bad network conditions.

4.2.2.2 Power, CPU and Memory. To measure the power drain, CPU load and memory occupied during the application runs, the values of CPU and memory were logged by setting a timer to collect this data every second.

Samsung offers an API that can callback CPU load changes only when the load value changes. However tests revealed that it can minimally report changes every second. Therefore, it was decided that both CPU and memory would be logged together every second in a single log line.

For power consumption, Tizen APIs offers a callback API “battery.onlevelchange” that only reports power changes when there is a power value change of one percent in terms of granularity, this API is based on HTML5 standards. All of the logs collected from the watch, including the transaction logs mentioned earlier, were queued in one stream to send the logs in batches, thus minimizing the battery drain caused by log collection.

4.2.2.3 Bitrate. To measure the bitrate between the watch and PONTE, Wireshark’s tshark process was used to collect the packets and filter them on port 1883 (MQTT) and 3000 (MQTT over Web socket). The outcomes were then exported through a modified version of pcap2csv that also outputs the episode number to get the bitrate per second between the watch and PONTE (and vice versa) and to perform statistical analysis on these measurements.

4.3. Experiments Definition

4.3.1. OnDemand Mode Experiment. In this experiment, all three of the transactions mentioned earlier were tested in “OnDemand” mode with multiple profiles of network conditions. The experiment was conducted on a single watch throughout all the profiles in the scope with one possible telemetry interval equal to 10 seconds. This results in only running 8 episodes for the on demand mode. Table 4.2 shows the independent variables that can be changed for each test run. Value changes were based on fixed profiles obtained from ATC [51]; these are shown in Table 4.3

Table 4.2: Independent Variables

Independent Variable	Description
Delay	Delay in milliseconds introduced to hold the packet
Loss	Percentage of dropped packets
Bandwidth Rate	Max allowed rate in kilobits per second

Table 4.3: Network Profiles

Network Profile	Download/Upload Delay in ms - Download/Upload bw in kbps - Loss %
2G-DevelopingRural	650/650-20/18-2
2G-DevelopingUrban	650/650-35/32-0
3G-Average	100/100-780/330-0
3G-Good	90/100-850/420-0
Edge-Average	400/440-240/200-0
Edge-Good	350/370-250/200-0
Edge-Lossy	400/440-240/200-1
Phy-Wi-Fi-baseline	0/0-0/0-0 (Physical network limitations only, no additional delay/loss to be added)

Note that there are other parameters that can be controlled to shape the network traffic: out of order and duplication and corruption. These parameters are beyond the scope of these experiments because they are not related to the network

profile tested. Rather, the values are related to the routing techniques used in an ISP. The research may be extended to test these parameters in the future. The network profile settings mentioned in Table 4.3 were made for use with a network limiter tcconfig [44]. Every network profile was combined with different control variable values. Each combination, which is called an “episode”, was executed for 30 minutes and the collected data was analyzed in terms of the dependent variables listed

Table 4.4: Control Variables

Control Variable	Description	Range
Timeout (on demand only)	Timeout in the event that not all sensors have finished their data collection	10 seconds fixed
Human Input Delay	Simulated time required for humans to select and submit in all screens; this delay is only applicable for the hit scenario	17 seconds total fixed
Wait Time (on demand only)	Waiting time between 2 runs, which will be fixed to 0	0 Seconds fixed
Hit/Miss Percentage	Hit ratio for returning eligible questions	100% hit fixed for Experiment A 0% and 100% hit for Experiment B
# of Eligible Questions	# of question that will meet the context eligibility criteria (light sensor scenario)	1 only fixed
Episode Time	Total time to complete test run	30 minutes fixed
Frequency of Telemetry Data Transmission (continuous only)	Number of seconds required for each telemetry message to send to context processor on logarithmic scale	10, 20, 40, 80, 160 seconds
Wait Time for Telemetry Collection (continuous only)	Number of seconds required between each sensor value read on change	0 seconds fixed
Watches	Number of watches 1-8	1 then 8
# of Questions to Scan	Number of questions the context processor must scan for every telemetry message received to match for eligibility	100

TABLE 4.5: Dependent Variables

Dependent Variable	Description	Unit
Response Time	Time taken to run each transaction type	millisecond
CPU Max	Maximum CPU Load while running each point in an episode	%Load
%CPU-Seconds	Total CPU percentage used while running each point	%Load x Seconds
Power Drain	Percentage lost during an episode run	Percentage per episode run
Memory Availability	Amount of available memory use during the episode run	Mega bytes
Bitrate	Bitrate collected from Wireshark conversation view	bits per second

4.3.2. Continuous Mode Experiment. This experiment tests running the question inquiry and answer posting in “Continuous” mode.

The authentication transaction occurs only once. It was excluded from the data analysis. In contrast, the question inquiry and answer posting run multiple times

during the test. The test will be run on a single watch. The control and independent variables are shown in Table 4.2 and Table 4.4. Variables not relevant to continuous mode are appropriately flagged with “OnDemand” in these tables.

While testing the continuous mode, if the test is concerned with the data integrity of the answer posting transaction, the answer posting result can be changed by controlling a light source that can be programmed to turn on and off at specific intervals. The easiest, most straightforward way to do this was to install Tasker [52] and add a task that turns on the LED light. The Android phone runs Tasker every two minutes. Tasker has a plugin that integrates it with a tiny LED application [53] to accomplish this task. This test can be performed as a potential extension of this research, but the message size for correct and wrong answer posting is nearly identical, and our focus was to test the performance aspects of the platform, so it was omitted in the current project.

When the tests for the continuous mode are being performed, they will vary in the interval of sending the telemetry data, in a logarithmic scale from 10 seconds to 160 seconds, every interval. The sensor data is stored in an array until the interval is met then the message is sent. This leads to different message sizes occurring in each test episode. These sizes are estimated in the Table 4.6

TABLE 4.6: Estimated T1.telemetrySubmission message size for each interval

T1.telemetrySubmission Interval (s)	Estimated Size (KB)
10	6.11
20	11.97
40	23.88
80	47.53
160	95.24

A total of 96 episodes were run for the single watch continuous mode tests, 48 episodes with 100% hit scenario where in every interval a question has been received and 48 episodes with 0% hit scenario where no questions have been received throughout the episode run, each group of 48 episodes were a combination of 8 network profiles and 6 different intervals of T1.telemetrySubmission interval length. While performing the tests, it was found that the interval of 320 seconds produces very few points that could not be properly statistically analyzed, so the 320-second interval containing 16 episodes has been omitted from all of the test results to bring down the episodes under analysis to 80.

4.3.3. Multiple Watches Experiment. This experiment tests running the logarithmic scale of the number of watches running in parallel, which are 1,2,4,8. However, the scope of the network profile combinations will be limited to only 3g-average and Wi-Fi profiles, the goal of this experiment is to assess the response time impact and server load impact, if any, when the number of watches increase. All of the tests will be based on the continuous mode with a 10 seconds interval for the T1.telemetrySubmission transaction. Table 4.8 shows the episodes that will be run for this test.

Table 4.7: Episodes run for the multiple watches experiment

Episode ID	Mode	Hit Percentage	Interval Telemetry Data Transmission (s)	Watches	Network Profile
EP109	continuous	0	10000	1	Phy-wifi-baseline
EP110		0		1	3G-Average
EP111		100		1	Phy-wifi-baseline
EP112		100		1	3G-Average
EP117		0		2	Phy-wifi-baseline
EP118		0		2	3G-Average
EP119		100		2	Phy-wifi-baseline
EP120		100		2	3G-Average
EP113		100		4	Phy-wifi-baseline
EP114		100		4	3G-Average
EP115		0		4	Phy-wifi-baseline
EP116		0		4	3G-Average
EP105		0		8	Phy-wifi-baseline
EP106		0		8	3G-Average
EP107		100		8	Phy-wifi-baseline
EP108		100		8	3G-Average

Chapter 5. Results and Analysis

For each of the experiments, multiple episodes were run and data was collected using the techniques described earlier. This section presents the results and details of the statistical analysis for both experiments performed on a single watch (OnDemand and Continuous) and on multiple watches in continuous mode.

5.1. Single Watch – OnDemand Mode Experimental Results.

This experiment was only performed on a single watch. There was also only one possibility of interval length for the wait time to collect data and send the message, which was 10 seconds. The results were analyzed for each network profile in the OnDemand mode for the independent variables described earlier in the previous chapter.

5.1.1. Response Time. As shown in Figure 5.1, which presents a graph of the average response time for each network profile and each transaction type, it was observed that the authentication transaction takes more time to complete the round trip than the telemetry submission transaction. This was an unexpected result comparing the amount of data being sent and processing time that has to be done in the context processor at the server side before sending the result back to the watch, but this additional delay in the authentication transaction can be explained by the time required for the watch to initialize application and for the authentication procedure to be completed.

The network profile greatly impacts the resulting response time for both transactions if we compare the extreme cases of 2g rural vs Wi-Fi. Under the worst network condition, the response time is highly acceptable since it is generally lower than 3 seconds under all network conditions.

For more detailed descriptive statistics for the response time of three types of transactions being measured (T0.authentication, T1.telemetrySubmission, and T2.answerPosting) Table 5.1, Table 5.2 and Table 5.3 can be referred to respectively.

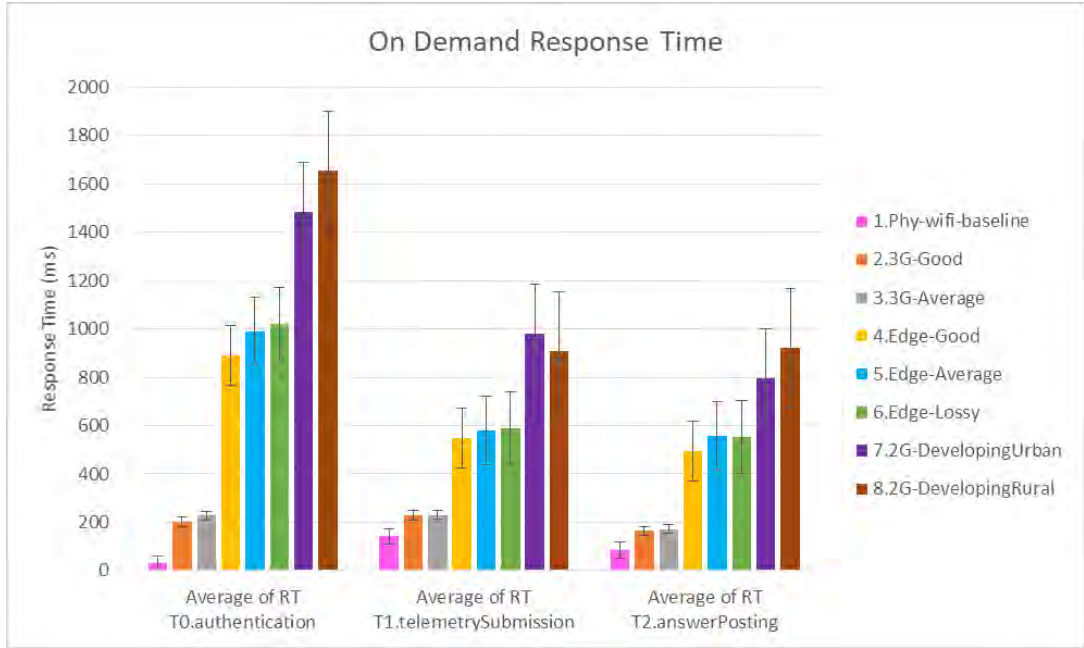


Figure 5.1: OnDemand average response time per network profile

Table 5.1: OnDemand descriptive statistics for the T0.authentication transaction

Network Profile	mean	median	min	max	SD
2G-DevelopingRural	1688.52	1474.50	1344.00	5230.00	679.36
2G-DevelopingUrban	1521.44	1488.00	1340.00	1815.00	105.76
Edge-Average	1009.56	983.00	857.00	1400.00	107.02
Edge-Good	905.00	874.00	754.00	1182.00	105.48
Edge-Lossy	1020.78	980.50	848.00	1821.00	154.63
3G-Average	227.05	225.50	184.00	279.00	16.19
3G-Good	207.68	205.50	176.00	267.00	15.03
Phy-wifi-baseline	30.43	28.00	13.00	59.00	10.78

5.1.2 CPU Usage, Bitrate, Power Drain and Memory Availability. When analyzing the distribution of max CPU, it was observed that the CPU load was almost maxing out at any point of time in the OnDemand mode and was averaging 95 for the Wi-Fi network profile and 96 for the 2g rural network profile. This needs to be addressed by optimizing the code in the OnDemand mode.

So we have ruled out performing statistical conclusions for the CPU load. Some of the observations were that the network profile does affect the CPU max on a

small margin. Also an increase in CPU time was observed when sending the data over the network under poor network conditions.

Table 5.2: OnDemand descriptive statistics for the T0.telemetrySubmission transaction

Network Profile	mean	median	min	max	SD
2G-DevelopingRural	908.86	819.00	762.00	2460.00	322.05
2G-DevelopingUrban	982.03	869.50	788.00	2446.00	329.41
Edge-Average	579.82	563.00	470.00	963.00	82.78
Edge-Good	550.05	527.00	437.00	892.00	91.84
Edge-Lossy	588.44	571.50	475.00	896.00	78.04
3G-Average	230.86	224.00	191.00	345.00	32.39
3G-Good	229.46	213.00	162.00	429.00	52.70
Phy-wifi-baseline	141.49	135.00	57.00	324.00	41.67

Table 5.3: OnDemand descriptive statistics for the T2.answerPosting transaction

Network Profile	mean	median	min	max	SD
2G-DevelopingRural	920.98	779.00	703.00	3572.00	534.20
2G-DevelopingUrban	797.25	784.00	709.00	1080.00	69.97
3G-Average	172.25	158.00	139.00	357.00	36.71
3G-Good	164.17	149.00	129.00	246.00	35.14
Edge-Average	558.40	510.00	445.00	949.00	121.46
Edge-Good	493.75	467.00	397.00	756.00	87.86
Edge-Lossy	551.91	530.50	447.00	1453.00	149.48
Phy-wifi-baseline	85.48	55.00	32.00	799.00	121.50

As for memory availability, bitrate and power drain, statistical conclusions cannot be drawn from the data collected because the timespan of each run is very short and the CPU was busy from start to finish for each point of the test episode. Analysis of these parameters was done later in the continuous mode because more data was available for analysis and there were gaps of different intervals that could be used to generate different resource usage patterns.

5.2. Single Watch – Continuous Mode Experiment Results

5.2.1. Response Time. When comparing the average response time for all three transactions, it was observed that the response time was linearly increasing whenever the frequency of sending the T1 message increased. This is due to queuing more telemetry data collected between intervals as per Table 4.6. As expected, the Wi-Fi profile had the lowest impact and 2g rural had the highest impact, Table F.1 in Appendix F can be referred for exact figures in average response time. This trend in the response time increase indicated that frequencies of more than 40 seconds should not be used to satisfy the worst network conditions and stay below 3 seconds of response time on average.

It was also observed in Figure 5.2 that the lossy edge connection delay equated to the 2g network at the 160 interval. This was explained by the higher possibility of packet retransmission for larger MQTT messages.

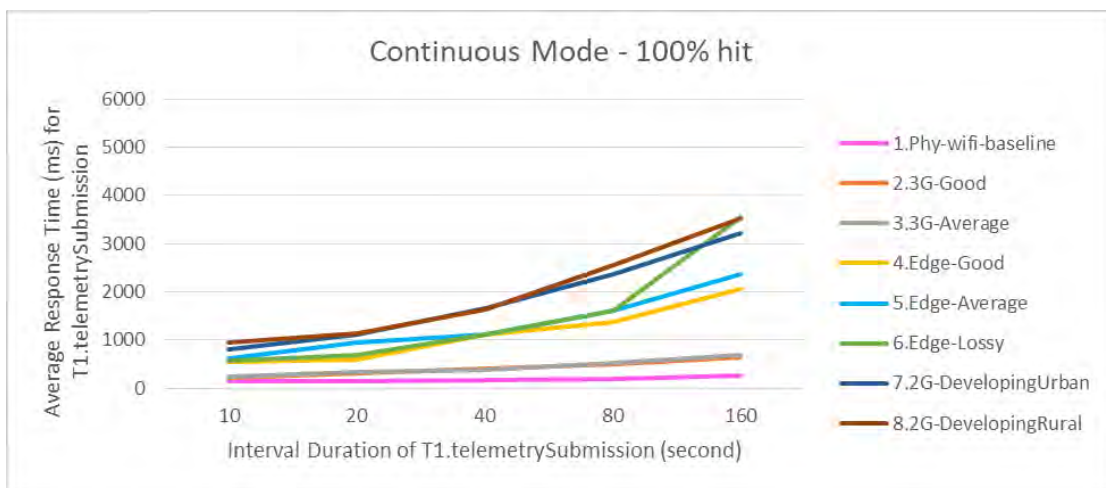


Figure 5.2: Continuous mode line chart, 100% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval

When comparing the 0% hit and the 100% hit, there was no clear difference in the averages because the conditions for the telemetry submission transaction were the same in both of these scenarios, but it can be observed that the 100% hit averages were lower than the 0% hit averages when Figure 5.3 and Figure 5.4 are compared.

In Figure 5.4, it was observed that it was better to limit the T1 message sending interval to less than 40 seconds in order to satisfy all network conditions and stay within the average response time of 1 second.

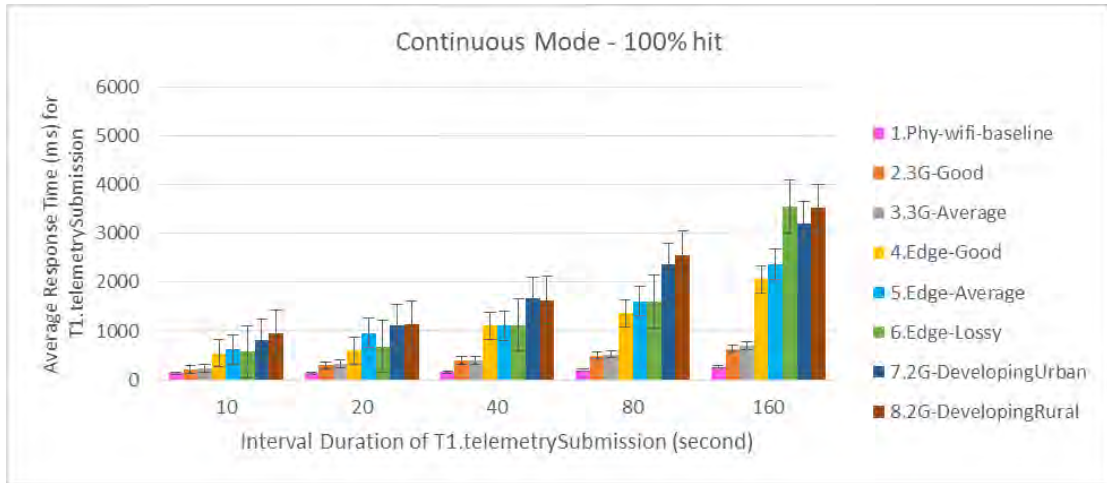


Figure 5.3: Continuous mode line chart, 100% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval

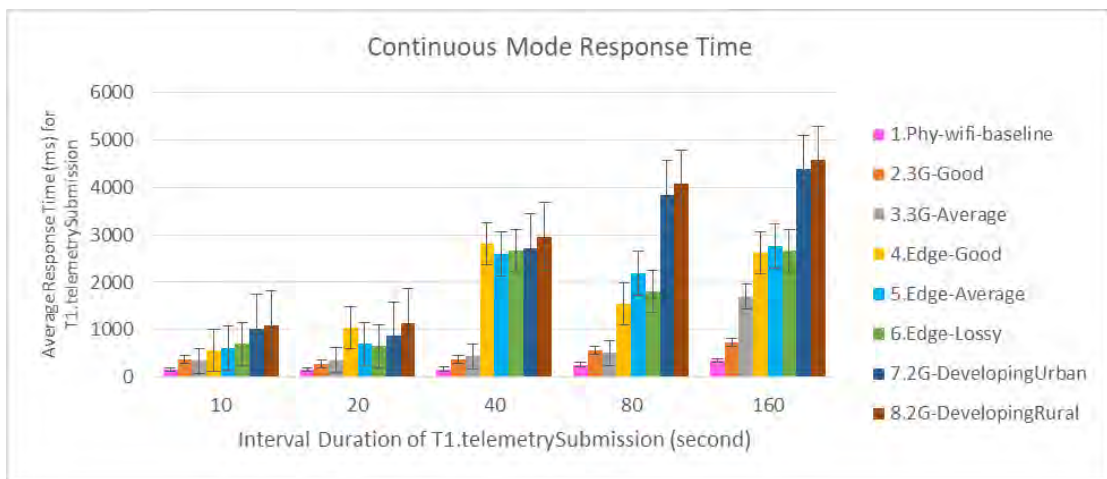


Figure 5.4: Continuous mode bar chart, 0% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval

Figure 5.5 shows that the response time trend was flat amongst the different telemetry intervals for the answer posting transaction. This was because the answer posting message size was not affected by the interval of the telemetry submission. The separation of the response time averages for each network profile is also shown.

Figure 5.6, shows the average response times of the three groups: 3g and Wi-Fi are in the first group, edge is in the second group and the 2g is in the 3rd group. But we notice the 100% hit scenarios were having lower averages for the 80 seconds interval if we compare Figure 5.6 and Figure 5.10.

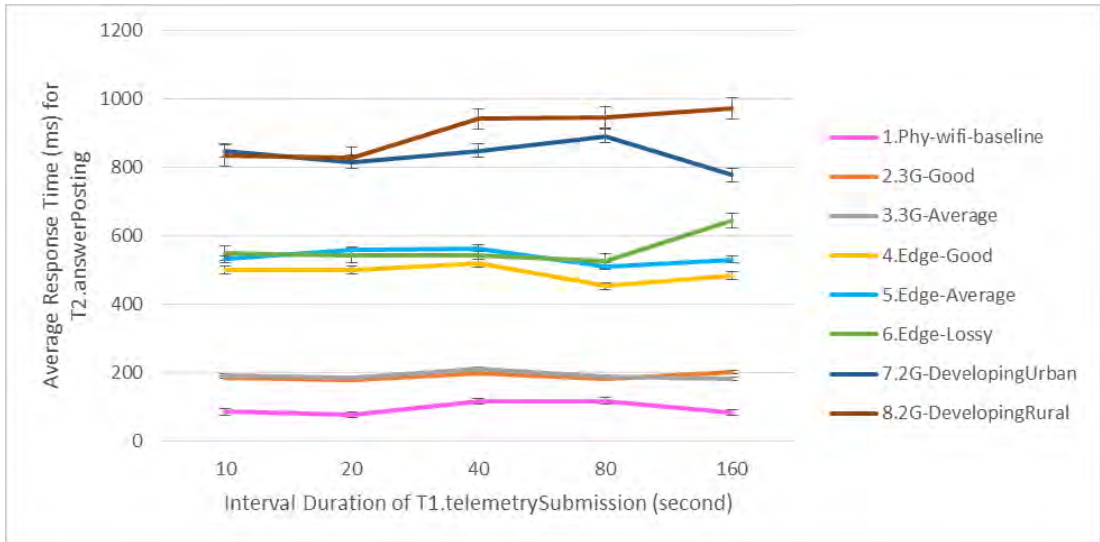


Figure 5.5: Continuous mode, 100% hit scenario average response time of T2.answerPosting per network profile and T1.telemetrySubmission interval

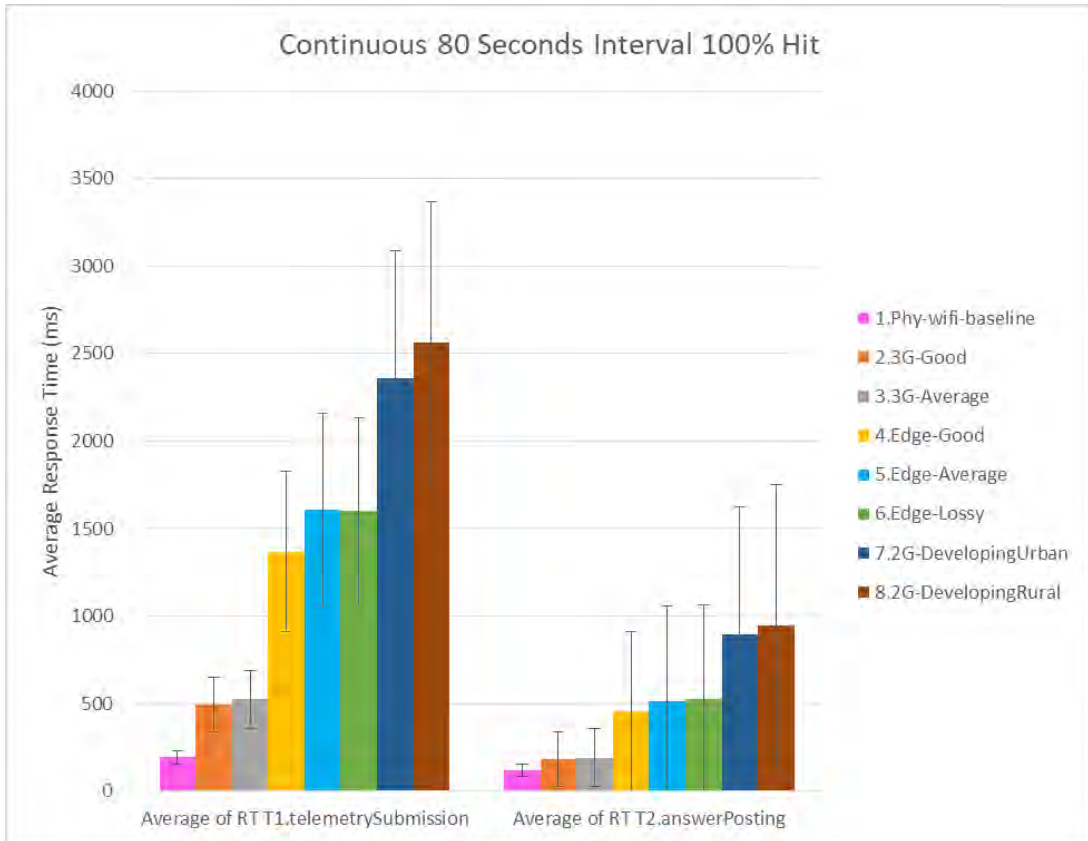


Figure 5.6: Average response time per network profile for each transaction type running in continuous mode in 80 seconds interval of T1.telemetrySubmission and 100% hit scenario

Figure 5.7 shows how the lossy network affected the stability of the response time of the 40-second interval. Also 2g-rural network is having the same effect because 2g-rural has the same percentage of loss equal to 2 percent.

Also if we compare the lower performing networks to the higher performing network, we notice the range in lower performing network is higher.

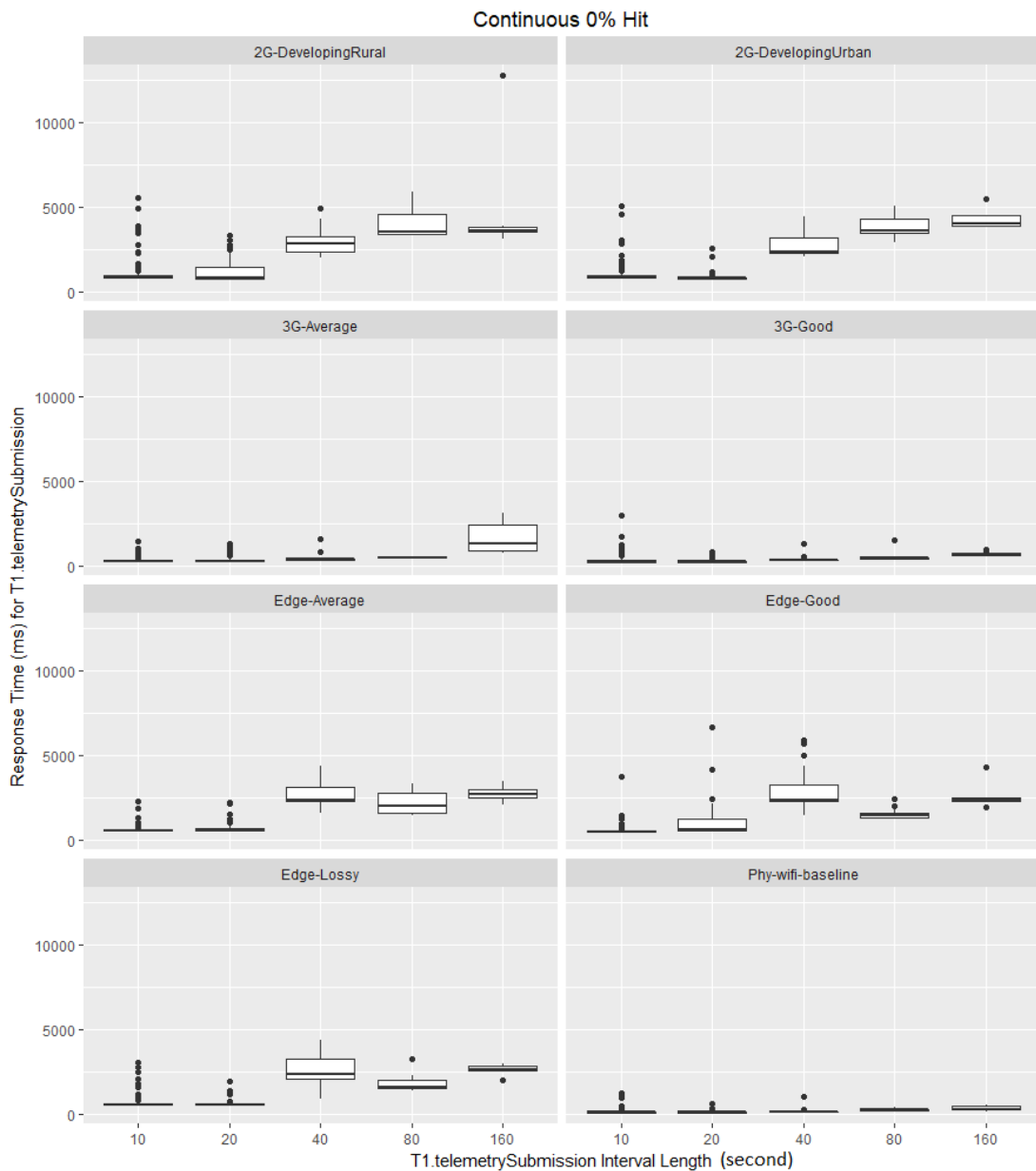


Figure 5.7: Box plot of response time for telemetry submission transaction for each interval and each network profile with 0% hit scenario

Also in Figure 5.8, the box plot shows that the lossy edge network lead to non-stable response times for intervals of 160 seconds.

Similar observation can be seen if compared to 0 percent hit which is expected that the ranges in the lower performing network profiles are higher than the better performing network profiles.

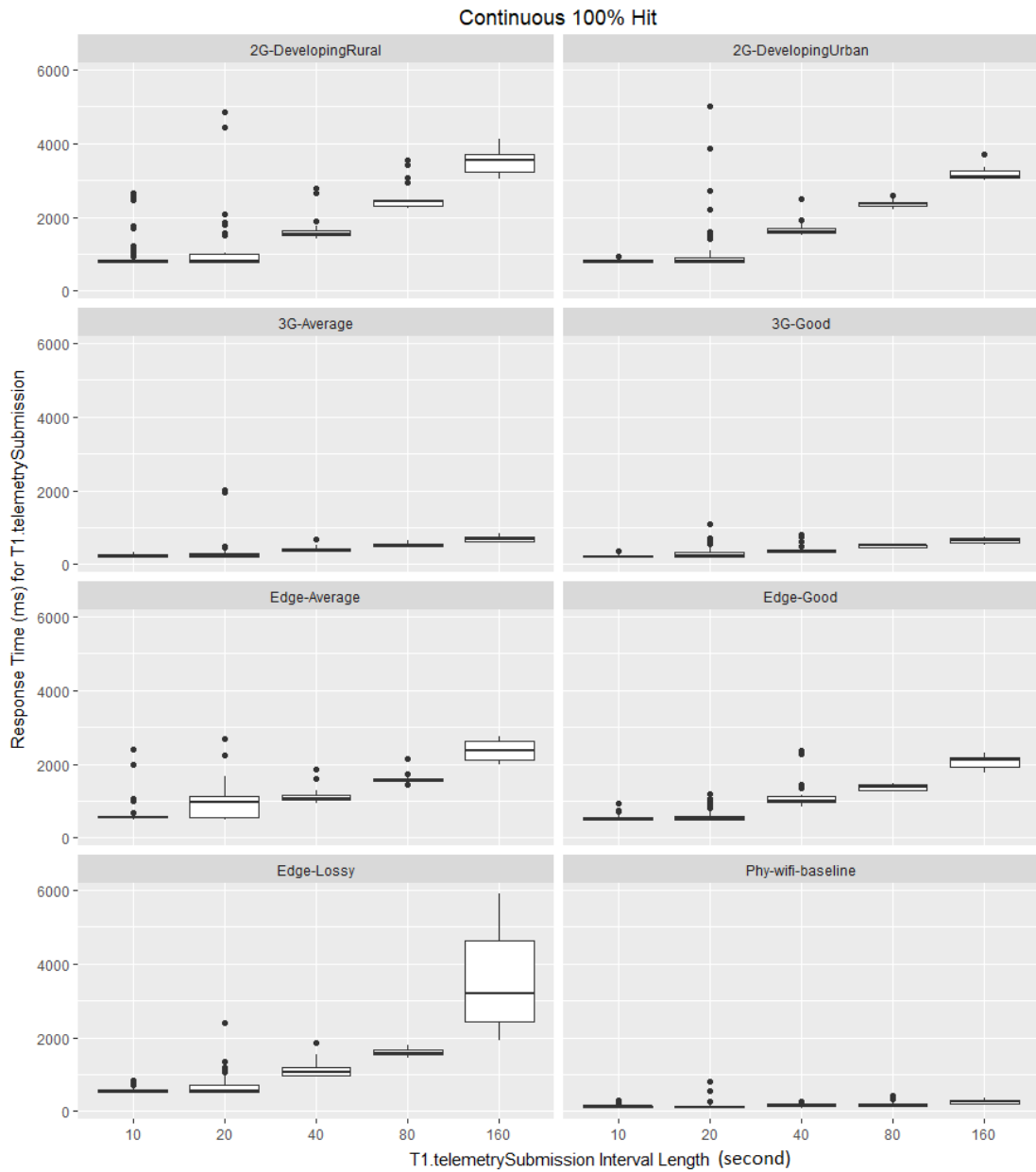


Figure 5.8: Box plot of response time for telemetry submission transaction for each interval and each network profile with 100% hit scenario

Referring to Figure 5.9, a strange behavior was observed in the Wi-Fi scenario: the ranges were quite high. This may have occurred because the watch was sometimes slow to respond in terms of application operation due to background running operations, so the delay in response from the watch itself may have contributed to the longer response times in the answer posting transaction.

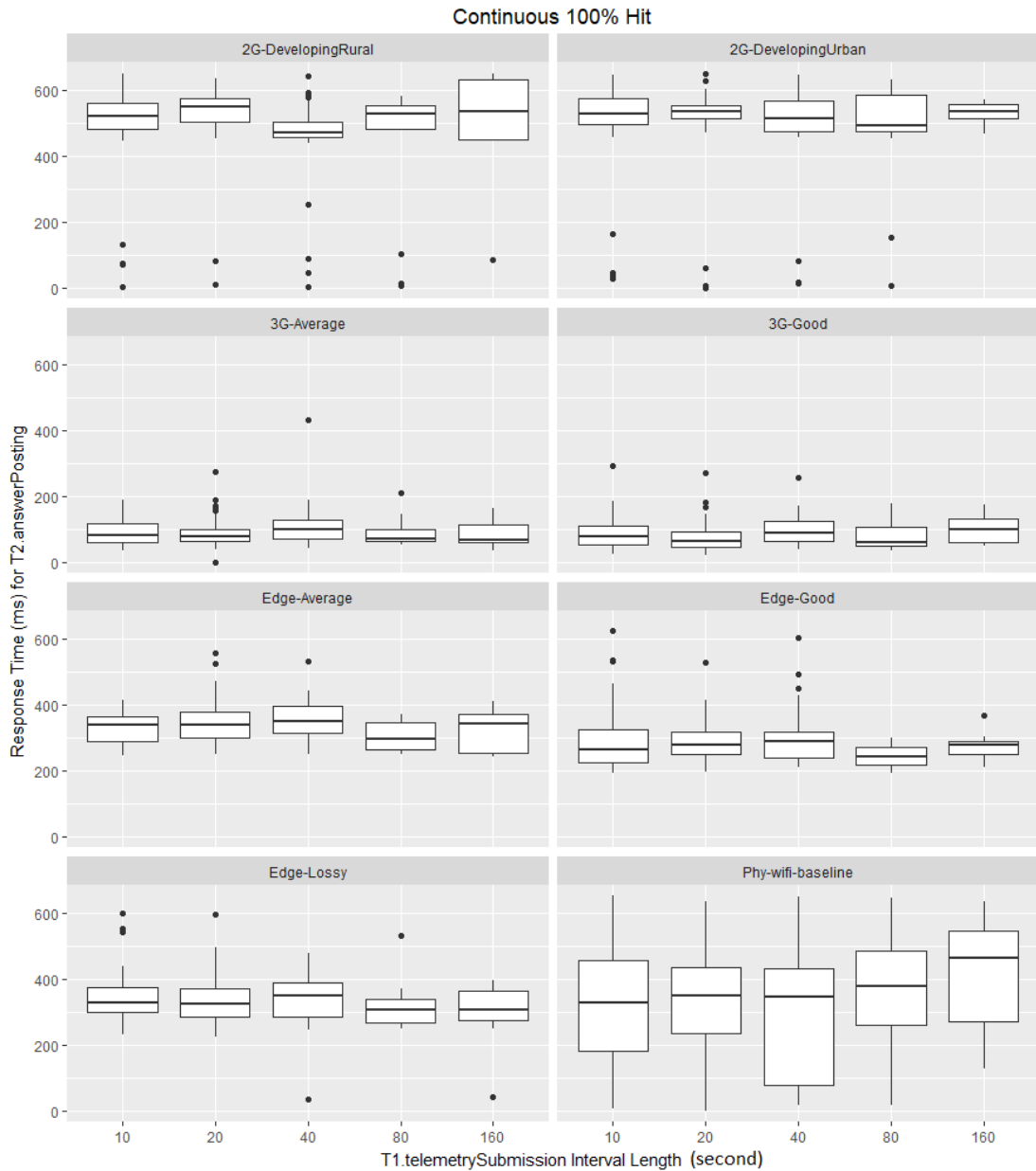


Figure 5.9: Box plot of response time for answer posting transaction for each interval and each network profile with 100% hit scenario

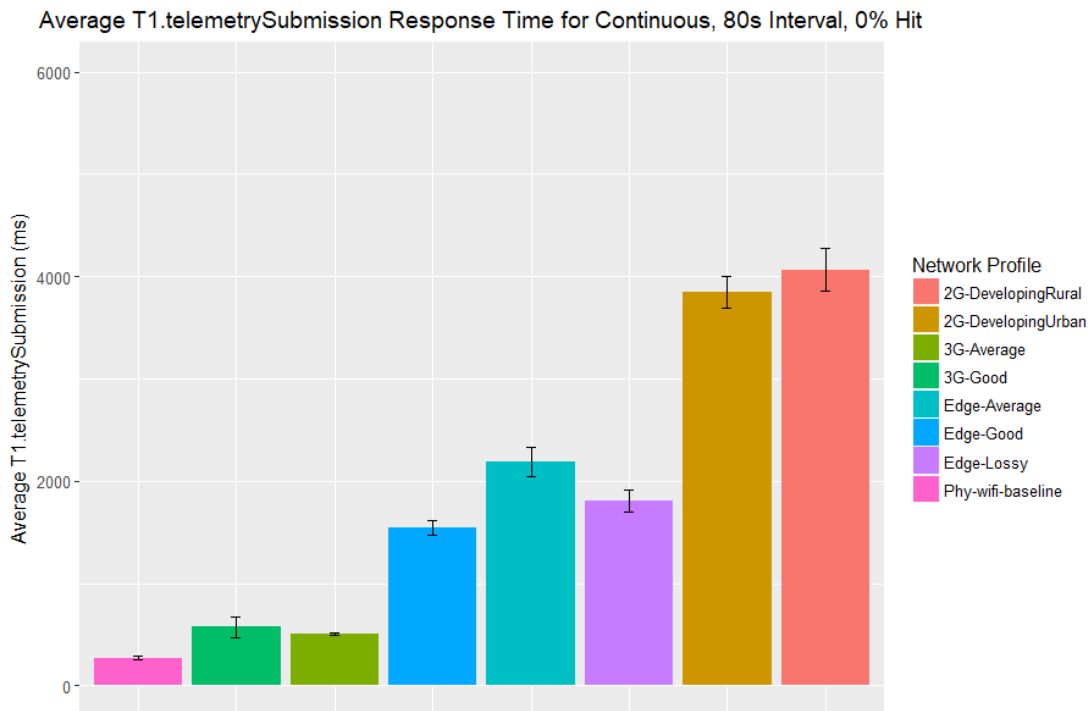


Figure 5.10: Average response time per network profile for T1.telemetrySubmission transaction running in continuous mode with 80-second interval of T1.telemetrySubmission and 0% hit scenario

5.2.2. CPU Usage. In Continuous mode, the CPU max percentage did not rise consistently above 90 percent, unlike the OnDemand mode where it average between 95 and 96 percent depending on the network condition. However, this observation is only applicable to the 0% hit scenario in continuous mode. In the continuous mode, from the CPU usage plot presented in Figure 5.11, we notice the CPU is fluctuating between 30% and 90%.

We can also notice from the CPU usage plots in Figure 5.11 and Figure 5.13, that the trend in max CPU usage shows how Wi-Fi is better than 2g. However, in both Wi-Fi and 2g, from Figure 5.12 and Figure 5.14 we notice the 100% hit scenario had higher max CPU percentages overall due to operations of moving between pages and rendering these pages occurring in the hit scenario. Appendix C contains additional CPU usage plots with more network conditions for reference.

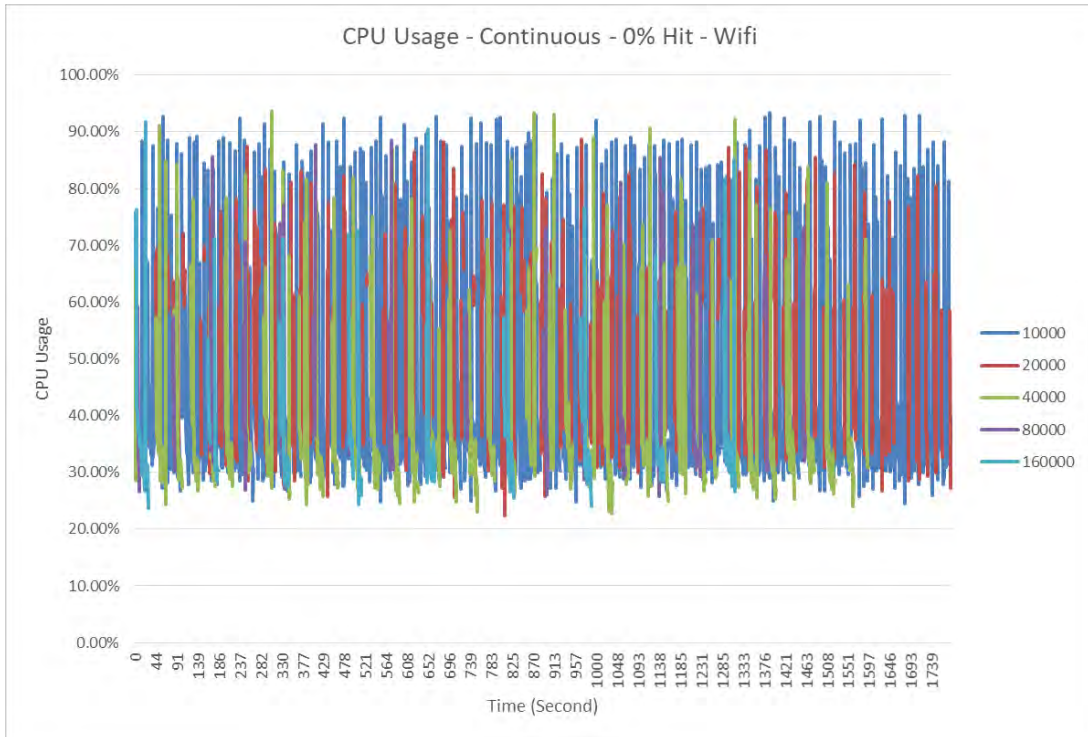


Figure 5.11: Continuous mode CPU usage within the 30-minute episode run with Wi-Fi network profile and 0% hit scenario

However in Figure 5.12 CPU usage increases above 90% due to rendering different pages in the 100% hit scenario

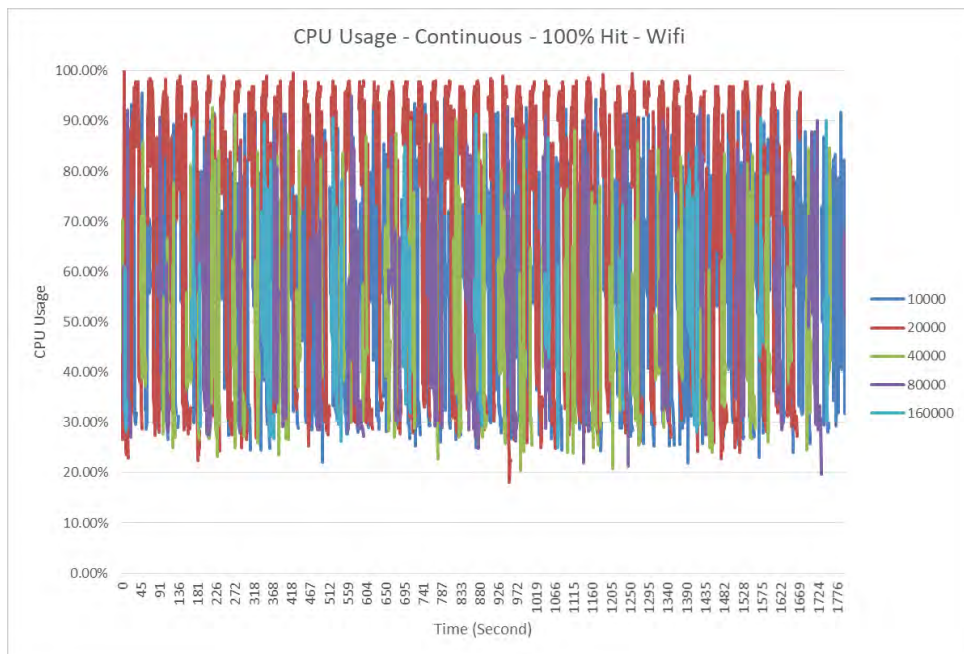


Figure 5.12: Continuous mode CPU usage within the 30-minute episode run with Wi-Fi network profile and 100% hit scenario

Figure 5.13 shows that the worst network profile have affected the CPU usage to increase also above 90%. And also in Figure 5.14 the 100% scenario has the same effect of higher CPU usage

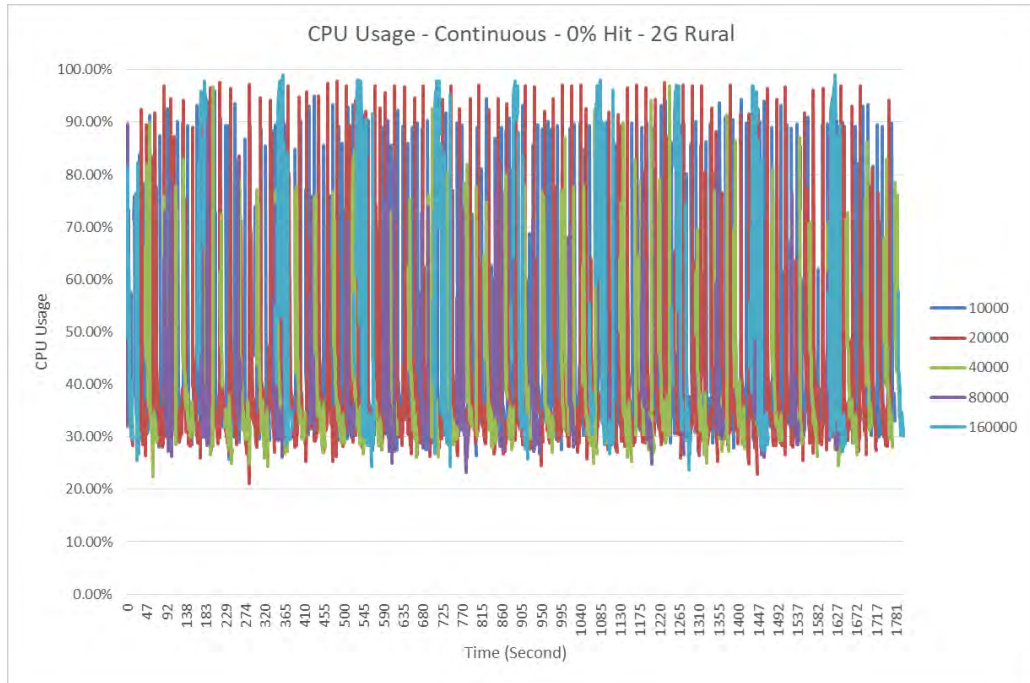


Figure 5.13: Continuous mode CPU usage within the 30-minute episode run with 2G rural network profile and 0% hit scenario

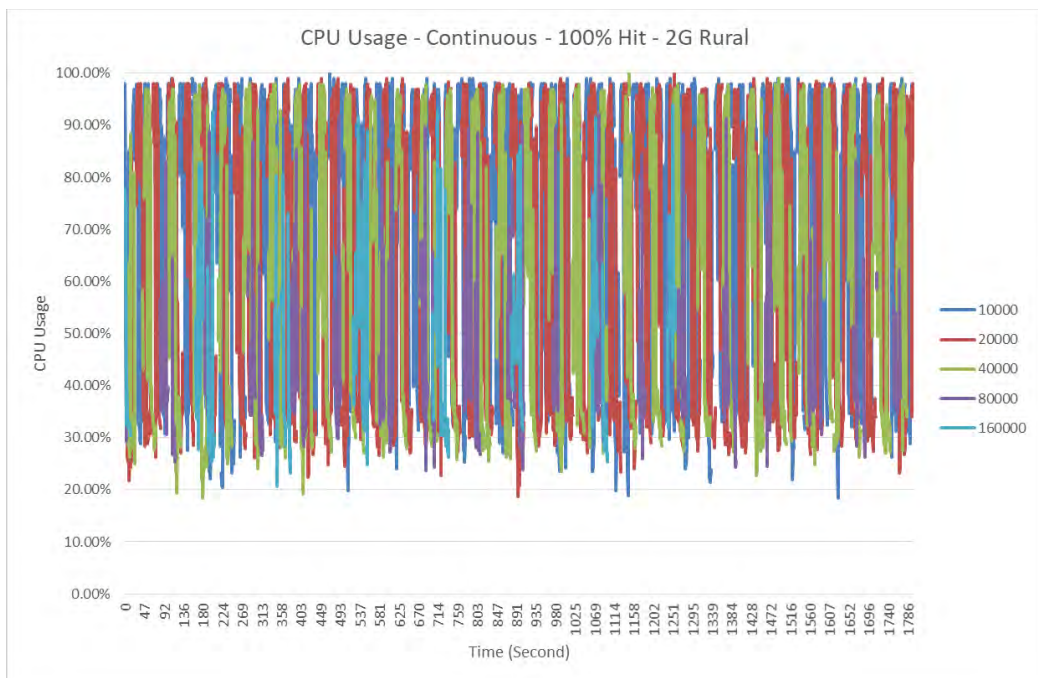


Figure 5.14: Continuous mode CPU usage within the 30-minutes episode run with 2G rural network profile and 100% hit scenario

In Figure 5.15, we notice the instability of the box plot position in some of the episodes, turning on the watch screen and rendering different pages can be a cause for this observation, but also the smartwatch has some background processes like the goal achievement notification that shows how far you have reached your goal of steps for today.

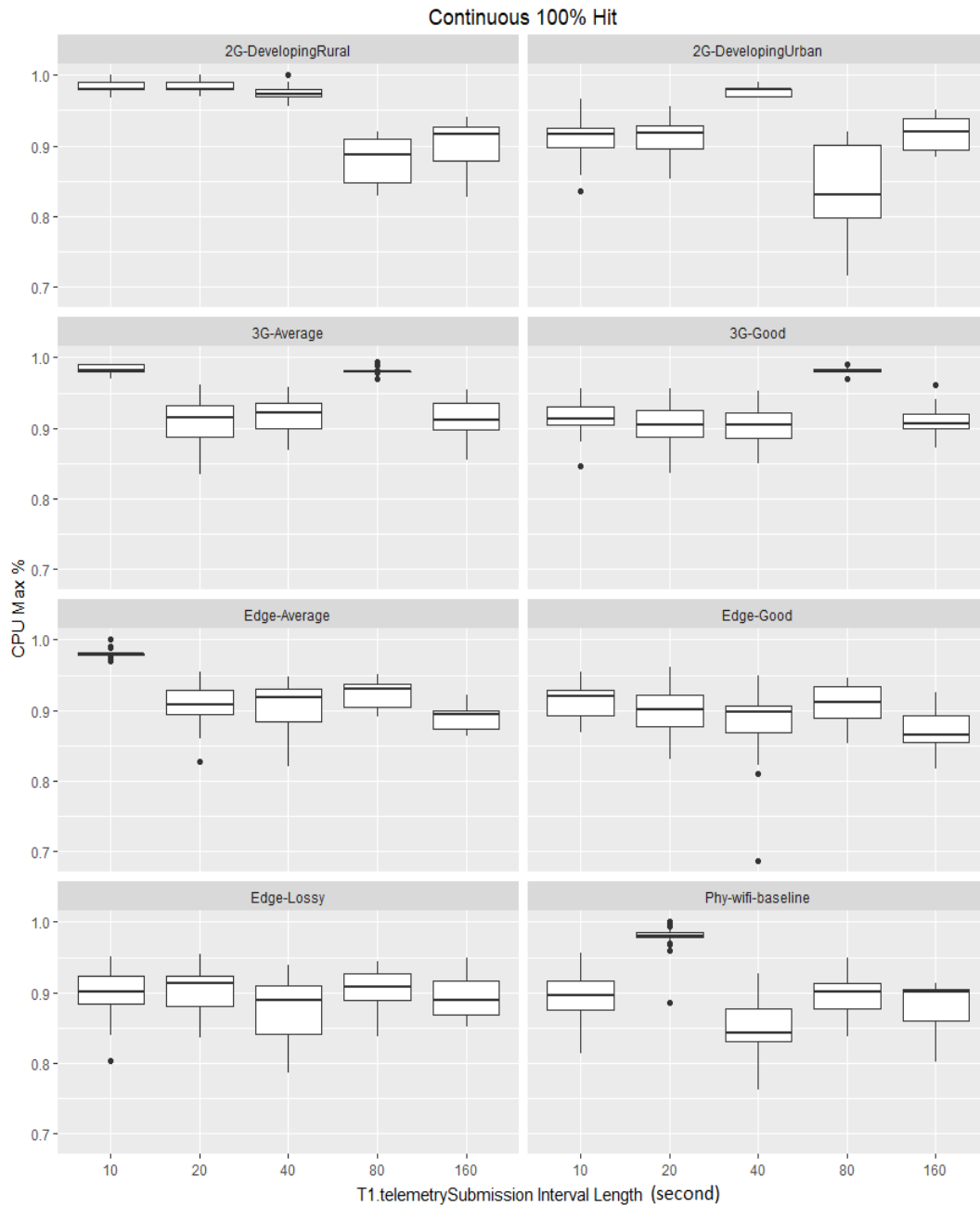


Figure 5.15: Box plot of CPU Max for telemetry submission transaction for each interval and each network profile with 100% hit scenario

In Figure 5.16, which shows the CPU max for the 100% hit scenario, shows that the CPU max in the low interval scenarios rose above 90% because the CPU was busy rendering questions and answers.

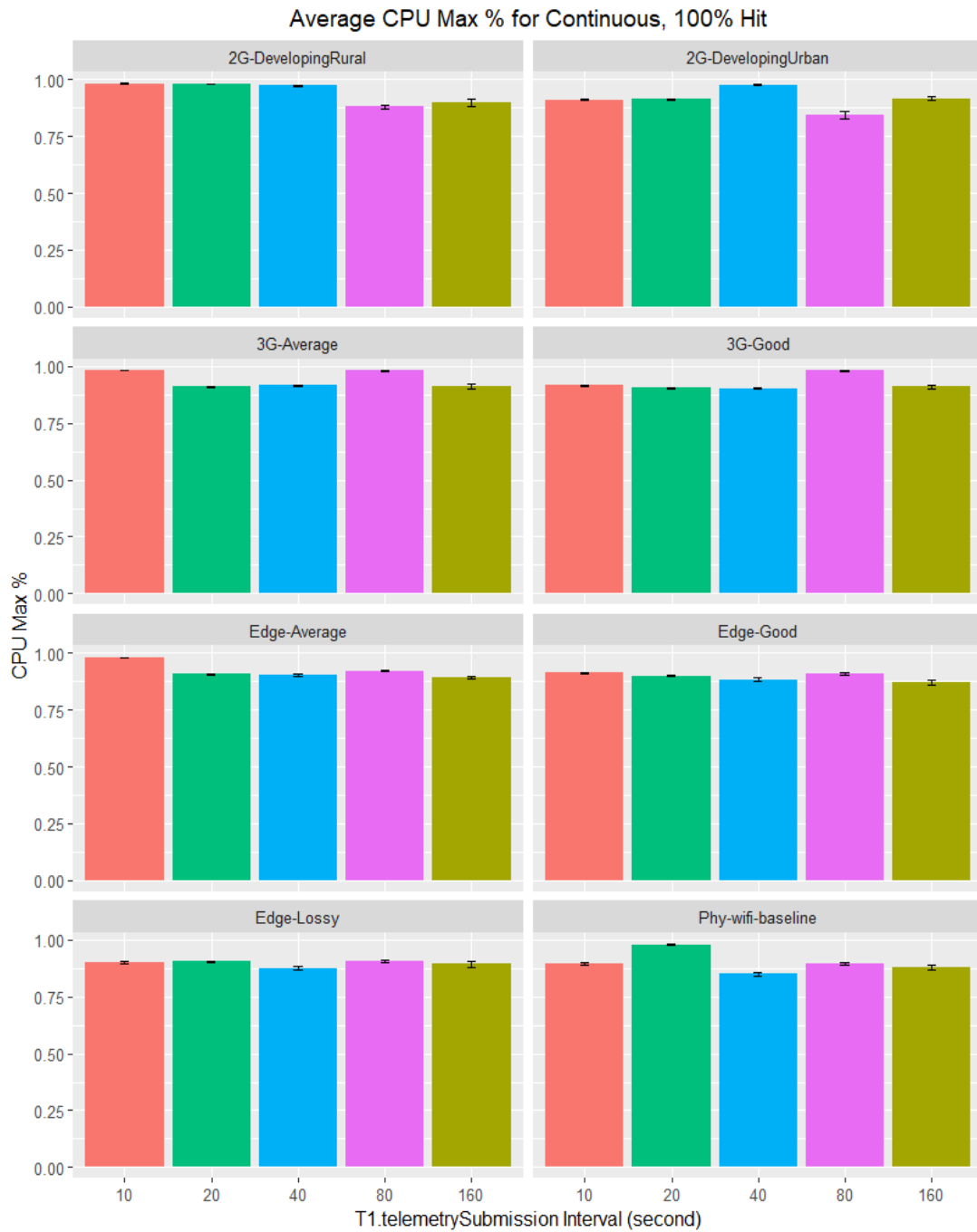


Figure 5.16: Average bar chart of CPU Max for telemetry submission transaction for each interval and each network profile with 100% hit scenario

In Figure 5.17, we notice the network profile did not affect the range of CPU max values as the box height and position is almost similar from one network profile to another, but we can notice the position change when moving from one interval to another.

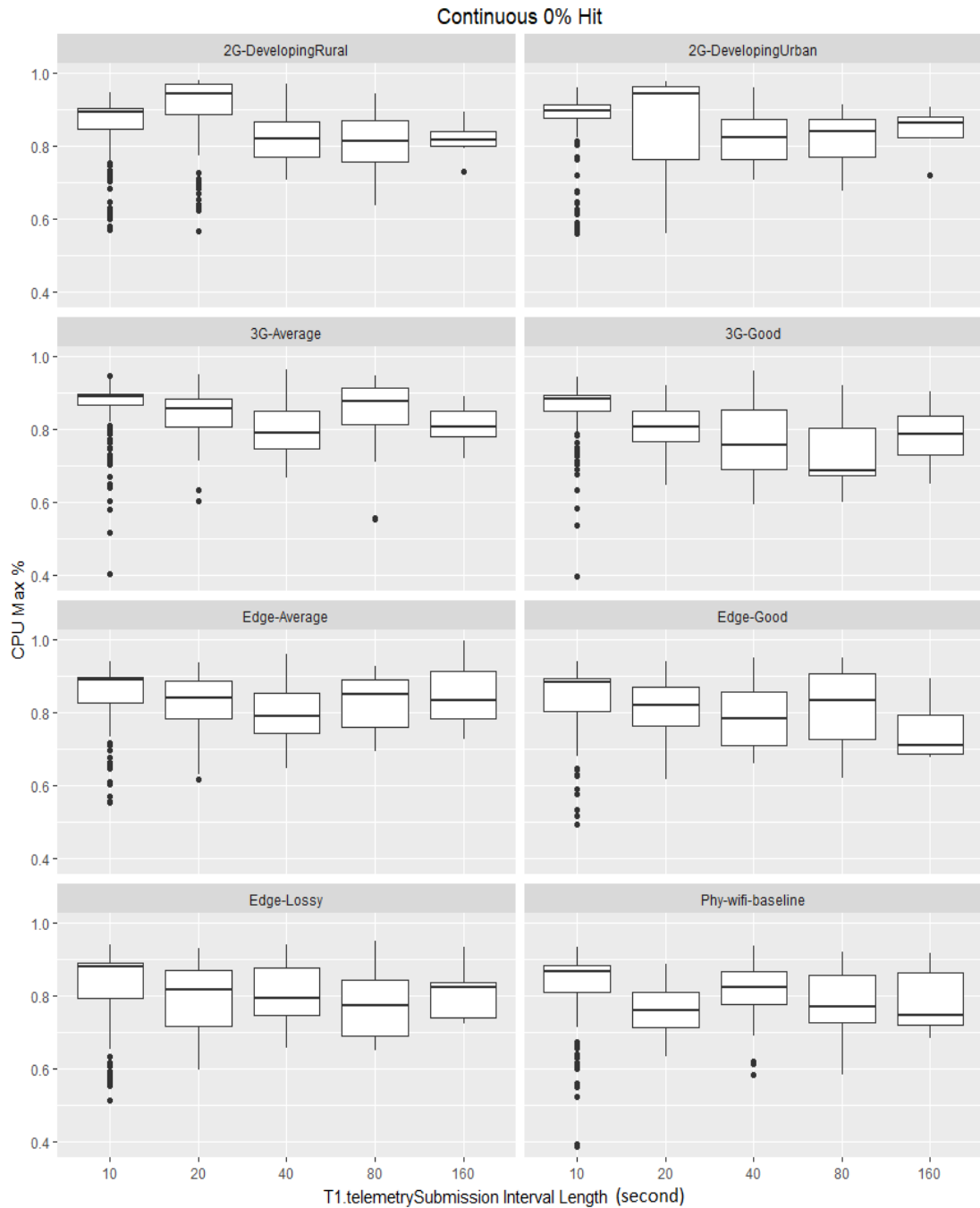


Figure 5.17: Box plot of CPU Max for telemetry submission transaction for each interval and each network profile with 0% hit scenario

In Figure 5.18, it was observed that network profiles did affect the CPU max percentage. Note that 2g had a CPU max of 85% while Wi-Fi had 76% for the 0% hit scenario in some of the intervals.

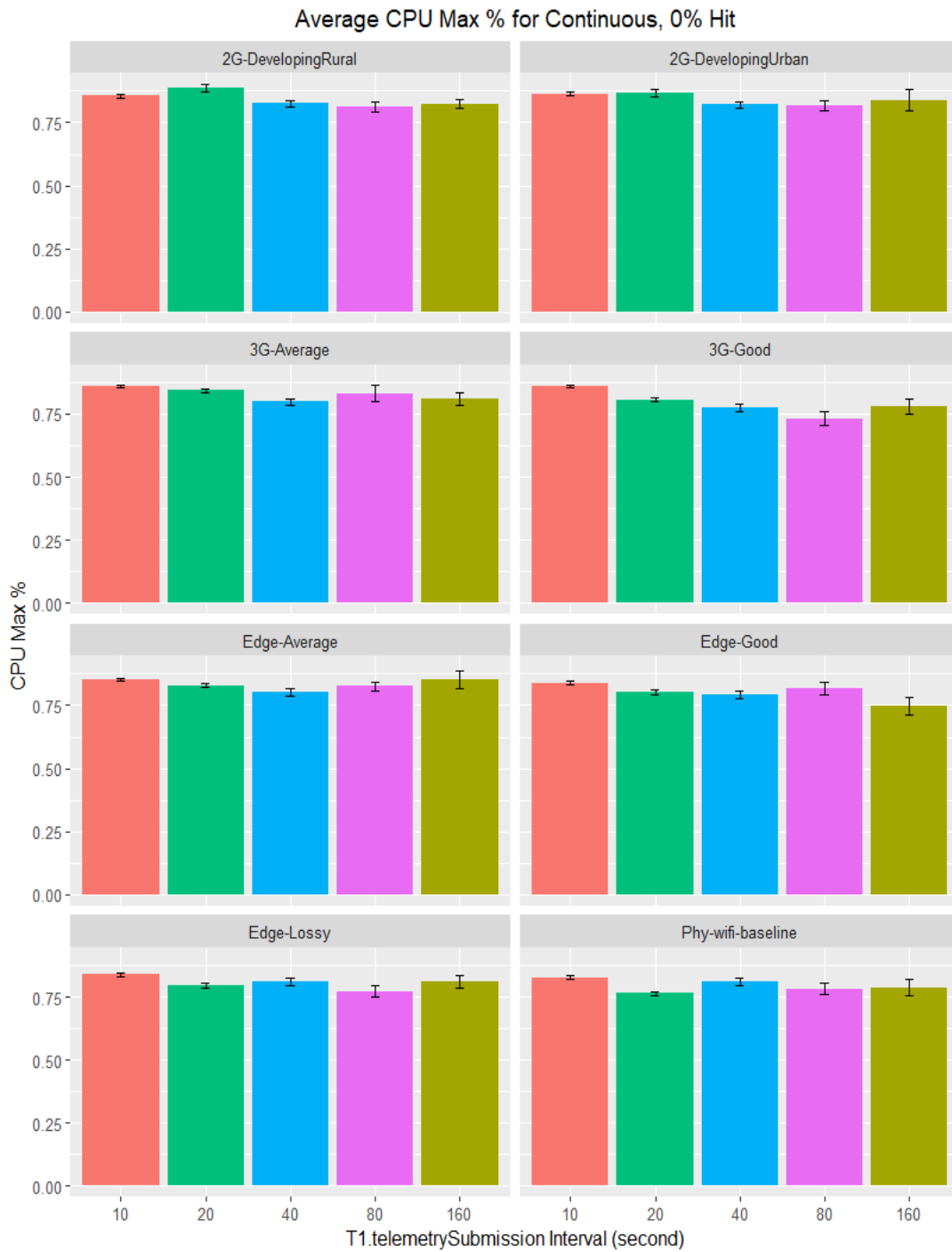


Figure 5.18: Average CPU max bar chart for telemetry submission transaction for each interval and each network profile with 0% hit scenario

The box plot in Figure 5.19 shows lower ranges of %CPU-Seconds than in Figure 5.20, because no page rendering is happening at all, but we notice one episode in edge-average 160 seconds interval that has outliers and a relatively larger distribution of values, this can be due to the lower number of samples but also can be caused by background Tizen OS operations running in the smartwatch.

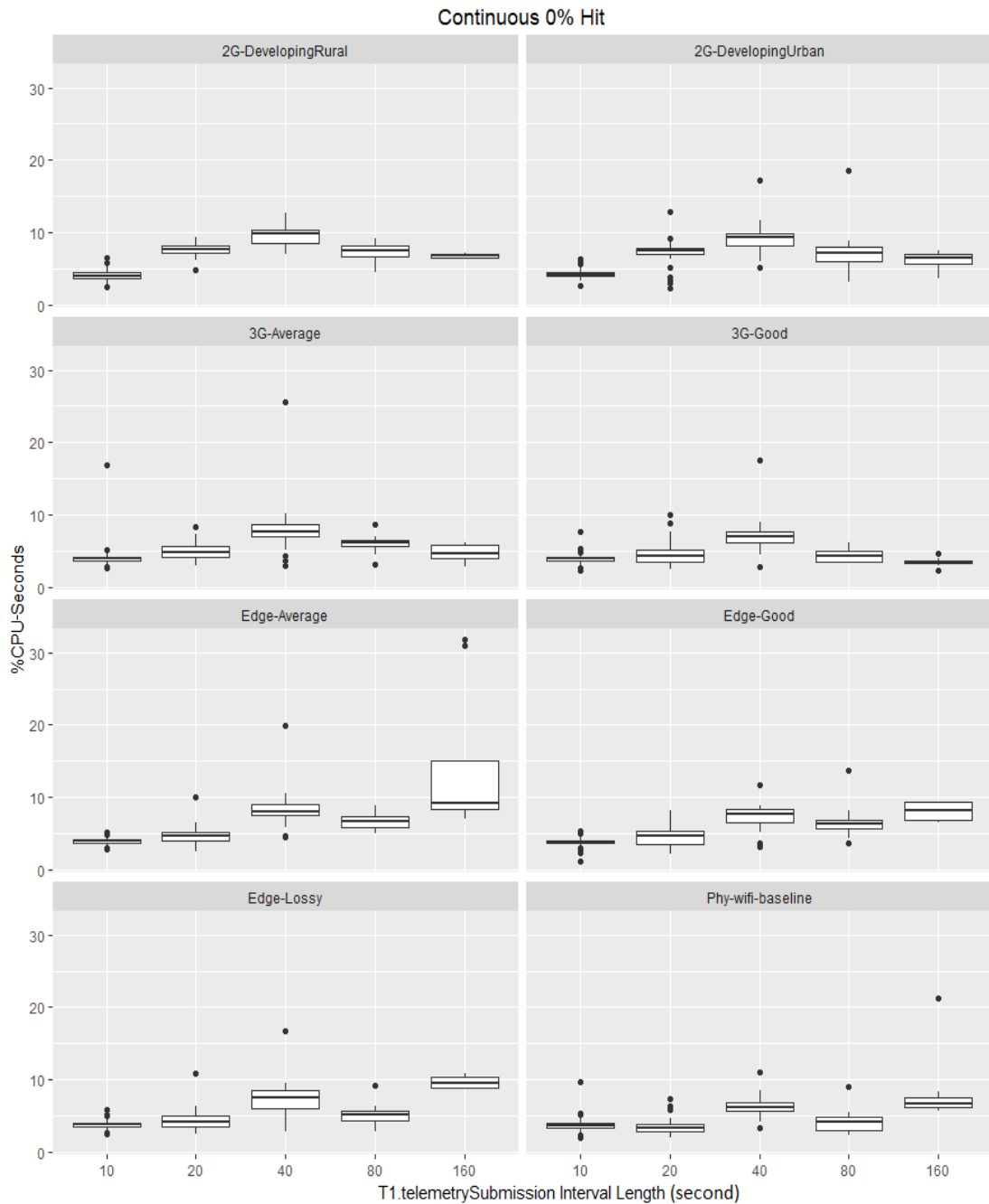


Figure 5.19: Box plot of %CPU-Seconds for telemetry submission transaction for each interval and each network profile with 0% hit scenario

Not only was the max CPU affected by the network profile, the %CPU-Seconds was also impacted. Note that the box plot for Wi-Fi %CPU-Seconds shows that some outliers dragged the range to 50. In fact, there were only a few outliers, as shown in the box plots in Figure 5.19 and Figure 5.20. The reason for this abnormality will be explained in the multiple watch experiment.

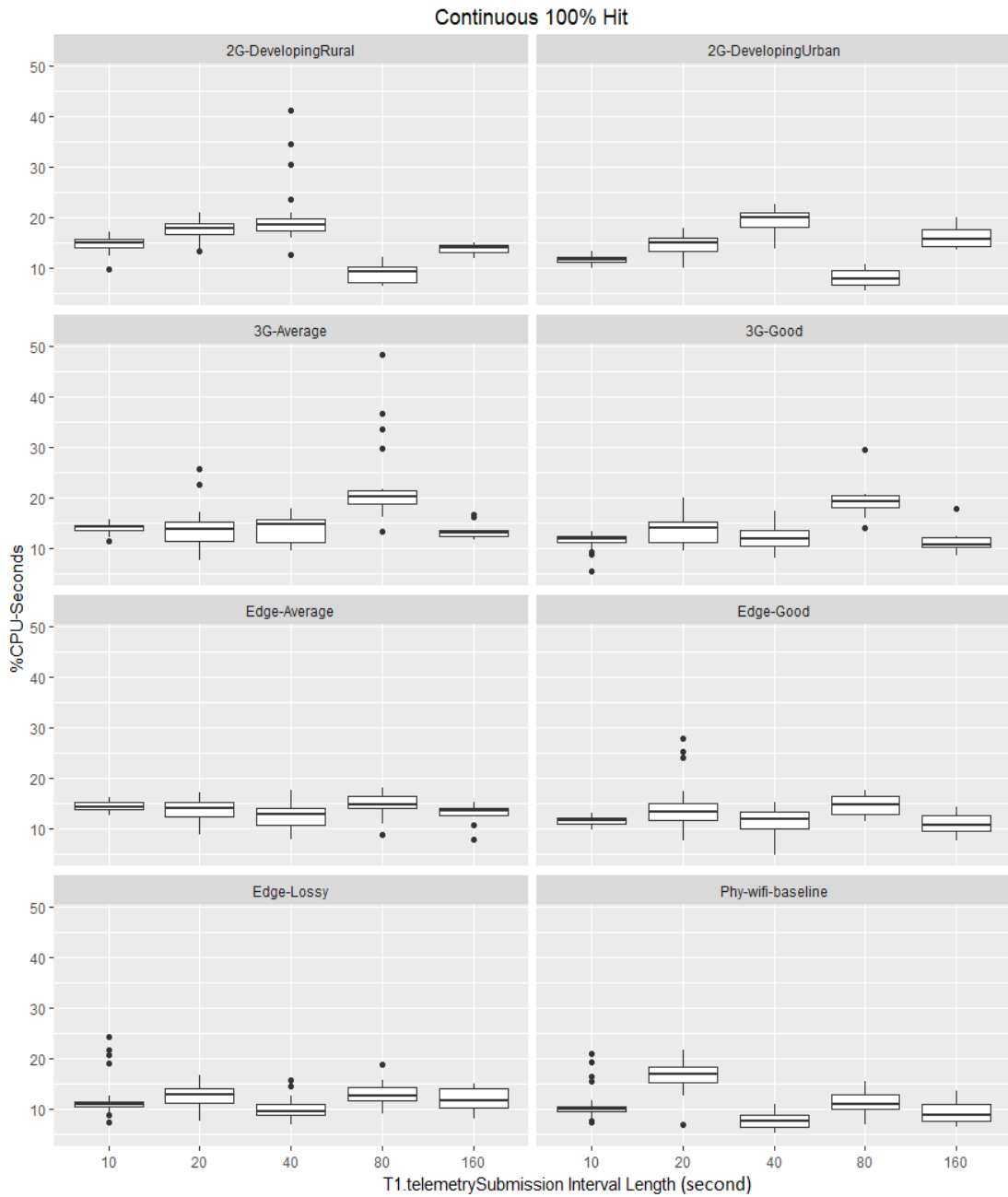


Figure 5.20: Box plot of %CPU-Seconds for each interval and each network profile with 100% hit scenario

The trend observed for %CPU-Seconds was that whenever the interval increased, the average CPU seconds also increased, as shown in Figure 5.21 for the 0% hit scenario.

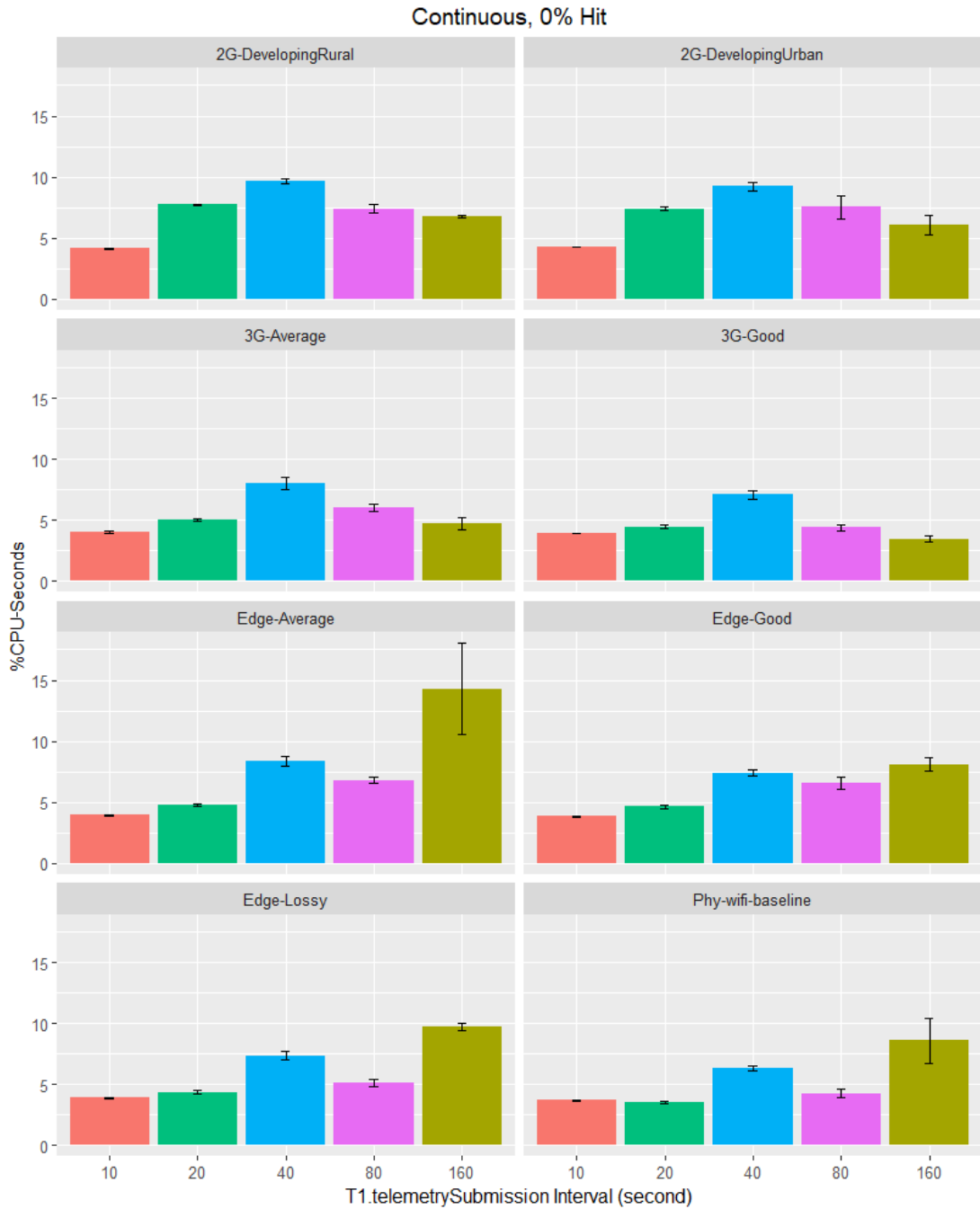


Figure 5.21: Average %CPU-Seconds bar chart for each interval of T1.telemetrySubmission and each network profile with 0% hit scenario

However, this result was weaker in the 100% hit scenario, as shown in Figure 5.22. Note also that the averages are all higher in the 100% hit scenario because rendering the new pages required more processing power than collecting the telemetry data within the same page of the smartwatch.

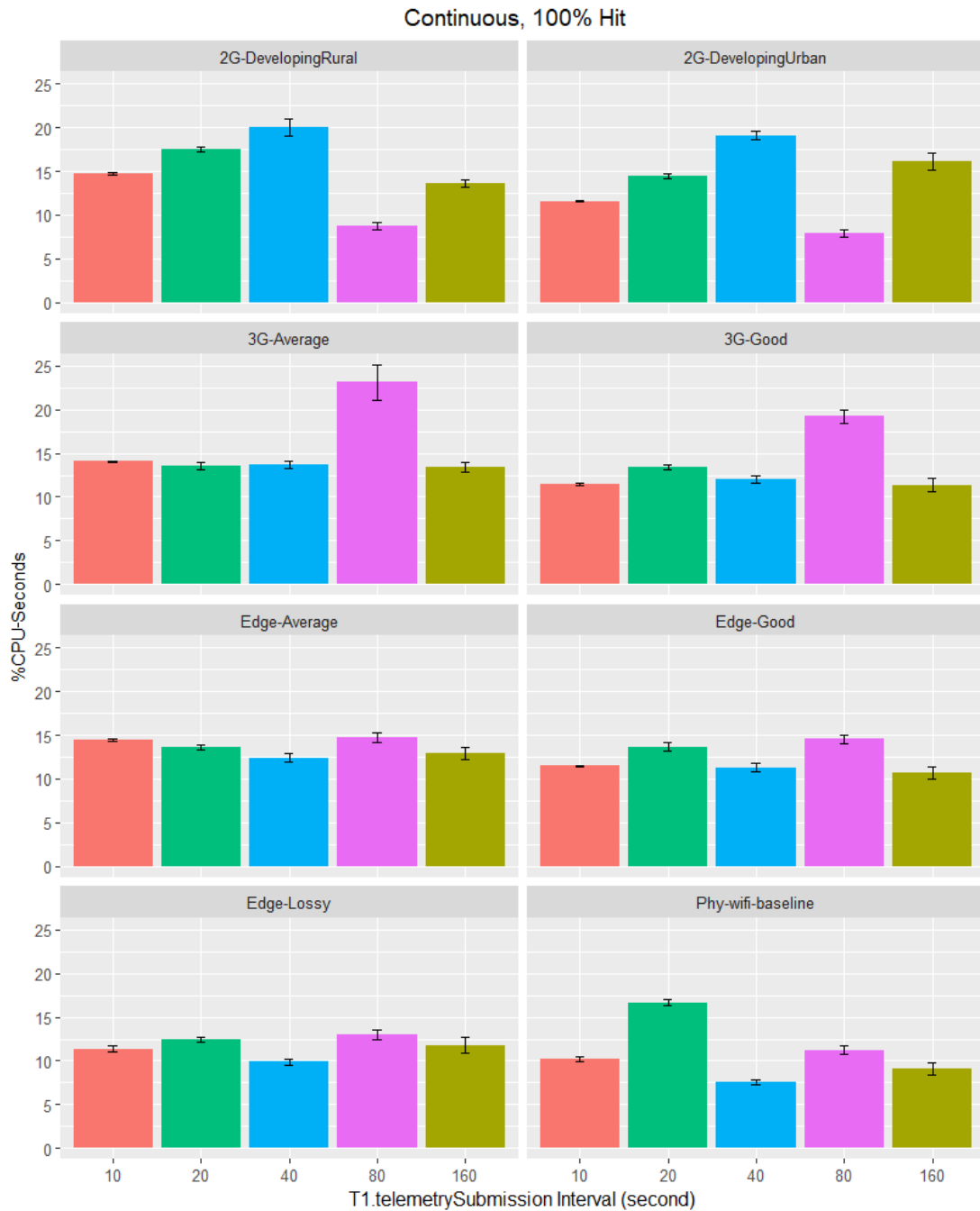


Figure 5.22: Average %CPU-Seconds bar chart for telemetry submission transaction for each interval and each network profile with 100% hit scenario

5.2.3. Power Drain. Figure 5.23 and Figure 5.24 show the battery usage during the 30-minute episode run for each network profile and frequency of sending the telemetry data. The power drain was found to increase whenever the interval of sending the telemetry data was increased. However, the 0% hit scenario had lower maximums for battery usage because the screen does not turn on to render new pages on the smartwatch. This consumes more battery than the 0% hit scenario. The increase in power drain with the interval increase can be explained due to the battery consumed as a result of keeping the network resources busy for a prolonged period when sending large amounts of data collected from the sensors during the intervals. It can also be noted how badly the battery is affected in the worst-case scenario, “2g-rural”, compared to better network conditions. The graphs also indicate that avoiding frequencies of more than 40 seconds may be a good choice for the platform setup.

Overall, there are no averages to compare for the power drain because each episode was only run once. Notice also that there is an outlier reading in the 20 second interval which lead into a reading of only 2 percent battery drain, percentages are only logged whenever a full one percent have depleted, therefore such errors are expected. These results can be further studied if the same episode is run at least 30 times to provide better statistical conclusions for the data. Currently, the battery usage results can only provide an idea of what is occurring, but they cannot strongly confirm any conclusions.

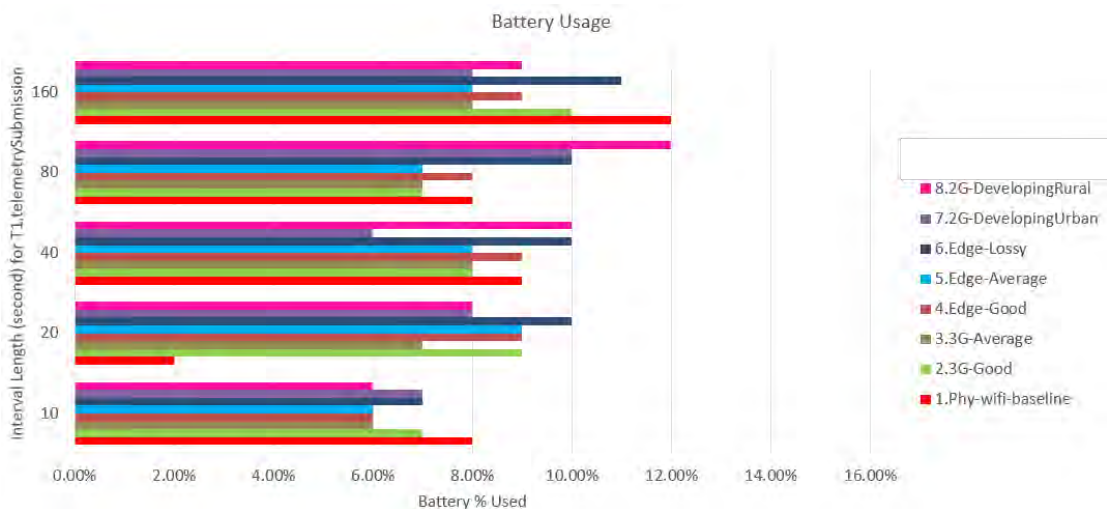


Figure 5.23: Continuous mode battery usage per network profile and T1 message frequency for 100% hit scenario

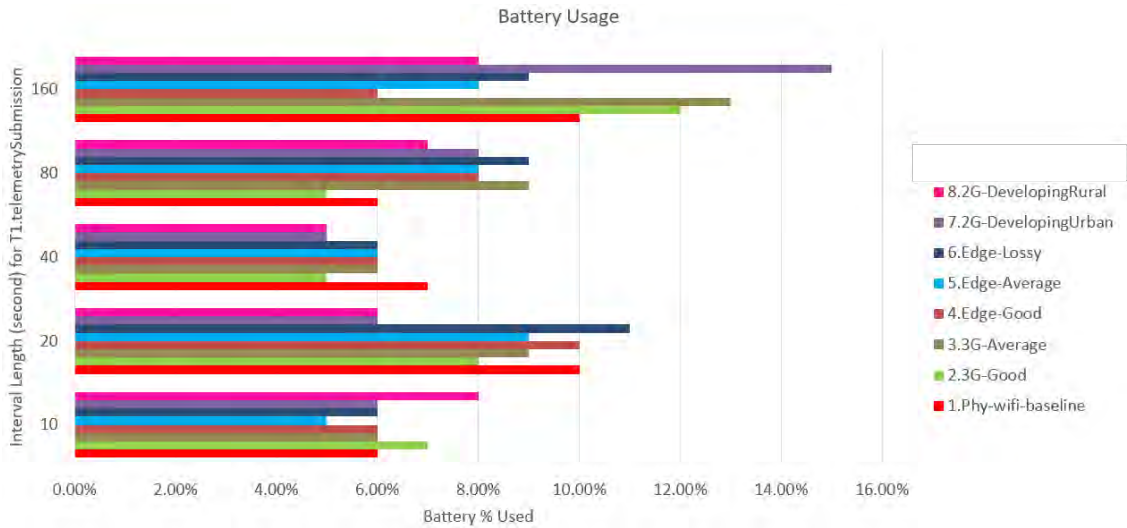


Figure 5.24: Continuous mode battery usage per network profile and T1 message frequency for 0% hit scenario

5.2.4. Memory Availability. In the continuous mode, there was no memory drain throughout the episode run, as shown in Figure 5.25, but every episode run had a different available memory at the start of the run. Therefore, normalizing the available memory by subtracting the starting value for each episode would make more sense for analysis, as per Figure 5.26.

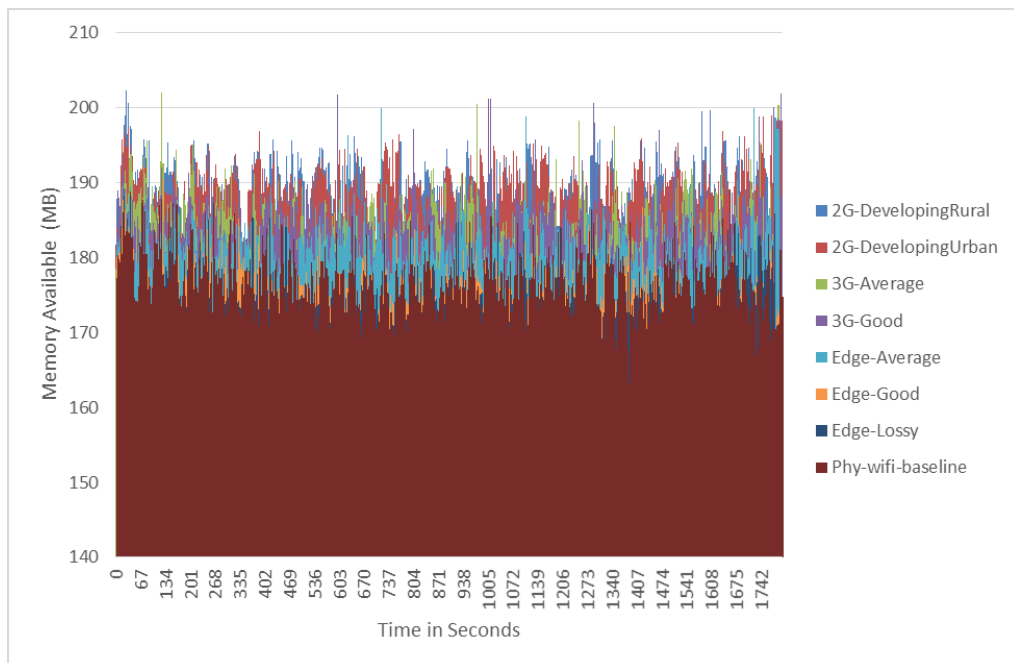


Figure 5.25: Continuous mode memory usage throughout the episode run per network profile

Figure 5.26 provides the initial observation that whenever the interval of T1.telemetrySubmission increases, the available memory decreases over the duration of the episode run. This can be explained by the increase in the size of the message being queued in the memory during the intervals. Another observation from Figure 5.28 is that the memory usage is slightly lower when the network condition is better.

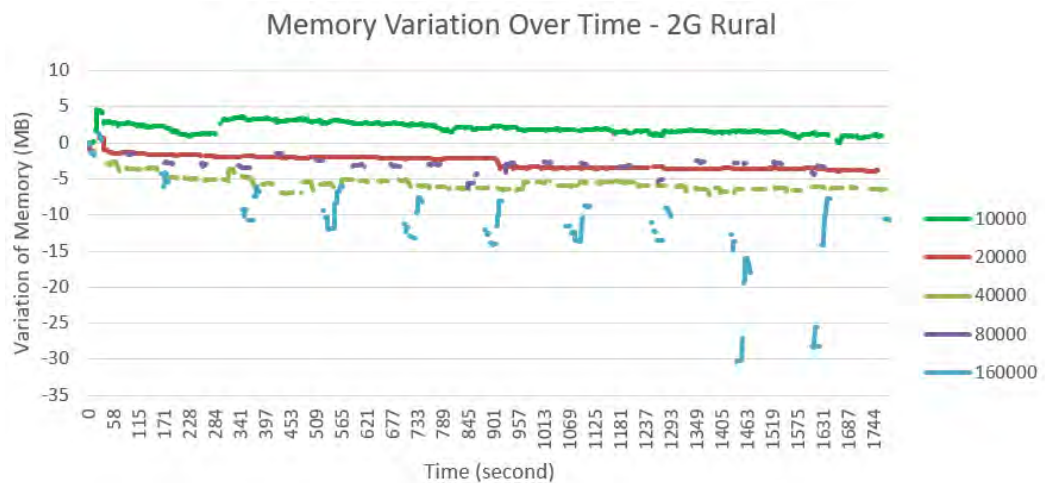


Figure 5.26: Continuous mode memory usage for each interval of T1.telemetrySubmission in 2G-Rural network with 0% hit scenario

In Figure 5.27, we can notice in the Wi-Fi network in contrast to 2G that the memory variation is more stable over the time across all intervals, also the memory usage is very similar from one interval to another.

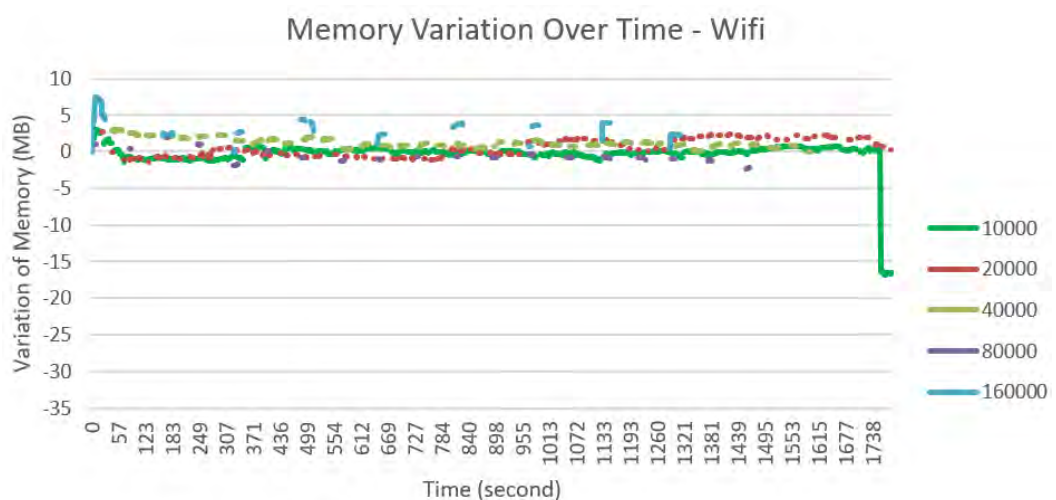


Figure 5.27: Continuous mode memory usage for each interval of T1.telemetrySubmission in Wi-Fi network with 0% hit scenario

In Figure 5.28 the observed difference is the higher variation of available memory over time due to moving between the sensor collection page, question page and answer page in the 100% hit scenario, in which each page contains operations that depend on manipulating data in variables stored in memory. For reference on more network profile memory plots, more figures can be found in Appendix E.

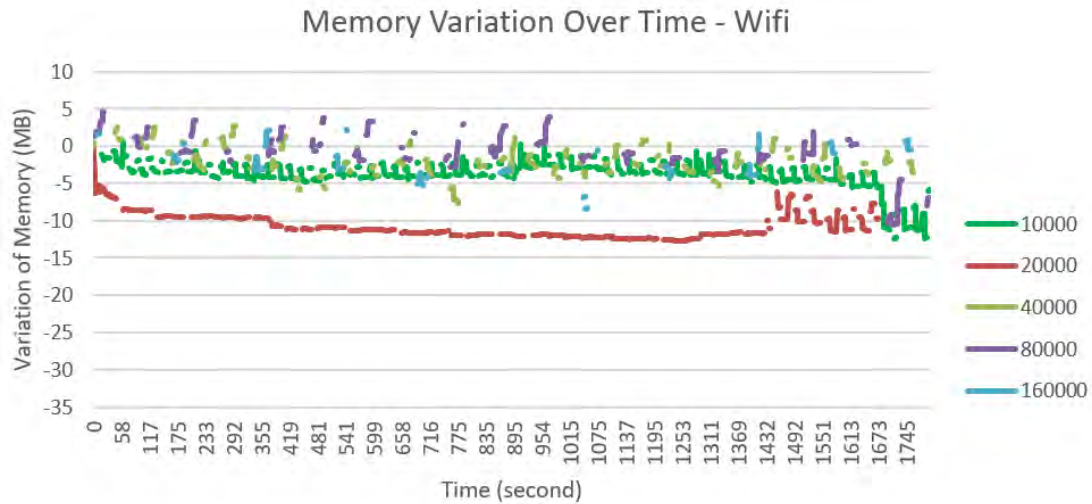


Figure 5.28: Continuous mode memory usage for each interval of T1.telemetrySubmission in Wi-Fi network with 100% hit scenario

Figure 5.29 shows that the Wi-Fi scenario is almost unaffected by the telemetry message interval increase. But generally speaking, the total variation in memory is within a 10-megabyte range, which is quite low considering the baseline available memory is 205 megabytes. This means the average memory being used in the application is around 5%.

Note that in Figure 5.30 and Figure 5.29 are normalized figures of the memory that were extracted by subtracting the starting available memory from each memory availability reading then averaging those results. So with respect to the Y axis, if the bar chart is above 0, that means that the memory availability throughout the application operation is higher than the initial available memory in each test point on average. And if the memory availability is below the X axis, that means the memory availability on average was lower than the starting point of available memory. So we can also consider that the higher the bar is, the better is the memory availability. Also we can consider that the point 0 on the y axis represents the starting available memory.

Also noticed in the 100% hit scenario in Figure 5.30 that the memory variation across all combinations is not exceeding 10 MBs. But we can notice in both Figure 5.29 and Figure 5.30 that when the network profile is better, then the memory usage is lower.

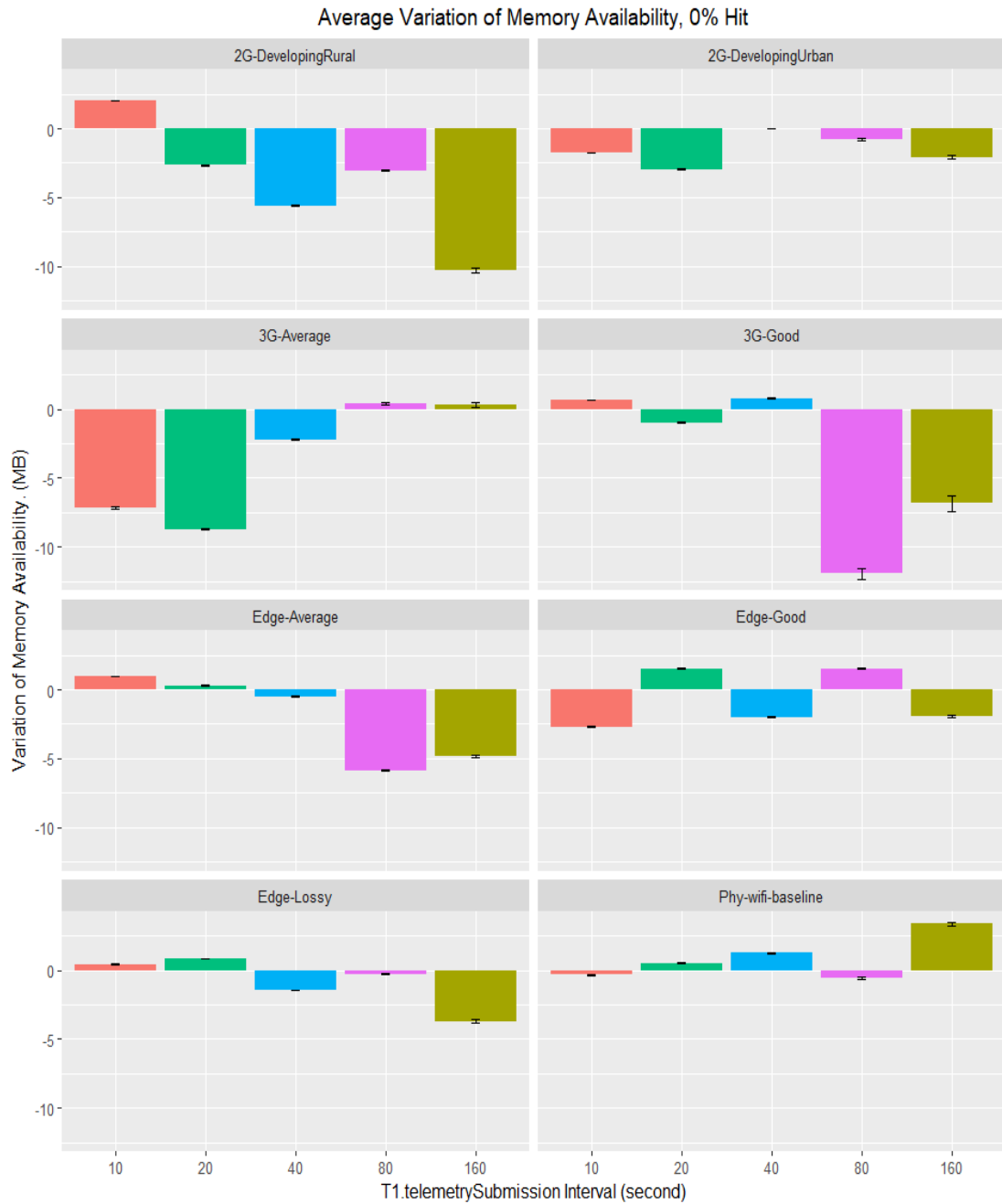


Figure 5.29: Continuous mode memory variation for each interval of T1.telemetrySubmission in Wi-Fi network with 0% hit scenario

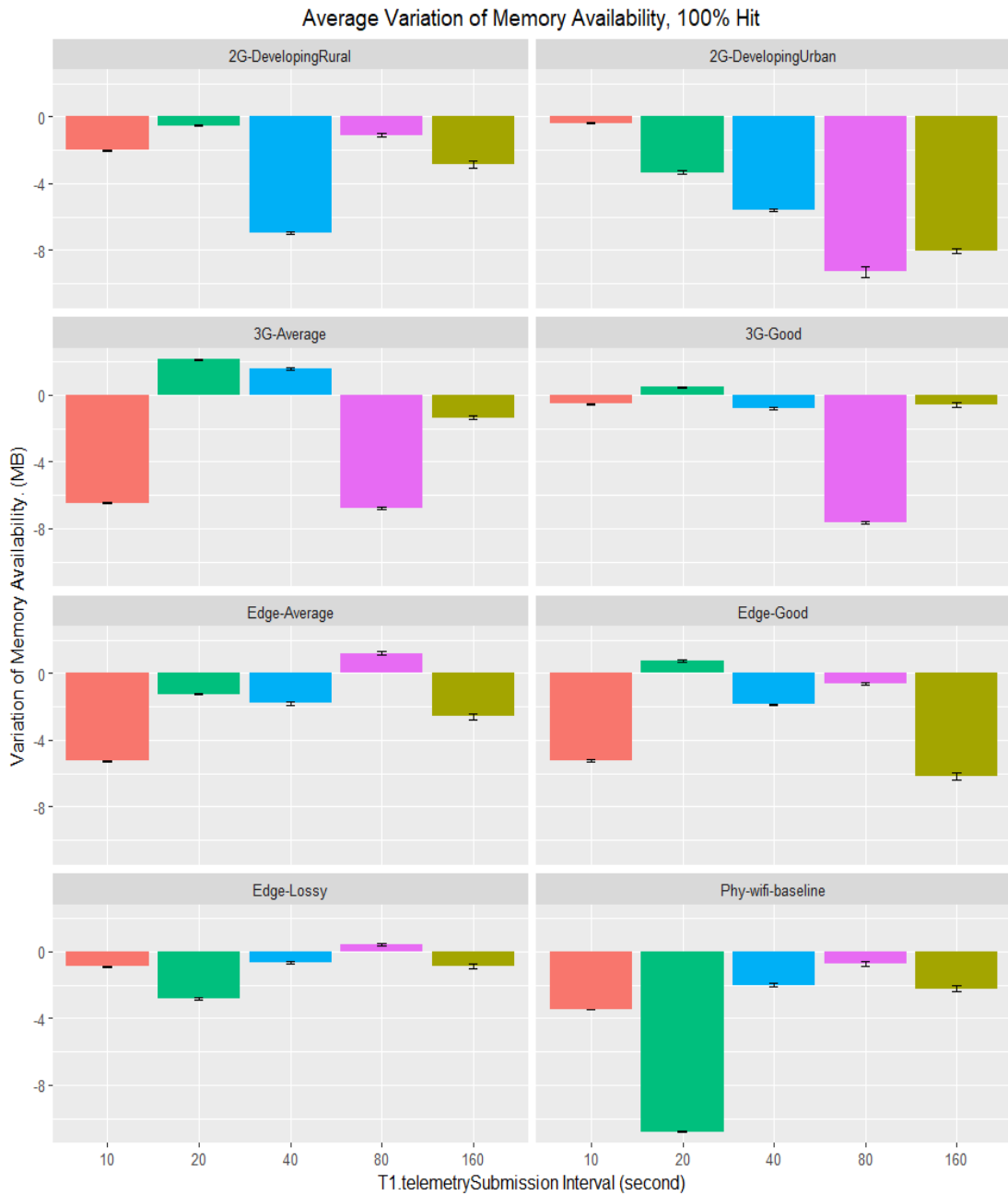


Figure 5.30: Continuous mode memory variation for each interval of T1.telemetrySubmission in Wi-Fi network with 100% hit scenario

This can be explained by the memory allocation required to send packets over slow networks, where in better performing networks the memory allocated is quickly deallocated when the packet is already sent, hence the better average available memory. There are a few exceptional cases in both the graphs which can be explained due to background operations running the watch to show achievement related notifications to the user.

5.2.5. Bitrate. Due to large amounts of data being sent every interval in contrast to the OnDemand mode, we clearly see the difference in bitrate measured in bits per second, where 2g is the worst and the Wi-Fi profile shows nearly double the performance of 3g. This result lead to a decision to change the sending mechanism to a more efficient method such as averaging the data collected by the smartwatch before sending it to the context processor to help speed up the response under poor network conditions.

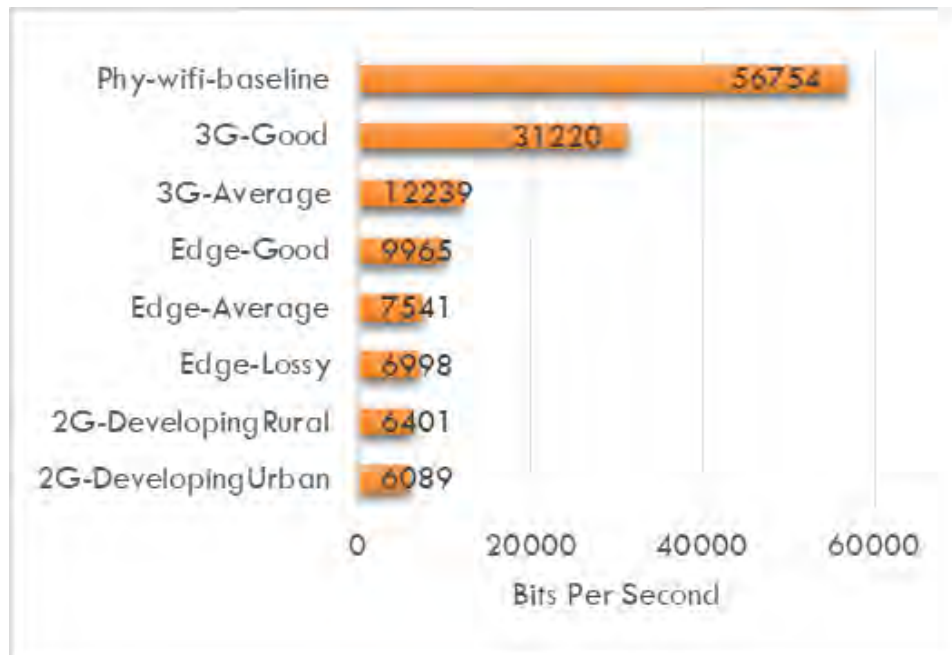


Figure 5.31: Continuous mode network throughput

5.3. Continuous Multiple Watches Experiment Results

During the data collection performed by the single watch, there was a bug in maintaining the transaction id. The transaction id was supposed to change in the continuous mode with every new telemetry reading procedure. Unfortunately, it was too late to rectify the bug and 104 episodes had already been run, so a virtual transaction id was created from the transaction data collected in the single watch mode. This virtual id has been correlated with the CPU usage and memory availability by matching time periods. Luckily, no two watches were operating at the same time, so it was feasible to do this correlation. Thus, the results discussed in the first two experiments were based on this correlation. The bug in the multiple watches mode has already been fixed, so the results are more stable. In this test, we executed 16 episodes for Wi-Fi and for 3g network profiles with either the variation of 100%

hit or 0% hit. In contrast, 104 episodes in total were run for the single watch experiments for both OnDemand and Continuous modes in the earlier experiments.

5.3.1. Server Load. The first thing that comes to mind when measuring the performance of multiple watches is the impact on the server CPU load.

It is important to note that the CPU load average can range from 0% to 400% due to the core count, which is equal to 4, where each core if it gets fully utilized represents 100% of the load, aggregated together the 4 cores when they are all utilized then the total server load would be 400%, the CPU load average can also be above 400%, which means there will be queued instructions waiting to be executed. The machine never reached this state in any of the tests.

Figure 5.33 and Figure 5.34 show the baseline load average in the Linux virtual machine. The virtual machine runs Ubuntu OS, couchDB, and PONTE in the baseline state, averaging at the rate of 21% for the 4 cores out of 400%. The CPU load average is not flat when the measurement was taken for 30 minutes which means there are background operations that affects the CPU usage, however measurement of the baseline will help understand how much of an increase occurs when running the context processor and middleware. For reference, Figure 5.32 shows the specs of the virtual machine allocated for the test.

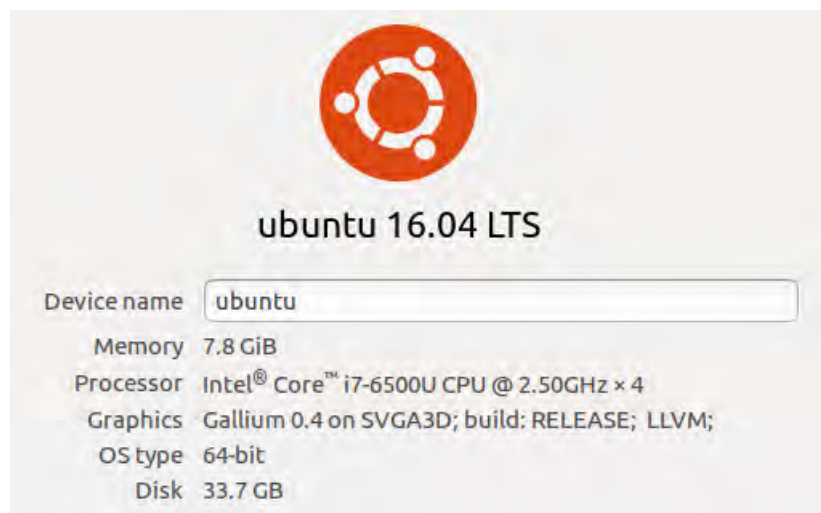


Figure 5.32: Server virtual machine specifications

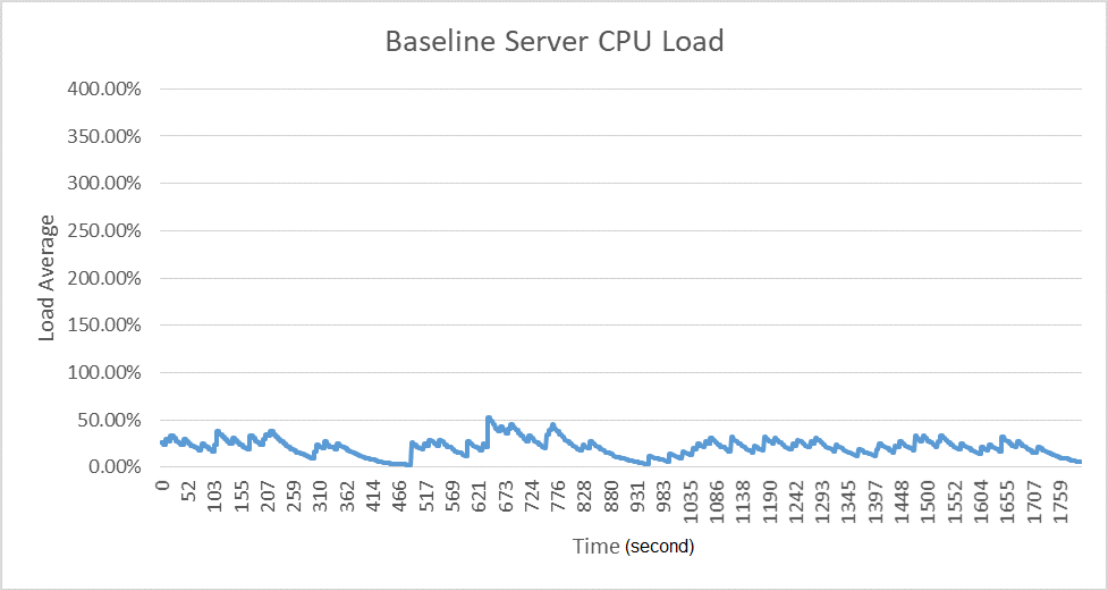


Figure 5.33: Server Load for the baseline scenario without the context processor running

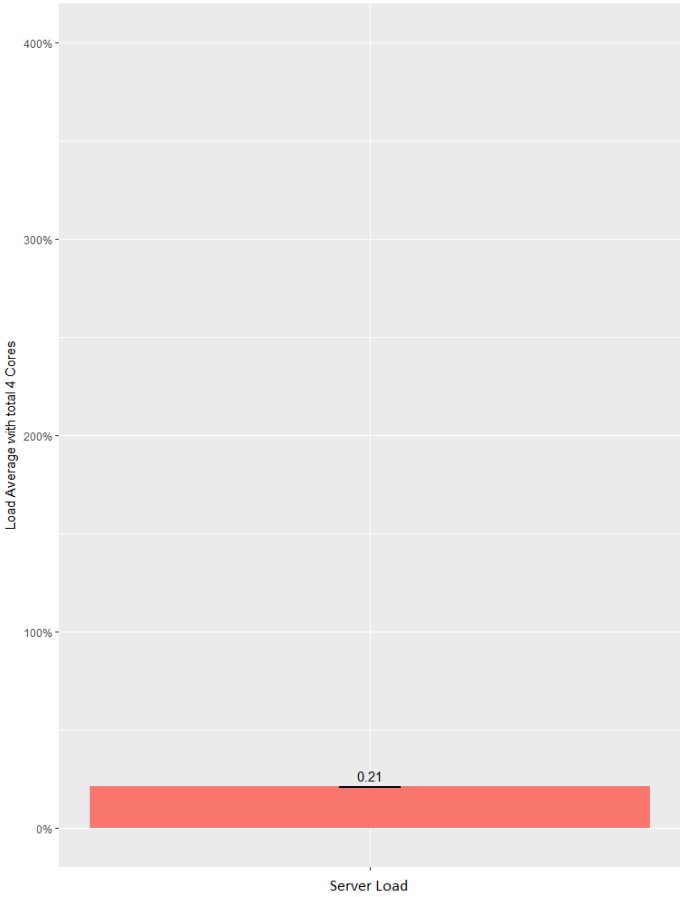


Figure 5.34: Average Server Load for the baseline scenario without the context processor running

In Figure 5.35, when observing the CPU load over the time of the episode run, the CPU usage is not exceeding 50% most of the time, however there are 3 spikes that have consumed the first core only, spikes can happen because the server hosts all the supporting applications for the automated test, including tcconfig which can affect the CPU usage when many packets have to be limited at the same time which is the case in the 8-watch test.

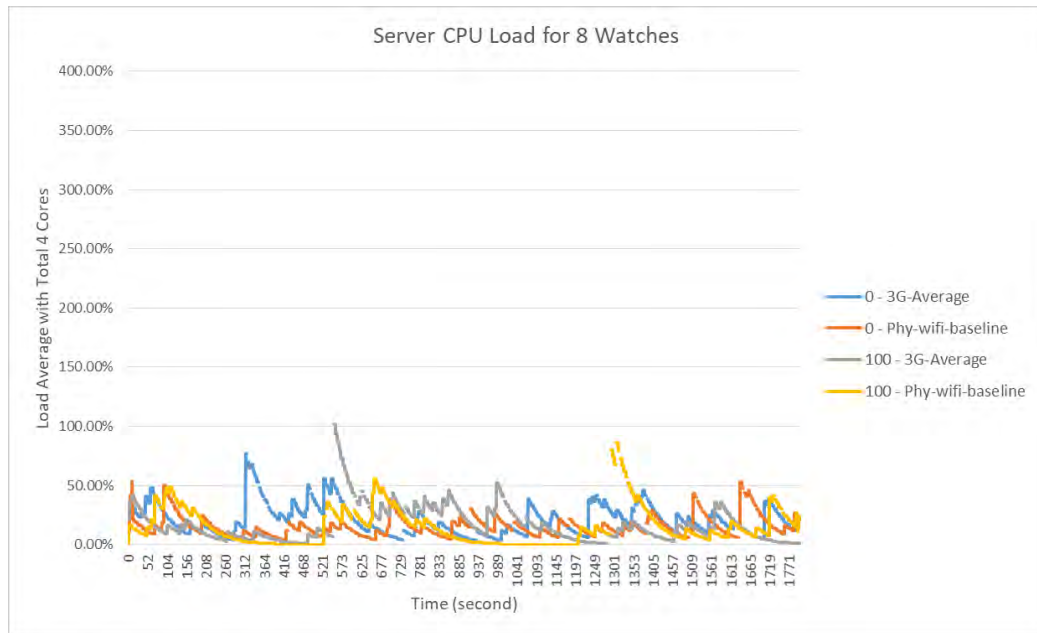


Figure 5.35: Server Load for each network profile with 0% and 100% hit scenario and 8 watches

In comparing 1, 2 and 4 watches, Figure 5.36 shows that the CPU usage was not observed to be increasing and was almost the same for all the tests of 1, 2, 4 and 8 watches. Even if the server load would have increased, the main concern of this research is the CPU load of the smartwatch because the server can always scale vertically or horizontally to satisfy the requirements of the number of hits expected on the platform.

5.3.2. Response Time. Comparing the response time between the single watch in the Wi-Fi scenario and multiple watches, and referring to the descriptive statistics in Table F.5.

It was observed that the average response times were very close for each network profile. Therefore, the response time was not impacted by increasing the

number of smartwatches, which is as expected since no abnormal server load impact was observed.

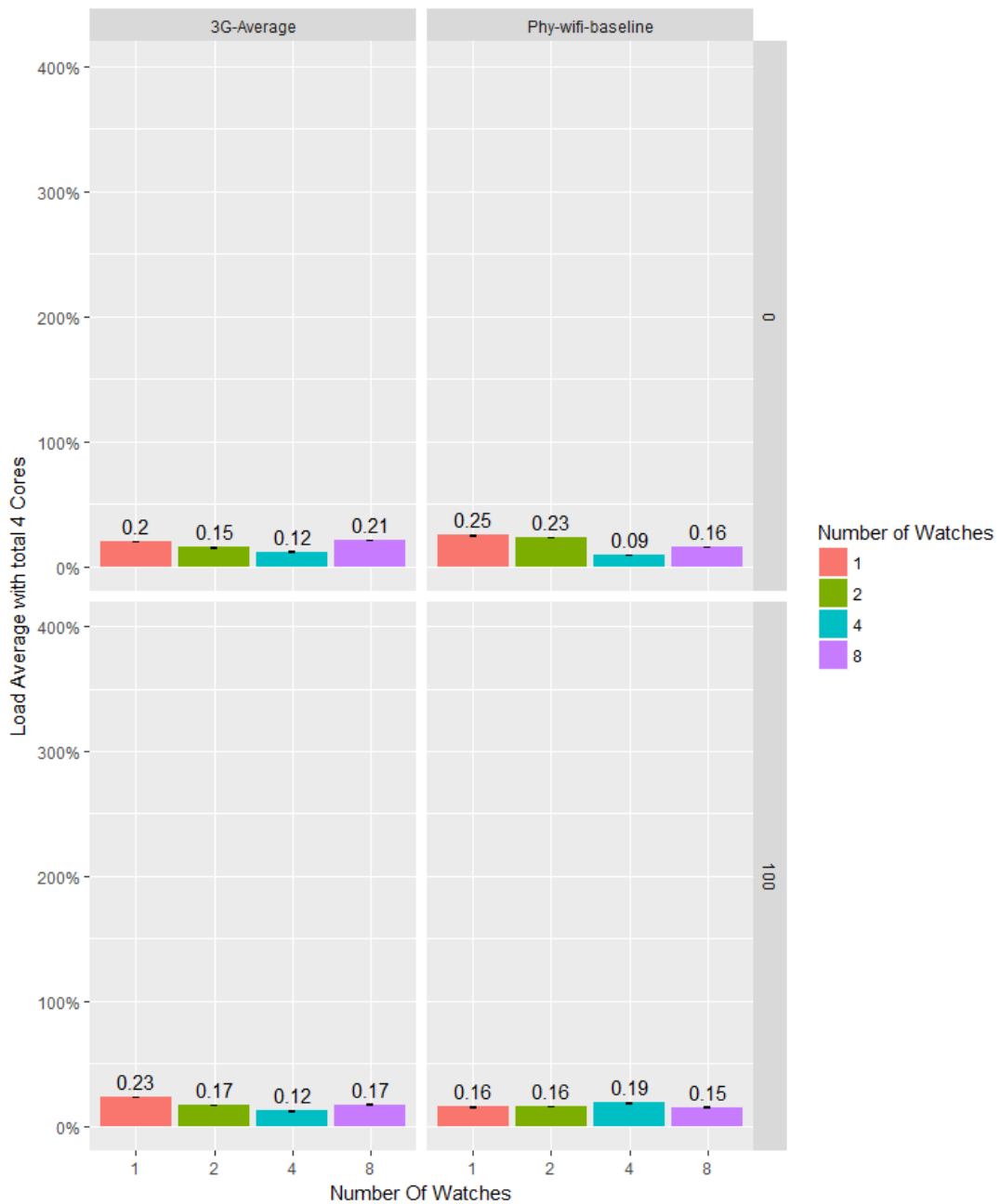


Figure 5.36: Bar chart Average CPU Load along number of watches for both 100% and 0% hit scenario for n watches (n = 1-8)

Referring to Figure 5.37, the authentication transaction was averaged amongst all the watches throughout both 100% hit and 0% hit scenarios for two reasons, first the authentication transaction has very low number of points to perform statistical analysis on, it only happens once per watch per episode, the second reason is that both

episodes with 0% hit and 100% hit scenarios have the authentication transaction as the first operation that occurs while starting the application, so the conditions for performing the authentication transaction are identical. For the results found on the authentication transaction, it is observed that the response time is relatively high, even higher in the 3g scenario than the wifi scenario which is expected, but the reason for this relatively high response time is not the packet transmission time only, the MQTT connection initiation and application startup operations also occur before the authentication transaction could proceed.

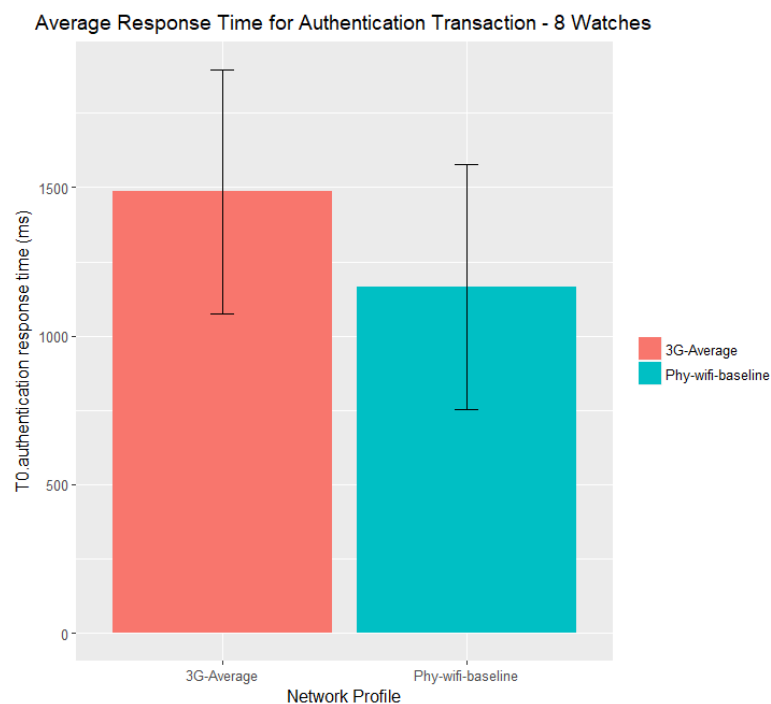


Figure 5.37: Average response time of T0.authentication in 100% and 0% hit scenario (n = 8)

Referring to Figure 5.38 and Figure 5.39, it was observed that the averages are close for the response time for answer posting and also for the telemetry submission transaction, but there were some variations in the average due to differences in the conditions of sensor data collection from one watch to the next. Note that the ids mentioned in the graphs are the Tizen IDs of the smartwatches. Also as expected, the response time under 3g network is higher than Wi-Fi, Table 5.5 and Table 5.6 provide the exact figures for differences in the mean between both Wi-Fi and 3g.

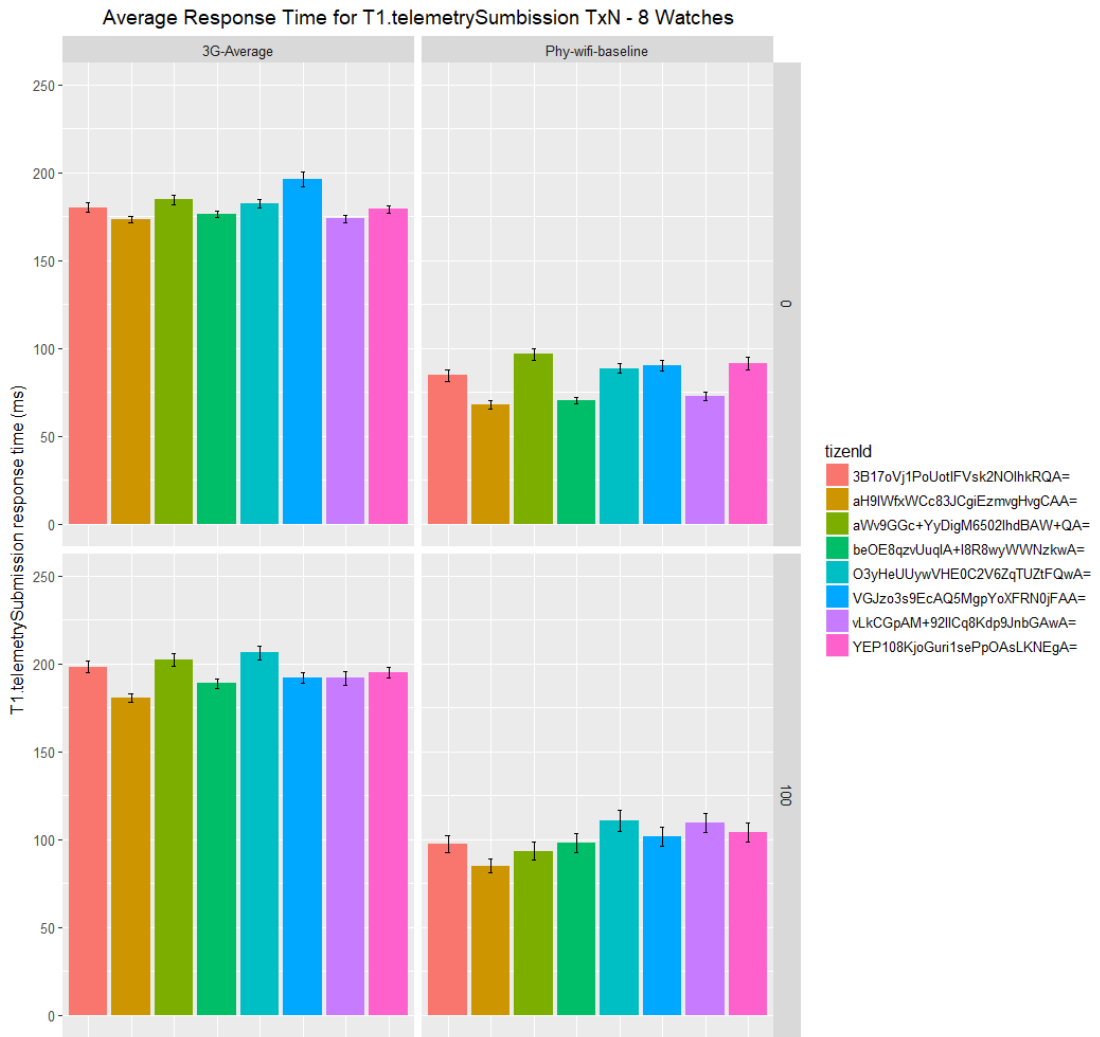


Figure 5.38: Average response time of T1.telemetrySubmission in 100% and 0% hit scenario per network profile

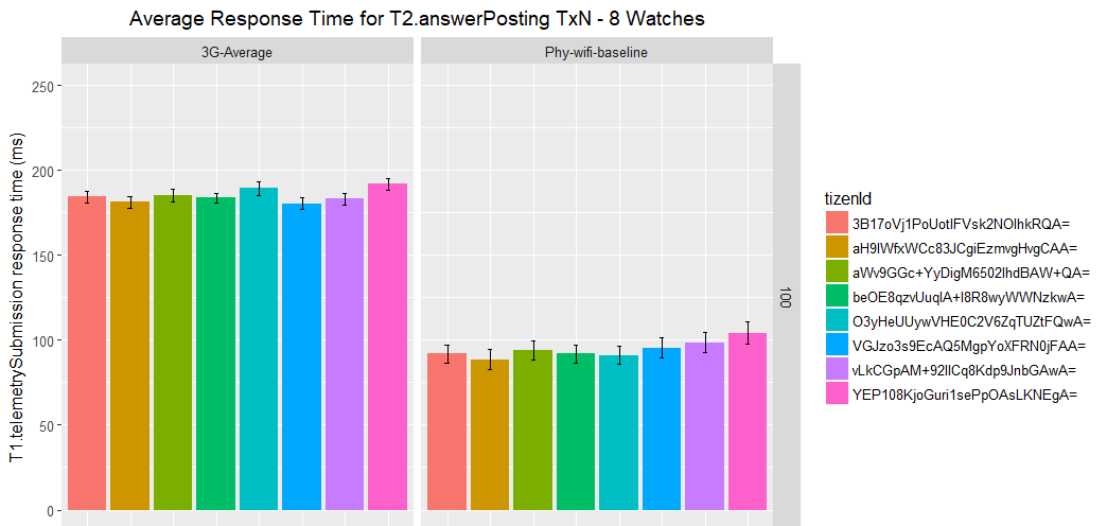


Figure 5.39: Average response time of T2.answerPosting in 100% hit scenario

Table 5.4: Summarized continuous mode descriptive statistics for T0.authentication transaction

Network Profile	Hit Percentage	mean	median	min	max	SD
3G-Average	0.00	1400.25	699.50	217.00	4188.00	1534.56
3G-Average	100.00	1584.86	686.00	244.00	4838.00	1769.14
Phy-wifi-baseline	0.00	52.63	45.00	25.00	106.00	25.78
Phy-wifi-baseline	100.00	2278.50	2378.00	48.00	4711.00	1729.33

Table 5.5: Summarized continuous mode descriptive statistics for T1.telemetrySubmission

Network Profile	Hit Percentage	mean	median	min	max	SD
3G-Average	0	305.45	187.00	125.00	5454.00	423.97
3G-Average	100	393.69	203.00	141.00	7190.00	650.17
Phy-wifi-baseline	0	164.63	80.00	25.00	2631.00	299.77
Phy-wifi-baseline	100	321.22	98.00	37.00	7296.00	770.01

Table 5.6: Summarized continuous mode descriptive statistics for T2.answerPosting

Network Profile	mean	median	min	max	SD
3G-Average	216.87	188.00	137.00	2919.00	162.65
Phy-wifi-baseline	150.66	87.00	31.00	3674.00	319.68

5.3.3. CPU Usage. In Figure 5.40, it is observed that the averages are very close to each other from one watch to another which confirms the stability the stability of the data transformation if compared with the single watch results, which is due to generating a proper transaction id in the multiple watch test.

As for the max CPU percentage averages, the hit scenario has higher CPU max and the 3g profile also produces higher results.

As for Figure 5.41 the %CPU-Seconds values are significantly higher in the hit scenario due to the page rendering operations, also the effect of the network profile on the %CPU-Seconds can be noticed in the figure, there are however a couple of inconsistent values that can be due to background operations in the smartwatch.

Table 5.7: Continuous mode descriptive statistics for T2.answerPosting for each watch

Profile	Hit %	TizenId	mean	median	min	max	SD
3G-Average	100	3B17oVj1PoUotlFVsk2NOlhkRQA=	225.23	184	142	1442	205.42
3G-Average	100	aH9IWfxWCc83JCgiEzmvghvgCAA=	236.89	185	140	2919	327.49
3G-Average	100	aWv9GGc+YyDigM6502lhdBAW+QA=	209.78	187	143	768	85.47
3G-Average	100	beOE8qzvUuqIA+I8R8wyWWNzkwA=	202.44	186	137	482	57.18
3G-Average	100	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	240.78	192	141	1345	184.79
3G-Average	100	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	202.22	187	140	440	56.22
3G-Average	100	vLkCGpAM+92llCq8Kdp9JnbGAWA=	199.5	186	141	513	56.72
3G-Average	100	YEP108KjoGuri1sePpOAsLKNEgA=	217.74	199	147	929	98.74
Wi-Fi	100	3B17oVj1PoUotlFVsk2NOlhkRQA=	165.2	84.5	31	2988	361.1
Wi-Fi	100	aH9IWfxWCc83JCgiEzmvghvgCAA=	115.37	73	31	622	106.56
Wi-Fi	100	aWv9GGc+YyDigM6502lhdBAW+QA=	126.73	80.5	35	1097	154.32
Wi-Fi	100	beOE8qzvUuqIA+I8R8wyWWNzkwA=	156.85	89	39	3600	429.26
Wi-Fi	100	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	163.05	89.5	38	2874	389.7
Wi-Fi	100	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	115.56	82	32	686	102.21
Wi-Fi	100	vLkCGpAM+92llCq8Kdp9JnbGAWA=	171.86	95	36	2587	337.61
Wi-Fi	100	YEP108KjoGuri1sePpOAsLKNEgA=	195.12	100	36	3674	461.45

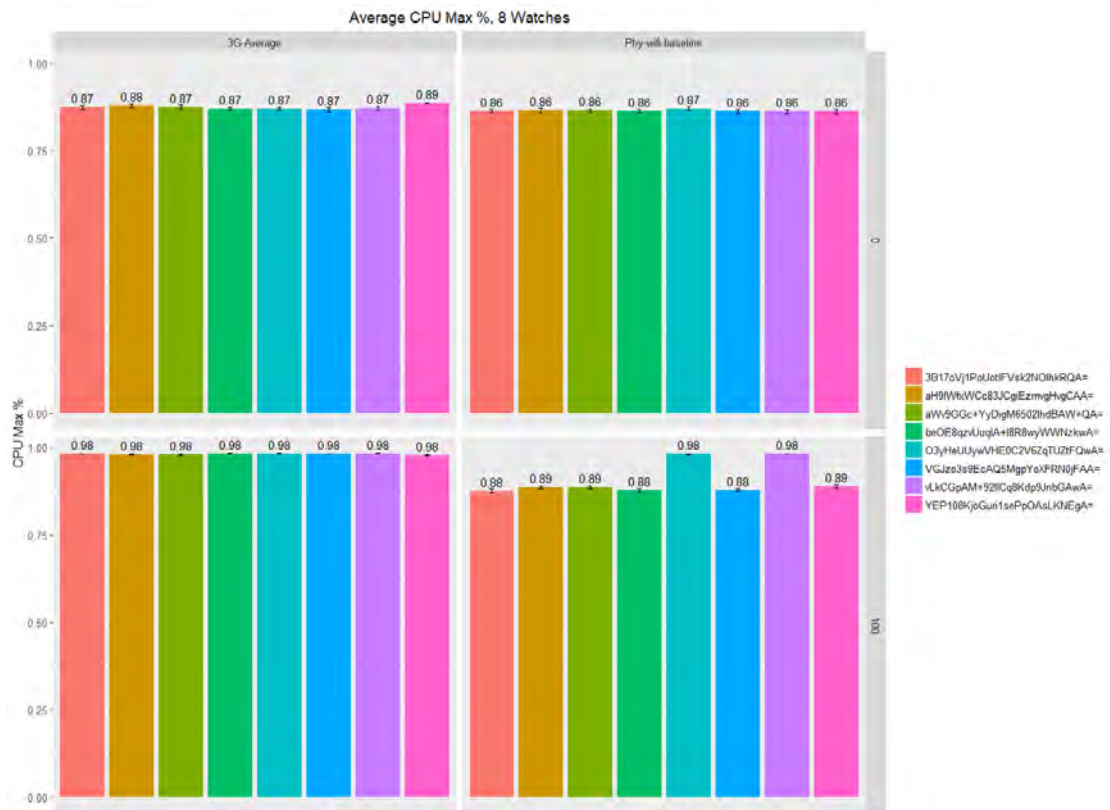


Figure 5.40: Average CPU max bar chart for each watch, each network profile and combination of 0% and 100% hit scenario

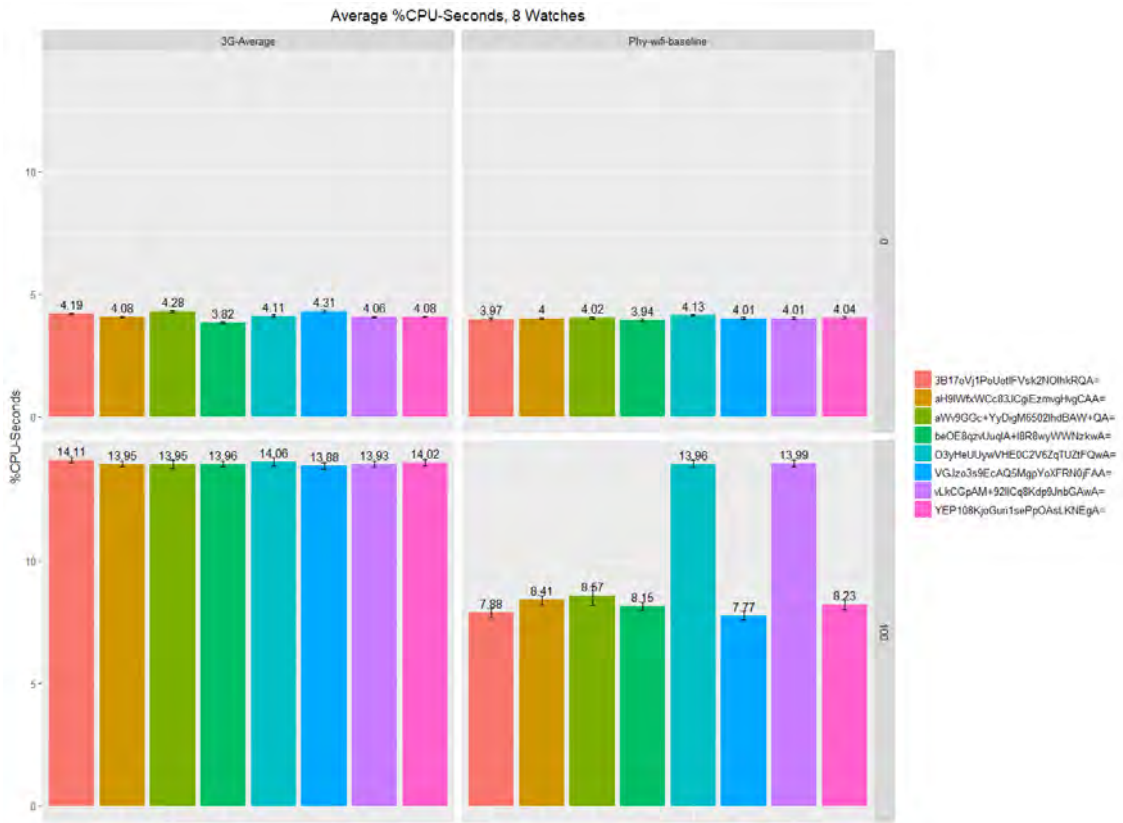


Figure 5.41: Average %CPU-Seconds bar chart for each watch, each network profile and combination of 0% and 100% hit scenario

5.3.4. Memory Availability. In Figure 5.42, the averages of the available memory are very close to each other, this observation is expected since there are no different intervals of T1.telemetrySubmission. As for the network profile effect on the memory availability it is not noticed in the results produced, but not many network profiles were tested so it may have a similar impact to the one observed in the single watch test. But for the hit scenario the impact is clear especially on the Wi-Fi network also due to rendering of pages in the 100% hit scenario.

5.4. Summary of Results

With respect to response time, the OnDemand mode showed sub-second response times on average for all network conditions. The Continuous mode showed similar results for T1.telemetrySubmission intervals lower than 40 seconds.

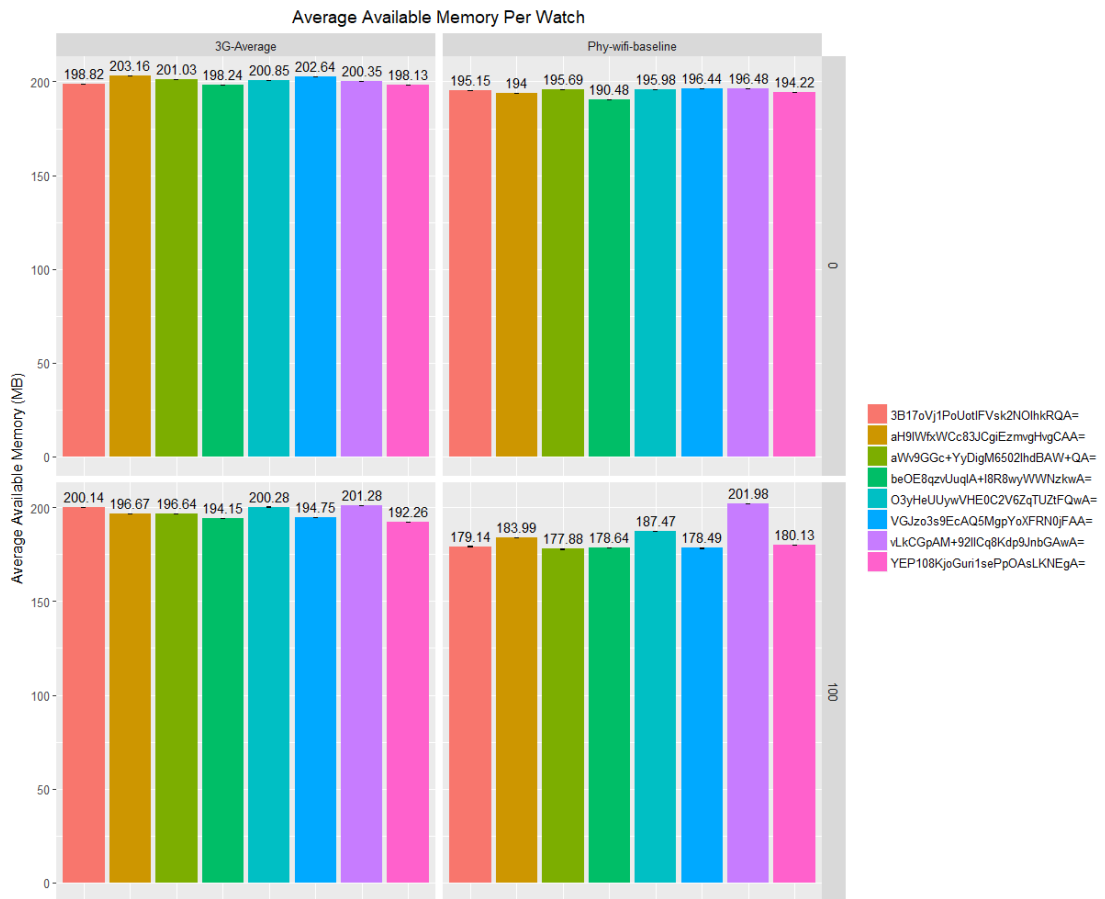


Figure 5.42: Average available memory bar chart for each watch, each network profile and combination of 0% and 100% hit scenario

The CPU usage results showed that the CPU max percentage was quite high, rising to 90 percent in the 0% hit scenario and more than 90 percent in the 100% hit scenario. The “1.0 GHz Cortex-A7” processor used in the SGS2 smartwatch nearly reached its limit running the Wearable School application, but the transaction and resource logging operations were also included in these tests, which may have partially contributed to this result. Moreover, the CPU was originally designed to be low power in order to handle running the watch for more than a day with a 250mAh battery. This means the CPU was relatively low power compared to the CPUs available in smartphones. As for the Exynos processor comparison [54], the “Exynos 2 Dual 3250” processor model achieved a performance at 7.2 gigaflops. However, current smartphone CPUs have reached performance levels of around 375 gigaflops, according to tests performed on the “Exynos 9 Octa 8895” CPU, which is used in the Samsung Galaxy S8 smartphone. These figures provide an idea of the limitations of the SGS2 smartwatch in terms of its CPU. Using the new Samsung Gear S3, which

has an “Exynos 7 Dual (7270) Cortex-A53 1.0GHz” processor, could lead to better CPU usage results because, according to the Arm Community [55], the Cortex-A53 performs approximately 20% better than the Cortex-A7. However, no information was found that reveals the exact number of gigaflops the CPU in the Samsung Gear S3 produces.

Aside from the high CPU usage observed, it was found that the telemetry interval and the network profile caused the CPU usage to increase. The CPU%-Seconds results also showed that changing the network profile or increasing the T1.telemetrySubmission interval increased the CPU%-Seconds, with some exceptions, due to the background processes running in the smartwatch.

In terms of power drain, the results are similar to the CPU usage because power drain is obviously related to the CPU usage and memory availability. However, as explained earlier, not enough data was available to draw conclusions.

In terms of memory availability, it was observed that higher T1.telemetrySubmission intervals lead to lower available memory and that memory availability fluctuated more in the 100% hit scenario than in the 0% hit scenario. In addition, the network profile reduced the impact of memory availability when T1.telemetrySubmission increased.

The bitrate was clearly affected by the network profile in clear proportions going from the worst “2g” to the best “Wi-Fi” network profile.

Chapter 6. Conclusion and Future Work

This chapter discusses possible areas of improvement to support the Wearable School platform. A summary of the findings from previous chapters is presented in the conclusion.

6.1. Future Work

In the previous chapters, options were presented for improving and extending the Wearable School platform by incorporating new features, such as developing authoring pages for questions rather than writing the JSON object for the in-situ questions. This chapter will discuss additional concepts for future research to enhance the Wearable School platform.

Further study on the practicality of extending the system to include surveys for students, teachers and the students' parents is required. A survey could be performed on a group of participants to gather feedback on user experiences of the platform to determine their perceptions of the affordances offered by the platform. For instance, students from two different classes in the same grade within a local school could be selected to provide feedback after using the platform. One group could answer questions in the classic MCQs style, while the other group uses the Wearable School Platform. In this case, participant answers would form part of the survey analysis regarding the affordances and benefits of the Wearable School Platform. In addition, subjective questions allowing the students and teachers to provide more open-ended feedback on their experiences could also be included. The sample data collected from these surveys can be studied to draw conclusions on the potential outcomes of this platform when used in real-world classroom settings.

Edge analytics could be performed on the platform to assess the data processing by the context processor or by the watches since the CPU load at the watch end was observed to be quite high, and some optimizations could be incorporated as part of edge analytics to solve this problem. Additionally, the study could include a comparison of the overhead of network vs overhead of processing.

Since the platform utilized readings from 6 of the 9 available sensors, further enhancements could be made to support all of the sensor readings listed in the Samsung APIs documentation.

To encourage students to use the smartwatch application, a ranking system could be developed to provide a gamification concept on top of the assessments. Seeing how they rank against their colleagues may encourage students to view the activities as fun and challenging rather than an obligatory.

In terms of other potential activities, IoT Sensor nodes could be placed at fieldtrip destinations. Sensors like the pressure and temperature sensors could be placed in strategic locations for high school students, such as inside an experimental pipe to answer questions about physics or advanced mathematics (e.g., questions about heat transfer) similar to the experiments described in [2]. However, a combination of the approaches from this paper and the cited reference papers would be required to achieve this. A sample use is to have a student answer mathematical questions on heat conduction and dissipation, in which thermal sensors placed on a plate that is being heated in the corner of the room measure the temperature on different parts of the plate. The student receives random notifications about a specific point on the plate that require a temperature calculation. When the student submits the answer, a mediation verification mechanism checks whether or not the student's answer is within an acceptable margin of error and shows the result of the assessment to the student.

Rating questions could be gathered from the students on their smartwatches. Ratings data could be used to help teachers provide questions to other teaching institutes, or a central repository could be developed for use across different institutes in order to develop a library similar to MERLOT [19]

Finally, the previous implementation developed in the Assessment Wiki for IoT [2] had an editorial flow of questions utilizing the crowdsourcing features of Wiki Media to enhance the questions database with input from multiple teachers and contributors. An editorial flow of this kind could be incorporated into the Wearable School platform.

6.2. Conclusion and Limitations

The comprehensive literature review presented at the beginning of this paper contributed to building the Wearable School platform through the synthesis of concepts and ideas from previous researches. The current research proposed a

practical assessment platform that combines education and sensor context with the latest technologies and online platforms available for use. The performance test conducted helped prove the feasibility of this platform with multiple students using it. The platform was also shown to perform well under a variety of network conditions with little impact on the user experience due to slow response times or failures. A video has been provided to demonstrate the tests performed and is available on YouTube [56]. The source code for the platform is also available in GitHub [57], along with the test automation code presented in Chapter 4 and is open for any modifications.

Recently, the price of the SGS2 smartwatch 4G variant has dropped below \$100. The price of this wearable would make a platform of this kind feasible for broader acceptance in schools or for parents seeking to enhance the educational experiences of their children. The results collected have shown that setting up the wearable school platform in continuous mode with 10 or 20 seconds interval is acceptable and provides the best user experience for presenting the eligible questions to students in the right situation and at the right time. The research conducted thus far would benefit from a variety of enhancements, particularly on the teacher portal side, which needs to be integrated with an editorial workflow for creating the in-situ questions. Hence, we look forward to the next iteration of improvements on this platform.

Regarding the network tests performed on the wearable device for evaluation of the MQTT protocol on smartwatches, the tests were conducted under the conditions of the worst-case scenario, in which all sensor data was aggregated every second then sent to the context processor. A more focused approach would involve sending the sensor data according to the sensor type. For example, readings from the light sensor are only required momentarily not throughout the entire interval, and a recommended interval for collecting data from the heart rate sensor should be added allowing some aggregation to be carried out before the data is sent to the context processor. The results of the responsiveness and effectiveness of the platform for educational use also suggest that the platform could be extended for use in other practical applications, such as urban planning, an application mentioned in the literature review. Alternatively, the system could be used to produce a heat map from

certain sensor values using a mixture of GPS and other sensor data. Thus the platform could be considered ubiquitous for any feasible application that involve sensor context. In addition, the results collected with respect to the server load have shown that the platform can be scaled to thousands of watches with acceptable resources, considering that the elements used (Node.JS, PONTE and CouchDB) are all known to be scalable, either horizontally or vertically, to suit the requirements of large-scale implementations.

However there are some limitations in this research. The proposed platform was not tested on real students so that a survey would have been performed to compare the classical style of assessments against the proposed wearable assessments. Also the number of watches was limited to 8, so tests on scalability would have been preferred to be performed with hundreds of watches perhaps in a simulated manner to test multiple instances of the server applications hosted on servers that are scaling horizontally or vertically. The OnDemand mode was not suitable to provide statistical conclusions for the smartwatch resources usage and an enhancement to its mechanism is required to collect more stable resource usage data. Also the data collection was limited to 120 episodes of 30 minutes run each, more tests are needed to assess the battery usage patterns in different scenarios. One more limitation was that the network profiles tested were not modeled to consider the mobility scenario where students are moving in a vehicle and handoffs for base stations are occurring during movement. These limitations can drive for a motivation to perform further research on the proposed platform.

References

- [1] M. Al-Solh and I. A. Zualkernan, "An MQTT-Based Context-Aware Wearable Assessment Platform for Smart Watches," in *Advanced Learning Technologies (ICALT), IEEE 17th International Conference*, pp. 98-100: IEEE, 2017.
- [2] I. Zualkernan, M. Albayed, M. Al Solh, M. Tuffaha, and H. Al Muhallabi, "An Assessment Wiki For Internet Of Things", *Education and Learning (EDULEARN) Proceedings*, pp. 2294-2302, 2014.
- [3] F. Qutaifan, I. Zualkernan, and K. Asad, "Crowd-Sourcing Assessments in a Developing Country: Why Do Teachers Contribute?," presented at the *Education and Learning (EDULEARN) Proceedings*, Barcelona, Spain, 1-3 July. 2013.
- [4] M. Bower and D. Sturman, "What are the educational affordances of wearable technologies?," *Computers & Education*, vol. 88, pp. 343-353, 2015.
- [5] *List of Wearables | Vandrico Inc.* Available: <http://vandrico.com/wearables/list>, 2016.
- [6] A. Labus, M. Milutinovic, Đ. Stepanic, M. Stevanovic, and S. Milinovic, "Wearable Computing In E-Education," *RUO. Revija za Univerzalno Odlicnost*, vol. 4, no. 1, p. A39, 2015.
- [7] P. Mistry and P. Maes, "SixthSense: a wearable gestural interface," in *ACM SIGGRAPH ASIA Sketches*, p. 11: ACM, 2009.
- [8] C. Romero, R. Cerezo, J. A. Espino, and M. Bermudez, "Using Android Wear for Avoiding Procrastination Behaviours in MOOCs," in *Proceedings of the Third ACM Conference on Learning@ Scale*, pp. 193-196: ACM, 2016.
- [9] H. W. Gellersen, A. Schmidt, and M. Beigl, "Multi-sensor context-awareness in mobile devices and smart artifacts," *Mobile Networks and Applications*, vol. 7, no. 5, pp. 341-351, 2002.
- [10] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 414-454, 2014.
- [11] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Handheld and Ubiquitous Computing: First International Symposium, HUC'99 Karlsruhe, Germany, Proceedings*, H.-W. Gellersen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 304-307, September 27–29, 1999.
- [12] M. Compton, C. Henson, L. Lefort, H. Neuhaus, and A. Sheth, "A survey of the semantic specification of sensors," in *Proceedings of the 2nd International Conference on Semantic Sensor Networks-Volume 522*, pp. 17-32: CEUR-WS.org, 2009.
- [13] M. Compton *et al.*, "The SSN ontology of the W3C semantic sensor network incubator group," *Web semantics: science, services and agents on the World Wide Web*, vol. 17, pp. 25-32, 2012.
- [14] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Mosden: An internet of things middleware for resource constrained mobile devices," in *47th Hawaii International Conference on System Sciences*, pp. 1053-1062: IEEE, 2014.
- [15] R. Buyya and A. V. Dastjerdi, *Internet of Things: Principles and paradigms*. Elsevier, 2016.

- [16] Y. Wang, "English interactive teaching model which based upon Internet of Things," in *International Conference on Computer Application and System Modeling (ICASM)*, vol. 13, pp. V13-587-V13-590: IEEE, 2010.
- [17] S. Martin *et al.*, "M2Learn open framework: Developing mobile collaborative and social applications," in *The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pp. 59-62, 2010.
- [18] B. Hou, H. Ogata, M. Miyata, M. Li, and Y. Yano, "Development of web-based japanese mimicry and onomatopoeia learning assistant system with sensor network," in *Wireless, Mobile and Ubiquitous Technologies in Education (WMUTE), 2010 6th IEEE International Conference on*, pp. 117-121: IEEE, 2010.
- [19] R. Cafolla, "Project MERLOT: Bringing peer review to web-based educational resources," *Journal of Technology and Teacher Education*, vol. 14, no. 2, p. 313, 2006.
- [20] S. M. Johnstone and R. Poulin, "Technology: What Is Open Course Ware And Why Does It Matter?," *Change: The Magazine of Higher Learning*, vol. 34, no. 4, pp. 48-50, 2002.
- [21] W. R. Watson, S. L. Watson, and C. M. Reigeluth, "Education 3.0: Breaking the mold with technology," *Interactive Learning Environments*, vol. 23, no. 3, pp. 332-343, 2015.
- [22] V. Vujovic and M. Maksimovic, "The Impact of the Internet of Things on Engineering Education," presented at the Second International Conference on Open and Flexible Education (ICOFE), Hong Kong, 2015.
- [23] G. Kortuem, A. K. Bandara, N. Smith, M. Richards, and M. Petre, "Educating the Internet-of-Things generation," *Computer*, vol. 46, no. 2, pp. 53-61, 2013.
- [24] D. Porcello and S. Hsi, "Crowdsourcing and curating online education resources," *Science*, vol. 341, no. 6143, pp. 240-241, 2013.
- [25] D. S. Weld *et al.*, "Personalized online education—a crowdsourcing challenge," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 1-31, 2012.
- [26] B. Guo, Z. Yu, X. Zhou, and D. Zhang, "From participatory sensing to mobile crowd sensing," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE International Conference*, pp. 593-598: IEEE, 2014.
- [27] *IMS Question & Test Interoperability Overview Version 2.2*. Available: https://www.imsglobal.org/question/qtiv2p2/imsqti_v2p2_overview.html, (Issued 1st September, 2015).
- [28] P. Santos, D. Hernández-Leo, M. Pérez-Sanagustín, and J. Blat, "Modeling the Computing Based Testing domain extending IMS QTI: Framework, models and exemplary implementations," *Computers in Human Behavior*, vol. 28, no. 5, pp. 1648-1662, 2012.
- [29] K. Gramatova, S. Stoyanov, E. Doychev, and V. Valkanov, "Integration of eTesting in an IoT eLearning ecosystem: Virtual eLearning Space," in *Proceedings of the 7th Balkan Conference on Informatics Conference*, p. 14: ACM, 2015.
- [30] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: The Definitive Guide: Time to Relax*. " O'Reilly Media, Inc.", 2010.

- [31] *MQTT Essentials Part 6: Quality of Service 0, 1 & 2*. Available: <http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>, 2015.
- [32] D. Hardt, "The OAuth 2.0 authorization framework". Available: <https://tools.ietf.org/html/rfc6749>, 2012.
- [33] P. Teixeira, *Instant Node.js Starter*. Packt Publishing Ltd, 2013.
- [34] G. Kunz, *Mastering Angular 2 Components*. Packt Publishing Ltd, 2016.
- [35] J. Mesnil, *Mobile and Web Messaging: Messaging Protocols for Web and Mobile Devices*. " O'Reilly Media, Inc.", 2014.
- [36] A. Sill, "Standards at the Edge of the Cloud," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 63-67, 2017.
- [37] H. Jaygarl, C. Luo, Y. Kim, E. Choi, and K. Bradwick, *Professional Tizen Application Development*. John Wiley & Sons, 2014.
- [38] *Node.js Quickstart, Classroom API, Google Developers*. Available: <https://developers.google.com/classroom/quickstart/nodejs>, 2017.
- [39] S. Y. Shapsough, "An IoT Architecture for Ubiquitous Context-Aware Assessments", American University of Sharjah. Available: <http://hdl.handle.net/11073/8910>, 2017.
- [40] *PrimeNG*. Available: <https://www.primefaces.org/primeng/#/>, 2017.
- [41] *Samsung Gear S2 classic - Full device specifications*. Available: https://www.gsmarena.com/samsung_gear_s2_classic-7677.php, 2017.
- [42] *exceljs nodejs library*. Available: <https://www.npmjs.com/package/exceljs>, 2017.
- [43] A. Tokmakoff, B. Sparrow, D. Turner, and A. Lowe, "AusPlots Rangelands field data collection and publication: Infrastructure for ecological monitoring," *Future Generation Computer Systems*, vol. 56, pp. 537-549, 2016.
- [44] *thombashi/tconfig*. Available: <https://github.com/thombashi/tconfig>, 2017.
- [45] *Polidea/Cellular-Data-Network-Simulator*. Available: <https://github.com/Polidea/Cellular-Data-Network-Simulator>, 2017.
- [46] *ATC - A tool to simulate network conditions*. Available: <http://facebook.github.io/augmented-traffic-control/>, 2017.
- [47] *ClockSync - Android Apps on Google Play*. Available: <https://play.google.com/store/apps/details?id=ru.org.amip.ClockSync&hl=en>, 2017.
- [48] *Time Server - Android Apps on Google Play*. Available: <https://play.google.com/store/apps/details?id=com.icecoldapps.timeserver&hl=en>, 2017.
- [49] M. Griffiths. *NetTime - Network Time Synchronization Tool*. Available: <http://www.timesynctool.com/>, 2017.
- [50] A. Pande, *JQuery 2 Recipes: A Problem-solution Approach*. Apress, 2014.
- [51] *augmented-traffic-control network profiles*. Available: <https://github.com/facebook/augmented-traffic-control>, 2017.
- [52] *Tasker - Android Apps on Google Play*. Available: <https://play.google.com/store/apps/details?id=net.dinglich.android.taskerm>, 2017.
- [53] *TF: Task Light - Android Apps on Google Play*. Available: <https://play.google.com/store/apps/details?id=com.devuni.flashlight.tasklight>, 2017.

- [54] *Exynos Processors Comparison- Wikipedia*. Available: <https://en.wikipedia.org/wiki/Exynos>, 2017.
- [55] *what are the main differences between cortex A7, A9, A53 - Discussions - Processors - Arm Community*. Available: <https://community.arm.com/processors/f/discussions/5967/what-are-the-main-differences-between-cortex-a7-a9-a53>, 2017.
- [56] M. Al-Solh. *Multiple Watches Test Environment Demo For the Wearable School Platform*. Available: <https://youtu.be/GcliTXkNbxM>, 2017.
- [57] *alsolh/wearable-school-workspace*. Available: <https://github.com/alsolh/wearable-school-workspace>, 2017.

Appendix A: Sample dynamic assessment objects

```
{
  "assessmentItem": {
    "identifier": "pedometer1",
    "adaptive": "false",
    "timeDependent": "false",
    "context": {
      "sensorNames": ["PEDOMETER"],
      "eligibility": "(<PEDOMETER.accumulativeDistance> > 0) && (<PEDOMETER.accumulativeTotalStepCount> > 0)"
    }
  },
  "responseDeclaration": {
    "identifier": "RESPONSE",
    "cardinality": "single",
    "baseType": "identifier",
    "correctResponse": "<PEDOMETER.accumulativeDistance> / <PEDOMETER.accumulativeTotalStepCount>"
  },
  "outcomeDeclaration": {
    "identifier": "SCORE",
    "cardinality": "single",
    "baseType": "float",
    "defaultValue": {
      "value": "0.0"
    }
  },
  "itemBody": {
    "p": "You have walked <PEDOMETER.accumulativeTotalStepCount> steps and total distance was <PEDOMETER.accumulativeDistance> meters",
    "choiceInteraction": {
      "responseIdentifier": "RESPONSE",
      "shuffle": "true",
      "maxChoices": "1",
      "prompt": "How long is each step in meters?"
    }
  },
  "responseProcessing": {
    "template": "http://www.imsglobal.org/question/qti_v2p1/rptemplates/match_correct"
  }
}
```

Figure A.1: Sample Dynamic Assessment Object: Pedometer Utilization

```

{"assessmentItem": {
  "identifier": "gps1",
  "adaptive": "false",
  "timeDependent": "false",
  "context": {
    "sensorNames": ["GPS"],
    "eligibility": "((<GPS.longitude> > -1000) && (<GPS.longitude> < 1000))"
  }
},
"responseDeclaration": {
  "identifier": "RESPONSE",
  "cardinality": "single",
  "baseType": "identifier",
  "correctResponse": "<var0>/500"
},
"outcomeDeclaration": {
  "identifier": "SCORE",
  "cardinality": "single",
  "baseType": "float",
  "defaultValue": {
    "value": "0.0"
  }
},
"itemBody": {"p": "if you are in a plane flying at the speed of 500 Km per hour and the Eiffel tower is currently <var0> Km away from you.",
  "variables":
["getDistanceFromLonLatInKm(<GPS.longitude>,<GPS.latitude>,2.2922926,48.8583736)"],
  "choiceInteraction": {
    "responseIdentifier": "RESPONSE",
    "shuffle": "true",
    "maxChoices": "1",
    "prompt": "how long will it take to reach there in hours time?"
  }
},
"responseProcessing": {
  "template": "http://www.imsglobal.org/question/qti_v2p1/vrptemplates/match_correct"
}
}

```

Figure A.2: Sample Dynamic Assessment Object: GPS Sensor Utilization

```

{
  "assessmentItem": {
    "identifier": "light1",
    "adaptive": "false",
    "timeDependent": "false",
    "context": {
      "sensorNames": [
        "LIGHT"
      ],
      "eligibility": "<LIGHT.lightLevel> > -1"
    }
  },
  "responseDeclaration": {
    "identifier": "RESPONSE",
    "cardinality": "single",
    "baseType": "identifier",
    "correctResponse": "if(<LIGHT.lightLevel> < 10) 'fonce' else 'lumire'"
  },
  "outcomeDeclaration": {
    "identifier": "SCORE",
    "cardinality": "single",
    "baseType": "float",
    "defaultValue": {
      "value": "0.0"
    }
  },
  "itemBody": {
    "p": "in french language, describe the current area lighting condition",
    "variables": null,
    "choiceInteraction": {
      "responseIdentifier": "RESPONSE",
      "shuffle": "true",
      "maxChoices": "1",
      "prompt": "is it light or dark?"
    }
  },
  "responseProcessing": {
    "template": "http://www.imsglobal.org/question/qti_v2p1/rptemplates/match_correct"
  }
}

```

Figure A.3: Sample Dynamic Assessment Object: Light Sensor Utilization

Appendix B: Frequency distributions and other graphs related to response time

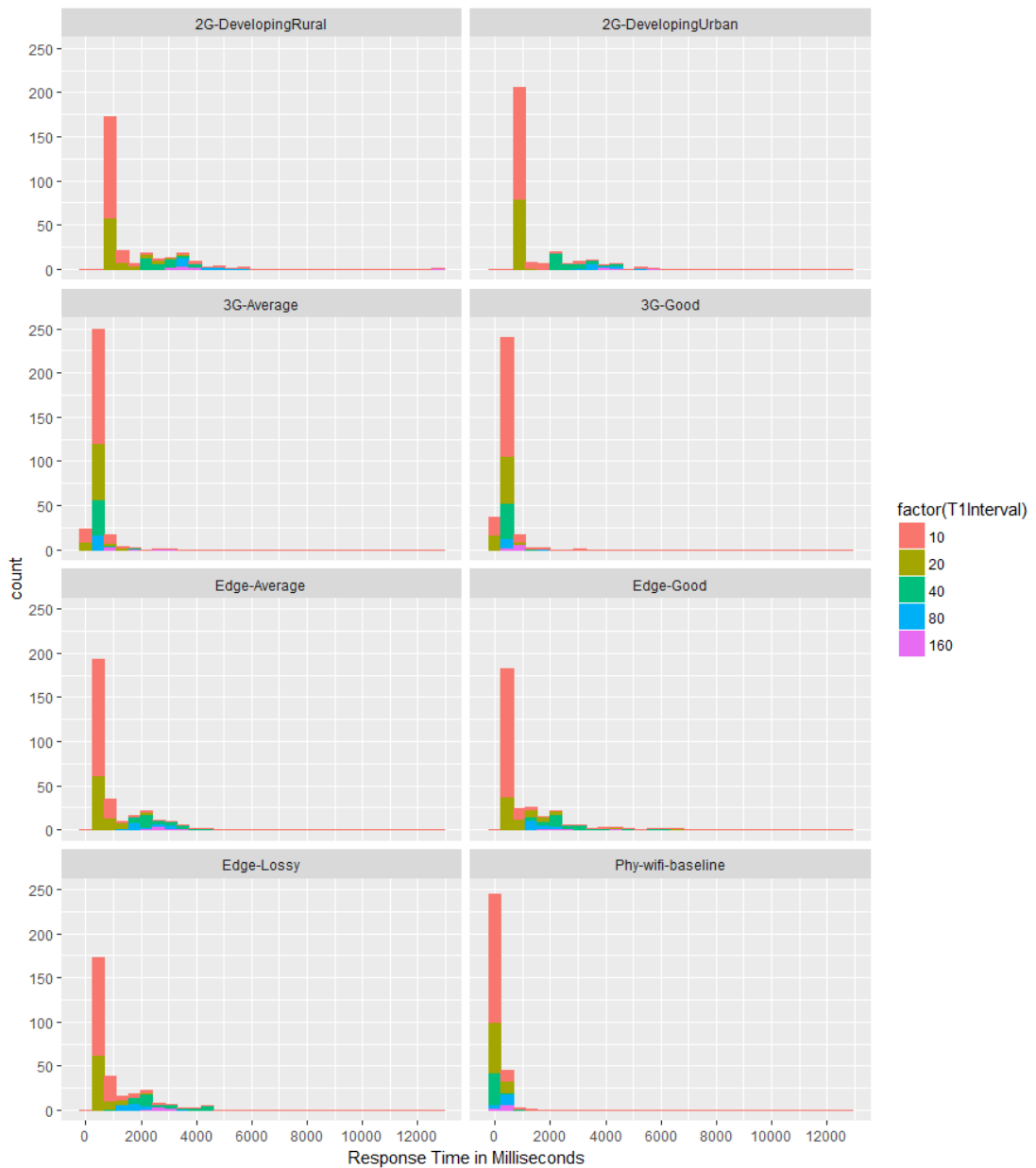


Figure B.1: Frequency distribution of response time for telemetry submission transaction for each interval and each network profile with 0% hit scenario

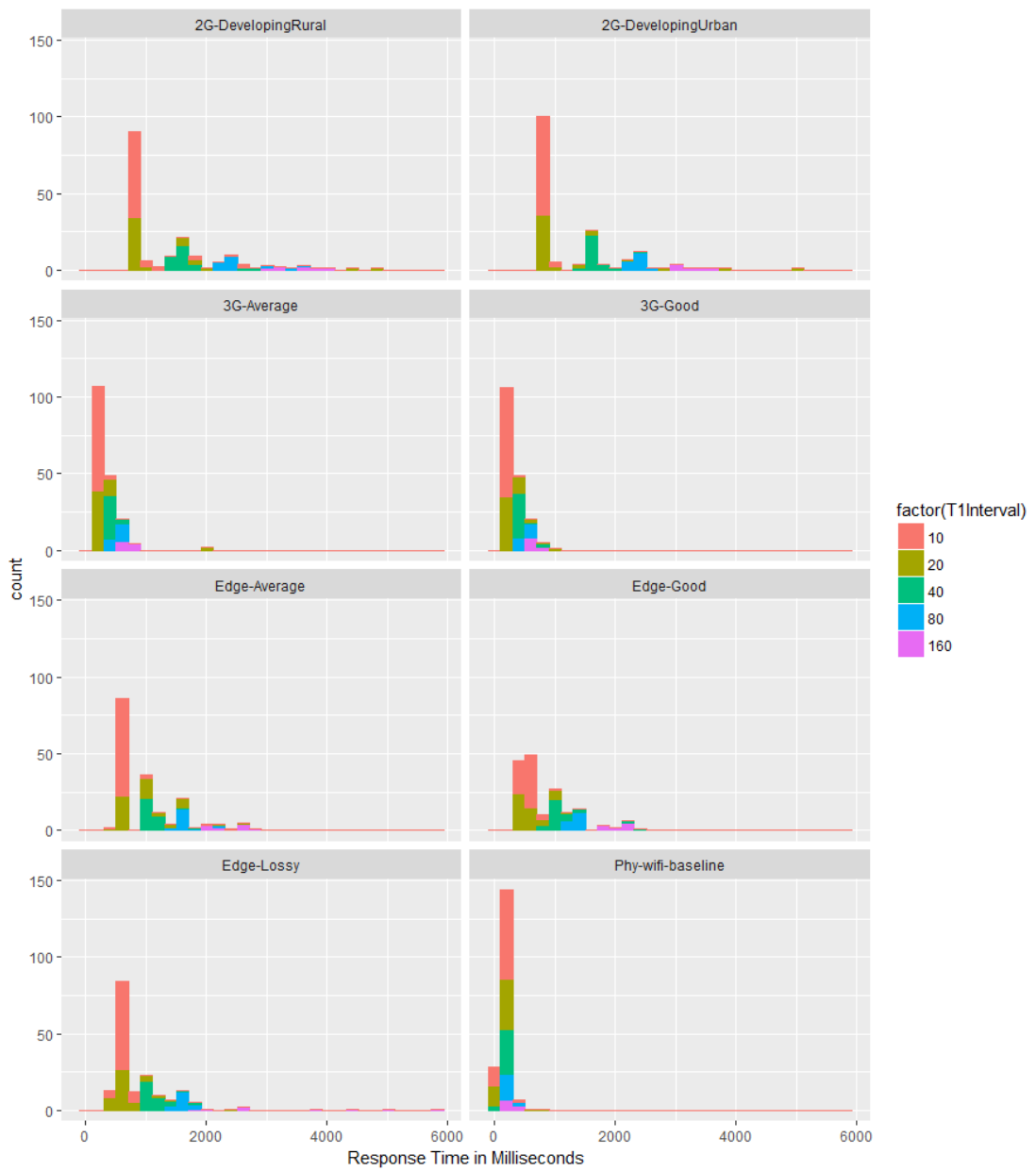


Figure B.2: Frequency distribution of response time for telemetry submission transaction for each interval and each network profile with 100% hit scenario

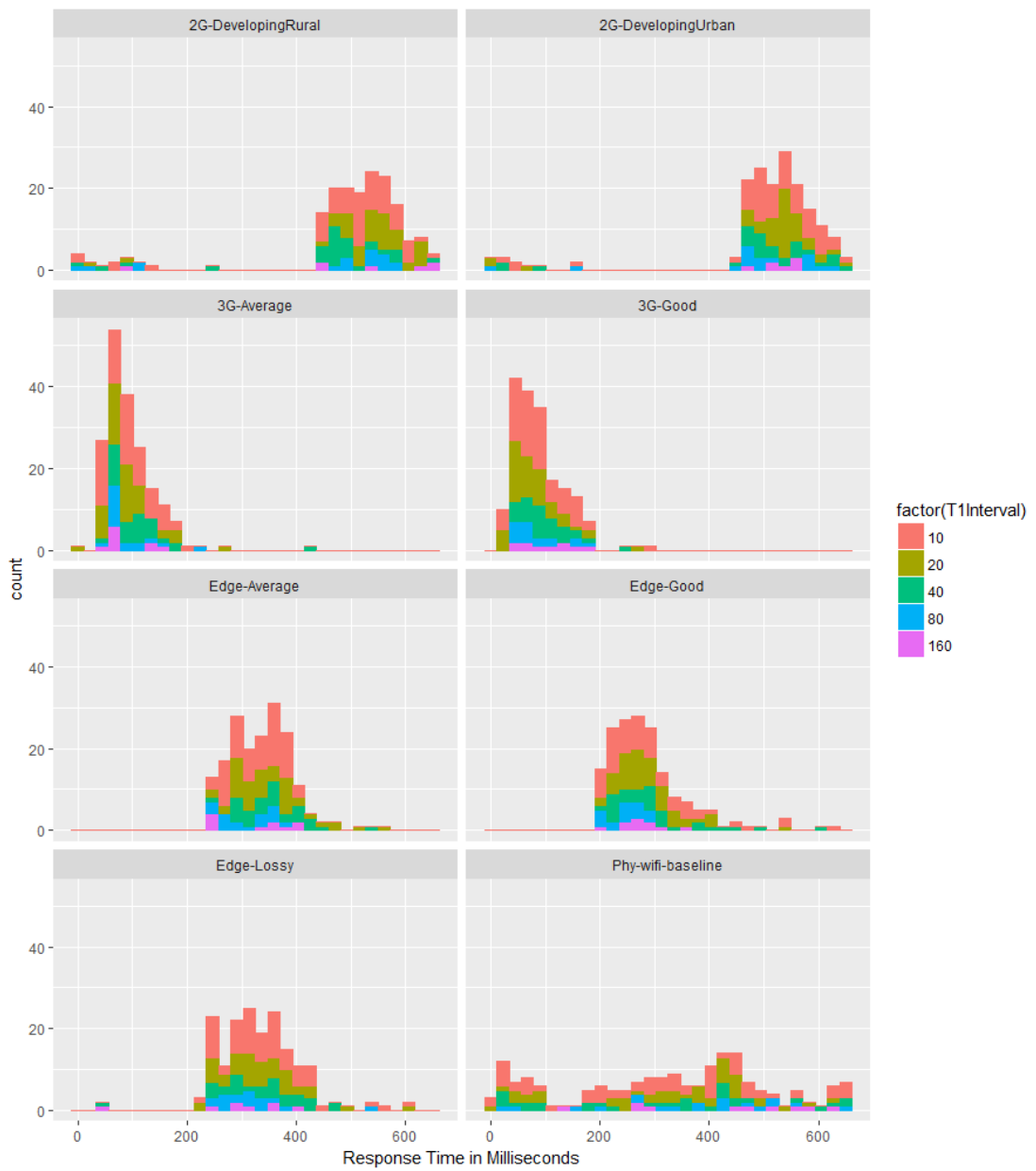


Figure B.3: Frequency distribution of response time for answer posting transaction for each interval and each network profile with 100% hit scenario

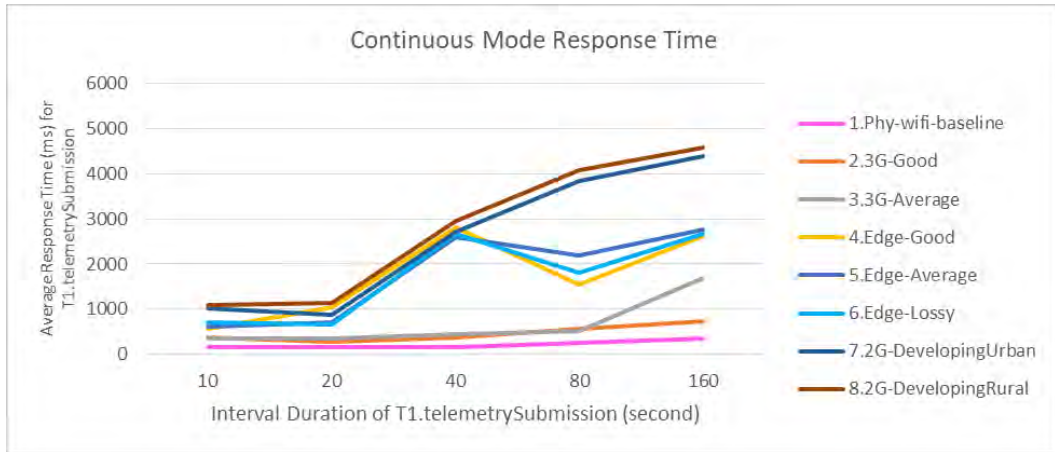


Figure B.4: Continuous mode line chart, 0% hit scenario average response time for T1.telemetrySubmission per network profile and T1.telemetrySubmission interval

Appendix C: Additional graphs for CPU usage over time

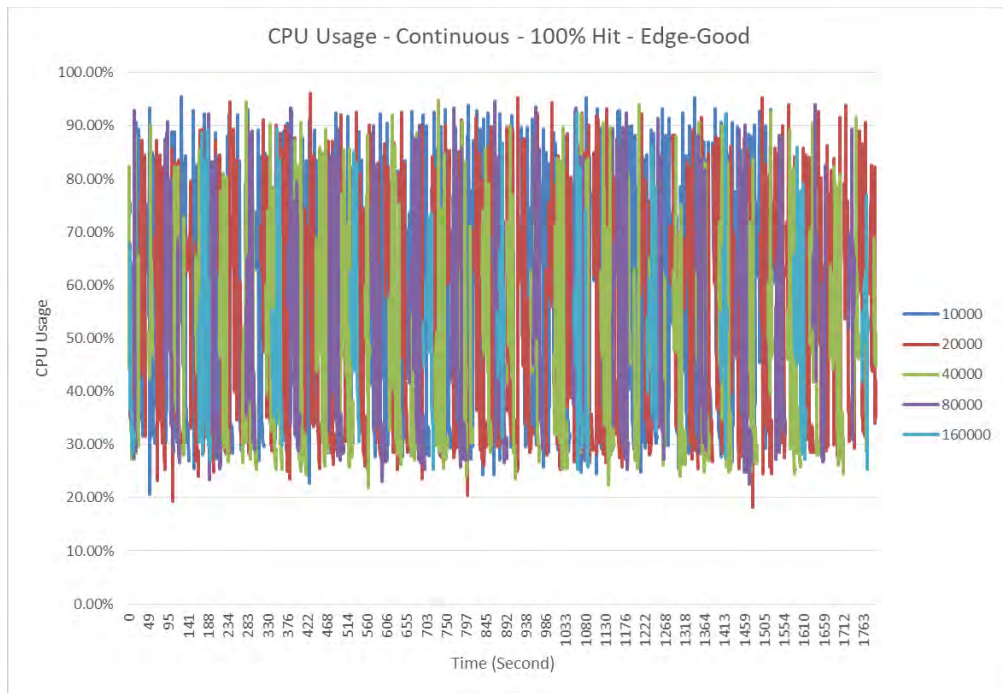


Figure C.1: Continuous mode CPU usage within the 30-minute episode run with Edge-Good network profile and 100% hit scenario

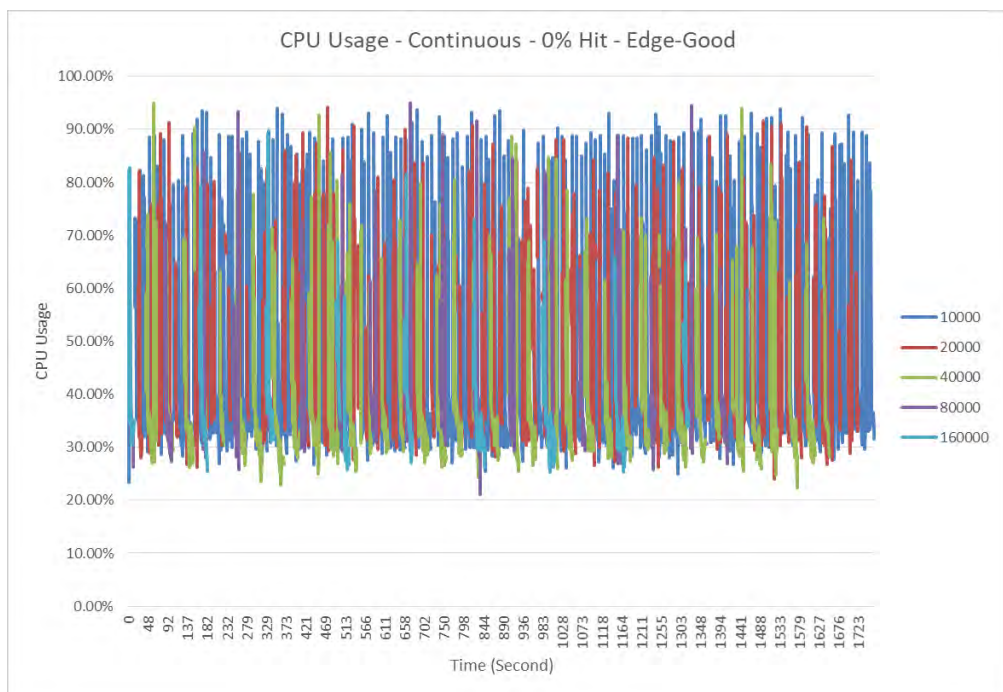


Figure C.2: Continuous mode CPU usage within the 30-minute episode run with Edge-Good network profile and 0% hit scenario

Appendix D: Frequency distributions for CPU Max and %CPU-Second

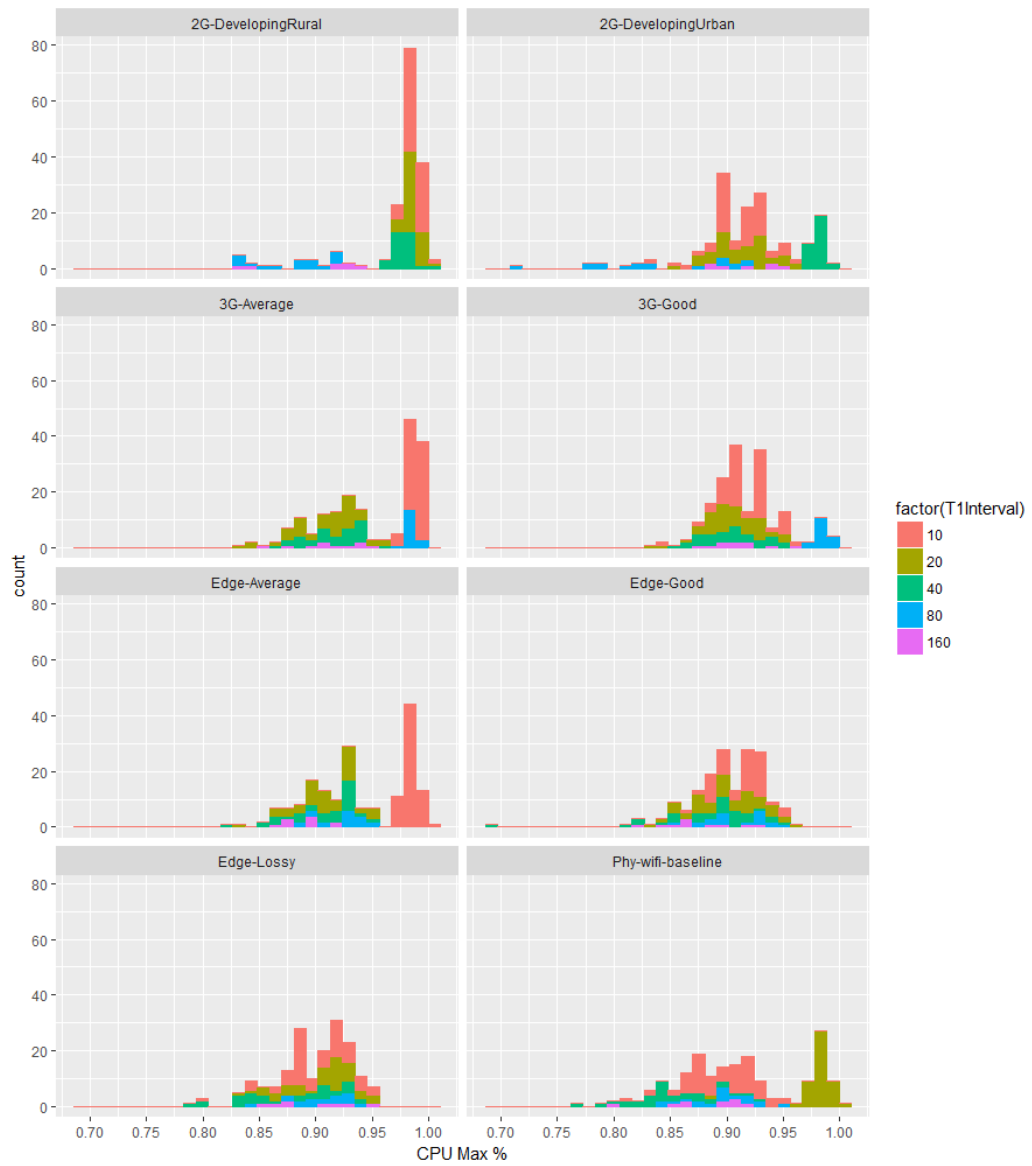


Figure D.1: Frequency distribution of CPU max for telemetry submission transaction for each interval and each network profile with 100% hit scenario

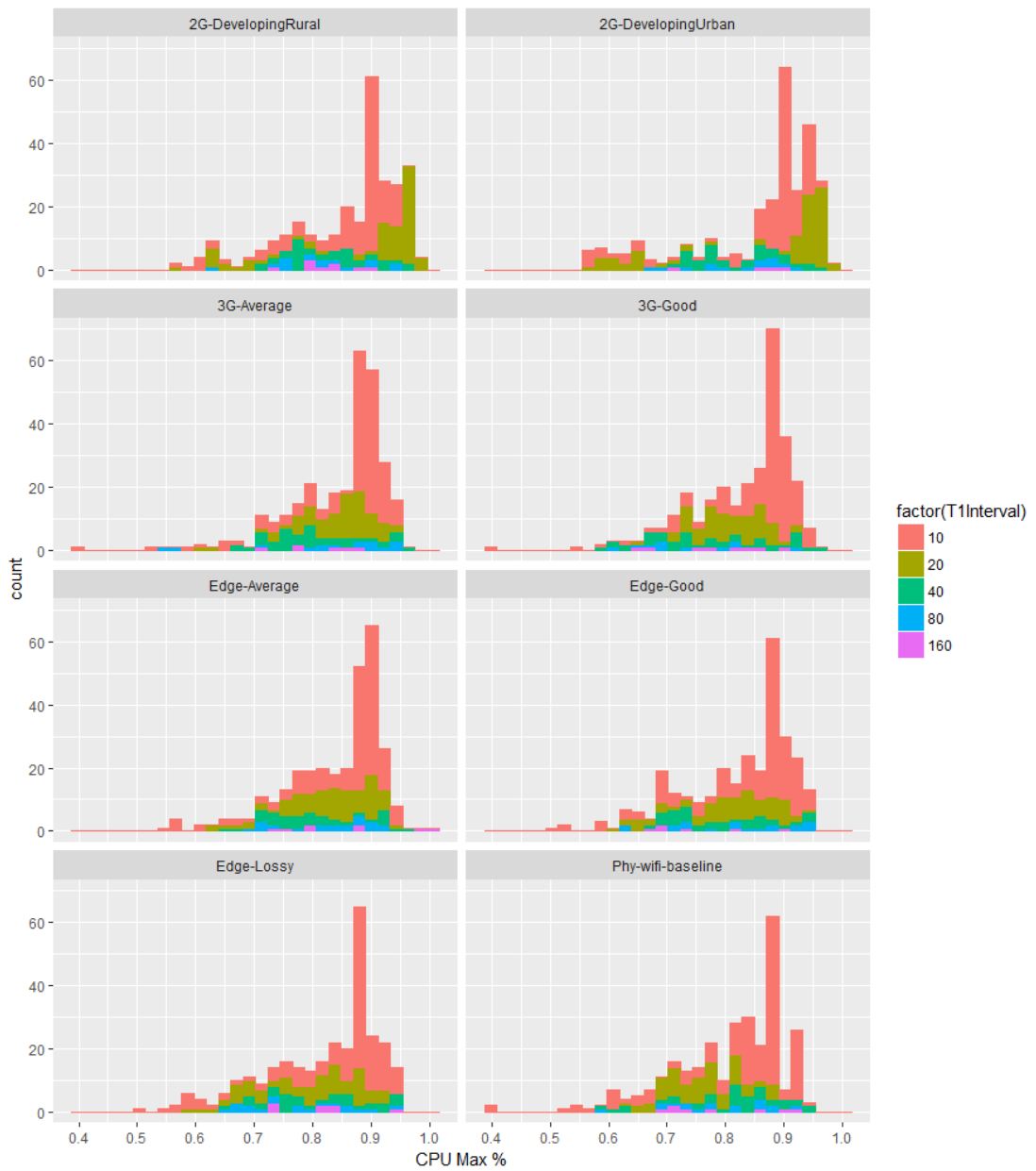


Figure D.2: Frequency distribution of CPU Max for telemetry submission transaction for each interval and each network profile with 0% hit scenario

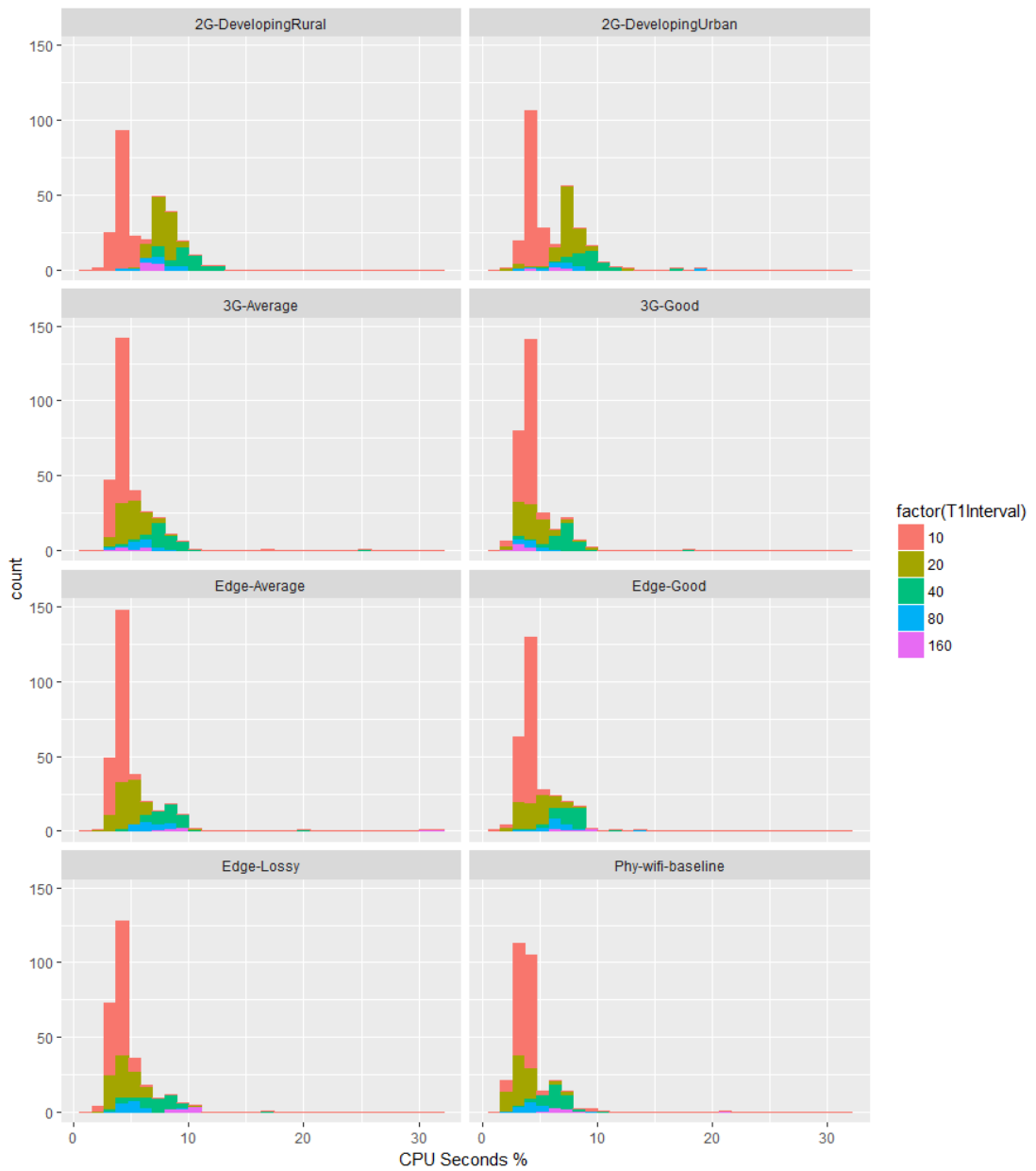


Figure D.3: Frequency distribution of %CPU-Seconds for telemetry submission transaction for each interval and each network profile with 0% hit scenario

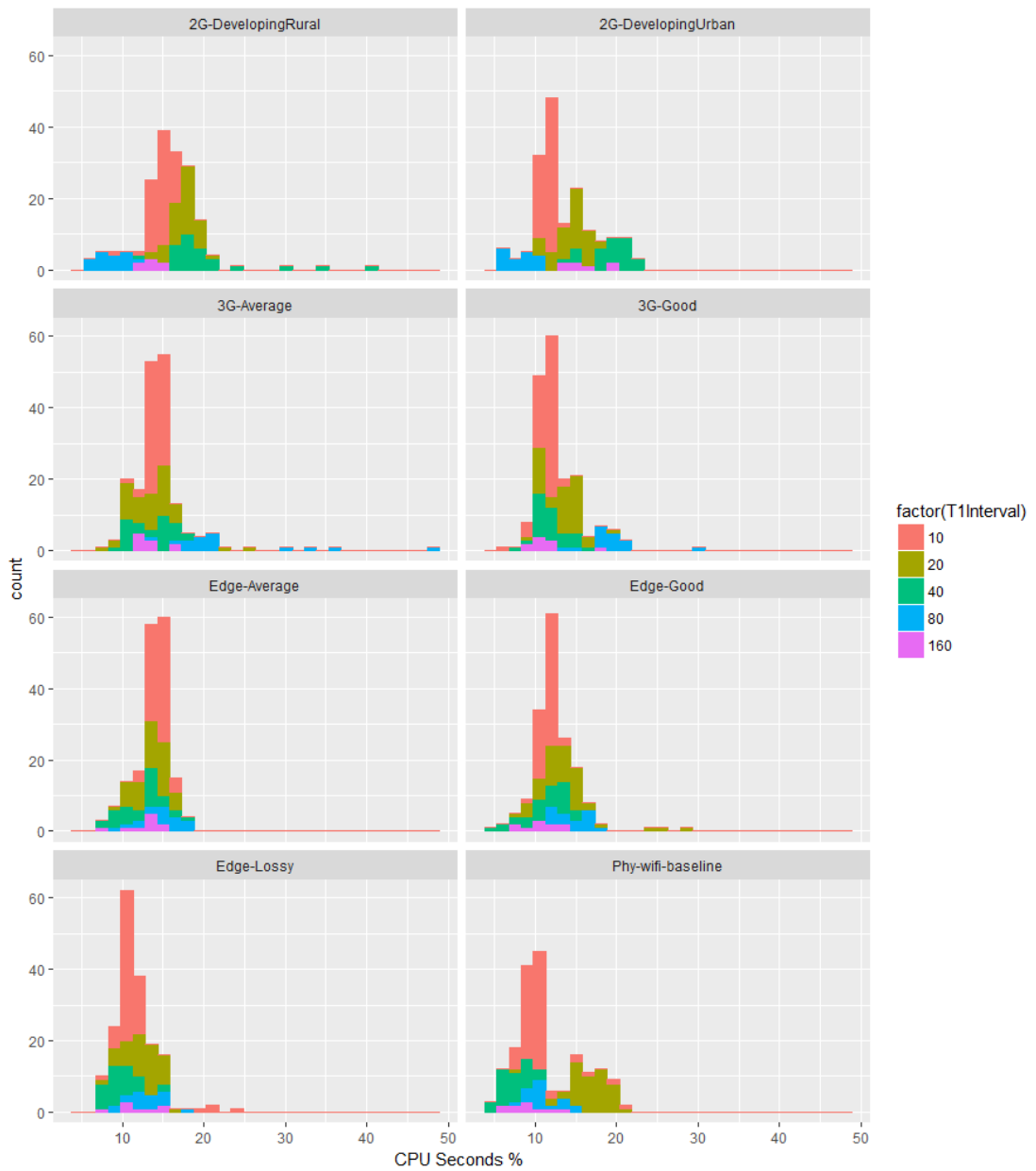


Figure D.4: Frequency distribution of %CPU-Seconds for each interval of T1.telemetrySubmission and each network profile with 100% hit scenario

Appendix E: Additional memory variation graphs

Memory Variation Over Time - Edge-Average

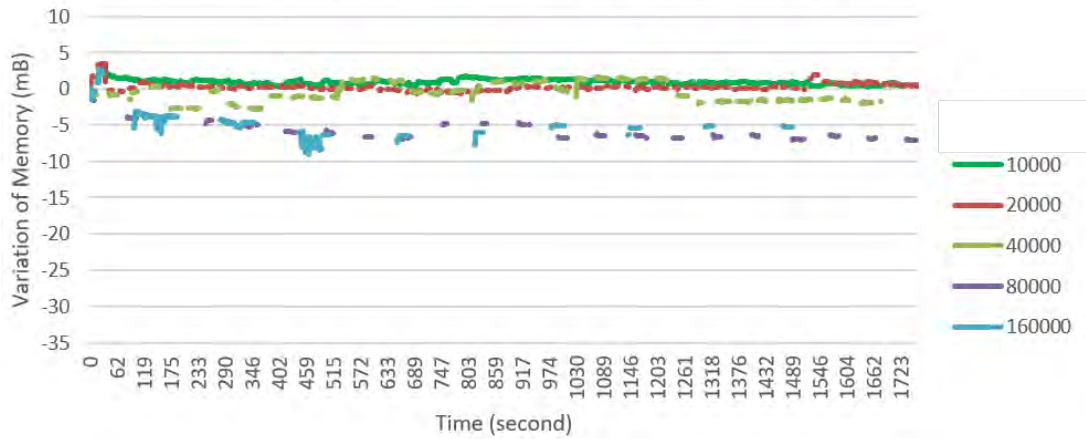


Figure E.1: Continuous mode memory usage for each interval of T1.telemetrySubmission in Edge-Average network with 0% hit scenario

Memory Variation Over Time - 3G-Good

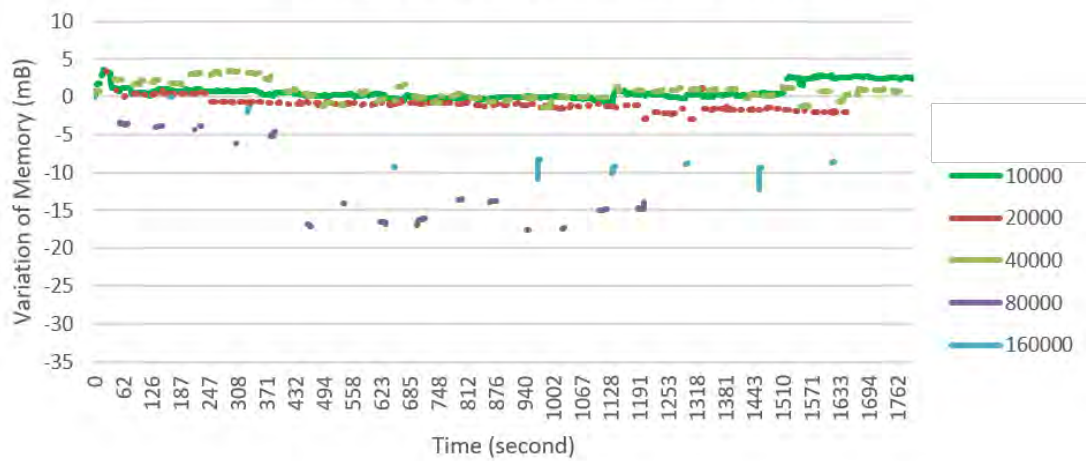


Figure E.2: Continuous mode memory usage for each interval of T1.telemetrySubmission in 3G-Good network with 0% hit scenario

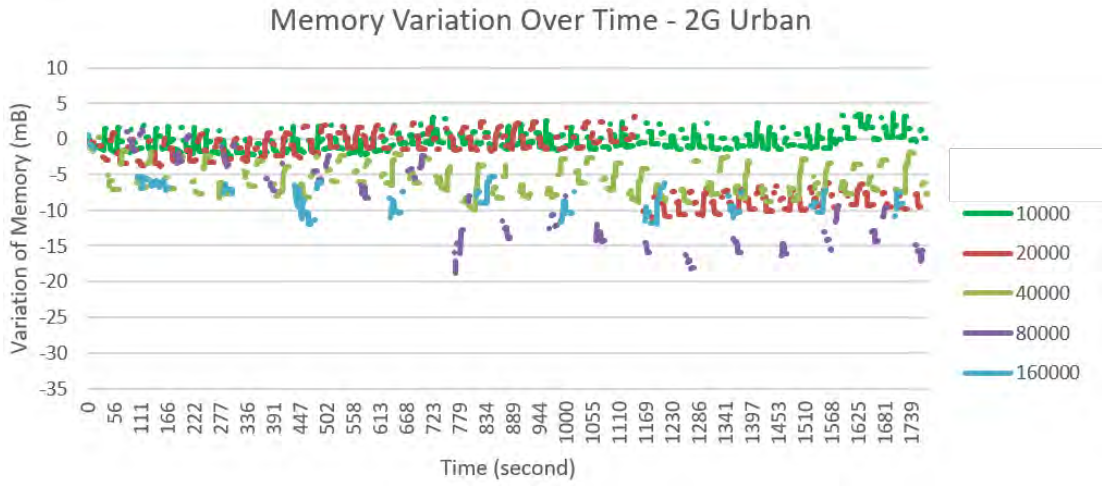


Figure E.3: Continuous mode memory usage for each interval of T1.telemetrySubmission in 2G-Urban network with 100% hit scenario

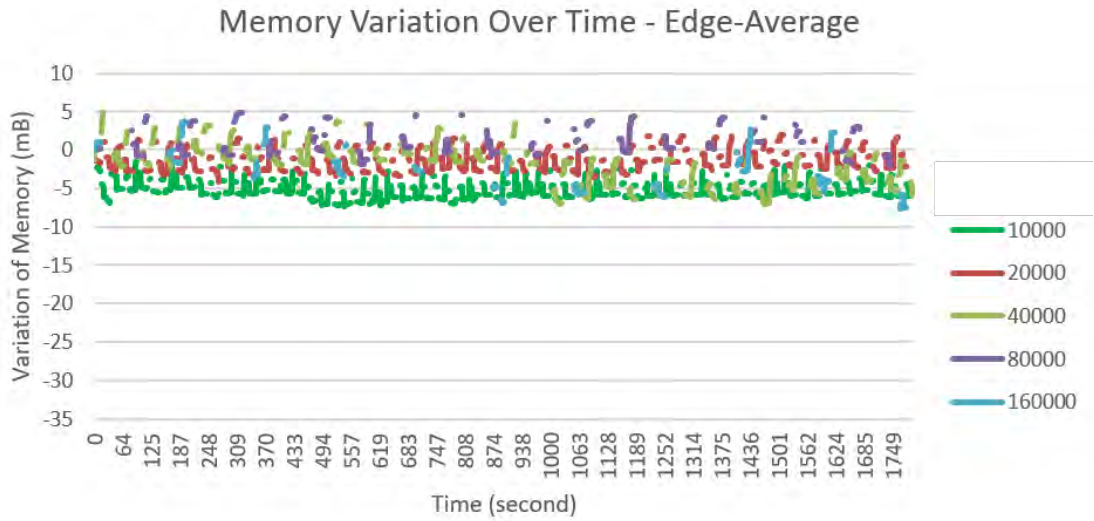


Figure E.4: Continuous mode memory usage for each interval of T1.telemetrySubmission in Edge-Average network with 100% hit scenario

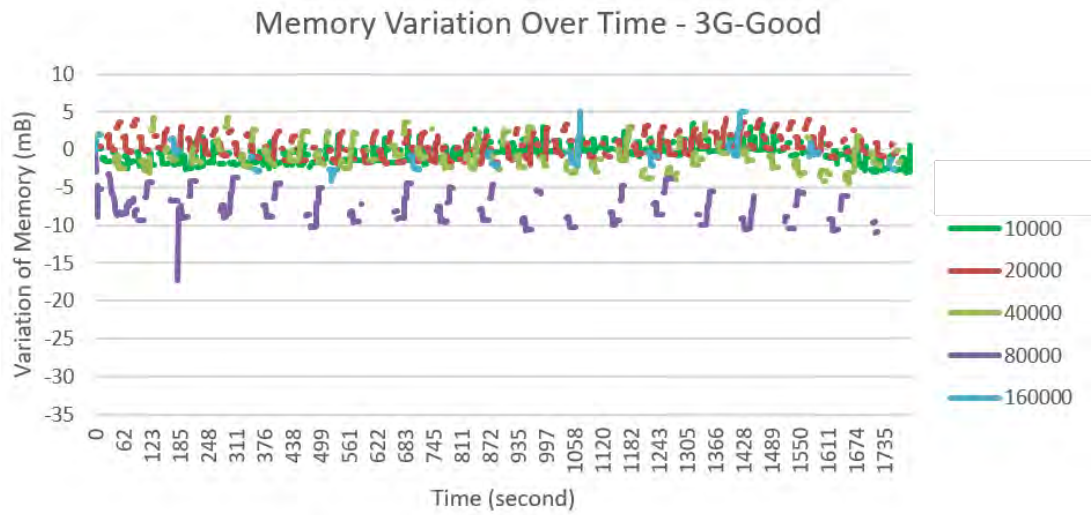


Figure E.5: Continuous mode memory usage for each interval of T1.telemetrySubmission in 3G-Good network with 100% hit scenario

Appendix F: Additional descriptive statistics

Table F.1: Continuous descriptive statistics for the T1.telemetrySubmission transaction

Network Profile	T1Interval	hit %	mean	median	min	max	SD
2G-DevelopingRural	10	0	1093.15	864	742	5541	739.31
2G-DevelopingRural	10	100	948.13	798	750	2669	443.06
2G-DevelopingRural	20	0	1145.76	832.5	715	3363	622.07
2G-DevelopingRural	20	100	1142.49	799	737	4838	831.54
2G-DevelopingRural	40	0	2961.49	2832	2043	4954	721.31
2G-DevelopingRural	40	100	1634.94	1546	1420	2784	309.36
2G-DevelopingRural	80	0	4067.19	3576.5	3380	5912	829.12
2G-DevelopingRural	80	100	2561.61	2428.5	2235	3533	396.72
2G-DevelopingRural	160	0	4566.78	3608	3116	12787	3093.83
2G-DevelopingRural	160	100	3517.86	3549	3048	4113	370.81
2G-DevelopingUrban	10	0	1032.21	862	751	5106	589.48
2G-DevelopingUrban	10	100	814.27	819	752	945	44.56
2G-DevelopingUrban	20	0	868.83	822	756	2579	247.26
2G-DevelopingUrban	20	100	1118.77	800	754	5005	814.87
2G-DevelopingUrban	40	0	2720.6	2405	2064	4485	622.01
2G-DevelopingUrban	40	100	1663.13	1610	1502	2493	183.3
2G-DevelopingUrban	80	0	3846.71	3592	2928	5086	586.99
2G-DevelopingUrban	80	100	2358.17	2375.5	2210	2585	94.95
2G-DevelopingUrban	160	0	4372	4051.5	3888	5497	760.77
2G-DevelopingUrban	160	100	3209.43	3111	3004	3693	244.17
3G-Average	10	0	345.14	290	182	1498	193.32
3G-Average	10	100	233.7	229	182	315	31.73
3G-Average	20	0	363.01	316.5	199	1334	196.62
3G-Average	20	100	325.39	238	179	2027	346.09
3G-Average	40	0	443.14	401	340	1602	201.83
3G-Average	40	100	392.94	373.5	312	675	72.06
3G-Average	80	0	510.25	513.5	454	573	41.13
3G-Average	80	100	523.94	522.5	447	651	50.39
3G-Average	160	0	1697.83	1316	784	3144	1021.84
3G-Average	160	100	693	663	616	827	78.23
3G-Good	10	0	369.08	298	182	3021	288.06
3G-Good	10	100	215.88	213	171	351	31

3G-Good	20	0	288.5	261.5	190	823	119.11
3G-Good	20	100	299.64	246	173	1096	162.23
3G-Good	40	0	386.67	353.5	293	1347	158.55
3G-Good	40	100	400.94	362.5	310	819	115.37
3G-Good	80	0	574.45	473	428	1564	329.87
3G-Good	80	100	496.35	507	410	566	43.17
3G-Good	160	0	736.75	694	628	985	125.13
3G-Good	160	100	639.7	648	531	726	67.5
Edge-Average	10	0	621.88	586	492	2302	199.76
Edge-Average	10	100	623.65	558	486	2388	288.26
Edge-Average	20	0	700.08	573	481	2216	327.64
Edge-Average	20	100	958.27	974	494	2700	487.77
Edge-Average	40	0	2595.3	2351	1599	4375	754.45
Edge-Average	40	100	1106.03	1051.5	948	1866	190.57
Edge-Average	80	0	2190.35	1992	1469	3315	655.55
Edge-Average	80	100	1609	1583.5	1446	2147	148.73
Edge-Average	160	0	2763	2733.5	2118	3463	405.57
Edge-Average	160	100	2364.9	2382.5	1999	2742	300.07
Edge-Good	10	0	568.52	511	434	3745	285.35
Edge-Good	10	100	543.41	514	453	944	89.04
Edge-Good	20	0	1041.46	640	470	6720	936.84
Edge-Good	20	100	597.96	515	440	1174	193.55
Edge-Good	40	0	2808.08	2386	1456	5939	1130.13
Edge-Good	40	100	1113.34	1007.5	841	2357	341.08
Edge-Good	80	0	1543.28	1495	1294	2410	286.58
Edge-Good	80	100	1367.94	1381	1244	1484	85.19
Edge-Good	160	0	2622.67	2360	1942	4289	839.05
Edge-Good	160	100	2058.9	2128	1757	2292	189.11
Edge-Lossy	10	0	701.05	589	489	3080	396.17
Edge-Lossy	10	100	574.77	548	491	835	77.37
Edge-Lossy	20	0	653.82	581	499	1978	246.44
Edge-Lossy	20	100	686.46	566.5	476	2409	331.28
Edge-Lossy	40	0	2660.59	2360	943	4360	968.2
Edge-Lossy	40	100	1125.88	1066.5	943	1865	208.87
Edge-Lossy	80	0	1812.42	1616	1409	3306	467.97
Edge-Lossy	80	100	1599.61	1578.5	1455	1804	102.79
Edge-Lossy	160	0	2657.75	2655	2006	2993	313.51
Edge-Lossy	160	100	3542.13	3203	1908	5893	1491.93
Phy-wifi-baseline	10	0	156.59	123	64	1284	146.92
Phy-wifi-baseline	10	100	134.38	124.5	73	311	46.26
Phy-wifi-baseline	20	0	161.63	136	74	617	83.81
Phy-wifi-baseline	20	100	141.35	114	71	815	120.5
Phy-wifi-baseline	40	0	172.05	140	82	1045	150.84
Phy-wifi-baseline	40	100	162.63	154.5	68	265	50.52

Phy-wifi-baseline	80	0	269.24	262	146	408	73.42
Phy-wifi-baseline	80	100	192.89	158	112	435	82.82
Phy-wifi-baseline	160	0	357.88	303	179	597	151.37
Phy-wifi-baseline	160	100	259.2	267.5	167	347	59.22

Table F.2: Continuous descriptive statistics for the T2.answerPosting transaction

Network Profile	T1Interval	hit %	mean	median	min	max	SD
2G-DevelopingRural	10	100	836.12	779.5	692	2224	240.02
2G-DevelopingRural	20	100	828.85	798	705	1766	155.80
2G-DevelopingRural	40	100	942.55	737	682	4651	727.40
2G-DevelopingRural	80	100	947.56	799	732	1937	375.22
2G-DevelopingRural	160	100	972.71	894	697	1791	378.98
2G-DevelopingUrban	10	100	850.12	787	706	2727	271.08
2G-DevelopingUrban	20	100	817.48	787	722	1590	132.41
2G-DevelopingUrban	40	100	849.70	777.5	708	1760	211.35
2G-DevelopingUrban	80	100	892.44	759.5	702	2522	417.05
2G-DevelopingUrban	160	100	778.57	783	719	817	34.15
3G-Average	10	100	191.45	177	138	387	48.49
3G-Average	20	100	186.00	174	0	483	65.07
3G-Average	40	100	213.31	190	145	662	93.43
3G-Average	80	100	190.28	167	153	418	61.31
3G-Average	160	100	182.30	163.5	139	271	41.35
3G-Good	10	100	187.76	172.5	130	500	58.15
3G-Good	20	100	178.46	163	126	479	59.76
3G-Good	40	100	198.78	181	143	467	61.83
3G-Good	80	100	183.82	158	140	320	53.62
3G-Good	160	100	202.20	190	147	308	53.62
Edge-Average	10	100	533.43	542	458	629	42.38
Edge-Average	20	100	558.94	543	462	804	74.84
Edge-Average	40	100	564.47	554.5	462	782	65.31
Edge-Average	80	100	511.78	502.5	460	572	41.43
Edge-Average	160	100	531.90	547	455	625	65.55
Edge-Good	10	100	502.29	474	393	901	100.43
Edge-Good	20	100	501.22	486	399	777	64.04
Edge-Good	40	100	520.03	497	415	870	100.86
Edge-Good	80	100	453.72	456	391	508	34.15
Edge-Good	160	100	485.30	485.5	416	571	40.66
Edge-Lossy	10	100	549.72	534	446	863	81.13
Edge-Lossy	20	100	542.50	529.5	431	855	78.33
Edge-Lossy	40	100	544.97	553.5	139	731	104.81
Edge-Lossy	80	100	526.17	515	461	782	72.67

Edge-Lossy	160	100	646.00	543.5	460	1445	326.15
Phy-wifi-baseline	10	100	86.13	70.5	30	496	62.91
Phy-wifi-baseline	20	100	78.06	65	0	181	41.93
Phy-wifi-baseline	40	100	116.22	83.5	40	448	88.78
Phy-wifi-baseline	80	100	118.53	66	41	759	163.84
Phy-wifi-baseline	160	100	83.90	74.5	47	219	50.06

Table F.3: Continuous descriptive statistics for max CPU results

Network Profile	T1Interval	hit %	mean	median	min	max	SD
2G-DevelopingRural	10	0	0.86	0.89	0.57	0.95	0.09
2G-DevelopingRural	10	100	0.97	0.98	0.93	1.00	0.01
2G-DevelopingRural	20	0	0.89	0.94	0.57	0.98	0.12
2G-DevelopingRural	20	100	0.98	0.98	0.97	1.00	0.01
2G-DevelopingRural	40	0	0.83	0.82	0.71	0.97	0.07
2G-DevelopingRural	40	100	0.97	0.97	0.96	1.00	0.01
2G-DevelopingRural	80	0	0.81	0.81	0.64	0.94	0.08
2G-DevelopingRural	80	100	0.88	0.89	0.83	0.92	0.03
2G-DevelopingRural	160	0	0.82	0.82	0.73	0.89	0.05
2G-DevelopingRural	160	100	0.90	0.92	0.83	0.94	0.04
2G-DevelopingUrban	10	0	0.86	0.90	0.56	0.96	0.10
2G-DevelopingUrban	10	100	0.93	0.93	0.78	0.99	0.04
2G-DevelopingUrban	20	0	0.87	0.94	0.56	0.98	0.14
2G-DevelopingUrban	20	100	0.91	0.92	0.85	0.96	0.03
2G-DevelopingUrban	40	0	0.82	0.82	0.71	0.96	0.07
2G-DevelopingUrban	40	100	0.98	0.98	0.97	0.99	0.01
2G-DevelopingUrban	80	0	0.82	0.84	0.68	0.91	0.08
2G-DevelopingUrban	80	100	0.84	0.83	0.72	0.92	0.06
2G-DevelopingUrban	160	0	0.84	0.86	0.72	0.91	0.08
2G-DevelopingUrban	160	100	0.92	0.92	0.88	0.95	0.03
3G-Average	10	0	0.86	0.89	0.40	0.95	0.08
3G-Average	10	100	0.97	0.98	0.90	0.99	0.02
3G-Average	20	0	0.84	0.86	0.60	0.95	0.06
3G-Average	20	100	0.91	0.92	0.83	0.96	0.03
3G-Average	40	0	0.80	0.79	0.67	0.96	0.08
3G-Average	40	100	0.92	0.92	0.87	0.96	0.02
3G-Average	80	0	0.83	0.88	0.55	0.95	0.12
3G-Average	80	100	0.98	0.98	0.97	0.99	0.01

3G-Average	160	0	0.81	0.81	0.72	0.89	0.06
3G-Average	160	100	0.91	0.91	0.86	0.95	0.03
3G-Good	10	0	0.86	0.88	0.40	0.94	0.08
3G-Good	10	100	0.93	0.93	0.70	0.99	0.03
3G-Good	20	0	0.81	0.81	0.65	0.92	0.06
3G-Good	20	100	0.91	0.91	0.84	0.96	0.03
3G-Good	40	0	0.77	0.76	0.59	0.96	0.11
3G-Good	40	100	0.90	0.90	0.85	0.95	0.03
3G-Good	80	0	0.73	0.69	0.60	0.92	0.09
3G-Good	80	100	0.98	0.98	0.97	0.99	0.01
3G-Good	160	0	0.78	0.79	0.65	0.90	0.09
3G-Good	160	100	0.91	0.91	0.87	0.96	0.03
Edge-Average	10	0	0.85	0.89	0.55	0.94	0.09
Edge-Average	10	100	0.98	0.98	0.95	1.00	0.01
Edge-Average	20	0	0.83	0.84	0.62	0.94	0.07
Edge-Average	20	100	0.91	0.91	0.83	0.95	0.03
Edge-Average	40	0	0.80	0.79	0.65	0.96	0.08
Edge-Average	40	100	0.91	0.92	0.82	0.95	0.03
Edge-Average	80	0	0.83	0.85	0.69	0.93	0.08
Edge-Average	80	100	0.92	0.93	0.89	0.95	0.02
Edge-Average	160	0	0.85	0.83	0.73	1.00	0.10
Edge-Average	160	100	0.89	0.90	0.86	0.92	0.02
Edge-Good	10	0	0.84	0.88	0.49	0.94	0.09
Edge-Good	10	100	0.93	0.93	0.87	0.99	0.03
Edge-Good	20	0	0.80	0.82	0.62	0.94	0.08
Edge-Good	20	100	0.90	0.90	0.83	0.96	0.03
Edge-Good	40	0	0.79	0.78	0.66	0.95	0.08
Edge-Good	40	100	0.88	0.90	0.69	0.95	0.05
Edge-Good	80	0	0.82	0.83	0.62	0.95	0.11
Edge-Good	80	100	0.91	0.91	0.85	0.95	0.03
Edge-Good	160	0	0.75	0.71	0.68	0.89	0.09
Edge-Good	160	100	0.87	0.86	0.82	0.92	0.03
Edge-Lossy	10	0	0.84	0.88	0.51	0.94	0.10
Edge-Lossy	10	100	0.92	0.93	0.80	1.00	0.04
Edge-Lossy	20	0	0.79	0.82	0.60	0.93	0.09
Edge-Lossy	20	100	0.90	0.91	0.84	0.95	0.03
Edge-Lossy	40	0	0.81	0.79	0.66	0.94	0.09
Edge-Lossy	40	100	0.88	0.89	0.79	0.94	0.04
Edge-Lossy	80	0	0.77	0.77	0.65	0.95	0.09
Edge-Lossy	80	100	0.91	0.91	0.84	0.94	0.03
Edge-Lossy	160	0	0.81	0.82	0.72	0.93	0.07
Edge-Lossy	160	100	0.89	0.89	0.85	0.95	0.03
Phy-wifi-baseline	10	0	0.83	0.87	0.39	0.93	0.10
Phy-wifi-baseline	10	100	0.92	0.92	0.81	0.99	0.04
Phy-wifi-baseline	20	0	0.76	0.76	0.64	0.89	0.06

Phy-wifi-baseline	20	100	0.98	0.98	0.89	1.00	0.02
Phy-wifi-baseline	40	0	0.81	0.82	0.59	0.94	0.09
Phy-wifi-baseline	40	100	0.85	0.84	0.76	0.93	0.04
Phy-wifi-baseline	80	0	0.78	0.77	0.58	0.92	0.09
Phy-wifi-baseline	80	100	0.90	0.90	0.84	0.95	0.03
Phy-wifi-baseline	160	0	0.79	0.75	0.68	0.92	0.09
Phy-wifi-baseline	160	100	0.88	0.90	0.80	0.91	0.04

Table F.4: Continuous descriptive statistics for %CPU-Seconds results

Network Profile	T1Interval	hit %	mean	median	min	max	SD
2G-DevelopingRural	10	0	4.15	4.05	2.52	6.48	0.61
2G-DevelopingRural	10	100	18.98	16.18	9.59	38.47	5.31
2G-DevelopingRural	20	0	7.71	7.78	4.87	9.34	0.80
2G-DevelopingRural	20	100	17.48	17.67	13.24	20.88	1.71
2G-DevelopingRural	40	0	9.67	9.86	7.03	12.75	1.47
2G-DevelopingRural	40	100	20.04	18.54	12.50	41.11	5.61
2G-DevelopingRural	80	0	7.37	7.53	4.54	9.23	1.42
2G-DevelopingRural	80	100	8.73	9.23	6.36	12.15	1.73
2G-DevelopingRural	160	0	6.78	6.80	6.38	7.12	0.26
2G-DevelopingRural	160	100	13.65	13.96	11.70	14.86	1.12
2G-DevelopingUrban	10	0	4.27	4.16	2.64	6.30	0.53
2G-DevelopingUrban	10	100	16.15	12.28	7.53	51.63	7.29
2G-DevelopingUrban	20	0	7.37	7.47	2.34	12.84	1.39
2G-DevelopingUrban	20	100	14.42	14.85	9.88	17.79	1.86
2G-DevelopingUrban	40	0	9.23	9.32	5.17	17.20	1.99
2G-DevelopingUrban	40	100	19.11	19.89	13.85	22.58	2.43
2G-DevelopingUrban	80	0	7.53	7.19	3.20	18.50	3.51
2G-DevelopingUrban	80	100	7.93	7.72	5.48	10.60	1.66
2G-DevelopingUrban	160	0	6.07	6.54	3.75	7.45	1.62
2G-DevelopingUrban	160	100	16.13	15.74	13.41	19.88	2.52
3G-Average	10	0	4.02	4.03	2.73	16.95	1.13
3G-Average	10	100	16.33	15.10	11.28	21.65	2.77
3G-Average	20	0	5.03	4.92	3.01	8.38	1.09
3G-Average	20	100	13.53	13.74	7.54	25.72	3.15
3G-Average	40	0	7.99	7.71	2.97	25.63	3.18
3G-Average	40	100	13.68	14.67	9.52	17.81	2.63
3G-Average	80	0	6.04	6.14	3.18	8.72	1.19

3G-Average	80	100	23.12	20.24	13.33	48.27	8.58
3G-Average	160	0	4.72	4.71	2.77	6.15	1.30
3G-Average	160	100	13.40	13.01	11.48	16.71	1.75
3G-Good	10	0	3.92	3.99	2.36	7.62	0.53
3G-Good	10	100	13.93	12.17	4.43	21.23	3.82
3G-Good	20	0	4.46	4.37	2.51	9.99	1.39
3G-Good	20	100	13.41	14.02	9.54	20.05	2.22
3G-Good	40	0	7.09	7.04	2.83	17.56	2.13
3G-Good	40	100	12.01	11.78	7.94	17.22	2.07
3G-Good	80	0	4.37	4.39	3.28	6.18	0.98
3G-Good	80	100	19.21	19.15	13.88	29.41	3.19
3G-Good	160	0	3.46	3.44	2.39	4.62	0.65
3G-Good	160	100	11.37	10.53	8.41	17.85	2.57
Edge-Average	10	0	3.94	3.96	2.82	5.17	0.39
Edge-Average	10	100	18.31	15.59	12.61	25.32	4.51
Edge-Average	20	0	4.74	4.71	2.56	10.08	1.09
Edge-Average	20	100	13.65	14.02	8.83	17.14	1.90
Edge-Average	40	0	8.37	8.05	4.61	19.83	2.32
Edge-Average	40	100	12.40	12.90	7.86	17.51	2.58
Edge-Average	80	0	6.80	6.73	5.03	8.94	1.04
Edge-Average	80	100	14.70	14.81	8.67	18.03	2.46
Edge-Average	160	0	14.30	9.23	7.08	31.71	10.53
Edge-Average	160	100	12.89	13.50	7.74	15.21	2.24
Edge-Good	10	0	3.86	3.92	1.16	5.31	0.49
Edge-Good	10	100	16.51	12.80	9.75	24.01	5.29
Edge-Good	20	0	4.64	4.70	2.25	8.14	1.34
Edge-Good	20	100	13.68	13.33	7.44	27.92	3.78
Edge-Good	40	0	7.38	7.64	3.13	11.74	1.51
Edge-Good	40	100	11.31	11.80	4.63	15.29	2.69
Edge-Good	80	0	6.56	6.39	3.64	13.69	2.11
Edge-Good	80	100	14.53	14.73	11.35	17.66	2.03
Edge-Good	160	0	8.10	8.22	6.59	9.45	1.35
Edge-Good	160	100	10.74	10.65	7.48	14.12	2.22
Edge-Lossy	10	0	3.90	3.90	2.50	5.80	0.52
Edge-Lossy	10	100	15.92	11.98	7.29	24.32	5.60
Edge-Lossy	20	0	4.35	4.13	2.53	10.82	1.24
Edge-Lossy	20	100	12.43	12.74	7.52	16.55	2.11
Edge-Lossy	40	0	7.36	7.56	2.84	16.78	2.29
Edge-Lossy	40	100	9.87	9.47	6.78	15.77	2.10
Edge-Lossy	80	0	5.15	5.27	2.79	9.15	1.36
Edge-Lossy	80	100	12.99	12.60	8.98	18.78	2.36
Edge-Lossy	160	0	9.71	9.63	8.89	10.93	0.81
Edge-Lossy	160	100	11.81	11.47	8.04	14.84	2.43
Phy-wifi-baseline	10	0	3.68	3.67	1.97	9.77	0.70
Phy-wifi-baseline	10	100	14.17	11.60	7.38	21.16	4.65

Phy-wifi-baseline	20	0	3.51	3.36	1.97	7.39	1.05
Phy-wifi-baseline	20	100	16.66	16.86	6.83	21.60	2.54
Phy-wifi-baseline	40	0	6.32	6.24	3.40	11.04	1.31
Phy-wifi-baseline	40	100	7.59	7.50	5.05	10.94	1.62
Phy-wifi-baseline	80	0	4.26	4.13	2.43	8.96	1.52
Phy-wifi-baseline	80	100	11.23	10.93	6.84	15.49	2.19
Phy-wifi-baseline	160	0	8.57	6.77	5.77	21.19	5.16
Phy-wifi-baseline	160	100	9.13	8.70	6.28	13.45	2.29

Table F.5: Continuous mode descriptive statistics for T1.telemetrySubmission for each watch

Network Profile	Hit %	TizenId	mean	median	min	max	SD
3G-Average	0	3B17oVj1PoUotIFVsk2NOlhkRQA=	386.47	200.00	134.00	3399.00	544.86
3G-Average	0	aH9IWfxWCc83JCgiEzmvghvgCAA=	175.79	171.00	132.00	279.00	28.37
3G-Average	0	aWv9GGc+YyDigM6502IhdBAW+QA=	426.59	212.00	131.00	5454.00	606.40
3G-Average	0	beOE8qzvUuqIA+I8R8wyWWNzkwA=	180.50	174.00	129.00	311.00	29.82
3G-Average	0	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	291.27	190.00	126.00	2009.00	317.38
3G-Average	0	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	602.43	307.00	139.00	4199.00	710.42
3G-Average	0	vLkCGpAM+92IICq8Kdp9JnbGAwA=	193.35	173.00	125.00	1290.00	121.14
3G-Average	0	YEP108KjoGuriIsePpOAsLKNEgA=	205.16	181.00	131.00	1385.00	139.23
3G-Average	100	3B17oVj1PoUotIFVsk2NOlhkRQA=	449.63	211.50	166.00	7118.00	901.86
3G-Average	100	aH9IWfxWCc83JCgiEzmvghvgCAA=	226.41	180.00	149.00	1809.00	225.91
3G-Average	100	aWv9GGc+YyDigM6502IhdBAW+QA=	405.96	242.00	166.00	3462.00	491.76
3G-Average	100	beOE8qzvUuqIA+I8R8wyWWNzkwA=	212.96	192.00	152.00	1567.00	165.45
3G-Average	100	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	618.70	251.50	168.00	7190.00	976.71
3G-Average	100	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	272.38	194.00	145.00	2191.00	301.23
3G-Average	100	vLkCGpAM+92IICq8Kdp9JnbGAwA=	527.14	225.00	141.00	6272.00	900.32
3G-Average	100	YEP108KjoGuriIsePpOAsLKNEgA=	437.83	219.00	166.00	2677.00	530.76
Phy-wifi-baseline	0	3B17oVj1PoUotIFVsk2NOlhkRQA=	168.04	81.00	29.00	1854.00	290.47
Phy-wifi-baseline	0	aH9IWfxWCc83JCgiEzmvghvgCAA=	80.78	60.00	25.00	1891.00	154.59
Phy-wifi-baseline	0	aWv9GGc+YyDigM6502IhdBAW+QA=	268.33	99.00	44.00	2631.00	447.16
Phy-wifi-baseline	0	beOE8qzvUuqIA+I8R8wyWWNzkwA=	75.06	66.50	31.00	728.00	59.23
Phy-wifi-baseline	0	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	276.95	102.00	41.00	1990.00	411.98
Phy-wifi-baseline	0	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	189.79	88.00	36.00	1977.00	291.12

Phy-wifi-baseline	0	vLkCGpAM+92lICq8Kdp9JnbGAwA=	72.85	64.50	28.00	210.00	28.28
Phy-wifi-baseline	0	YEP108KjoGuri1sePpOAsLKNEgA=	190.31	87.50	37.00	2003.00	335.01
Phy-wifi-baseline	100	3B17oVj1PoUotIFVsk2NOlhkRQA=	231.39	98.00	45.00	3389.00	458.89
Phy-wifi-baseline	100	aH9IWfxWCc83JCgiEzmvGhvgCAA=	263.80	82.00	45.00	7296.00	919.83
Phy-wifi-baseline	100	aWv9GGc+YyDigM6502IhdBAW+QA=	128.97	88.00	37.00	636.00	122.02
Phy-wifi-baseline	100	beOE8qzvUuqIA+H8R8wyWWNzkwA=	428.53	95.00	46.00	3731.00	794.11
Phy-wifi-baseline	100	O3yHeUUywVHE0C2V6ZqTUZtFQwA=	543.66	133.50	49.00	6763.00	1156.59
Phy-wifi-baseline	100	VGJzo3s9EcAQ5MgpYoXFRN0jFAA=	242.47	93.00	52.00	6960.00	823.45
Phy-wifi-baseline	100	vLkCGpAM+92lICq8Kdp9JnbGAwA=	420.00	113.50	55.00	5306.00	790.71
Phy-wifi-baseline	100	YEP108KjoGuri1sePpOAsLKNEgA=	366.91	112.50	41.00	4051.00	705.64

Appendix G: Additional graphs for sever load

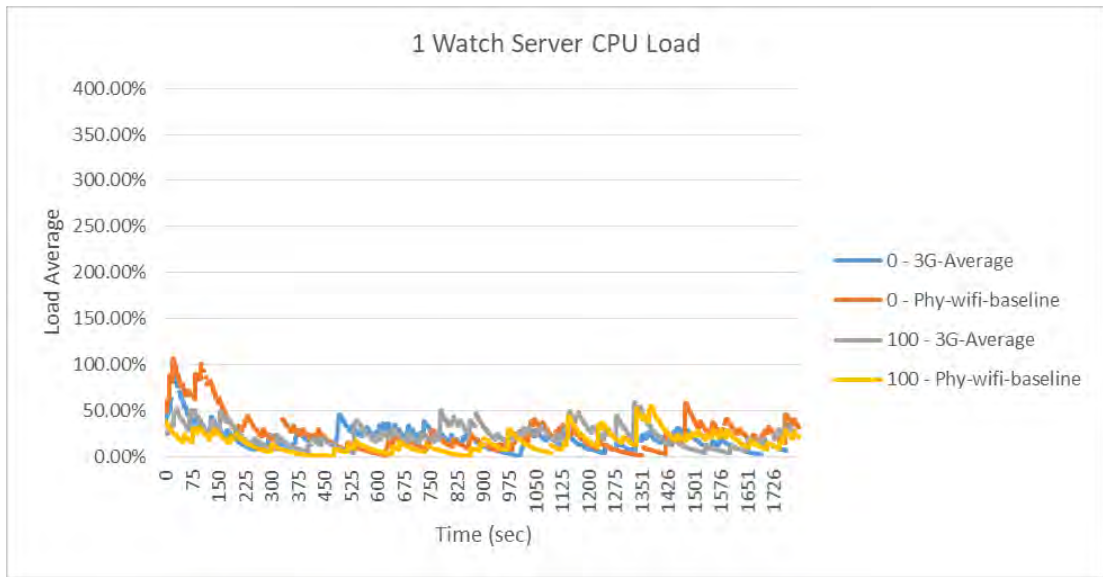


Figure G.1: Server Load for each network profile with 0% and 100% hit scenario and 1 watch

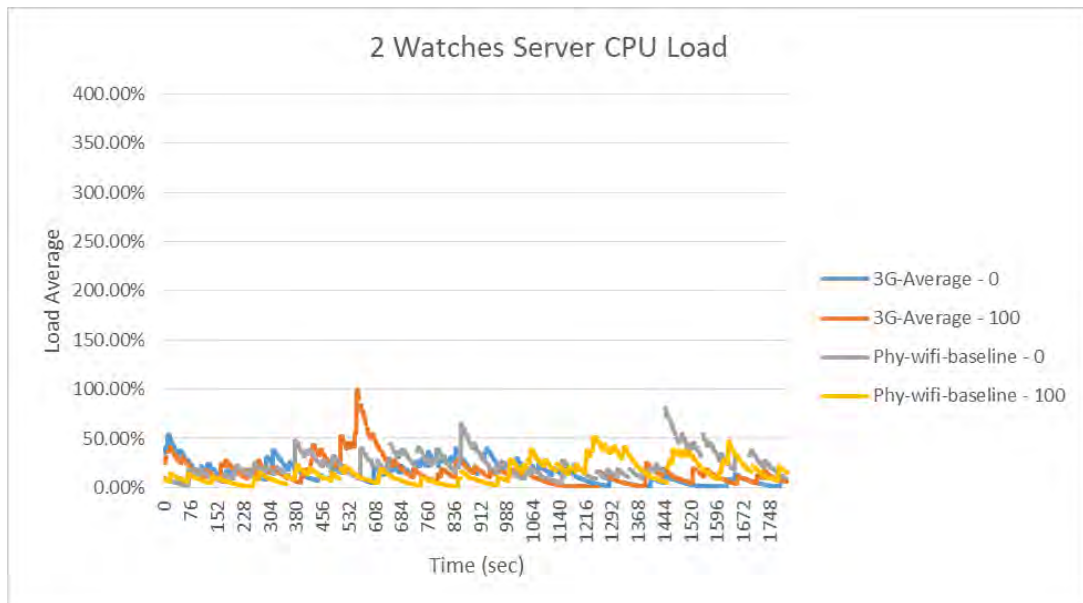


Figure G.2: Server Load for each network profile with 0% and 100% hit scenario and 2 watches

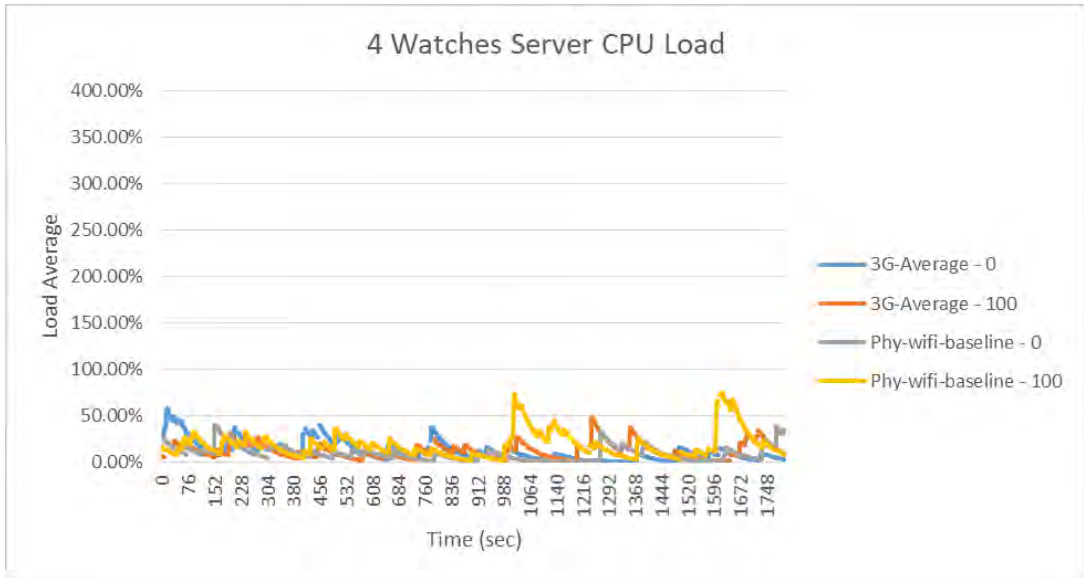


Figure G.3: Server Load for each network profile with 0% and 100% hit scenario and 4 watches

Vita

Mohamad Naeem Al Solh was born in 1987, in Dubai, United Arab Emirates. He received his primary and secondary education in Dubai, UAE. He received his B.Sc. degree in Computer Engineering from Ajman University of Science and Technology in 2012. He currently works at Etisalat Telecommunications Corporation as a solution architect.

In September 2012, he joined the master's program in Computer Engineering at the American University of Sharjah. During his master's studies, he co-authored "An MQTT-Based Context-Aware Wearable Assessment Platform for Smart Watches", which was presented at the 17th IEEE International Conference on Advanced Learning Technologies (ICALT 2017) and "An Assessment Wiki For Internet Of Things", which was presented at the 6th International Conference on Education and New Learning Technologies, 2014. His research interests are IoT, continuous integration, telecommunications, ubiquitous learning and ubiquitous computing.