

**QOS AWARE REAL TIME PLATFORM FOR MASSIVELY
MULTIPLAYER ONLINE GAMING (MMOG) OVER RTPS**

BY

Bashar Mohammad Khatib

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

December 2016

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN- 31261, SAUDI ARABIA
DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Bashar Mohammad Khatib** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.



Dr. Ahmad Almulhem
Department Chairman



Dr. Salam A. Zummo
Dean of Graduate Studies

10/1/17

Date



Dr. Basem Almadani
(Advisor)



Dr. Tarek Sheltami
(Member)



Dr. Marwan Abu-Amara
(Member)

© Bashar Mohammad Khatib
2016

To my lovely family; parents, brother, and sisters.

ACKNOWLEDGMENTS

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

My deepest gratitude is to my advisor, Dr. Basem Almadani. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time the guidance to recover when my steps faltered. Almadani taught me how to question thoughts and express ideas. His patience and support helped me overcome many crisis situations and finish this dissertation. I hope that one day I would become as good an advisor to my students as Almadani has been to me.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Tarek Sheltami and Dr. Marwan Abu-Amara, for their encouragement and insightful comments. I would also like to thank King Fahd University of Petroleum and Minerals for the support extended towards my research and for granting me the opportunity to pursue graduate studies, special thanks are given to the Department of Computer Engineering. Thanks to the Real-Time Innovation Inc. (RTI) for its support with all tools that I have used in this thesis work. Heartfelt thanks to my family, Father, Mohammad khatib, my mother, Hadya Khatib, brother Feras khatib, and sisters, Shorouq, Shayma, and Shatha Khatib for their support and encouragement throughout my study.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
LIST OF ABBREVIATIONS	xvii
ABSTRACT	xix
ملخص الرسالة	xxi
CHAPTER 1: INTRODUCTION.....	1
1.1 History of MMOG.....	1
1.2 Technical requirement.....	2
1.2.1 Scalability.....	2
1.2.2 Consistency	3
1.2.3 Reliability.....	3
1.2.4 Fairness	3

1.3	Communication models	4
1.3.1	Client/Server model	4
1.3.2	Synchronous/Asynchronous model	5
1.3.3	Fan Out/In Model	6
1.3.4	P2P/M2N Models	8
1.4	Middleware software	8
1.4.1	Middleware usages	9
1.4.2	Middleware types	11
1.5	Publish/Subscribe Middleware	15
1.6	Distributed and Real-Time systems	17
1.6.1	Real-Time Publish/Subscribe Middleware (RTPS)	17
1.6.2	Publication	24
1.6.3	Subscription	25
1.7	MMOG Classifications	26
CHAPTER 2: LITERATURE REVIEW		28
CHAPTER 3: WORK OBJECTIVES		39

CHAPTER 4: DDS MIDDLEWARE ARCHITECTURE	42
4.1 Publish-Subscribe model	44
4.2 QoS Architecture for MMOG.....	45
4.2.1 Durability.....	46
4.2.2 Presentation.....	46
4.2.3 Deadline	47
4.2.4 Latency-Budget	48
4.2.5 Reliability.....	49
4.2.6 Transport-Priority	50
4.2.7 History.....	51
4.2.8 Resource-Limits	52
CHAPTER 5: PROBLEM STATEMENT AND PROPOSED SOLUTION	53
5.1 Durability.....	54
5.2 Presentation.....	55
5.3 Deadline	55
5.4 Latency-Budget	55

5.5	Reliability	56
5.6	Transport-Priority	56
5.7	History	56
5.8	Resource-Limits	56
5.9	Proposed algorithm	57
5.10	Proposed algorithm for RTinDDS	59
5.10.1	RTinDDS Proposed Solution	60
5.10.2	Algorithm Proposed Solution	63
CHAPTER 6: EXPERIMENTAL SETTINGS/IMPLEMENTATION AND RESULTS		64
6.1	Experiment setup and Performance Metrics	65
6.2	Experiment setup for AIRTinDDS	72
6.3	Experiment scenarios:	75
6.3.1	Scenario 1:	75
6.3.2	Scenario 2:	77
6.3.3	Scenario 3:	79
6.3.4	Scenario 4:	81

6.4	Experiment QoS Profiles	84
6.4.1	Profile 1:	84
6.4.2	Profile 2:	87
6.4.3	Profile 3:	90
6.4.4	Profile 4:	92
6.4.5	Profile 5	96
6.4.6	Profiles discussion:	98
6.5	Comparison between RTinDDS and other Platforms	100
6.5.1	Conclusion	102
CHAPTER 7: CONCLUSIONS AND FUTURE WORK		103
7.1	Conclusions	103
7.2	Future work	105
References		106
Vitae		109

LIST OF TABLES

Table 6-1: Tools and Programs.....	65
Table 6-2: Configuration table	69
Table 6-3: Topics statistics.....	69
Table 6-4: Network statistics.....	70
Table 6-5: Configuration Scenario 1.....	75
Table 6-6: latency scenario 1.....	76
Table 6-7: latency scenario 2.....	78
Table 6-8: latency scenario 3.....	80
Table 6-9: latency scenario 4.....	82
Table 6-10: configuration table.....	84
Table 6-11: profile 1 - QoS configuration.....	84
Table 6-12: profile 2 - QoS configuration.....	87
Table 6-13: profile 3 - QoS configuration.....	90
Table 6-14: profile 4 - QoS configuration.....	93
Table 6-15: profile 5 - QoS configuration.....	97

LIST OF FIGURES

Figure 1-1: Architecture view of client-server	5
Figure 1-2: synchronous communication model[8].....	5
Figure 1-3: asynchronous communication model[8].....	6
Figure 1-4: Fan-out model	7
Figure 1-5: Fan-in model.....	7
Figure 1-6: Middleware Architecture.....	9
Figure 1-7: PS vs. Polling.....	16
Figure 1-8: RTPS Structure.....	19
Figure 1-9: RTSP Behavior[9].....	21
Figure 2-1: Peer-to-Peer Architecture [18].	34
Figure 3-1: PMS over DDS middleware, two players are playing.	40
Figure 4-1: DDS Middleware	42
Figure 4-2: Architectural view of Publisher-Subscriber.....	45
Figure 5-1: DDS infrastructure.	57
Figure 5-2: The hierarchy of the designed model.....	58

Figure 5-3: RTinDDS Algorithm flowchart	59
Figure 5-4: AI Algorithm (AIRTinDDS) flowchart	62
Figure 6-1: PerfTest echo requests/replies	66
Figure 6-2: PMS Throughput	66
Figure 6-3: Data load Vs. Throughput	67
Figure 6-4: Data load Vs. Avg. packet/sec	68
Figure 6-5: PMS Latency	68
Figure 6-6: latency per each topic	71
Figure 6-7: latency per each topic	72
Figure 6-8: unstable RTinDDS (a) Vs AIRTinDDS (b) throughput	73
Figure 6-9: AIRTinDDS vs (unstable) RTinDDS	74
Figure 6-10: RTinDDS vs. AIRTinDDS latency	74
Figure 6-11: Throughput (scenario 1)	76
Figure 6-12: Latency (scenario 1)	76
Figure 6-13: Throughput (scenario 2)	78
Figure 6-14: Latency (scenario 2)	78

Figure 6-15: Throughput (scenario3)	80
Figure 6-16: Latency (scenario 3)	80
Figure 6-17: Throughput (scenario 4)	82
Figure 6-18: Latency (scenario 4)	82
Figure 6-19: Throghput (All scenarios)	83
Figure 6-20: Latency (All scenarios)	83
Figure 6-21: Latency - Profile 1	86
Figure 6-22: profile 1 – latency per message	87
Figure 6-23: Latency (Profile 2)	89
Figure 6-24: profile 2 – latency per message	90
Figure 6-25: profile 3 – latency	92
Figure 6-26: profile 4 – latency	96
Figure 6-27: Latency - profile 5	97
Figure 6-28: Profiles latency	98
Figure 6-29: mean, min, and max/profiles	99

LIST OF ABBREVIATIONS

MMOG	:	Massively Multiplayer Online Games
DDS	:	Data Distribution Service
ACK	:	acknowledgment
NACK	:	negative acknowledgment
P2P	:	Point-To-Point (P2P)
TPMs	:	Transaction Processing Monitors
SQL	:	Structured Query Language
DCE	:	Distributed Computing Environment
CORBA	:	Common Object Request Broker Architecture
DCOM/COM	:	Distributed/Component Object Model
IDL	:	interface definition language
TP	:	Transaction processing
OLTP	:	on-line transaction processing
RPCs	:	Remote Procedure Calls
ORBs	:	Object Request Brokers
OMG	:	Object Management Group
MDA	:	Model Driven Architecture
UML	:	Unified Modeling Language
MOF	:	Meta-Object Facility
CWM	:	Common Warehouse Meta-model
RMI	:	Remote Method Invocation

PS	:	Publish-Subscribe
RTPS	:	Real-Time Publish/Subscribe Middleware
MMORPG	:	Massively Multiplayer Online Role-playing Game
MMOFPS	:	Massively Multiplayer First-person Shooter
MMORTSG	:	Massively Multiplayer Online Real-time Strategy Game
MMOTBSG	:	Massively Multiplayer Online Turn-based Strategy Game
MMOSG	:	Massively Multiplayer Online Sports Game
MMORG	:	Massively Multiplayer Online Racing Game
MMORG	:	Massively Multiplayer Online Rhythm Game
MMOMG	:	Massively Multiplayer Online Management Game
MMOSG	:	Massively Multiplayer Online Social Game
MMOBBG	:	Massively Multiplayer Online Bulletin Board Game
EE	:	Enterprise Edition
ANFIS	:	adaptive neural fuzzy
MLP	:	multi-layer Perceptron network
FTLFN	:	focused time lagged network (
FPS	:	first person shoot
RT	:	real time
PMS	:	Plane Model Simulation
AI	:	artificial intelligent
NS	:	Notification service

ABSTRACT

Full Name : [Bashar Mohammad Mousa Khatib]

Thesis Title : [QoS Aware Real-Time Platform for Massively Multiplayer Online
Gaming (MMOG) over RTPS]

Major Field : [Computer Engineering]

Date of Degree : [May 2016]

Recently, Massively Multiplayer Online Game (MMOG) has become very popular; there are thousands of games built to be played online, where hundreds or thousands of users from all around the world can play the game at the same time. The hottest issue in the MMOG is the real-time (RT) service, where the players can share their game status in a real-time manner. The main need for the RT platform is to guarantee the quality of services, the platform should deliver a robust and excellent quality, besides, the service should be loosely couple oriented to support the growth of MMOG applications.

We present a new platform solution to deal with the real-time online gaming using a DDS middleware. An RTinDDS is a real-time platform built based on the DDS middleware. This middleware can guarantee to provide the developers, designers, and end users with a suitable platform to build their online real-time gaming and to deliver it in a robust manner to the end users. The platform also allows designers to make the best decision for a specific situation.

The results of the experimental work show that RTinDDS improved the reliability, scalability, throughput, and latency. Chapter 6.

We improved RTinDDS by implementing a new smart algorithm based on the artificial neural network AIRTinDDS. This algorithm used to detect and resolve any issue will occur during the game running to guarantee to deliver a robust platform, the system able to adapt itself to any unexpected behavior. This will effect on many factors such as, sharing the resources fairly among the players, describing the issue to the players and inform them what to do and solve the issue without any break in the game.

We have applied our algorithms RTinDDS/AIRTinDDS to build a game (PMS) over DDS middleware to measure and test the power of RTinDDS/AIRTinDDS, we tested many scenarios and profile to make sure that we covered most cases and circumstances. The results show very good throughput with a very reasonable latency, with the ability to scale the number of players.

ملخص الرسالة

الاسم الكامل: بشار محمد موسى خطيب

عنوان الرسالة: بناء منصة قوية لألعاب الفيديو ذات الاعداد الهائلة من اللاعبين عبر الانترنت باستخدام أدوات تحسين الجودة المدعومة من (دي دي اس الوسيطة) التي تدعم تقنية الوقت الحقيقي.

التخصص: هندسة الحاسب الآلي

تاريخ الدرجة العلمية: December 2016

مما لا شك فيه أنه في الأونة الأخيرة، كثر عدد الألعاب التي تدعم اعداداً هائلة من المستخدمين عبر الانترنت. هناك الآلاف من الألعاب التي بنيت لتدعم اعداداً كبيرة من المستخدمين ليلعبوا ويشاركوا حالاتهم عبر شبكة الانترنت، حيث المنات أو الآلاف من المستخدمين من جميع أنحاء العالم يمكن أن يلعبوا نفس اللعبة في نفس الوقت. أهم قضية في هذه الأنواع من الالعاب هي الخدمة في الوقت الحقيقي (RT) ، حيث يمكن للمستخدمين مشاركة حالتهم بنفس الوقت تماماً. الحاجة الرئيسية لمنصة RT هي لضمان جودة الخدمات، حيث يجب على هذه المنصة ان تقدم نوعية قوية وممتازة، لضمان نمو اعداد اللاعبين بنفس الوقت دون حدوث أي خلل يؤثر على جودة وأداء اللعبة .

في هذه الرسالة، قمنا بالعمل لتقديم منصة قوية قادرة على توفير جميع الاحتياجات اللازمة للتعامل مع هذه النوعية الألعاب (الوقت الحقيقي عبر الإنترنت) باستخدام الوسيطة DDS , RTinDDS هي منصة تعمل في الوقت الحقيقي بنيت على أساس الوسيطة DDS ، وهذا يمكن الوسيطة ضمان تزويد المطورين والمصممين واللاعبين، أرضية مناسبة لبناء هذه النوعية من الالعاب وتقديمها بطريقة قوية للمستخدمين النهائيين . هي منصة قادرة أيضا على السماح للمصممين القدرة على اختيار انساب القرارات اثناء بناء اللعبة لتحقيق أفضل النتائج .

في هذه الرسالة، قمنا بتجارب عملية عن طريق بناء لعبة مدعومة بالوسيطة DDS، حيث أظهرت النتائج أن RTinDDS عملت على تحسين ال . Throughput, latency

عادة ما يستهلك اللاعبون اثناء لعبهم موارد الشبكة (bandwidth, CPU, and memory) اثناء اللعب في نفس الوقت مع

عدد كبير من اللاعبين من كل أنحاء العالم. لهذا، قمنا بتحسين RTinDDS من خلال تنفيذ خوارزمية ذكية جديدة تعمل باستخدام تقنية الذكاء الاصطناعي، لكشف وحل ما قد يحدث من مشاكل غير متوقعة أثناء اللعب.

لقد قمنا بتطبيق هذه الخوارزميات لبناء لعبة (PMS) على DDS الوسيطة لقياس واختبار قوة RTinDDS / AIRTinDDS ، وضعنا العديد من السيناريوهات للتأكد من شمول وتغطية معظم الحالات والظروف، فقد بينت النتائج ان هذه الخوارزميات قادرة على إعطاء Throughput عالي بالتناسب مع اعداد اللاعبين، وفي نفس الوقت Latency منخفض .

CHAPTER 1

INTRODUCTION

Massively Multiplayer Online Games (MMOG) defined as a virtual world where a massive number of players can set together virtually and interact with each other in a real-time manner. One of the most popular challenges in this area is the huge amount of the real-time exchange messages over a huge number of users[1]. Many technologies in the literature tried to deal with this kind of challenges by applying different algorithms and different middlewares, no one used the Data Distribution Service (DDS) middleware to build MMOG, which is one of the lead middleware in delivering real-time messages with a very powerful quality of services.[1].

1.1 History of MMOG

The first game in the MMOG world is called Ultima Online7, it was released in 1997. The main objective of this game was the commercials; it showed a very successful achievement in the commercial world.[2].

In 1998, a new game called Lineage was released in Korea. It was popular all over Korea. In 1999 EverQuest was released; this game was overall expectation. It had unexpected subscribers. EverQuest was considered a break point in MMOG world. Many new MMOGs have been released after this game. The next successful game in that period was called Asheron's Call; it was popular wide world.[2].

Many games have been released up until now were considered a role-playing game (MMORPGs). We have different kinds of MMOG categories, such as shooter players, first person shoot game, race players, and many others. There were many issues and difficulties regarding many factors, one of the most important factor was the real-time issue. Many research and works are done in order to overcome all these issues.

In a typical MMOG, players collaborate or compete in a virtual world. Each player sees a graphical representation of the world and controls an avatar that performs various actions, such as moving, picking up objects, or communicating with other players. To provide a shared sense of space among players, each player must maintain a copy of the (relevant) game state on his computer. When one player performs an action that affects the world, the game state of all other players affected by that action must be updated. This can be done by sending either the action or its effects over the network.

In DDS middleware, by using the suitable QoS policies in terms of all MMOG circumstances, it can provide a Scalable, Consistent, Reliable, and fairness systems.

1.2 Technical requirement

1.2.1 Scalability

The biggest challenge in MMOGs is the scalability. Ideally, thousands of players all over the globe could play together in the same virtual world, regardless of varying network connection quality. Bad network conditions can result in high latency, meaning that it takes more time for a message to reach its destination.[3][4] and [5]. Moreover, the bandwidth that is, the maximum throughput of data—is also limited and varies depending on

connection quality and the number of joining players, our platform should be able to handle the growing number of players where this affect on the latency, the bandwidth, and the quality of service.

1.2.2 Consistency

MMOGs are complex distributed systems. Each player interacts with the game in real time, so the player's machine must know about the state of the game in near real time at least, that part of the game state most relevant to the player.[6]. Owing to the network latency problem, this game state is, unfortunately, never 100 percent up to date because of other players constantly and concurrently modify the game world. The challenge is to nevertheless provide a consistent view of the virtual world to players or to provide a means to tolerate inconsistencies so that they do not negatively affect gameplay.[7].

1.2.3 Reliability

The probability of a single node's failure in a distributed system is low, but it grows along with the number of nodes. It is almost certain that in MMOG with thousands of participating nodes, failures will occur fairly often. In addition to that, network connections can also fail. MMOG must, therefore, be able to cope with node and network failures with minimal disturbance to gameplay.[7].

1.2.4 Fairness

The basis for an enjoyable game experience is that all players are treated equally despite the game implementation's distributed nature. This requires coping both with the imbalance of player machines' resources in terms of CPU power and network bandwidth and with

cheating players who exploit weaknesses in the game design and implementation to their advantage.

1.3 Communication models

There are different kinds of communication architecture where MMOG can be built over.

1.3.1 Client/Server model

C/S model is a network architecture where each node on the network is either a client or a server. Servers usually have chosen to be powerful computers or processes abled to manage the huge number of client requests and updates. [8]. Figure **1-1**.

- Thin-Client model: Applications and ‘heavy work’ resides on server(s). Client(s) serve as an interface for the server(s).
- Fat-Client model: Most of the programming logic resides on the client(s) side. Server(s) handles data.
- Applet model: intermediate model.

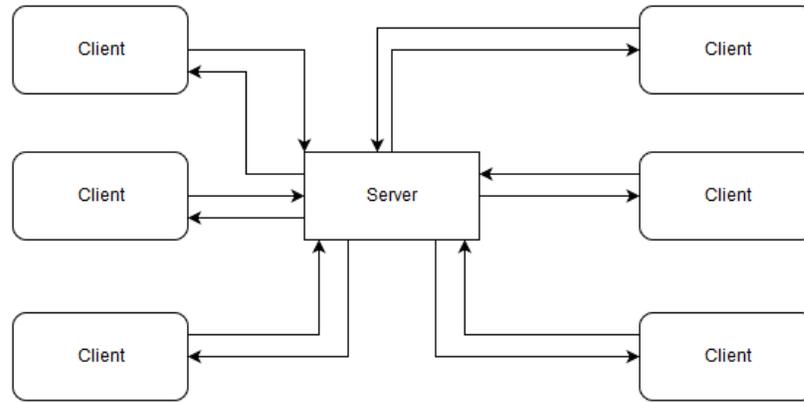


Figure 1-1: Architecture view of client-server

1.3.2 Synchronous/Asynchronous model

In synchronous communication, the source sends the data and waits for a reply by the destination. If the transmission is successful, the destination sends a positive acknowledgment (ACK) otherwise, the sender receives a negative acknowledgment (NACK). Synchronous communication is used in middleware documentations to point to reliable communication features.[8].

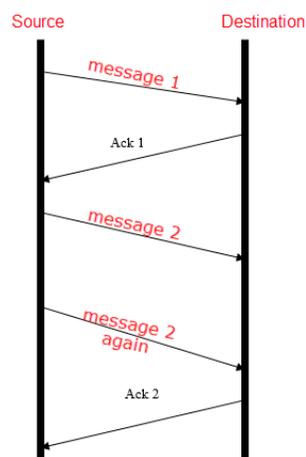


Figure 1-2: synchronous communication model[8]

On the other hand, in asynchronous communication, senders keep pushing data to destinations without waiting for replies. Non-Reliable communication is used to describe asynchronous middleware. Different techniques are used to assure reliable communication when the asynchronous model is used. One of the simplest tricks used is to number the messages and on the destination side the sequence of the message is checked and missed messages are requested again.

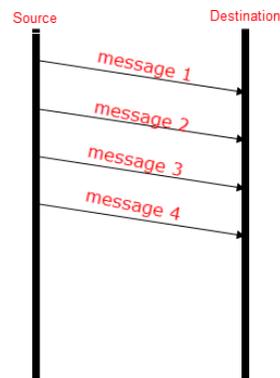


Figure 1-3: asynchronous communication model[8]

1.3.3 Fan Out/In Model

The main feature of the fan out model is the limited number of data sources and a huge number of destinations. This model is more suitable for data distributions, stocks feeds, and news services applications.

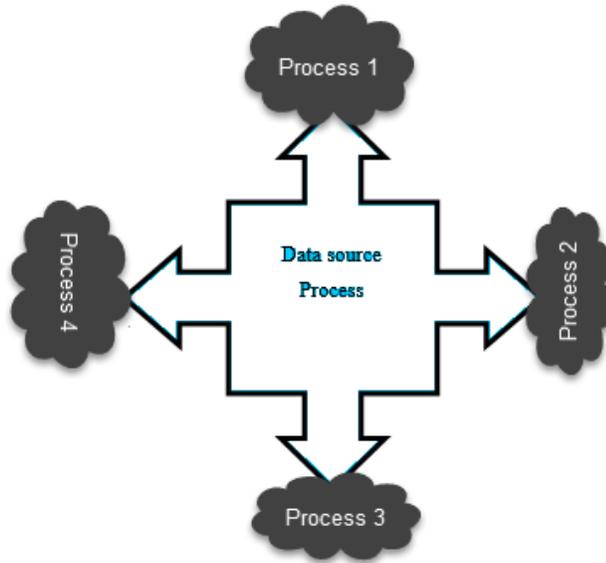


Figure 1-4: Fan-out model

On the other hand, the fan-in model is found in applications such as network management, data acquisition, fraud detection, and data collection. The common feature of these applications is the huge number of data sources (publishers) and a limited number of receivers (subscribers). For example, the publisher could be a network device signaling particular events and subscribers could be network administrator responsible for the systems. An industrial example for this model is the oil wells drilling column where sensors measure certain values and send them to a central server for analysis.[8]

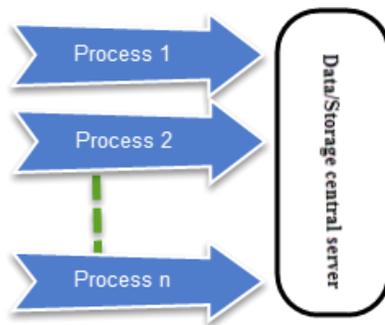


Figure 1-5: Fan-in model

1.3.4 P2P/M2N Models

Point-To-Point (P2P) model is usually seen in industrial environment applications where thousands of processes perform a certain task and many of them send (publish) data to limited number (mostly one) process.

Many to many is a combination of the fan-in and fan-out model. Many data sources send data to many destinations. Production planning and scheduling systems are one of the good examples for this model where production lines have to send production report for each material to several systems such as quality control, production monitoring and logistic management system.

1.4 Middleware software

Middleware is defined as a “Software that facilitates interoperability by mediating between an application program and a network, thus masking differences or incompatibilities in network transport protocols, hardware architecture, operating systems, database systems, remote procedure calls, etc.”[8]. In other words, it is a kind of delivering a service between different platforms, including both hardware and software, and the applications. The services shared common protocols and standards between these different platform taking into account the different types of software, hardware, and applications. Middleware used to be reusable, where developers and designers can update, edit, modify the services as it fits their applications.

The main role of middleware is to ease the task of designing, developing and managing distributed systems by providing simple and consistent integrated programming

environment. Middleware increases the interoperability, portability and flexibility of distributed systems.

The main usages of middleware in developing distributed systems are to be used by applications to locate transparently across the network by providing interaction with another application or server since it is independent of network services. In the other hand, the middleware is always reliable and available which can Scale up in capacity without losing function.

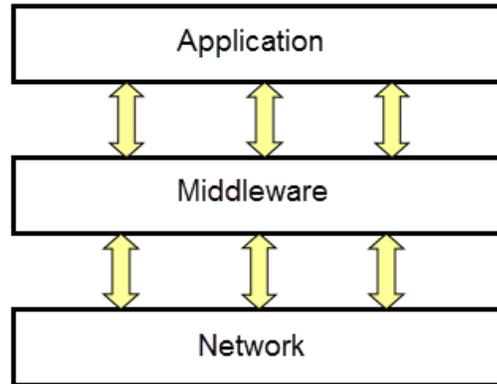


Figure 1-6: Middleware Architecture

1.4.1 Middleware usages

Middleware can be classified into three groups from a usage point view, distributed system services which include critical mission systems, peer-to-peer communication, and data management services. RPCs, MOMs, and ORBs are suitable for those usages. And the second one is the application enabling services which give applications access to distributed services and the underlying network. Transaction Processing Monitors (TPMs) and Structured Query Language (SQL) are the most used services in this category. The last

type is the middleware management services, which provide transparent network communication between heterogeneous systems.

The most commonly used middleware technologies are:

- Distributed Computing Environment (DCE) by Open Software Foundation
- Common Object Request Broker Architecture (CORBA) by Object Management Group
- Distributed/Component Object Model (DCOM/COM) by Microsoft

The different programming model is supported by different middleware platforms. One of the most popular models is object-oriented middleware in which applications are structured into objects that communicate via location transparent method invocation. The main examples of this type of middleware are the OMG's CORBA and Microsoft's Distributed COM., Both of these platforms offer an interface definition language (IDL) which is used to abstract over the fact that objects can be implemented in any suitable programming language. An object request broker which is responsible for transparently directing method invocations to the appropriate target object, and a set of services (e.g. naming, time, transactions, replication etc.) which further enhance the distributed programming environment.[8].

Not all middleware is object based. Two other popular paradigms are event-based middleware and message-oriented middleware, both of which mainly employ 'single shot' communications rather than the request-reply style communication found in object-based middleware.

Event based middleware is particularly suited to the construction of non-centralized distributed applications that must monitor and react to changes in their environment. Examples are process control, Internet news channels and stock tracking. It is claimed that event-based middleware has potentially better scaling properties for such applications than object-based middleware.

Message-oriented middleware, on the other hand, is biased toward applications in which messages need to be persistently stored and queued. Workflow and messaging applications are good examples.

1.4.2 Middleware types

1.4.2.1 Transaction processing

Transaction processing (TP) middleware are used to provide a complete environment for transaction application that accesses a relational database. In TP middleware, clients call remote procedures stored on the server, which contain a set of SQL statements (transaction). Procedures execute set of SQL statements, which all succeed or all fail as a unit.

Application based on TP used to be mission-critical applications with strong controls over database's security and integrity and usually are known as on-line transaction processing (OLTP).

TP is considered efficient from the communication overhead side because it requires only a single request/reply statement and the multiple SQL statement are kept on the server side. It is not suitable for program-to-program communications because it tends to be

heavyweight and expensive, and it requires deep experience to implement and tune. TP middleware cannot run properly without service contracts from the vendor in most cases, and the most important features provided by TP middleware are[8]:

- Application development tools: User interaction and database interfaces
- System administration: Security, tuning and users management
- Transaction execution: Scheduling and load balancing

1.4.2.2 Remote Procedure Calls (RPCs)

RPCs have been used for a long time, they are one of the earliest forms of inter-program communication mechanism. they are easy to understand although they operate in low level. The code calls a procedure that resides on a remote system and the results are returned. This is considered to be a synchronous communication while the programs call and wait for results.

RPCs are point-to-point communication rather than one data source to many destinations and this makes them work well for small and simple applications. Experts stated that RPCs do not scale well to large, mission-critical systems as they leave important to the programmer such as managing network and system failures, managing multiple connections, flow control and buffering, portability and synchronization between processes.

1.4.2.3 Object Request Brokers

Object Request Brokers (ORBs) are similar to RPCs but they are language independent and object-oriented. They are strictly object-oriented and point-to-Point communication.

The main technology providers for ORBs are the Object Management Group (OMG) and Object Web (Open Source), where Open CORBA Component Model Platform (OpenCCM) is defined. It consists of an open development tool set, an open deployment infrastructure and an open run-time container.

1.4.2.4 Object Management Group (OMG)

Where two technologies are provided, the Common Object Request Broker Architecture (CORBA) where the CORBA is vendor-independent architecture and infrastructure that can be used by almost any computer, operating systems, Programming language and network to communicate together. OMG-specified also CORBA Event Service as a standard service layered over CORBA. It is clear that CORBA requires strict object-oriented approach and it uses request-reply (Synchronous) communication.

In addition, the other technology is the Model Driven Architecture (MDA), it provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG's established modeling standards: Unified Modeling Language (UML); Meta-Object Facility (MOF); and Common Warehouse Meta-model (CWM). Platform-independent Application descriptions built using these modeling standards can be realized using any major open or proprietary platform, including CORBA, Java, NET, XMI/XML, and Web-based platforms.[8]

1.4.2.5 Sun Microsystems

Which has two main technologies, the Java Remote Method Invocation (RMI), which it enables programmers to develop Java-based and distributed applications. In RMI, methods of remote java objects can be invoked by another java virtual machine on same or different hosts. A reference to the remote object is required, which can be obtained from the bootstrap naming service provided by RMI or as an argument value.

And the other one is the JINI Network Technology which it is an open architecture, which helps developers to program network centric services that are highly adaptive to changes. Jini is suitable for the dynamic computing environment. Its main uses are scalable, evolvable and flexible systems.

1.4.2.6 Microsoft

Which has also two main technologies, the Distributed Component Object Model (DCOM) which it is a protocol enables network objects to communicate directly in reliable, secure and efficient manner. DCOM designed to be used across several network transport including Internet protocols. DCOM is based on the Open Software Foundation's DCE-RP spec. Component Object Model (COM) can be used with Java applets and ActiveX components.

The other technology is the .NET, it is a set of Microsoft software technologies, which enables a high level of programs integration by using XML web services.

1.5 Publish/Subscribe Middleware

Publish-Subscribe (PS) communication concept is very similar to printed periodicals publication and subscription mechanism. Applications communicate by sending and receiving issues of named publication. An issue consists of

- **Topic:** A string chosen by the user to identify the publication. The topic identifies the user data that will be sent to the distributed system without a need for network address or port number.
- **Type:** which defined the data format of an issue. It is used to provide automatic type conversion between computer architectures. Type is usually a structure's name.
- **Issue identification:** a number used internally by the middleware to uniquely identify each issue of a publication.
- **User data:** The actual content defined by the application for a certain issue, which changes in every issue.

Communication in PS has three phases, the declaration of intent to generate data by the publisher, declaration of interest by the subscriber and the transporting the issue from publisher to subscriber.

When a certain issue is generated, the middleware routes the issue to all subscribers over the net. These three steps can happen in any order. For example, a publisher sent a topic to all subscribers and then a new subscriber joined the network. All issues those are still valid (the persistence period specified by the publisher did not expire) will be sent to the new

1.6 Distributed and Real-Time systems

A distributed system is a collection of autonomous nodes, connected through a network and distribution middleware, which enables nodes to communicate and coordinate between each other and to share their resources, so we can see the system as one integrated system. There are many advantages and disadvantages for this kind of distributed systems, one of the most common issue is the complexity, since the system will be consisting of thousands and sometimes hundreds of thousands of nodes as in our case where we have a massive number of players. Also, since we have a huge number of nodes, the security will become an issue as the system become difficult to be managed and predicted.

On the other hand, distributed systems gave us the ability to scale our systems, where hundreds of thousands were able to join the system simultaneously. In addition to the scalability, the nodes can share the resources easily, this will affect on the resource limitation and can help us to manage and control the resources.

As we studied all other middlewares, we chose to use Real-Time Publish/Subscribe Middleware (RTPS), since it is also used in the DDS middleware.

1.6.1 Real-Time Publish/Subscribe Middleware (RTPS)

RTPS was specifically developed to support the unique requirements of data distributions systems. As one of the application domains targeted by DDS, the industrial automation community defined requirements for a standard publish- subscribe wire-protocol that closely match those of DDS. As a direct result, a close synergy exists between DDS and

the RTPS wire-protocol, both in terms of the underlying behavioral architecture and the features of RTPS.

The RTPS protocol is designed to be able to run over multicast and connectionless best-effort transports such as UDP/IP. The main features of the RTPS protocol include:[9].

- Performance and quality-of-service properties to enable best-effort and reliable publish-subscribe communications for real-time applications over standard IP networks.
- Fault tolerance to allow the creation of networks without single points of failure.
- Extensibility to allow the protocol to be extended and enhanced with new services without breaking backward compatibility and interoperability.
- Plug-and-play connectivity so that new applications and services are automatically discovered and applications can join and leave the network at any time without the need for reconfiguration.
- Configurability to allow balancing the requirements for reliability and timeliness for each data delivery.
- Modularity to allow simple devices to implement a subset of the protocol and still participate in the network.
- Scalability to enable systems to potentially scale to very large networks.
- Type-safety to prevent application programming errors from compromising the operation of remote nodes.

The above features make RTPS an excellent match for a DDS wire-protocol. Given its publish-subscribe roots, this is not a coincidence, as RTPS was specifically designed for meeting the types of requirements set forth by the DDS application domain.

All RTPS entities are associated with an RTPS domain, which represents a separate communication plane that contains a set of Participants. A Participant contains local Endpoints. There are two kinds of endpoints: Readers and Writers. Readers and Writers are the actors that communicate information by sending RTPS messages. Writers inform of the presence and send locally available data on the Domain to the Readers which can request and acknowledge the data. Figure 1-8.

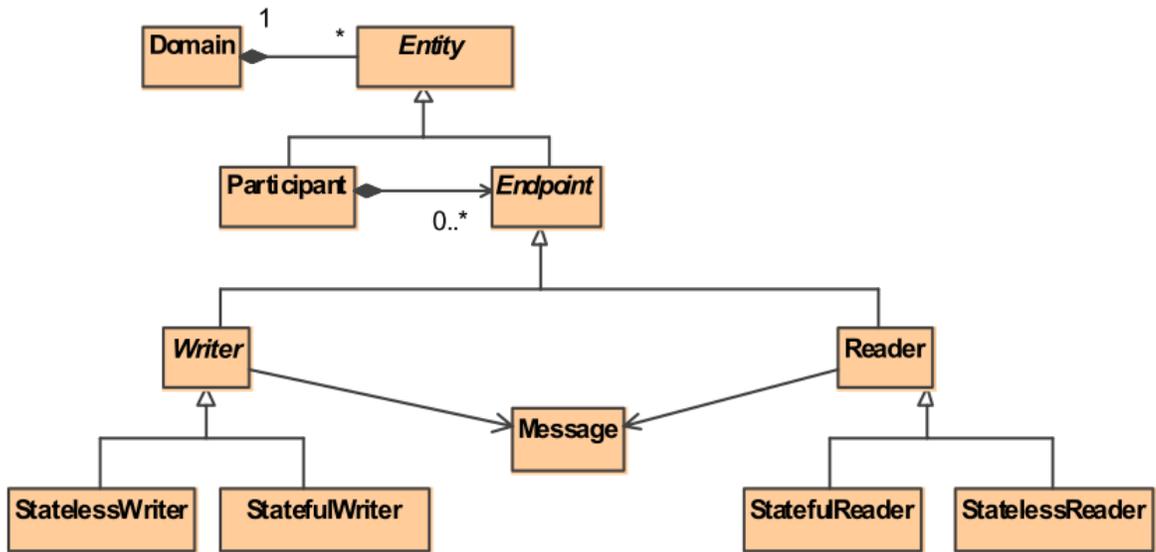


Figure 1-8: RTPS Structure

RTPS adds few features to PS to overcome conflicting goals of timing, memory, and reliability. Those features are:

- Timestamps and numbers every issue

- Allows the application to trade-off deterministic timing with reliable delivery.
- Adds bi-directional request-reply communication over PS
- Controls memory usage
- Works well in RTOS environment
- The main communication objects in RTPS are publications, subscription, clients and servers. These objects together support unidirectional (PS send-receive) and bi-directional (request-reply) communications.

In unidirectional communication, information flows in one way from publishers to subscribers while publication and subscriptions specify how the issues are sent and received.[8].

A typical sequence illustrating the exchanges between an RTPS Writer and a matched RTPS Reader is shown in Figure **1-9**.

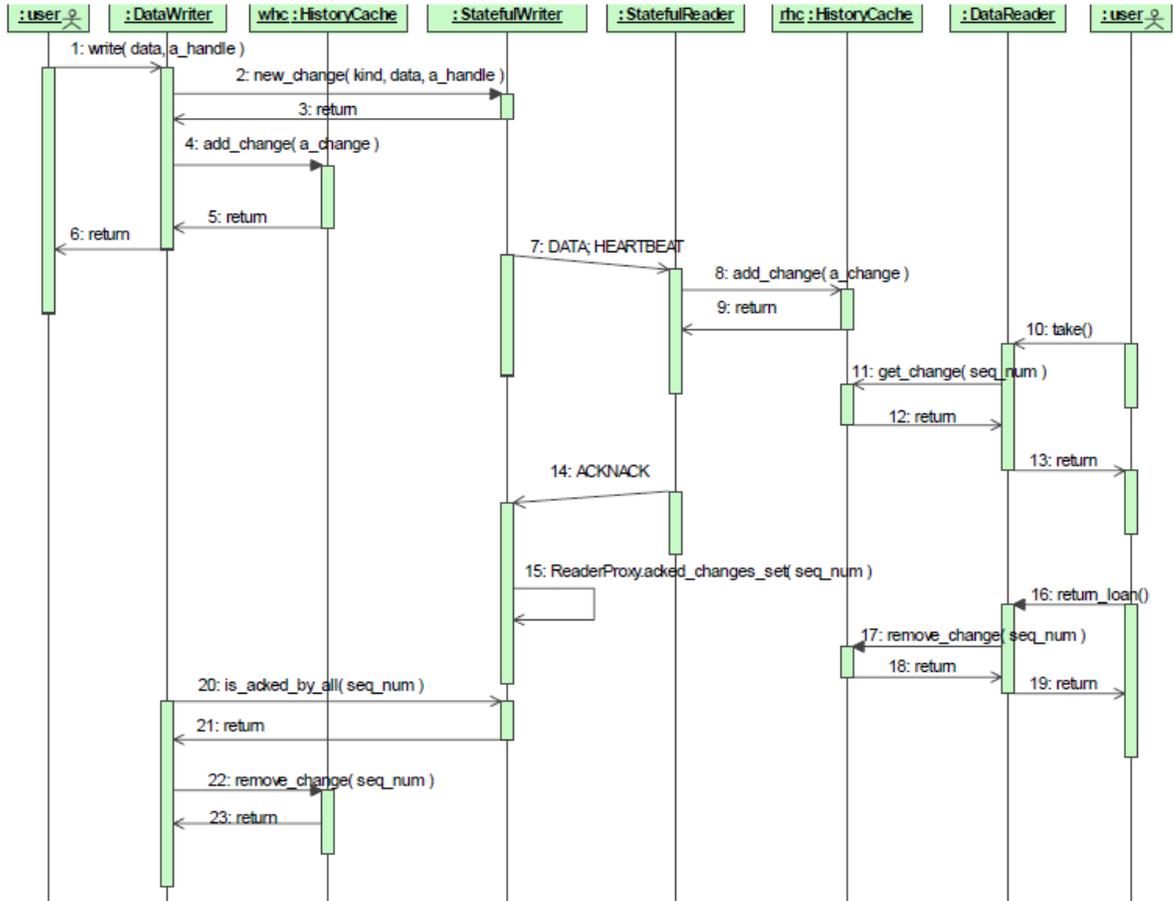


Figure 1-9: RTSP Behavior[9].

1. The DDS user writes data by invoking the write operation on the DDS DataWriter.
2. The DDS DataWriter invokes the new_change operation on the RTPS Writer to create a new CacheChange. Each CacheChange is identified uniquely by a SequenceNumber.
3. The new_change operation returns.
4. The DDS DataWriter uses the add_change operation to store the CacheChange into the RTPS Writer's HistoryCache.
5. The add_change operation returns.

6. The write operation returns; the user has completed the action of writing Data.
7. The RTPS Writer sends the contents of the CacheChange changes to the RTPS Reader using the Data Sub message and requests an acknowledgment by also sending a Heartbeat Sub message.
8. The RTPS Reader receives the Data message and, assuming that the resource limits allow that, places the CacheChange into the reader's HistoryCache using the add_change operation.
9. The add_change operation returns. The CacheChange is visible to the DDS DataReader and the DDS user. The conditions for this depend on the reliabilityLevel attribute of the RTPS Reader.
 - a. For a RELIABLE DDS DataReader, changes in its RTPS Reader's HistoryCache are made visible to the user application only when all previous changes (i.e., changes with smaller sequence numbers) are also visible.
 - b. For a BEST_EFFORT DDS DataReader, changes in its RTPS Reader's HistoryCache are made visible to the user only if no future changes have already been made visible (i.e., if there are no changes in the RTPS Receiver's HistoryCache with a higher sequence number).
10. The DDS user is notified by one of the mechanisms described in the DDS Specification (e.g., by means of a listener or a WaitSet) and initiates reading of the data by calling the take operation on the DDS DataReader.

11. The DDS DataReader accesses the change using the `get_change` operation on the `HistoryCache`.
12. The `get_change` operation returns the `CacheChange` to the `DataReader`.
13. The `take` operation returns the data to the DDS user.
14. The RTPS Reader sends an `AckNack` message indicating that the `CacheChange` was placed into the Reader's `HistoryCache`. The `AckNack` message contains the GUID of the RTPS Reader and the `SequenceNumber` of the change. This action is independent from the notification to the DDS user and the reading of the data by the DDS user. It could have occurred before or concurrently with that.
15. The `StatefulWriter` records that the RTPS Reader has received the `CacheChange` and adds it to the set of `acked_changes` maintained by the `ReaderProxy` using the `acked_changes_set` operation.
16. The DDS user invokes the `return_loan` operation on the `DataReader` to indicate that it is no longer using the data it retrieved by means of the previous `take` operation. This action is independent from the actions on the writer side as it is initiated by the DDS user.
17. The DDS `DataReader` uses the `remove_change` operation to remove the data from the `HistoryCache`.
18. The `remove_change` operation returns.
19. The `return_loan` operation returns.

20. The DDS DataWriter uses the operation `is_acked_by_all` to determine which CacheChanges have been received by all the RTPS Reader endpoints matched with the StatefulWriter.

21. The `is_acked_by_all` returns and indicates that the change with the specified 'seq_num' SequenceNumber has been acknowledged by all RTPS Reader endpoints.

22. The DDS DataWriter uses the operation `remove_change` to remove the change associated with 'seq_num' from the RTPS Writer's HistoryCache. In doing this, the DDS DataWriter also takes into account other DDS QoS such as DURABILITY.

23. The operation `remove_change` returns.

1.6.2 Publication

The publication represents the sending side and can be described by the following properties:

- **Topic:** a string label that uniquely identifies the distributed issues.
- **Type:** a string label that identifies the issue's data format to help the middleware in serialization and de-serialization process.
- **Strength and persistence:** these two values allow subscription to arbitrate among issues of the same topic sent by multiple publishers. Strength expresses the relative priority of one publisher relative to another. And persistence indicates the time for which the issue remains valid after its publishing time. Subscription

receives issues from the strongest publication and accepts issues from a weaker publication when the issues of the stronger are expired.

- Time to keep: is a value to specify how long to store new issues. New subscribers receive all issues that have not surpassed the publication's time to keep period.
- Publication mode: which specify when the publication should send new issues to subscribers.
 - Synchronous mode: issues are sent immediately and in the same context of the application thread.
 - Signaled mode: issues are sent immediately but by another thread provided by the middleware. This feature is important when the application thread is time-critical and cannot afford the non-deterministic delay caused by making a network call.
 - Asynchronous mode: issues should not be sent immediately but stored temporarily and another middleware's process sends all stored issues periodically.

1.6.3 Subscription

Subscription represents the receiving side in unidirectional communication. Subscription has the following properties:

- Topic: to specify the issue to be received.
- Type: to help the middleware to select the appropriate operations to serialize and de-serialize the issues.

- Minimum separation time: which specify the smallest time period between two issues that the receiving application can handle. Minimum separation is timing constraint that helps to control the issues flow. The middleware should not deliver any issue before the minimum separation time since the last issue.
- Deadline: which is the maximum waiting time the receiving application can wait without new data. The middleware should notify the application when new issues do not arrive before deadline time.

1.7 MMOG Classifications

The classifications of MMOG type has many different definitions. The main definition of MMOG or MOG is to be able to support thousands or hundreds of thousands of players playing at the same time.

Based on the type of the game, we can categorize the MMOG into:

- Massively Multiplayer Online Role-playing Game (MMORPG).
- Massively Multiplayer First-person Shooter (MMOFPS).
- Massively Multiplayer Online Real-time Strategy Game (MMORTSG).
- Massively Multiplayer Online Turn-based Strategy Game (MMOTBSG).
- Massively Multiplayer Online Sports Game (MMOSG).
- Massively Multiplayer Online Racing Game (MMORG).

- Massively Multiplayer Online Rhythm Game (MMORG).
- Massively Multiplayer Online Management Game (MMOMG).
- Massively Multiplayer Online Social Game (MMOSG).
- Massively Multiplayer Online Bulletin Board Game (MMOBBG).

MMOGs have two main categorizations based on the game time aspects. They are real time and turn based games. In real-time games, the time is kept continuously without any stop regardless of what the users are doing. This kind of games is usually considered to be more realistic because it reflects the real life. On the other hand, real-time games usually consume more resources than other categories during its operation. With the game time keeps running all the time, the demand of game calculations are surely higher.[2].

Turn based games is not a real-time game, it runs the game by giving each user time to make his decision about what to do. The best example of this type of game is the chess. In a chess game, one player can take an action at a time. Because of this mechanism, the use of turn based time system is not always a choice for many games, it is kind of limited type with these games like chess.[2].

CHAPTER 2

LITERATURE REVIEW

Lately, MMOG has come to be as an online game that allows hundreds or thousands of players from all world wild to interact with each other at the same time.

Because of the dramatic increase in the number of the online players, the classic implementation architecture is no longer appropriate for this kind of real-time situations, a scalable number of users, network bandwidth, high-quality graphics, reliable, fault-tolerant, low-cost, load-balancing, and secured network. All this require a loosely coupled architecture to support evolution in Massive Multiplayer Online Game application.

The need for the specific requirements of the online multiplayer game makes the well-known protocols belong to TCP/IP family is very limited to be used to overcome all issues may be faced. Based on this fact, many researches have been done to come up with suitable protocols to provide the exact functionalities required and to eliminate any overhead that may affect the performance.

It is assumed that a game service provider can have access to a number of servers located over the world, finding the most suitable way to partition the world into servers is a very hot topic on the subject of multiplayer online gaming. Choosing the locale servers in such a way that reduces the total interactive communication delay perceived by all users is the main objectives of many researches.[1].

Client-server model introduces a single way to handle thousands of users and point of failure to the game, by distributing the players over many servers remove this bottleneck, but require special synchronization techniques to provide a consistent game for all players. The classic synchronization methods are not optimized for the requirements of fast-paced multiplayer games. Using multiple copies of the game state and rollbacks is one way to deal with this kind of low-latency consistency.[10].

Some efforts have been done to manage overall system power consumption while users playing online games, ARIVU is a special middleware tries to capture both longer possible sleep times and shorter ones for efficient power management, it also collects data about the status of the game based on already existing information.[11].

The work has been done to minimize the energy consumption by adaptively varying resource consumption.

Gendu is a middleware used to transform asynchronous remote writes into synchronous local operations using local locks only, to accomplish this, a consistency strategy consisting of three parts has been developed, Position-based, Asynchronous Remote-Write and Write operations. The main idea of this middleware is to transform the Locking systems into Lockless. This is because the lockless systems have very special properties over the locking systems such as simplicity, ideally and improved performance. The actual implementation Gendu meets the expected high-level design by implementing a cell-scape and a server network, and by enforcing the consistent communication protocol, all of which are encapsulated into Gendu's middleware. It manages the cells, the server network and all of the other consistency-specific behavior.[12].

Massiv middleware is a middleware used to shorten the implementation process of the distributed MMOGs. Furthermore, this middleware is implemented to run on more than one server in the worldwide. Therefore, and to make sure that these servers are consistent, it takes into account the issues of latency, security, and synchronization.[13].

ZeroC Internet Communications Engine is a powerful middleware as Corba, and to avoid the complexity in the CORBA, ICE used many features to simplify this complicated cores in CORBA. This helps in MMOG maintenance and development, like the update of the software, protocols capability, and it supports Multilanguage.[6].

Microforté is a BigWorld Technology includes a reliable, scalable and customizable, platform that can deal with thousands of thousands of users, it uses special tools to build the virtual globe, since it is a 3D based, it uses a special 3D engine. The main feature of the BigWorld server is that it can be reconfigured at runtime.[13].

DoIT is a middleware that attempts to meet most of the required features for the MMOGs, it is slightly close to our design in one main feature, it is customizable, designer or developer can customize it easily. While still it does not use pub/sub model. It is an event driven model, since that it uses a message-oriented rather than an RPC architecture, because of this feature, it ensures to provide a smooth gaming states rather than blocking/waiting model for the updates from the server.[13].

Cloud-DReAM is a cloud-based dynamic resource provisioning middleware; it automatically allocates resources from a cloud-computing provider whenever the current resources prove insufficient to handle the load imposed on the system. In addition, it

disposes of previously acquired resources when they are no longer necessary. This way, CloudDReAM allows game companies to deploy their own private server clusters without over-provisioning them, and resort to the services of a cloud provider when their own resources are not sufficient to handle the load. As a result, resources are used more efficiently and the costs of supporting the game are reduced, benefiting every stakeholder involved. Cloud-DReAM allows game infrastructures to be deployed conservatively and scale continuously according to runtime load, allowing game managers to pay for infrastructure according to demand.[14].

The Lucid middleware affords different layers for the game architecture. One of the most important features is the use of the special protocol in an attempt to use advanced routing, but the model itself represented as a c/s model.[15].

Atlas provides virtual globe platform based on either a c/s or a peer-server architecture. The virtual globe is divided among many servers. Atlas providing a special interface and different components to support management, replication control, scalability, mutual exclusion, and to balance the load.[3].

The Real-Time platform middleware is slightly close to our design in its use of a pub/sub model for controlling the management issues. RTF handles the replication issue in the broadcasted objects shadow, if there is any replication it can handle it automatically. It uses the serialization model to pass the messages in an object-based messaging, even though the design is mainly organized toward server-based messaging.[5].

Colyseus is a middleware is also slightly close to our design, filtering data based on some specific constrains with regard to the management interest while it does not use the pub/sub model. The design built based on an interest range requests, such as an object discovery model should gather specific data at runtime constrained with some conditions, enhanced by a predictive and replica mechanisms.[16].

Journey is a middleware architecture for massively multiplayer gaming, it attempts to address the ability to satisfy the MMOG fundamental requirement such as load balancing, failure discovery, game cheating, capability, and scalability by applying layers model. It is considered as a multilayer middleware.[7].

Lamoth is a cloud middleware for MMOGs that provides an interface for in-game message dissemination. Lamoth handles the exchange of game messages between nodes by making use of an arbitrary number of off-the-shelve pub/sub servers deployed in the cloud depending on the game scenario. Lamoth used Redis, a popular ready-to-use open-source cloud middleware, which provides, amongst other things, efficient pub/sub capabilities. The evaluation shows that Lamoth allows the MMOG to scale to high numbers of players and can properly handle extremely demanding in-game situations if enough resources are provided[4].

Many peer-to-peer architectures for massively multiplayer gaming have been proposed in the literature trying to minimize host traffic and accessing data, many middleware plugins used to transform the network into a hybrid architecture to partition some of the server tasks to the client nodes. The middleware appears as a transparent middle layer for the both

client and the server. It allows it to be modified without any significant changes in the client or server sides.

In the client/server massively multiplayer online game architecture, client nodes communicate directly to the server where it hosts the game. Multiple users running the client nodes are allowed to communicate just with the server to be able to interact with each. The server preserves the state of the game, aware of every change in the virtual worldwide, while each client preserves the status of its local state, only aware of what happening in its area.

When a user does an action, the client generates an event, and send it to the server. The server then processes the event and update the state based on the game logic, which is performed to the worldwide state. Moreover, the server should broadcast the changes to all clients in the same interest to modify it to the local states. This is called an update based massively multiplayer online game model.

When the middleware broadcasted some of the jobs of the server to all connected peers, it should have similar jobs to the game's node. It needs kind of a state awareness, functionality to spread both changes and actions, the capability to produce the new state changes from specific actions, and the ability to keep the server's status consistent with the middleware state.[17].

Another peer-to-peer work aims to expand upon Homura and Net Homura frameworks, with a focus towards developing a unified deployment and networking system. Which combines Java Enterprise Edition (EE) 5/6 application technologies with low-level

networking support to provide a unified platform for the deployment and execution of both P2P and Client-Server based games, which can be analyzed within a real-world context.[18].

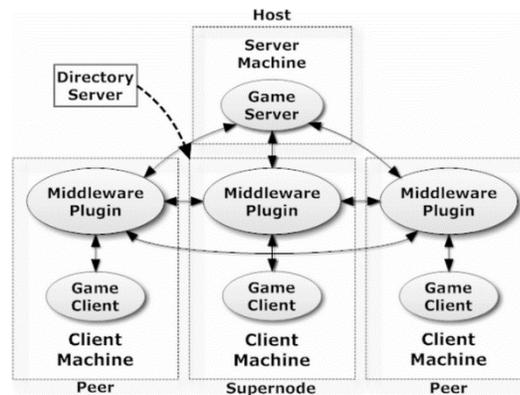


Figure 2-1: Peer-to-Peer Architecture [18].

PKTown 2.0 is a third party peer-peer middleware, it is the second version of PKTown 1.0, and it is developed in order to solve old issues in version 1. The main contributions of the PKTown are to make the task of the server side is loosely coupled and remodeled them to be as simple-tasks, like Forward, NAT, and Meta Server. Furthermore, it lets each server to work in parallel, according to which area the users are located. In addition to that, it provides a solution for the distributed database to make them work in parallel, it splits the two main tables into sub stables and spread them to different servers of databases. With these features, the application will scale up to help thousands of users to be at the same time.[19].

The other side of this work is applying the neural network to make the work smarter and adaptive as in some cases it is required to change some QoS immediately at the runtime, Chapter 4 will explain more.

Because of the increase in the number of the online players, the traditional implementations for the MMOG is no longer appropriate for this kind of real-time situations, a scalable number of users, network bandwidth, high-quality graphics, reliable, fault-tolerant, low-cost, and load-balancing. This requires a loosely coupled architecture to support the evolution in the MMOG.

Lin and Shen have proposed an algorithm called CloudFog, which is used the cloud computing as clouds can free players from huge requirements. They proposed a new node called supernode where this node connects the end user to the cloud.

The cloud sends the data and the updated information to the supernodes, and then the supernodes generate the game streaming and stream it to the users. They have their own topology to select the most suitable supernode taking into account the QoS.[20].

Carter et al. have proposed a novel hybrid peer-to-peer platform and its requirements. They did a comparison with existing platforms to show how their work addressed the issues. They conclude that the combinations of grouping and topology are required in order to deduce the efficacy and to determine the gameplay scenarios.[21].

Carlini and Ricci have focused on the interest management problem in MMOG; they proposed a gossip-based work to a build a P2P overlay with the best effort. They tested their protocol on the game “Second life” and it showed very encouraging results.[22].

Croucher and Engelbrecht have proposed a middleware for an existing MMOG client-server architecture; it can transform the games network into a hybrid that delivers data between players and the server. If there is any change in the server side, the players will not be affected, just reconfigure the middleware.[18].

Weng and Wang used the artificial neural network to propose a dynamic resource allocation in the MMOG, they divided the method into two phases, the load prediction phase, and resource allocation phase, they collected the data, which includes the CPU, network load, and the memory. They have designed and implemented an ANN and an adaptive neural fuzzy (ANFIS) to detect and predict the suitable resource allocation based on each zone, their work shows that the ANN behaves quite better than the ANFIS.[23].

Prasetya used the neural network to detect pot cheating on MMOG, the author showed that the ANN is potentially able to detect the pot, but it needs a proper training and input data to perform well. ANN can be used as an additional filter for the cheater, also the results show that ANN could be used to detect not only the pot cheater, but also another type of cheating, but it needs different models of ANN.[24].

Gaspareto et al. applied ANN to deal with the speed cheating in the MMOG, they applied two different approaches, the multi-layer Perceptron network (MLP) and the focused time lagged network (FTLFN), it was possible to conclude that their utilization avoiding speed cheating in MMOG is possible, once good results were found in this work. The MLP network showed the best performance, where less than one percent of false positive were obtained.[25].

Nae, Iosup, and Prodan proposed a dynamic resource provisioning and data management resources, they proposed a prediction algorithm based on neural network, they tested many scenarios that focused on MMOG-specific properties and data center hosting policies. It is shown that the static resource provisioning is less efficient than the dynamic provisioning by 5 to 10 times.[26].

Again, Nae, Prodan, and Fahringer presented a neural network prediction service, but this time for the management purposes in the MMOG, they focused on the first person shoot (FPS) games. Their approach was based on the distributing the game for sub-areas in a suitable size. The role of the neural network here is to adapt the entity counts accurately and quickly.[27].

Gorlatch et al. proposed a load prediction platform that based on ANN, their approach differs from the others that the ANN realize the load balancing at a generic level, the system loads the balance at real time using the collected information of the application.[28].

Shi et al. used a DR algorithm which is based on the ANN to predict and calculate more accurate readings about the player's motion, they employed the Q-Learning to the platform to decide the impact of the ANN on prediction, their results showed that their method decreased the data transfer rate, the network traffic, and improve the prediction accuracy.[29].

Negrão et al. proposed a middleware called CloudDReAM used to monitor the load on the servers; it also can load the balance when it is reached to the provided threshold. CloudDReAM can use new resources if needed from the cloud provider.[14].

Chen et al. investigated that the network traffic analysis could be different when a bot is playing rather than the player, Bednar and Miller proposed a hybrid GRNN to get the same result by using less information and data. Their GRNN showed that the bots were identified correctly with 99% while the player only by 6%.[30].

CHAPTER 3

WORK OBJECTIVES

Massively Multiplayer Online Game (MMOG) has become very popular; there are thousands of games built to be played online, where hundreds/thousands of users from all around the world can play at the same time for the same game. The hottest issue in the MMOG is the real-time (RT) service, where the players can share their game status in a real-time manner. The main need for the RT platform is to guarantee the quality of service, the platform should deliver a robust and excellent quality in terms of throughput, latency and all other technical requirements as in section 1.2, on besides, the service should be loosely coupled oriented to support the growth of MMOG application.

We are presenting a new platform solution to deal with the real-time online gaming using the DDS middleware. RTinDDS and AIRTinDDS are real-time platforms based on the DDS middleware, this middleware can guarantee to provide the developers, designers, and end users a suitable platform to build their online real-time gaming and to deliver it in a robust manner for the end users. The platform able to allow designers to make the best decision for a specific situation. RTinDDS is a real-time platform that has been implemented by choosing the best QoS in the DDS middleware for the default situation (stable situation) to deliver the best performance. We have implemented a game we call it Plane Model Simulation (PMS). After running our game scenario. Figure **3-1**, we have chosen the best QoS and we applied it on the PMS.

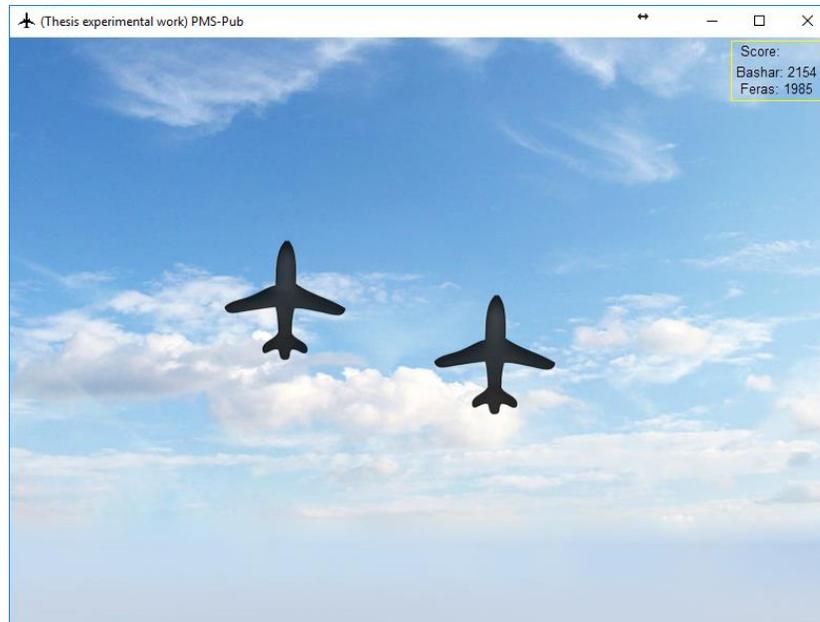


Figure 3-1: PMS over DDS middleware, two players are playing.

PMS shows a very robust real-time platform, the results show that RTinDDS in PMS delivered a very good QoS in terms of reliability, throughput, and latency.

The second part of the work is by making RTinDDS smart enough to adapt itself when any issue occurred at runtime; we have implemented a new platform called AIRTinDDS, this platform can adapt and correct itself once an issue occurred. This is done by choosing the best QoS in that situation using artificial intelligent (AI). Sometimes, it is needed to change some QoS at runtime to accommodate the new unexpected changes. We have applied the AIRTinDDS algorithm in the RTI Connex simulation; it shows a very robust and adaptive platform.

We can list our objective as below.

- The main objective is to prepare and build a full infrastructure (platform) to work with the massively multiplayer online gaming (MMOG) which we can prepare and

deliver it as a provider for anyone needs to work with MMOG systems under DDS middleware.

- The main aim of DDS middleware is to deliver the correct data at the right place at the precise time, even between asynchronous publishers and subscriber. We used DDS to guarantee to deliver correct data at a correct time.
- Build a simulation, plane model simulation (PMS) to prove that the used QoS supported by DDS is working as expected.
- Use the suitable QoS to deal and resolve all issues faced by the network to deliver a real-time behavior
- Apply the AI (Artificial intelligent) to make the system smart to resolve any unexpected issue or behavior by itself.

CHAPTER 4

DDS MIDDLEWARE ARCHITECTURE

The main aim of DDS middleware is to deliver the correct data to the right place at the precise time, even between asynchronous publishers and subscriber. The DDS Middleware is a middle layer that attempts to hide all complex detail of the OS for our MMOG platform. The beauty of this middleware is that it provides APIs for many different languages so you can use whatever supported programming languages to build your own application. All low-level details such as the connection protocols, nodes discovery, IP broadcasting, data managements, special formats, QoS, etc., are managed by this DDS middleware. Figure 4-1.

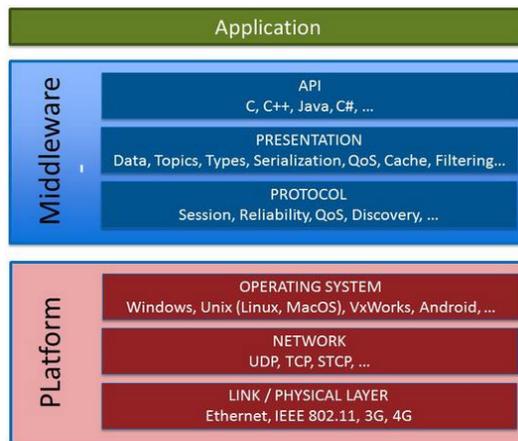


Figure 4-1: DDS Middleware

Usually, real-time applications have their own requirement to model their communication protocols as a central data exchange, where nodes publish data which is then available to the other remote nodes that are waiting for. Real-time applications can be found in many

areas, it could be found in industrial automation, sensor network, simulation, network management systems, distributed control, and telecom control. Generally, any application needs information spreading is a candidate for this kind of network architecture. As in our work we use it to develop MMOG.

Predictable distribution of information with very low overhead is a primary concern in these real-time systems. Because of it is not feasible to add as much as infinite resources, it is important to be able to decide what is the available resources and provide them policies makes the middleware able to assign the resources to the highest important requirements. This need leads to the ability to control Quality of Service properties and values that affect resource utilization, overhead, and predictability.

The necessity to scale the publisher and subscribers to hundreds or hundreds of thousands in a robust manner is also an important need. This is actually not only affects the scalability but also the flexibility: on many of these systems, applications can be added with any necessity to reconstruct the whole system. Data-centric communications architecture decouples publishers from subscribers; the less coupled the senders and the receivers are, the easier these extensions become.

Distributed shared memory is a traditional model that allows data-centric exchanges. However, this kind of models is not able to implement efficiently over a network and does not afford the needed flexibility and scalability. Therefore, another model, the Data-Centric Publish-Subscribe (DCPS) model, has become very common in many real-time systems. This model builds on the idea of a “global data space” that is accessible to all concerned applications. Applications that want to share information to this "global data space"

announce their intent to become “Publishers.” On the other hand, applications that want to read this data space announce their intent to become “Subscribers.” Then, each time this Publisher application posts new data into this “data space,” the middleware propagates the information and the data to all interested applications (Subscribers)[31].

4.1 Publish-Subscribe model

Because of the complexity of these days’ applications, the architecture for servers or nodes is no longer acceptable to be one single machine, the trends nowadays to make the nodes be distributed across many places. As program needs became more complex, some parts of the program were distributed to separate systems. Nowadays applications often utilize dozens to thousands of single computers that interact with each other. These systems are using synchronous communication and point-to-point connections in most cases. As a result, new problems like limited scalability, fragile reliability, or restricted flexibility arise. Further, this approach led to difficult large-scale application development as the aspects of communication were dominating. To cope with the issues mentioned, Pub/Sub systems have been developed. The main idea behind Pub/Sub is, to loosely couple the whole system using a messaging service component (or event notification service). This is achieved by decoupling the creators of messages, referred to as publishers, from the consumers of these messages, referred to as subscribers. Publishers and subscribers do not require any direct mutual knowledge and act independently. A publisher offers a service for which subscribers can express their interest in. In the case of new data, a new message is created by the publisher and propagated automatically and asynchronously to all interested subscribers via the event notification service.

DDS uses the pub/sub model for discovery and management of data flows between the entities of the DDS, including publisher, subscriber entities, services of durability, and databases. Many other patterns are built on this powerful model. Figure 4-2.

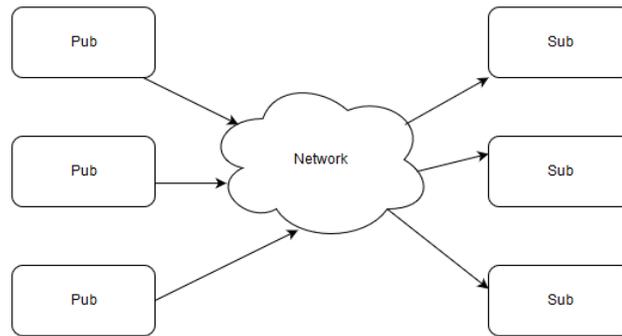


Figure 4-2: Architectural view of Publisher-Subscriber

4.2 QoS Architecture for MMOG

The DDS depends on the use of QoS. A QoS can be defined as a set of characteristics that controls some aspect of the behavior of the DDS Service. QoS is comprised of individual QoS policies.[31].

We discuss the suitable QoS of DDS used to improve MMOG and get the best effort of the network with regard to its reliability, availability, scalability, robustness, and congestion issues.

Many QoS implemented in the DDS middleware to make it as adaptive as possible in a different situation depends on the application. In MMOG, by changing a specific QoS, that can affect directly to the performance of the MMOG application if used wisely.

4.2.1 Durability

The decoupling between DataReader (player reads data) and DataWriter (player writes data) offered by the Pub/Sub model allows the system to write data even if there are no current subscribers on the network. Moreover, a subscriber who joins the system after some data has been written could potentially be interested in accessing the most current values of the data as well as some history. This QoS policy controls whether the service will actually make data available to late-joining players. Note that although related, this does not strictly control what data the game will maintain internally. That is, the game may choose to maintain some data for its own purposes and yet not make it available to late-joining players if the DURABILITY QoS policy is set to VOLATILE.[31].

Sometimes it could be a case where the new joining player needs some old information about the game while it is already started for a while, this QoS can handle this situation, and it would send all old required information to the new joining ones.

4.2.2 Presentation

This policy controls the extent to which changes to data instances can be made dependent on each other and also the kind of dependencies that can be propagated and maintained by the game.

The Presentation QoS can be set with different flavors, the setting of `coherent_access` controls whether the game will preserve the groupings of changes made by the publishing application by means of the operations `begin_coherent_change` and `end_coherent_change`.

The setting of `ordered_access` controls whether the game will preserve the order of changes. The granularity is controlled by the setting of the `access_scope`. [31].

We use this QoS to handle the way of how the information should be delivered to the joining players, some data could be delivered as a group and some as an individual, it depends on the type of the data.

4.2.3 Deadline

This QoS policy is useful for cases where the data is expected to have each instance updated periodically. On the publishing side, this setting establishes a contract that the application must meet. On the subscribing side, the setting establishes a minimum requirement for the remote publishers that are expected to supply the data values.

When the Service ‘matches’ a `DataWriter` and a `DataReader` it checks whether the settings are compatible (i.e., $\text{offered deadline period} \leq \text{requested deadline period}$) if they are not, the two entities are informed (via the listener or condition mechanism) of the incompatibility of the QoS settings and communication will not occur.

Assuming that the reader and writer ends have compatible settings, the fulfillment of this contract is monitored by the Service and the application is informed of any violations by means of the proper listener or condition.

The value offered is considered compatible with the value requested if and only if the inequality “ $\text{offered deadline period} \leq \text{requested deadline period}$ ” evaluates to ‘TRUE.’

The setting of the DEADLINE policy must be set consistently with that of the TIME_BASED_FILTER. For these two policies to be consistent the settings must be such that “deadline period \geq minimum_separation.”[31].

We use this QoS to detect when a link is overloaded, a network congestion occurs and affects the packet loss and the latency. In DDS middleware, a deadline quality of service can be used to control and detect the congestion in the network.

4.2.4 Latency-Budget

This policy provides a means for the game to indicate to the middleware the “urgency” of the data-communication. By having a non-zero duration the Service can optimize its internal operation.

This policy is considered as a hint. There is no specified mechanism as to how the service should take advantage of this hint.

The value offered is considered compatible with the value requested if and only if the inequality “offered duration \leq requested duration” evaluates to ‘TRUE.’[31].

We used this QoS, to determine if the connection is stable and can handle the game load or not, it can be used to send the players a message to tell them that your connection is not stable; you can reconnect to get a better connection.

4.2.5 Reliability

This policy indicates the level of reliability requested by a DataReader or offered by a DataWriter. These levels are ordered, BEST_EFFORT being lower than RELIABLE. A DataWriter offering a level is implicitly offering all levels below.

The setting of this policy has a dependency on the setting of the RESOURCE_LIMITS policy. In case the RELIABILITY kind is set to RELIABLE the write operation on the DataWriter may block if the modification would cause data to be lost or else cause one of the limits in specified in the RESOURCE_LIMITS to be exceeded. Under these circumstances, the RELIABILITY max_blocking_time configures the maximum duration the write operation may block.

If the RELIABILITY kind is set to RELIABLE, data-samples originating from a single DataWriter cannot be made available to the DataReader if there are previous data samples that have not been received yet due to a communication error. In other words, the service will repair the error and re-transmit data samples as needed in order to reconstruct a correct snapshot of the DataWriter history before it is accessible by the DataReader.

If the RELIABILITY kind is set to BEST_EFFORT, the service will not re-transmit missing data samples. However, for data- samples originating from any one DataWriter the service will ensure they are stored in the DataReader history in the same order they originated in the DataWriter. In other words, the DataReader may miss some data samples but it will never see the value of a data object change from a newer value to an older value.[31]

To achieve a reliable system, a Reliable QoS policy can provide settings in different flavors to track the system reliability. It can ignore some unimportant data while it can ensure to deliver the critical ones.

4.2.6 Transport-Priority

The purpose of this QoS is to allow the application to take advantage of transports capable of sending messages with different priorities.

This policy is considered a hint. The policy depends on the ability of the underlying transports to set a priority on the messages they send. Any value within the range of a 32-bit signed integer may be chosen; higher values indicate higher priority. However, any further interpretation of this policy is specific to a particular transport and a particular implementation of the Service. For example, a particular transport is permitted to treat a range of priority values as equivalent to one another. It is expected that during transport configuration the application would provide a mapping between the values of the `TRANSPORT_PRIORITY` set on `DataWriter` and the values meaningful to each transport. This mapping would then be used by the infrastructure when propagating the data written by the `DataWriter`.

This QoS used to prioritize sending the information to the players. High priorities could be set for the hot data while lower priority for usual data.

4.2.7 History

This policy controls the behavior of the Service when the value of an instance changes before it is finally communicated to some of its existing DataReader entities.

If the kind is set to `KEEP_LAST`, then the Service will only attempt to keep the latest values of the instance and discard the older ones. In this case, the value of `depth` regulates the maximum number of values (up to and including the most current one) the Service will maintain and deliver. The default (and most common setting) for `depth` is one, indicating that only the most recent value should be delivered.

If the kind is set to `KEEP_ALL`, then the Service will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that the Service can use to keep this history are limited by the settings of the `RESOURCE_LIMITS` QoS. If the limit is reached, then the behavior of the Service will depend on the `RELIABILITY` QoS. If the reliability kind is `BEST_EFFORT`, then the old values will be discarded. If reliability is `RELIABLE`, then the Service will block the DataWriter until it can deliver the necessary old values to all subscribers.[31].

This QoS used where some important data should be delivered even not in the real time manner; this QoS can handle such these cases. It can save some data in the history to be delivered when the player is ready. This can ensure a reliable and a robust service.

4.2.8 Resource-Limits

This policy controls the resources that the service can use in order to meet the requirements imposed by the application and other QoS settings.

If the DataWriter objects are communicating samples faster than they are ultimately taken by the DataReader objects, the middleware will eventually hit against some of the QoS-imposed resource limits. Note that this may occur when just a single DataReader cannot keep up with its corresponding DataWriter. The behavior, in this case, depends on the setting for the RELIABILITY QoS. If reliability is BEST_EFFORT, then the service is allowed to drop samples. If the reliability is RELIABLE, the Service will block the DataWriter or discard the sample at the DataReader in order not to lose existing samples.

This QoS used to determine if the joining player has the minimum required resources to be able to launch the game, The system sends him a message to inform him that these resources you are using are not good enough to join the game.

We have chosen the suitable QoS of DDS to improve PMS and get the best effort of the network with regard to its reliability, availability, scalability, robustness, and congestion issues.

CHAPTER 5

PROBLEM STATEMENT AND PROPOSED SOLUTION

Because of the dramatic increase in the number of the players in online gaming, the classic implementation architecture is no longer appropriate for this kind of real-time situations, as it may raise scalability and interdependence issues [1][32]. A scalable number of users, network bandwidth, high-quality graphics, reliable, fault-tolerant, low-cost, load balancing, and security. All these features require a loosely coupled architecture to support the evolution of improvement in the Massive Multiplayer Online Gaming application.

The need for the specific requirements of online multiplayer games makes the well-known protocols belong to TCP/IP family is very limited to be used to overcome all issues may be faced [1]. E.g. peer to peer consumes a lot of bandwidth. It is also less reliable in terms of security as global game state is stored in local peer. So malicious peers can modify the game state and propagate to other peers [1]. Based on this, many researches have been done to come up with suitable protocols to provide the exact functionalities required and to eliminate any overhead that may affect the performance.

A DDS middleware is one of the most suitable solutions to deal with this kind of applications, DDS middleware ensures to deliver a very high-performance, scalable application, reliable, robust, high availability rate and low congestion issues. This is can be done by using the QoS policies integrated with this DDS middleware.

DDS middleware specifies a very powerful QoS that can be used in a smart way to make the MMOG powerful compared to other platforms.

In this work, we make a comparative and an experimental study to show the power of this middleware in the MMOG field. We create a complete infrastructure contains the suitable QoS and the suitable configuration in different situations depends on the need of the game, this will help the designers of the game, the developers and for sure the end users.

There are many quality of services implemented in the DDS middleware used to make adaptive and robust behavior for different situation depends on the application. In MMOG, there are main QoSs that can affect directly to the performance of the MMOG application. We can improve the performance as we use them wisely.

5.1 Durability

As described the benefit of this QoS in the previous section, we set the suitable value based on the game scenario. This will be shown in details in chapter 6.

Durability = TRANSIENT_LOCAL

History = KEEP_LAST

Or

History = KEEP_ALL

Two types of design, this can be chosen based on the game scenario, Chapter 6 will explain scenarios and QoS profiles.

5.2 Presentation

This QoS can be set as follows:

Presentation = INSTANCE

Coherent_access = true and order_access = true in different situations depends on the configuration.

In the case of sending information data to the players, Coherent_access = true and order_access = false could be used. And on the other hand, if orders or actions have to be sent, then Coherent_access = false and order_access = true could be used.

5.3 Deadline

This QoS should be set a suitable amount of time, based on the game configuration and scenario; it could be changed from game to game.

If there is no data sent during this deadline period, we can detect that there is an issue (congestion) on the network.

5.4 Latency-Budget

This QoS policy is used to determine whether the connection is stable and can handle the game load or not, it can be used to send a message to the player to inform that your connection is not stable, you can reconnect to get a better connection.

Latency-Budget = 'maximum acceptable delay' based on the game configuration. In this work 400ms is used.

5.5 Reliability

Scenarios could be best effort or reliable.

Reliability = RELIABLE and in some cases, it could be Reliability = BEST_EFFORT

5.6 Transport-Priority

High priorities could be set for the hot data while lower priority for usual data.

Transport-priority = 100 for hot data and Transport-priority = 10 for ordinary data

5.7 History

This QoS can save some data in the history to be delivered when the player is ready. This can ensure reliable and robust service.

History = KEEP_LAST with depth=100, the depth can be changed based on the configuration.

Or

History = KEEP_ALL

5.8 Resource-Limits

A warning message could be sent to the player to inform him that the resources you use are not good enough to join the game.

5.9 Proposed algorithm

Figure 5-1, Figure 5-2 show the infrastructure of RTinDDS/AIRTinDDS implementation for the MMOG/PMS. Seven topics have been implemented, as we believe these topics will cover all required data in the MMOG/PMS context.

- 1- Instruction Topic (the main topic), this topic used to send the instructions and orders between the players, e.g. location updates, levels, ...
- 2- Configuration Topic, this topic used to update some configuration related to the shape, color, size, etc.
- 3- Score Topic, this topic used to deal with the scores of the players.
- 4- Game Status Topic, this topic used to show the status of the game, e.g. start, restart, stop, exit, etc.
- 5- Interest management Topic, this topic used to specify which objects will be interested between the players.
- 6- Location topic, description of object's location.
- 7- Velocity, Obtain object's current velocity.

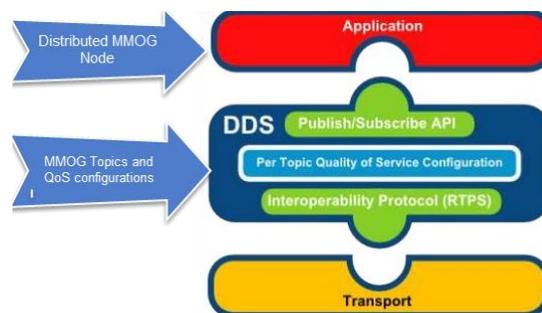


Figure 5-1: DDS infrastructure.

Where each topic has its own QoS configuration based on the type of the topic, Figure 5-2 shows the hierarchy of the designed model.

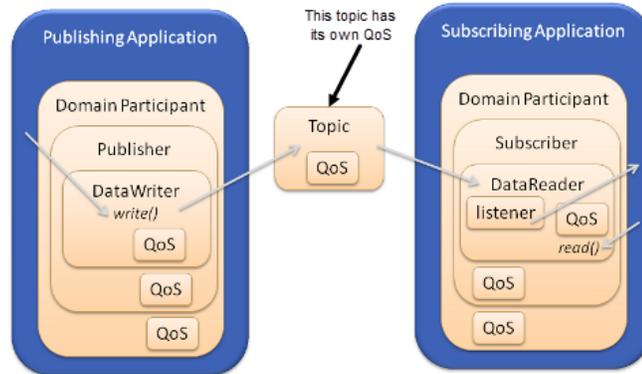


Figure 5-2: The hierarchy of the designed model

As all these QoS have been set in the work, the result shows that the DDS middleware is a suitable solution to deal with the MMOG application. DDS middleware ensures to deliver a very high-performance, scalable application, reliable, robust, high availability rate and low congestion issues. This can be done by using the mentioned QoS policies with their suitable values integrated with the DDS middleware.

An experimental study has been done to show the power of this middleware in the MMOG field. A complete infrastructure has been implemented that contains the suitable QoS and the suitable configuration in different situations depends on the need of the game, The proposed solution will help the designer of the game, the developers and for sure this will reflect to the end users.

5.10 Proposed algorithm for RTinDDS

This algorithm has been applied on the PMS. Any issue can be detected and resolved by changing the suitable QoS in that situation.

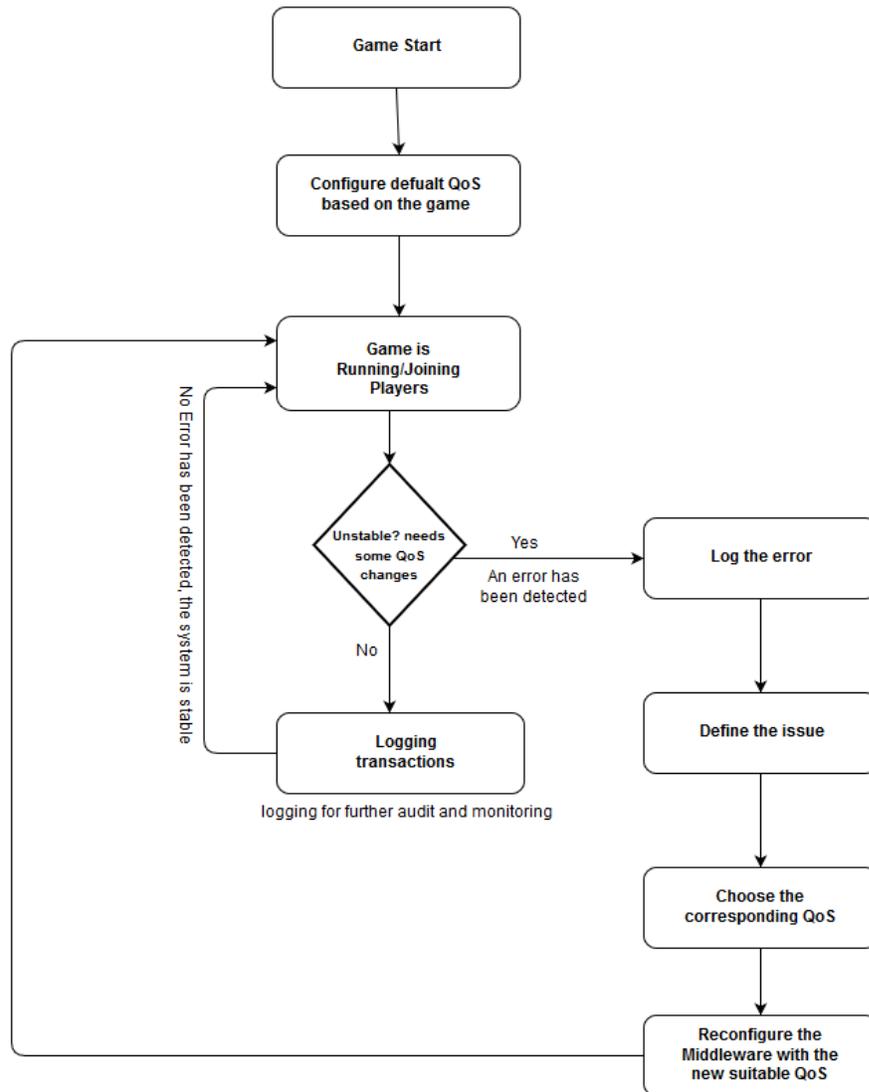


Figure 5-3: RTinDDS Algorithm flowchart

At the starting of PMS, the system read the default QoS file which contains all default values for the required QoS based on the scenario. While running PMS for a long time, sometimes the system read some warnings and hints that informs us that the PMS could

go in an unstable state (Deadline QoS sends receives call-back messages from the subscriber as a warning, Latency-Budget, and Resource-limit do the same), the system keeps checking these messages and warnings and do the required changes with regards to the suitable QoS based on that situation.

The system logs that issue in a log file, then define the issue and chose the corresponding QoS that fixes that issue. All this handled at run time while the game is running. The following is a pseudo code for this algorithm.

5.10.1 RTinDDS Proposed Solution

```
Initial state - Set default QoS for the system
```

```
While      running/joining player
```

```
    if an error has been detected: error = true
```

```
        Log the error for later checking
```

```
        Define the issue and the cause
```

```
        Choose the corresponding QoS with the defined issue
```

```
        Reconfigure the middleware
```

```
    If error = false
```

```
        Log the transaction to keep track the system
```

As we wanted to make PMS adaptive to any issue at runtime, and as mentioned in the above sections, we come up with the second part of the work, which is AIRTinDDS. A new algorithm has been developed based on Artificial Neural Network (ANN) to make the PMS smart enough to adapt itself.

The following algorithm is applied to the experimental work. The AIRTinDDS detects if there is a specific issue and solve it with its corresponding solution by changing the suitable QoS using the ANN. Figure **5-4**.

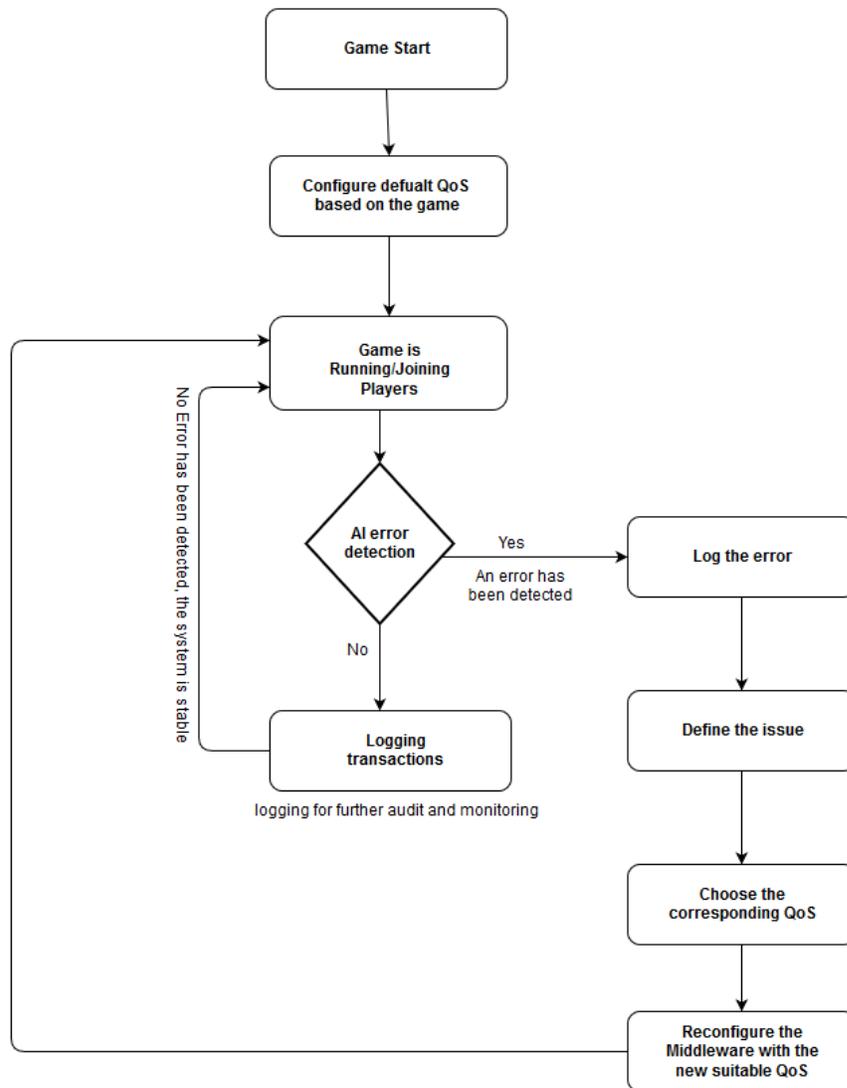


Figure 5-4: AI Algorithm (AIRTinDDS) flowchart

The above flowchart shows that the system now has its own capability to adapt itself to any unexpected behavior. The system trained in good circumstances to make it, as it is the stable system. Now, an error AI detector used to detect any error based on what the system learned. The algorithm can reconfigure the middleware with the new suitable QoS based on the error logging setup.

The following is the new pseudo code for the AIRTinDDS.

5.10.2 Algorithm Proposed Solution

```
Initial state - Set default QoS for the system

While      running/joining player

    if AI detect an error: error = true

        Log the error for later checking

        Define the issue and the cause

        Choose the corresponding QoS with the defined issue

        Reconfigure the middleware

    If error = false

        Log the transaction to keep track the system
```

From the pseudo code above we can notice that the AI unit can detect any error while the player is joining, when an error occurs, The AI unit can define the type of the issue and define the cause, once this is done, AI can tell what is the best QoS needs to be changed. When all this is happening, PMS can reconfigure itself with the new QoS values at runtime.

CHAPTER 6

EXPERIMENTAL SETTINGS/IMPLEMENTATION AND RESULTS

In this chapter, we describe the environment and methodology we use to perform the experiments. The goal of the experiment is as below:

- Evaluate and compare between both RTinDDS and AIRTinDDS in our PMS.
- Evaluate the feasibility of using DDS middleware in MMOG context.
- Compare results while the number of players increases, test the scalability.
- Prove that AIRTinDDS can adapt itself.
- Prove that RTinDDS/AIRTinDDS can work on real-time manner.
- Set different scenarios with different profiles, and record the observations.

In this section, we evaluate the performance of the proposed solution by implementing a game, Plane Model Simulation (PMS) and apply the algorithms on it. We carried the experiment using the following software and monitor tools as in Table **6-1**.

Table 6-1: Tools and Programs

Tool	Version	Use
Eclipse Java Mars	Mars Release (4.5.0)	To implement the game based on the DDS middleware and the suitable QoS
RTI Connex	5.1.0	Real Time
Wireshark	1.2.3	Used to evaluate the proposed experiment.
RTI PerfTest	5.2.0	Combined Latency and Throughput Performance Test
NeurophStudio	2.92	Java neural network framework

6.1 Experiment setup and Performance Metrics

We discuss the result of applying our suggested QoS in the DDS middleware, and how it improved the performance of the MMOG system. It shows that it gets the best effort of the network concerning its reliability, availability, scalability throughput, and latency.

These charts below show the latency for publish/subscribe messaging in applications. Latency was measured, in microseconds, by having the consumer (DDS DataReader) echo messages back to the producer (DDS DataWriter). This allowed round-trip latency to be measured on the sending machine, avoiding time synchronization issues. Figure **6-1**

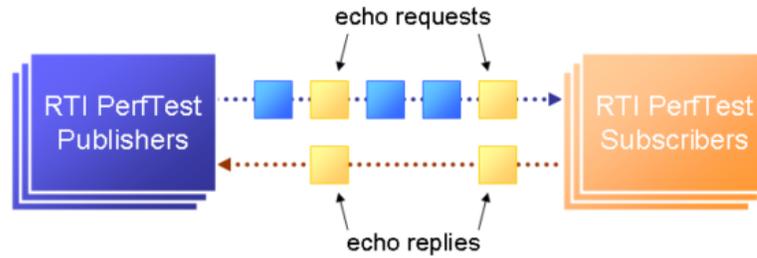


Figure 6-1: PerfTest echo requests/replies

As shown in Figure 6-2, this is the throughput graph for applying different data payload on the same game setup for PMS; this shows the throughput for the publisher/subscriber nodes on the network while running the game.

We have repeated the simulation (running the game and play our PMS) six times and each time we increased the load exponentially, we started by sending data load 32 bytes and ended up by sending 1024 bytes.

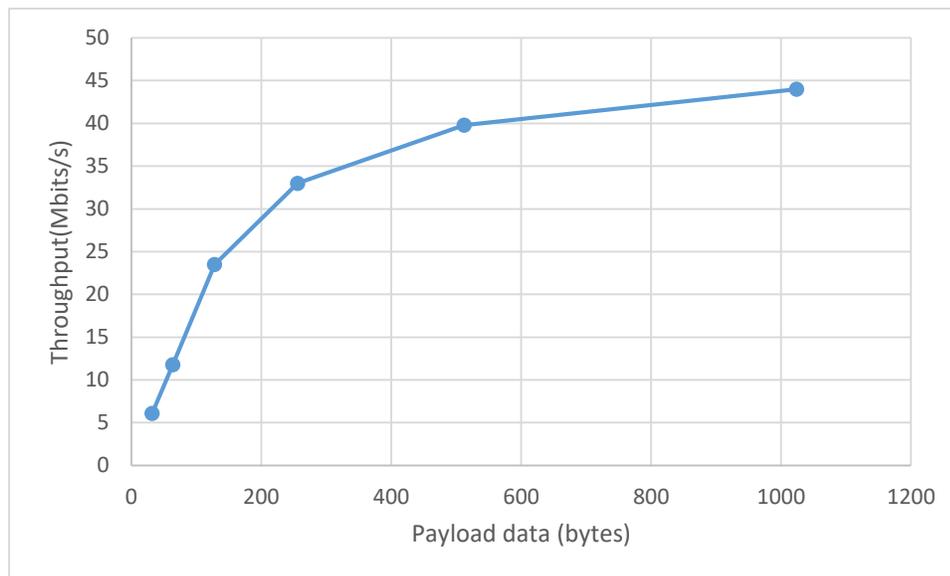


Figure 6-2: PMS Throughput

By applying different scenarios and different data load, the throughput keeps showing a very good result based on the QoS setup.

The results show that the system is always available where there is no such failure recorded, also no packet loss for those topics where we assigned their priority to be the highest. This proved the availability and the reliability for our MMOG platform.

We repeat the experiment with different data size set, the results keep showing that each time we doubled the data load; we got an increased throughput without any loss in the packets. Figure 6-3.

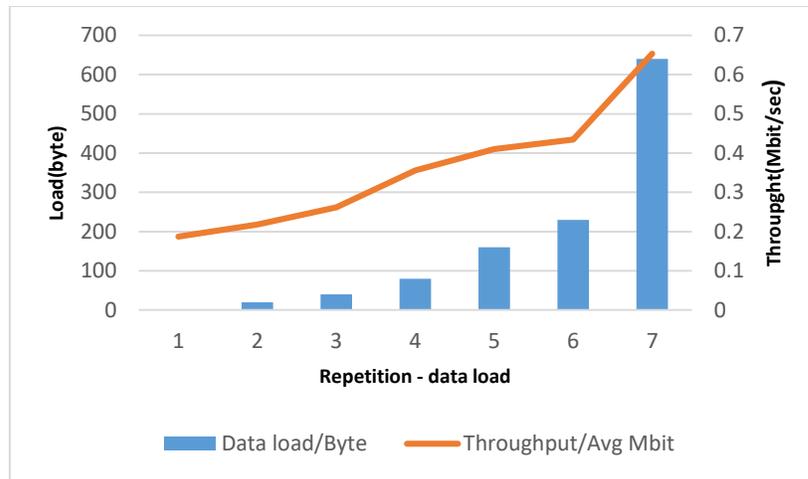


Figure 6-3: Data load Vs. Throughput

Figure 6-4 shows the average number of packets per seconds where we doubled the transmitted data and repeated seven times. It shows that there is no packet loss, whenever the data load increased, the Avg. packet increased.

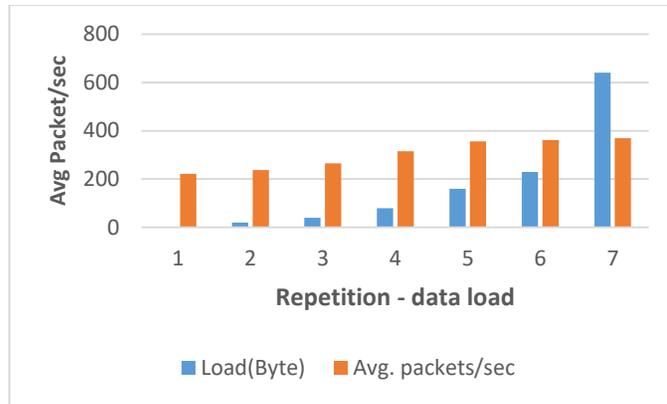


Figure 6-4: Data load Vs. Avg. packet/sec

On the other hand, we have measured the latency for PMS, as mentioned earlier, we used a perfTest tool powered by RTI DDS middleware, this tool allows us to measure the round trip time as it sends an echo requests with timestamp and waits for the echo replies with the same timestamp and then at the publisher side, it calculates the latency by subtracting the timestamps on the same machine. Figure 6-5

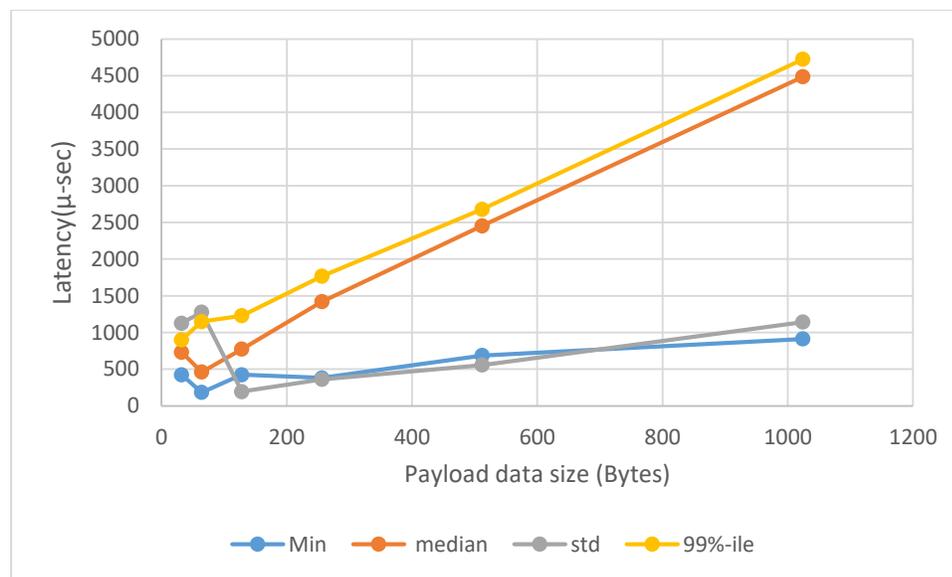


Figure 6-5: PMS Latency

We can notice from Figure 6-5 that we can guarantee by 99% that the latency is equal or less than 4724 microseconds.

However, we were not able to focus on the scalability in our PMS experimental work, even though we sent more than million sample per the running time (few minutes), so we have carried out another experimental work using RTI Connex (platform compliant with the Data Distribution Service (DDS) standard) to allow thousands of players to join the game with the same QoSs we have already chosen for the experiment work.

As we mentioned in chapter 5 we set 7 topics to be published in our PMS, our experiment shows that these seven topics can cover all data needed to make the PMS stable and works as expected. We have carried out our work with the following configuration. Table 6-2.

Table 6-2: Configuration table

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	2	2	1	2	0.5	0.5	1
Simulation Period	~4 hours						
Number of sent samples per second	~1000 samples/sec						
Number of pub/sub per simulation period	~10,000						
Topic size	65KB						
Avg. data sent/sec	65MB						

We've calculated some important statistics. And based on the algorithm and the choice of the QoS, We can ensure that the experiment has shown a robust QoS configuration with very high throughput with very low latency.

Table 6-3: Topics statistics

Topics	Mean Latency μ -sec	median latency μ -sec	std latency μ -sec	max latency μ -sec	min latency μ -sec
--------	-------------------------	---------------------------	------------------------	------------------------	------------------------

LoctionTopic	2.40	2.40	637	2.42	2.38
VelocityTopic	84.2	84	163	84.2	84
InstructionTopic	84.2	84	163	84.2	84
IntrestTopic	84.2	84	163	84	84
ConfigTopic	90.8	91	953	91	91
ScoreTopic	90.8	91	953	90.8	90
GameStatusTopic	2.41	2.4	476	2.41	2.4
max latency value	91 μ-sec				
min latency value	2.38 μ-sec				

Table 6-4: Network statistics

Packets:	1000000
Avg. packets/sec:	9718
Avg packet size:	485.878 bytes
Avg Mbps:	40

We address the performance evaluation for DDS QoS by carried out an experimental work and as mentioned above we have implemented a Plane model simulation (PMS) where many pub/sub-nodes can join and play the game.

We calculated the average throughput by changing the data load each time; we ran the game many times and recorded the changing in the number of bytes per second while it is changed as the data load changed. The results show a very high throughput on the other side, a very low latency. In addition to that, we have carried out another experimental work where we wanted to prove the ability of our proposed platform to scale the number of nodes. We set the QoS to be exactly the same as in our PMS, we were able to scale the

number of nodes as there was a 10000 publish/subscribe players playing at the same time on different platforms, the results show also a very high throughput with very low latency.

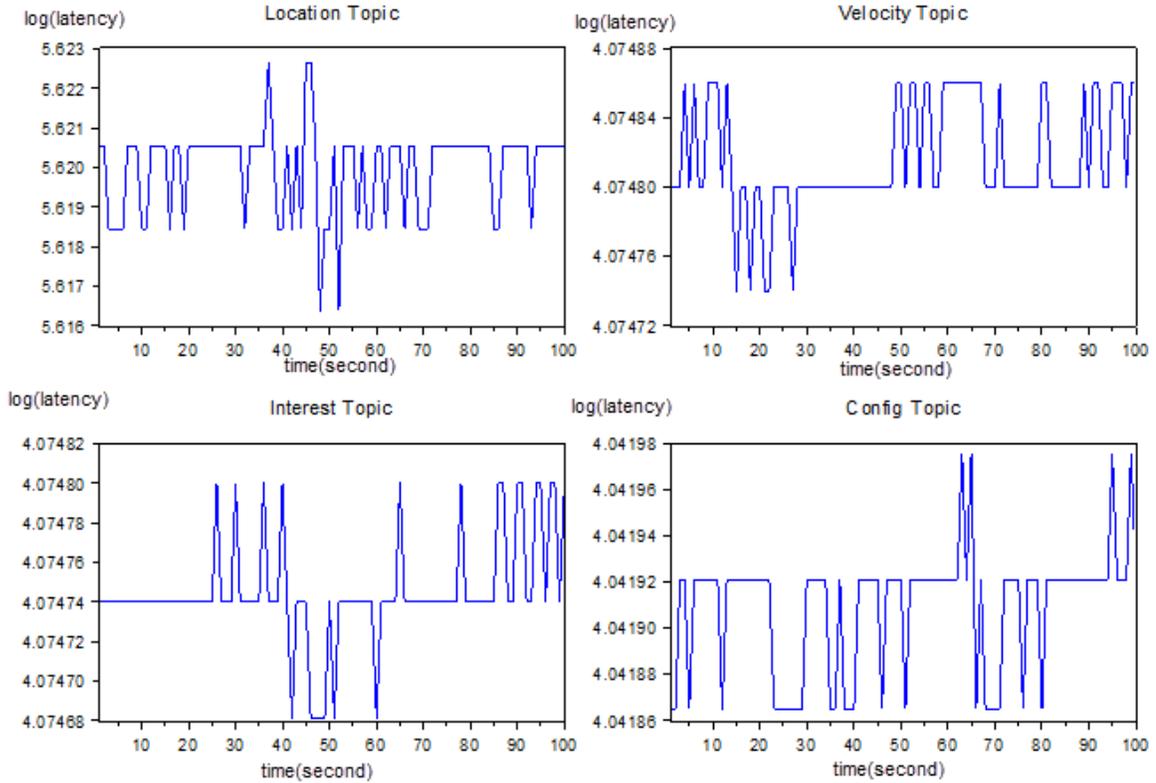


Figure 6-6: latency per each topic

As shown in Table 6-3, we have calculated the max latency achieved by all topics, it shows that IntrestTopic shows the max latency and it is 90 μ -sec. As it is the hot topic in the massively online gaming. The other topics show very similar low latencies. Figure 6-7.

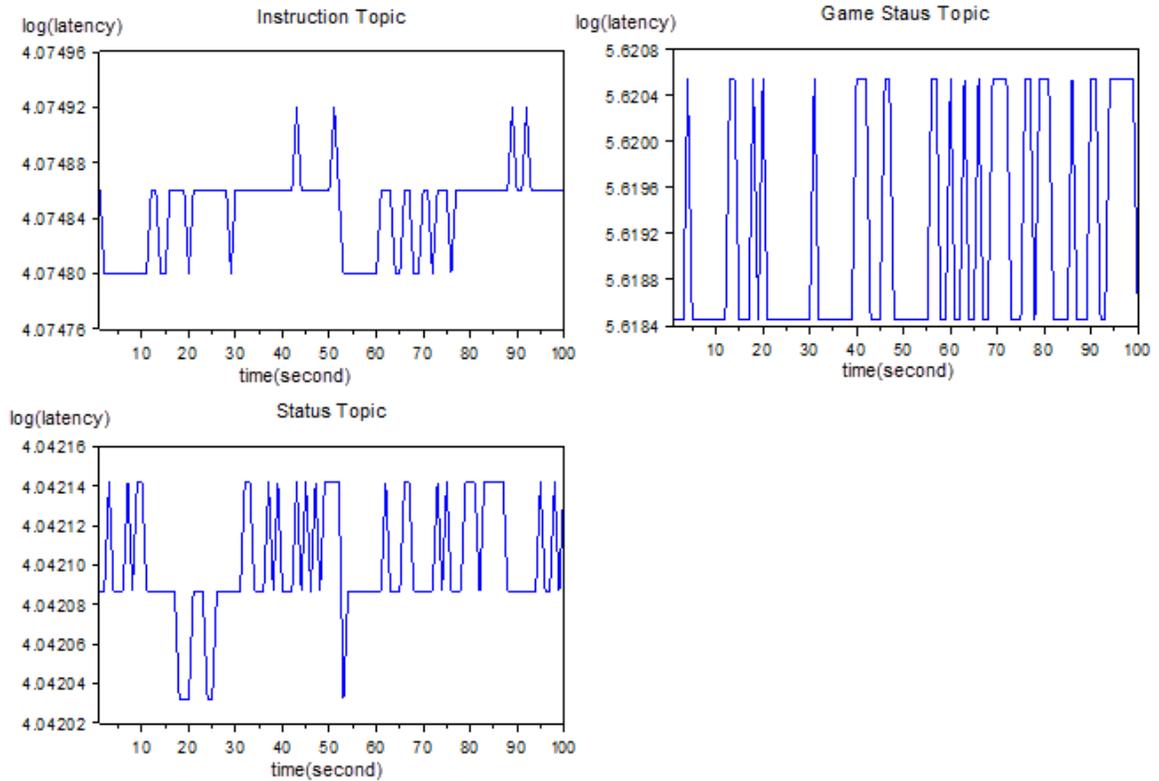


Figure 6-7: latency per each topic

We tried a couple of scenarios to assure that our PMS still afford a very high throughput with low latency.

In the other side of our work, the AIRTinDDS, this platform where it has, an AI module used to adapt and reconfigure the system from any unexpected behavior.

6.2 Experiment setup for AIRTinDDS

In this section, we discuss and compare the result of applying our suggested QoS in the DDS middleware (PMS) with before and after applying the AI algorithm. It shows that it gets the best effort of the network concerning its throughput, latency. In addition, it shows that after applying AI algorithm, we get a better performance in 12.11%.

As shown in Figure 6-8 (a), this is the throughput graph for unstable RTinDDS (we made our systems unstable by flood the network and by unexpected cut made on purpose) by applying different data load on the same game setup; this shows the throughput for the publisher/subscriber nodes on the network.

By applying different scenarios and different data load, the throughput keeps showing a very good result based on our network configuration.

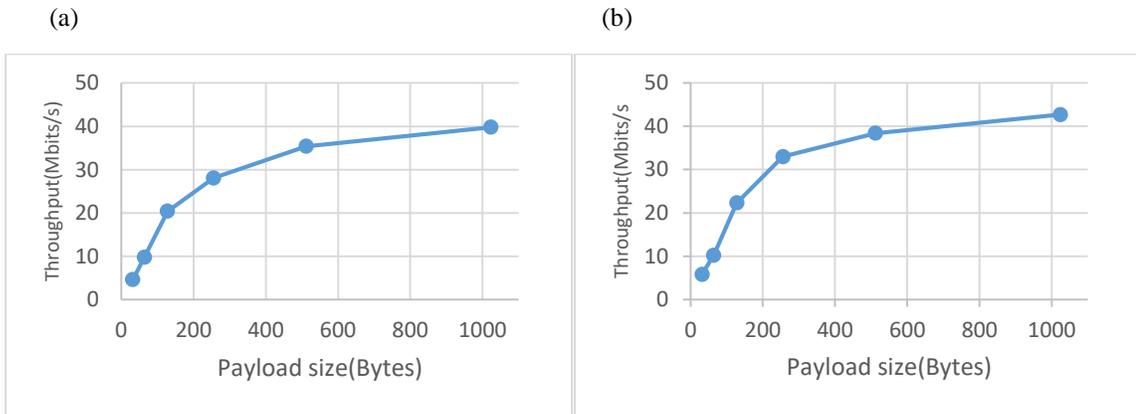


Figure 6-8: unstable RTinDDS (a) Vs AIRTinDDS (b) throughput

Figure 6-8 (b) shows the throughput for our network after applying the AIRTinDDS algorithm.

It is clear from the Figure 6-9 that the throughput is higher in AIRTinDDS than in RTinDDS (with unstable circumstances)

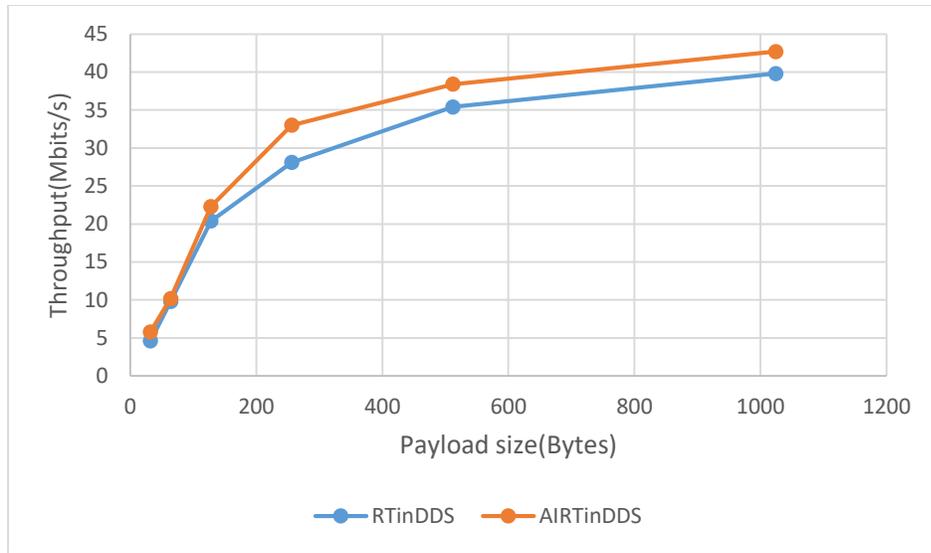


Figure 6-9: AIRTinDDS vs (unstable) RTinDDS

Figure 6-9 shows that after applying AIRTinDDS, the throughput is increased by 12.11%.

As we measured the throughput, we also measured the latency to see the different between unstable RTinDDS and AIRTinDDS. Figure 6-10.

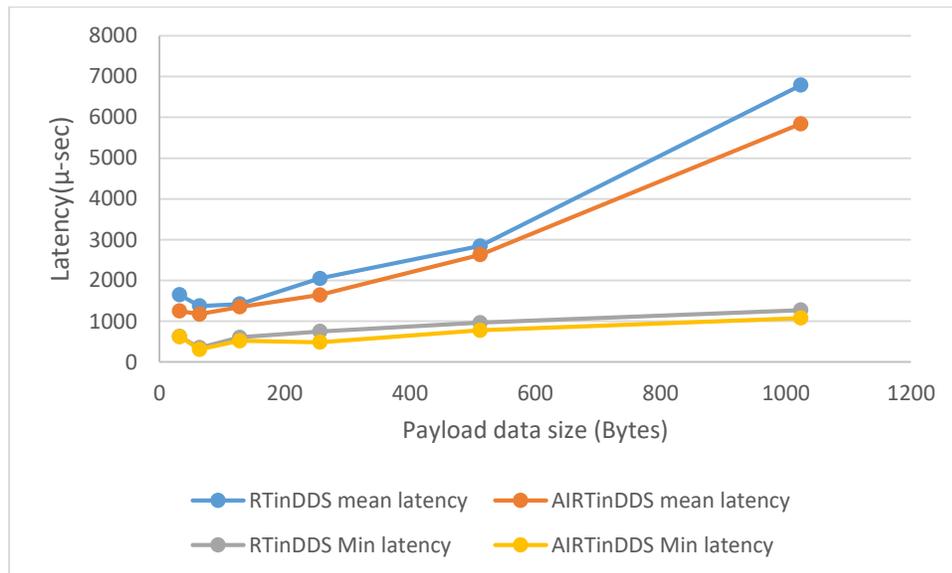


Figure 6-10: RTinDDS vs. AIRTinDDS latency

Figure 6-10, show that after applying AIRTinDDS, the latency improved by 14.19%.

6.3 Experiment scenarios:

We tried a couple of scenarios to assure that our PMS still afford a very high throughput with low latency.

6.3.1 Scenario 1:

Table 6-5: Configuration Scenario 1

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	2	2	1	2	0.5	0.5	1
Simulation Period	~4 hours						
Number of sent samples per second	~500 samples/sec						
Number of pub/sub per simulation period	50						
Topic Size	65KB						

In this scenario, we made changes across the topics frequencies, as we considered that some states or conditions we have a higher priority for some topic over the others. In addition to that, we played with the number of sent topics as we increased the number of pub/sub for this scenario. We calculated the average throughput by changing the data load each time; we ran the game many times and recorded the changing in the throughput while the data load changed. The results still show a very high throughput on the other side, a very low latency with zero loss packet. Figure **6-11**.

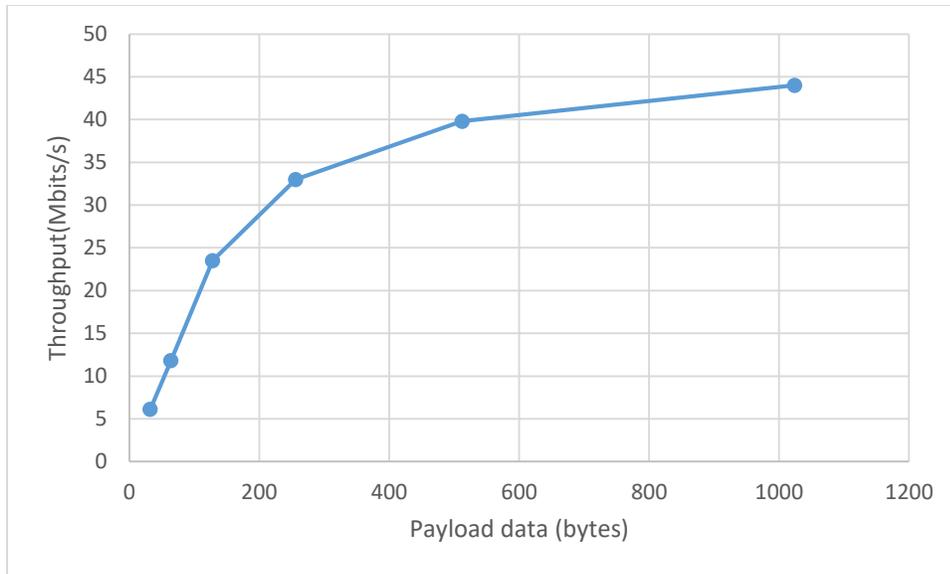


Figure 6-11: Throughput (scenario 1)

We measured the latency in this scenario; we found that the max latency and the min latency was as follows. Table 6-6

Table 6-6: latency scenario 1

max latency value	4724 μ -s
min latency value	138 μ -s

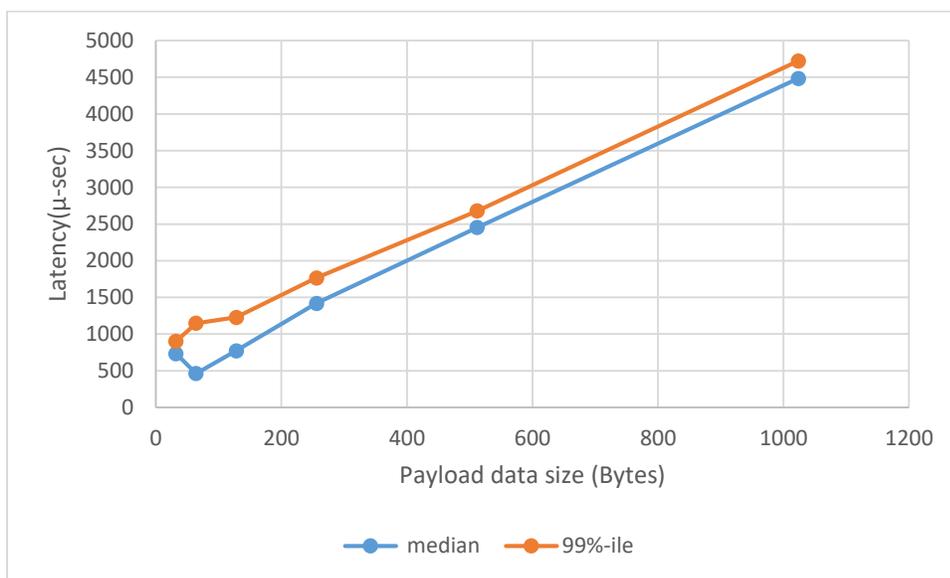


Figure 6-12: Latency (scenario 1)

6.3.2 Scenario 2:

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	1	1	1	1	1	1	1
Simulation Period	~4 hours						
Number of sent samples per second	~700 samples/sec						
Number of pub/sub per simulation period	100						
Topic size	65KB						

In this scenario, we made changes across the topics frequencies and the number of pub/sub per running period, as we considered in this case that all topics have the same priority, no one has a higher priority than other does. In addition to that, we played with the number of sent topics as we increased the number of pub/sub for this scenario. We measured the average throughput by changing the data load each time. The results still show a very high throughput, and on the other side, a very low latency with zero loss packet. Figure **6-13**, Figure **6-14**.

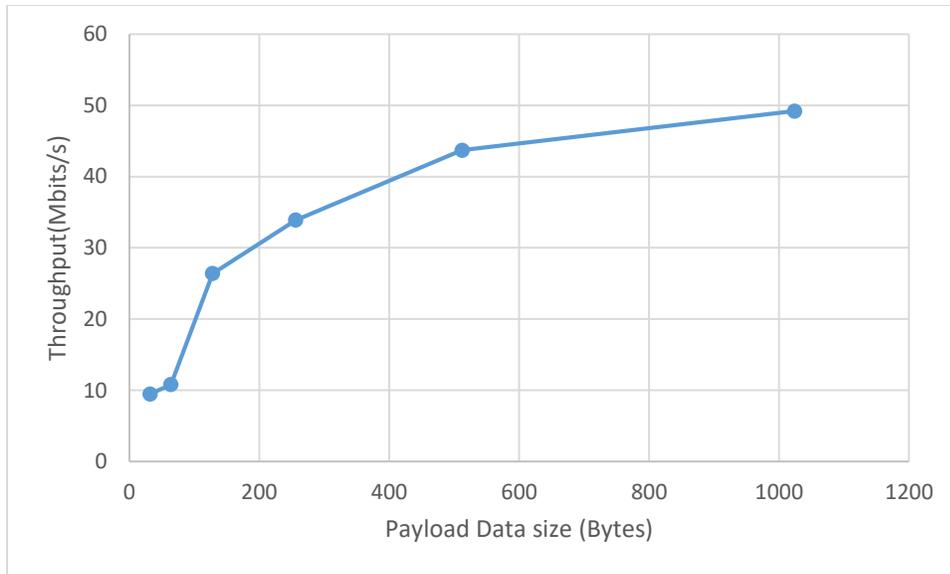


Figure 6-13: Throughput (scenario 2)

Table 6-7: latency scenario 2

max latency value	5249 μ -s
min latency value	284 μ -s

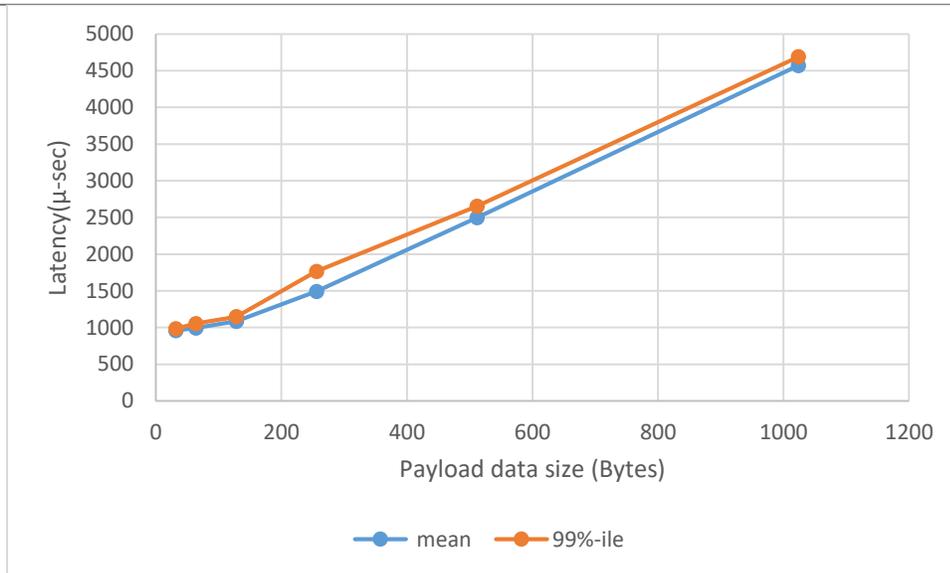


Figure 6-14: Latency (scenario 2)

6.3.3 Scenario 3:

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	3	3	1	2	1	1	2
Simulation Period	~4 hours						
Number of sent samples per second	~2000 samples/sec						
Number of pub/sub per simulation period	150						
Topic size	65KB						

In this scenario, we have increased the priority for location, velocity, and interest management topics, as we believe in this scenario; we have a very high frequency for the location and velocity as well as interest management. This kind of configuration fits those games that need to share the velocity and the location of the objects, such as race car, flight models. (Any race games).

In addition to that, we changed the number of sent topics as we increased the number of pub/sub for this scenario. We measured the average throughput by changing the data load each time. The results still show a very high throughput, and on the other side, a very low latency with zero loss packet. Figure 6-15, Figure 6-16.

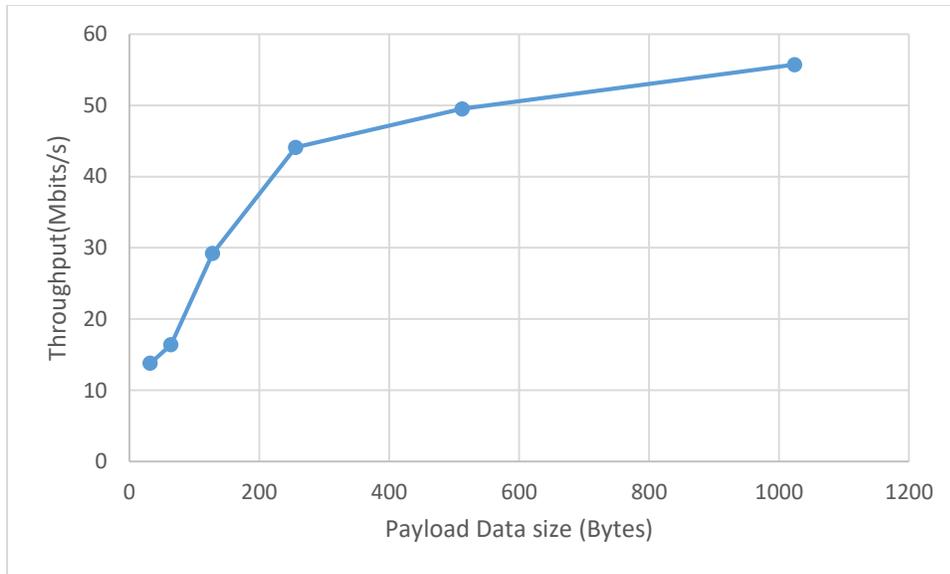


Figure 6-15: Throughput (scenario3)

Table 6-8: latency scenario 3

max latency value	5921 μ -s
min latency value	327 μ -s

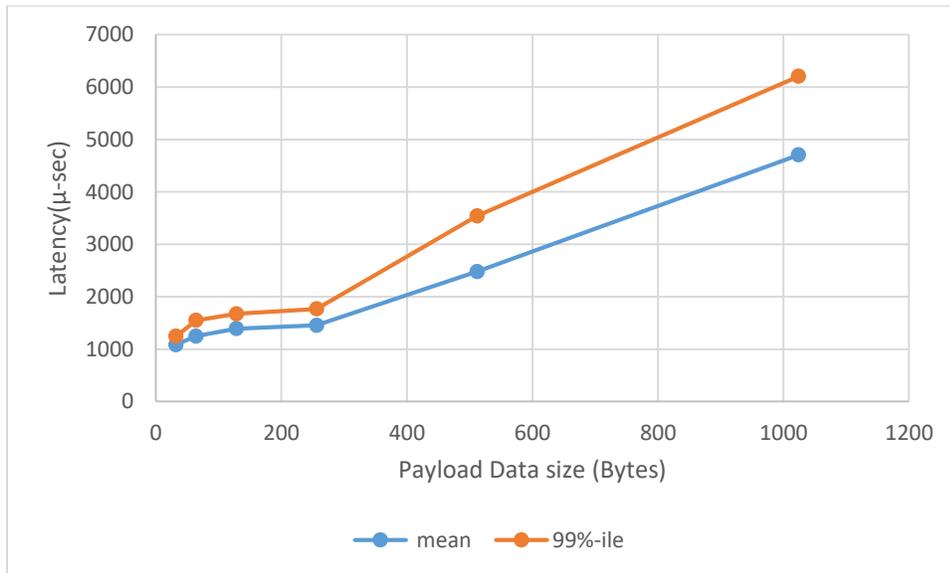


Figure 6-16: Latency (scenario 3)

6.3.4 Scenario 4:

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	3	1	1	6	1	1	1
Simulation Period	~4 hours						
Number of sent samples per second	~3000 samples/sec						
Number of pub/sub per simulation period	200						
Topic size	65KB						

In this scenario, we have increased the priority for the Interest management topic (IntT), as we considered this scenario to be as we have a very hot data (shared environment. Where the other topics still send their samples in low frequency. This kind of configuration fits those games that need to share specific environment (shapes, colors, existing objects, static objects...etc.).

In addition to that, we changed the number of sent topics as we increased the number of pub/sub for this scenario. Again, we measured the average throughput by changing the data load each time. The results still show a very high throughput, and on the other side, a very low latency with zero loss packet. Figure **6-17**, Figure **6-18**.

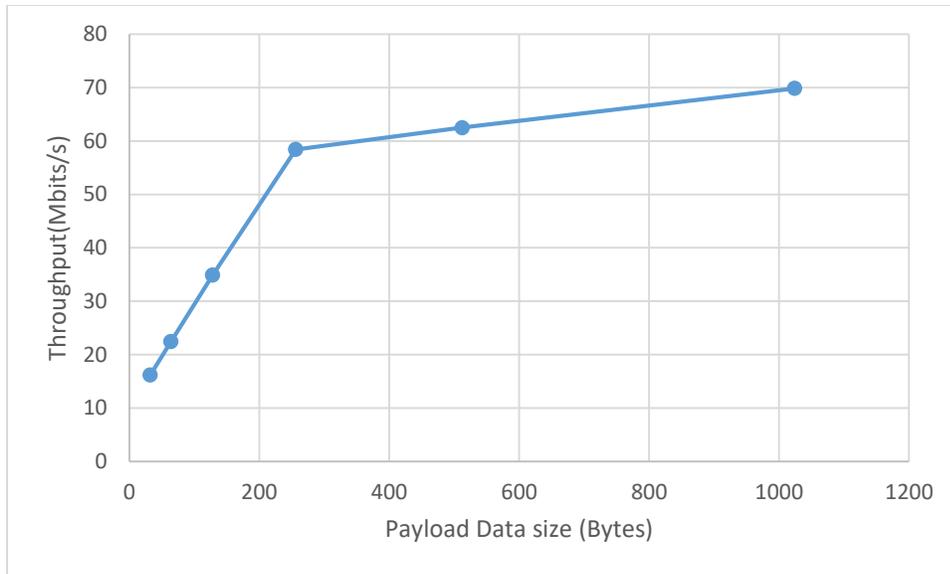


Figure 6-17: Throughput (scenario 4)

Table 6-9: latency scenario 4

max latency value	9428 μ -s
min latency value	4195 μ -s

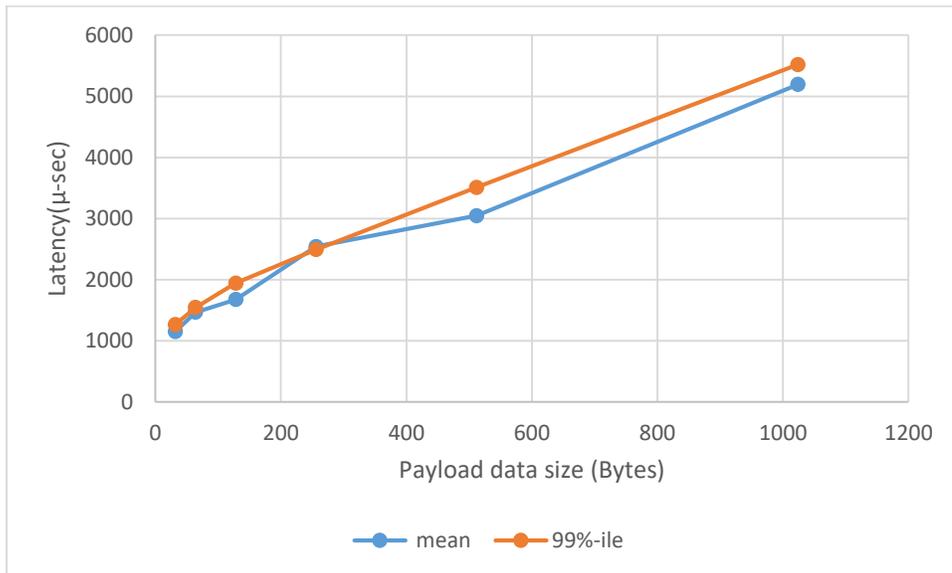


Figure 6-18: Latency (scenario 4)

All previous scenarios (scenario 1,2,3 and 4) have been done under the standard QoS we have chosen, we can conclude that in all scenarios, the throughput keeps showing a very good result, as it keeps growing without any packet loss. In very reasonable latency. Figure 6-19, Figure 6-20.

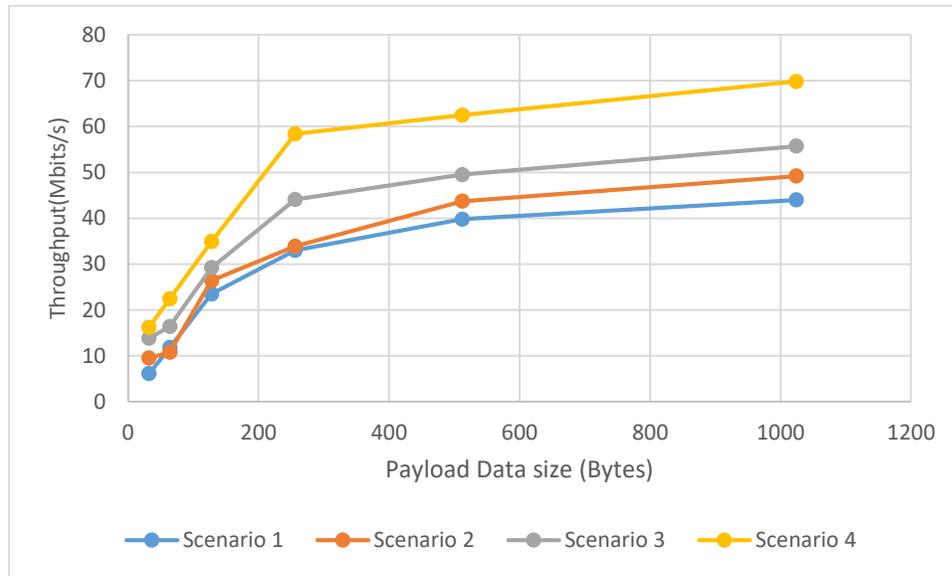


Figure 6-19: Throughput (All scenarios)

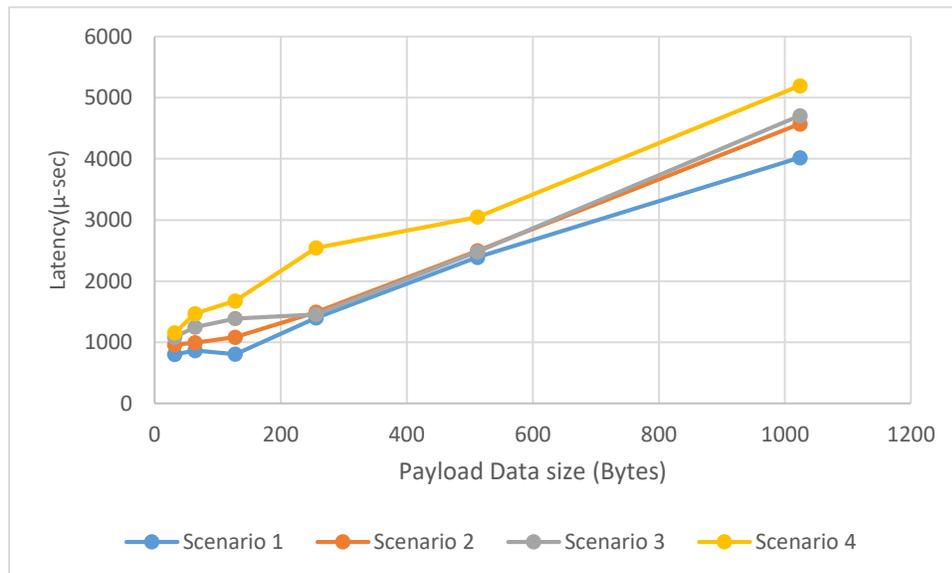


Figure 6-20: Latency (All scenarios)

The second part of our experimental work using different profile by repeating our scenarios with changes in QoS. We have changed some critical QoS and we observe the results.

6.4 Experiment QoS Profiles

Configuration Table **6-10** used for all profiles.

Table 6-10: configuration table

Topics	LT	VT	IT	IntT	CT	ST	GST
Frequencies	2	2	1	2	0.5	0.5	1
Simulation Period	~4 hours						
Number of sent samples per second	~500 samples/sec						
Number of players per simulation period	50						
Topic size	65KB						

6.4.1 Profile 1:

We fixed our configuration table 11 to be used for all profiles, the following is a set of QoS, and we are going to set for each profile with its suitable QoS values. Table **6-11**.

Table 6-11: profile 1 - QoS configuration

QoS	Possible values	Chosen value	What our chosen will affect
Durability	VOLATILE/ TRANSIENT_LOCAL/ PERSISTENT	VOLATILE	The game will only attempt to provide the data to the existing players. Any new joining player will start with the

			new coming data, there is nothing in the history for those players.
History	KEEP_LAST/ KEEP_ALL	KEEP_LAST "depth"=100	The game will only attempt to keep the most recent samples of each instance of data.
Deadline	Any suitable value	1 second	The player expects a new sample updating the value of each instance at least once every deadline period. The game indicates that the application commits to write a new value for each instance at least once every deadline period.
Latency-Budget	the maximum acceptable delay	400ms	Specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache and the receiving application is notified of the fact. This policy is a hint to the Service, not something that must be monitored or enforced. The Service is not required to track or alert the user of any violation. The default value of the <i>duration</i> is zero indicating that the delay should be minimized.
Reliability	RELIABLE/ BEST_EFFORT	BEST_EFFORT	It is acceptable to not retry sending any samples. Presumably, new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples.

Transport-Priority	We can set priorities for the hot data while we can reduce the priority rank for some usual data.	Transport-priority = 100 Varies between different players.	This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data
--------------------	---	---	---

We ran our experimental work with the above QoS configuration, we calculated the latency for each profile. Figure 6-21, Figure 6-22.

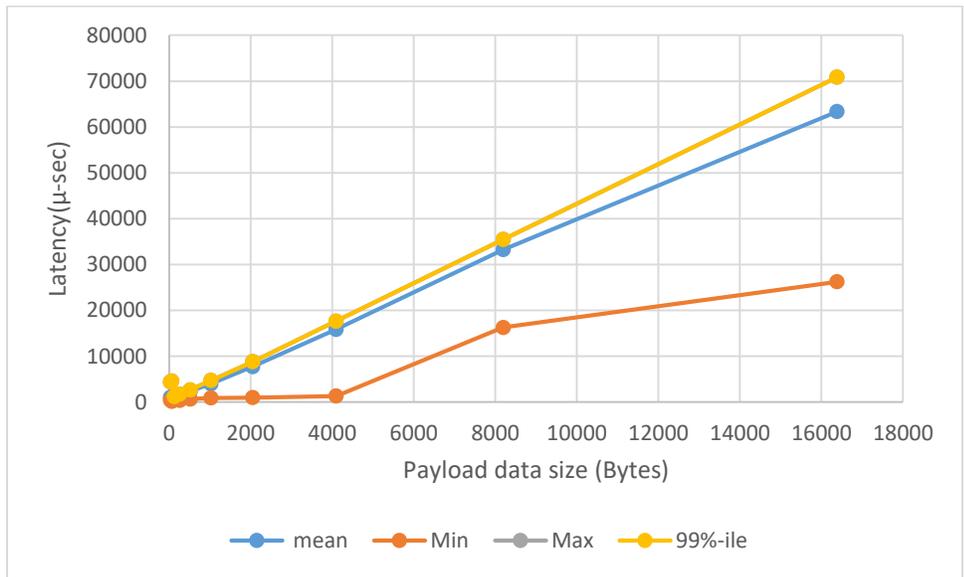


Figure 6-21: Latency - Profile 1

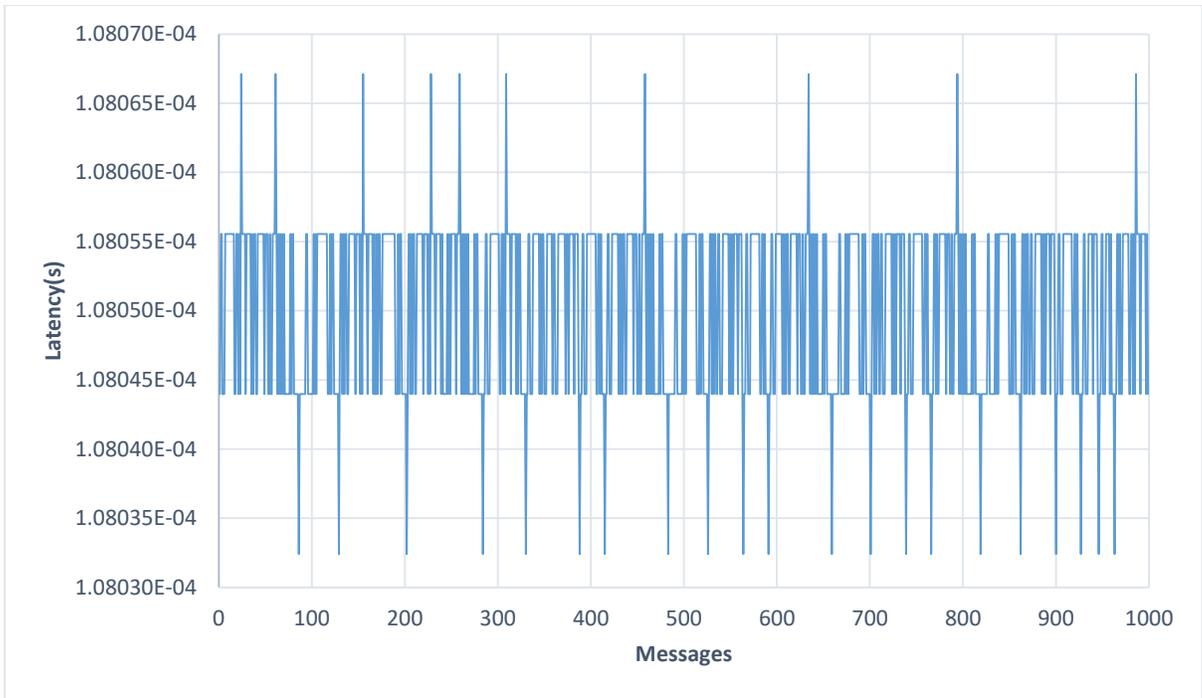


Figure 6-22: profile 1 – latency per message

6.4.2 Profile 2:

In this profile, we changed the durability, reliability, and history, as we believe profile one will behave better, as we set the reliability to be reliable and the history to be "keep_all".

Table 6-12.

Table 6-12: profile 2 - QoS configuration

QoS	Possible values	Chosen value	What our chosen will affect
Durability	VOLATILE/ TRANSIENT_LOCAL/ PERSISTENT	PERSISTENT	The game will keep the data on permanent storage so that new joining players can outlive a system session.

History	KEEP_LAST/ KEEP_ALL	KEEP_ALL	The game will attempt to keep all samples until they can be delivered to the end players.
Deadline	Any suitable value	1	The player expects a new sample updating the value of each instance at least once every deadline period. The game indicates that the application commits to write a new value for each instance at least once every deadline period.
Latency-Budget	the maximum acceptable delay	400ms	Specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache and the receiving application is notified of the fact. This policy is a hint to the Service, not something that must be monitored or enforced. The Service is not required to track or alert the user of any violation. The default value of the <i>duration</i> is zero indicating that the delay should be minimized.
Reliability	RELIABLE/ BEST_EFFORT	RELIABLE	The game will attempt to deliver all samples in its history. Missed samples may be retried.
Transport-Priority	We can set priorities for the hot data while we can reduce the priority rank for some usual data.	Transport- priority = 100 Varies between different players.	This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data. The default

			value of the <i>transport_priority</i> is zero.
--	--	--	---

Figure 6-23, Figure 6-24 shows the latency.

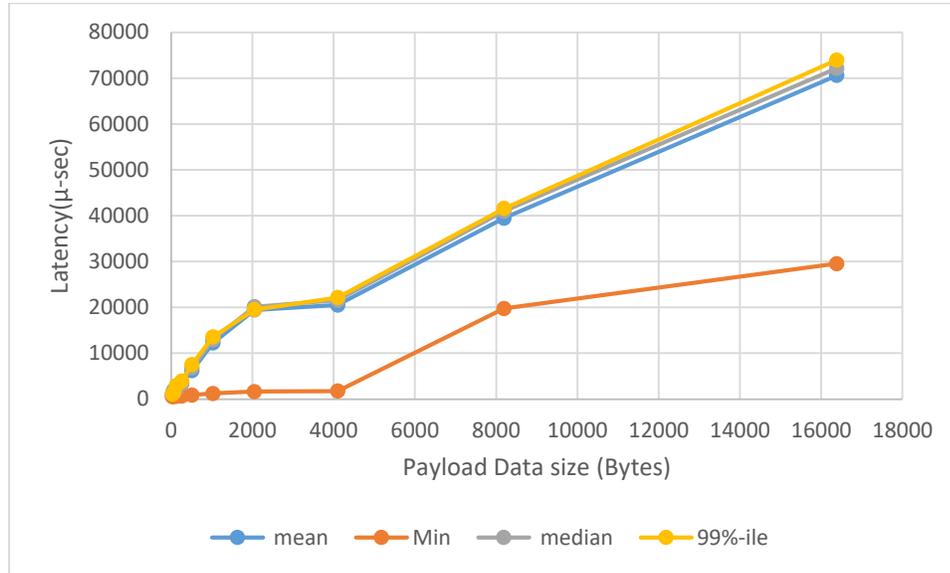


Figure 6-23: Latency (Profile 2)

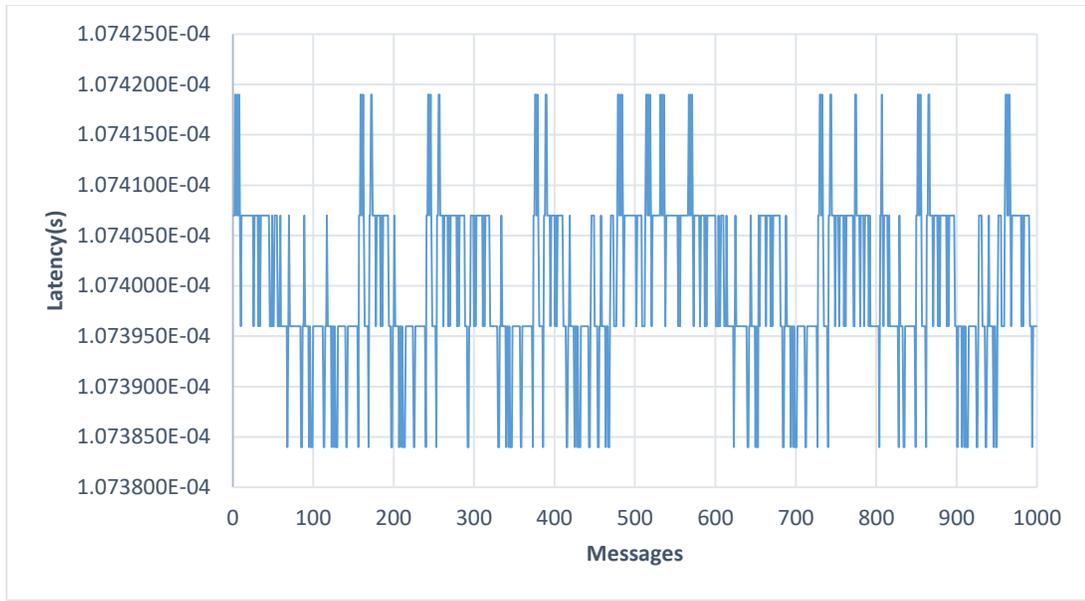


Figure 6-24: profile 2 – latency per message

6.4.3 Profile 3:

In this profile, we set the history to be "keep_all", the durability to be persistent and we set the reliability to be persistent. In this profile we show how QoS should be consistent with each other to give us a reasonable result as here we set the durability to be persistent which means it will keep all data on a permanent storage and the history is keep all which consist with each other while in the other hand, we set the reliability to be best_effort with does not. Table 6-13.

Table 6-13: profile 3 - QoS configuration

QoS	Possible values	Chosen value	What our chosen will affect
Durability	VOLATILE/ TRANSIENT_LOCAL/	PERSISTENT	The game will keep the data on permanent storage so that new

	PERSISTENT		joining players can outlive a system session.
History	KEEP_LAST/ KEEP_ALL	KEEP_ALL	The publisher will attempt to keep all samples until it can be delivered to the end players.
Deadline	Any suitable value	1	The player expects a new sample updating the value of each instance at least once every deadline period. The game indicates that the application commits to write a new value for each instance at least once every deadline period.
Latency-Budget	the maximum acceptable delay	400ms	Specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache and the receiving application is notified of the fact. This policy is a hint to the Service, not something that must be monitored or enforced. The Service is not required to track or alert the user of any violation. The default value of the <i>duration</i> is zero indicating that the delay should be minimized.
Reliability	RELIABLE/ BEST_EFFORT	BEST_EFFORT	it is acceptable to not retry sending any samples. Presumably, new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples.

Transport-Priority	We can set priorities for the hot data while we can reduce the priority rank for some usual data.	Transport-priority = 100 Varies between different players.	This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data. The default value of the <i>transport_priority</i> is zero.
---------------------------	---	---	--

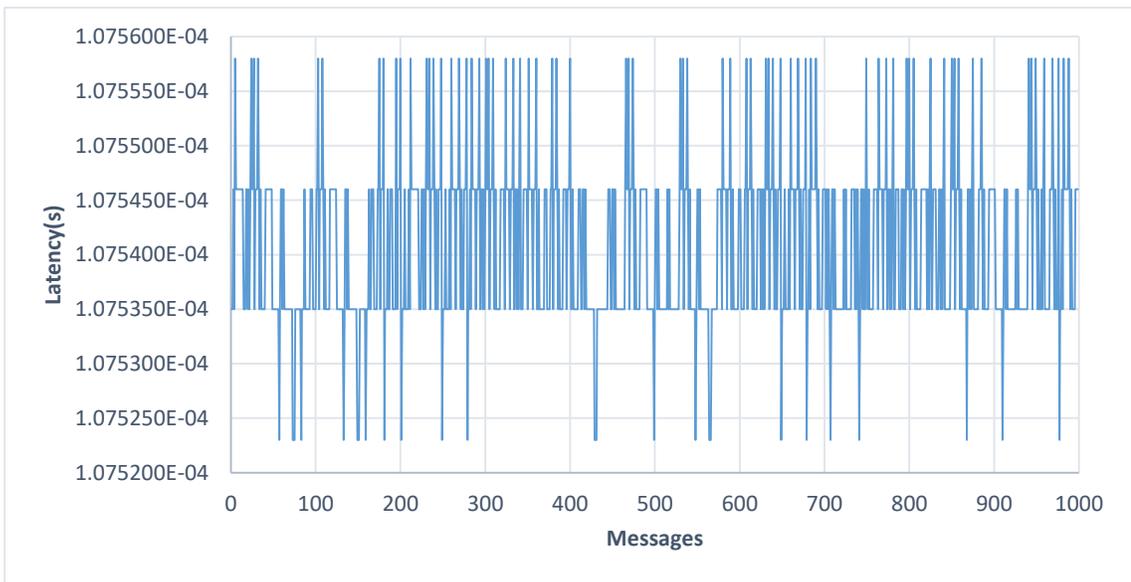


Figure 6-25: profile 3 – latency

6.4.4 Profile 4:

In this profile, we changed the deadline to be 0.5 seconds. We also did changes across the reliability, durability, and the history. In this profile, we prove that the durability should be consistent with the history as here we set the durability to be *transient_local* with *keep all* history which we can notice that the latency is so high which indicates that these sets do not consist. Also, we set the reliability to be *best_effort* while the history is *keep_all* which makes it also does not make much sense. Table **6-14**.

Table 6-14: profile 4 - QoS configuration

QoS	Possible values	Chosen value	What our chosen will affect
Durability	VOLATILE/ TRANSIENT_LOCAL/ PERSISTENT	TRANSIENT_LOCAL	The game will attempt to keep some samples so that they can be delivered to any late-joining players.
History	KEEP_LAST/ KEEP_ALL	KEEP_ALL	The publisher will attempt to keep all samples until it can be delivered to the end players.
Deadline	Any suitable value	0.5	The player expects a new sample updating the value of each instance at least once every deadline period. The game indicates that the application commits to write a new value for each instance at least once every deadline period.
Latency-Budget	the maximum acceptable delay	400ms	Specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache and the receiving application is notified of the fact. This policy is a hint to the Service, not something that must be monitored or enforced. The Service is not required to track

			or alert the user of any violation. The default value of the <i>duration</i> is zero indicating that the delay should be minimized.
Reliability	RELIABLE/ BEST_EFFORT	BEST_EFFORT	
Transport-Priority	We can set priorities for the hot data while we can reduce the priority rank for some usual data.	Transport-priority = 100 Varies between different players.	This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data. The default value of the <i>transport_priority</i> is zero.

Figure 33. Shows the latency for this profile.

QoS	Possible values	Chosen value	What our chosen will affect
Durability	VOLATILE/ TRANSIENT_LOCAL/ PERSISTENT	VOLATILE	The game will only attempt to provide the data to existing players.
History	KEEP_LAST/ KEEP_ALL	KEEP_ALL	The publisher will attempt to keep all samples until it can be delivered to the end players.
Deadline	Any suitable value	10 seconds	The player expects a new sample updating the value of each instance at least once every deadline period. The game indicates that the application commits to write a new value for each instance at

			least once every deadline period.
Latency-Budget	the maximum acceptable delay	400ms	Specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache and the receiving application is notified of the fact. This policy is a hint to the Service, not something that must be monitored or enforced. The Service is not required to track or alert the user of any violation. The default value of the <i>duration</i> is zero indicating that the delay should be minimized.
Reliability	RELIABLE/ BEST_EFFORT	BEST_EFFORT	
Transport-Priority	We can set priorities for the hot data while we can reduce the priority rank for some usual data.	Transport-priority = 100 Varies between different players.	This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data. The default value of the <i>transport_priority</i> is zero.

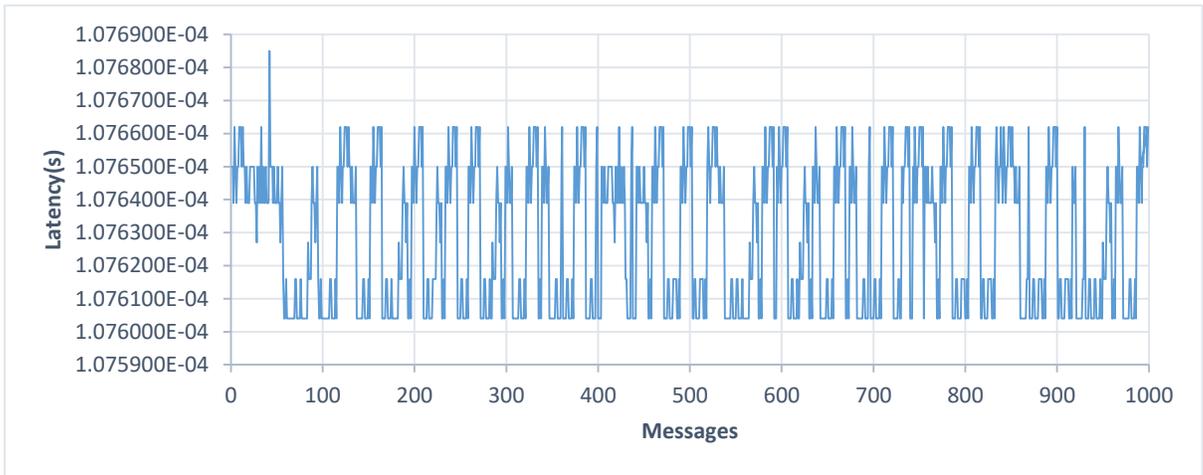


Figure 6-26: profile 4 – latency

6.4.5 Profile 5

The main changes in this profile are the deadline and we set the durability to be volatile with keep all history, we can notice the difference between this profile and profile 1. Table 6-15.

Table 6-15: profile 5 - QoS configuration

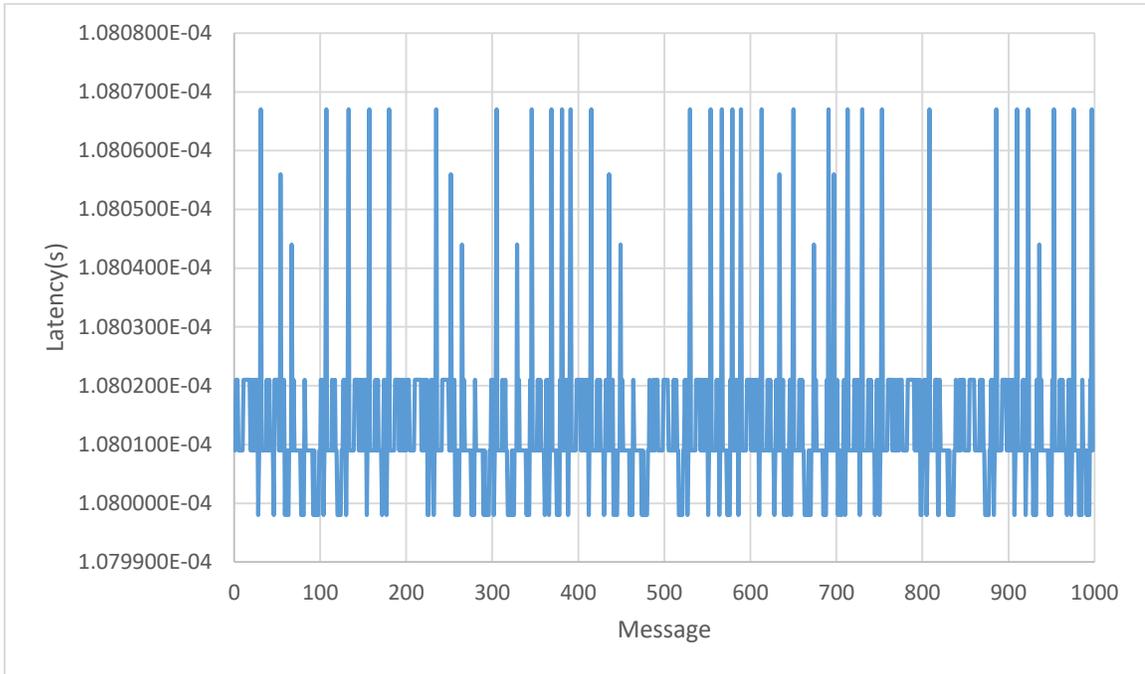


Figure 6-27: Latency - profile 5

6.4.6 Profiles discussion:

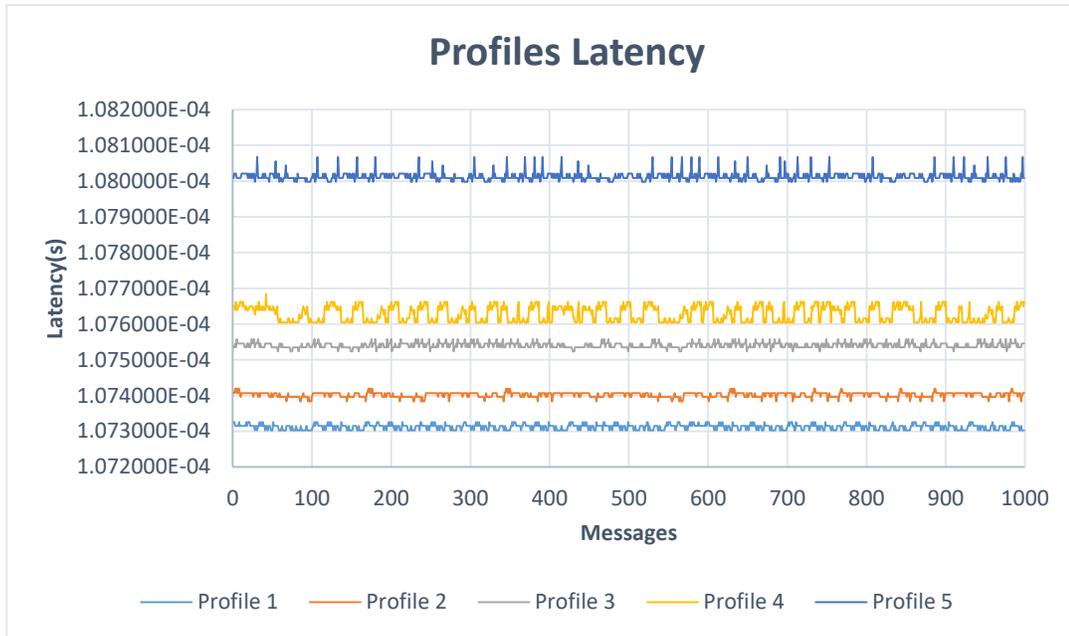


Figure 6-28: Profiles latency

As we can have noticed from the figure above that the profile one is having the lowest latency, as it also makes much sense because we set the reliability to be 'best effort' and the history is 'keep last' with deadline equals to 1 second and with volatile durability. With this combination, we can get the lowest latency. We decided to set our PMS to be as the profile 1 in its default case. Figure 6-29 also shows the mean values and the min/max values for the latency for all profiles.

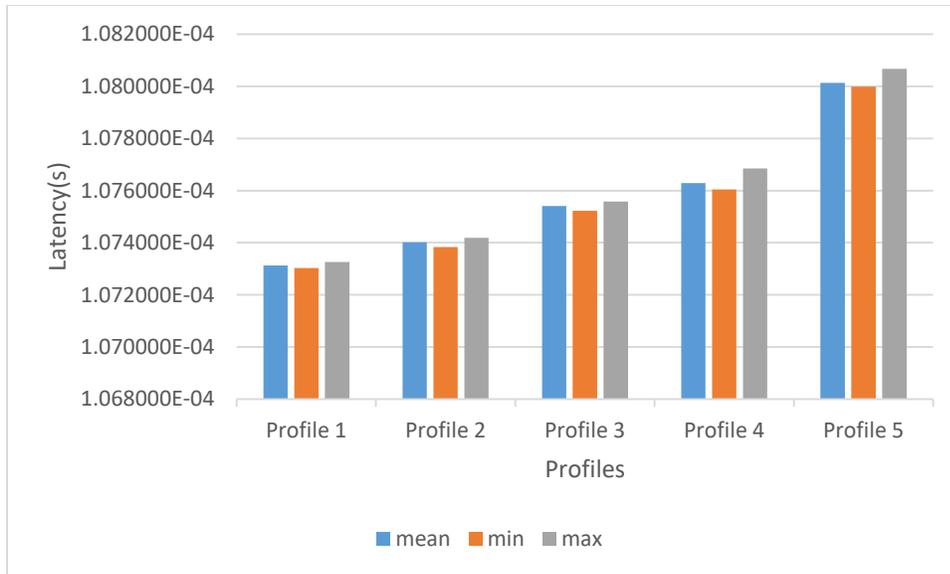


Figure 6-29: mean, min, and max/profiles

6.5 Comparison between RTinDDS and other Platforms

Latency dependencies have been a subject of research for all interactive networked multimedia applications since their inception. Acceptable latency values for MMORPGs appear to be game-specific

This section compares between RTinDDS and other platforms to prove that RTinDDS can fulfill the requirement of an MMOG.

- CloudCraft: Cloud-based Data Management

CloudCraft presents the experimental results with a single game server in [33]. When the client number is not more than 500, the response time for each read/write command is under 15ms. That means, 500 concurrent clients put a little pressure on the game server as well as the 5-node Cassandra cluster. However, when the client number is up to 600, the game server throws many “time-out” exceptions, which block the acceptance of subsequent commands. So the maximum number of concurrent clients in the case of single game server is around 500. Each client randomly sends 500 read/write commands.

In our case, the player number is around 200 players, the response time is less than 11ms. Since we have seven different topics, we have around 2000 read/write commands.

As we have different profiles with different scenarios, all profile/scenarios have latency lower than 11ms Figure **6-28**. We can tell that RTinDDS fulfills the requirement with compared to CloudCraft MMOG.

- CloudFog: Cloud-based gaming

The general response latency requirement is 100ms [20], 20ms is attributed to play out and processing delay and 80ms is the network latency. A user is covered by a data center or a supernode if the response latency is no more than the latency requirement of the user's game.

We showed that one of the highest case scenario Figure 6-30, we got latency less than 70ms.

- **World of Warcraft based on TCP**

They evaluated the TCP delay based on packet dumps from a Wireshark client running on the WoW client. The RTT is the time a message needs to travel to a remote host and back again. This time is important to data systems which adapt their throughput rate based on the delay. In TCP the RTT is measured between segment transmission and ACK receipt. It is then estimated using Karn's Algorithm, TCP timestamps or Jacobson's algorithm. In their work, they estimated the RTT according to the Karn's algorithm. The delay is the RTT divided by two, 40ms of delay correspond to 80ms of RTT.[34].

We showed that one of the highest case scenario Figure 6-31, we got latency less than 70ms.

- **Other Games**

Latency dependencies have been a subject of research for all interactive networked multimedia applications since their inception. Acceptable latency values for MMORPGs appear to be game-specific but within the certain bounds. Fritsch, it's been determined that

test game Everquest2 runs smoothly with latencies of up to 1250ms. Dick, Wellnitz, and Wolf performed a player survey and analysis of how latency affects player performance in different games, and among them a single RPG Diablo 2 which showed that players think of latencies around 80ms as the optimal, and those around 120ms as the maximum tolerable value.[33].

6.5.1 Conclusion

We have different profiles with different scenarios, all profile/scenarios have latency lower than 70ms Figure 6-32, Figure **6-28**. We can tell that RTinDDS fulfills the requirement compared with some of the well-known MMOGs.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

We are presenting a new platform solution to deal with the real-time online gaming using the DDS middleware. RTinDDS and AIRTinDDS are real-time platform based on the DDS middleware guarantees to provide the developers, designers, and end users a suitable platform to build their online real-time gaming and to deliver it in a robust manner for the end users. The platform also able to allow designers to make the best decision for a specific situation. RTinDDS is a real-time platform that has been implemented by choosing the best QoS in the DDS middleware for the default situation (stable situation) to deliver the best performance. We have implemented a game we call it Plane Model Simulation (PMS). After running our simulation scenario. We have chosen the best QoS and we applied it on the PMS.

PMS shows a very robust real-time platform, the simulation results show that RTinDDS in PMS improved the reliability, scalability, throughput, and less packet loss.

The second part of the work is by making RTinDDS is smart enough to adapt itself when any issue occurred at runtime; we have implemented a new platform called AIRTinDDS, this platform can adapt and correct itself once an issue occurred. This is done by choosing the best QoS in that situation using artificial intelligent (AI). Sometimes, it is needed to change some QoS at runtime to accommodate the new unexpected changes. We have

applied the AIRTinDDS algorithm in the RTI Connex simulation; it shows a very robust and adaptive platform

This thesis addressed a performance evaluation for DDS QoS by carrying out an experimental and simulation work.

We calculated the average throughput by changing the data load each time, we ran the game many times and recorded the changing in the number of bytes per second while it is changed as the data load changed. The results show a very high throughput while a very low latency. In addition to that, we have implemented a simulation where we wanted to prove the ability of our proposed platform to scale the number of nodes. We set the QoS to be exactly the same as in our PMS game, we were able to scale the number of nodes as there was a 10000 publish/subscribe players playing at the same time on different platforms, the results show also a very high throughput with very low latency.

The other part of this thesis is AIRTinDDS, our algorithm is smart enough to detect and adapt any issue related to our assigned QoS. The results show a very high throughput while a very low latency. In addition to that, we have implemented a simulation where we wanted to prove the ability of our proposed platform to scale the number of nodes. We set the QoS to be exactly the same as in our PMS game, we were able to scale the number of nodes as there was a 10000 publish/subscribe players playing at the same time on different platforms, the results show also a very high throughput with very low latency.

7.2 Future work

Our work can be extended to cover the following:

- Build a web-based application for PMS and apply both algorithms, RTinDDS and AIRTinDDS where you can test experimentally the scalable number of players.
- Build a 3D game and focus on the Interest management issue, where the environment itself could be shared amongst all users which increased the load on the network.
- Propose more special profiles and scenarios where it fits the previous point. So you will be able to manage any unexpected fault.

References

- [1] F. Arslan, “Towards Service Oriented Architecture (SOA) for Massive Multiplayer Online Games (MMOG),” 2012.
- [2] B. Master, “Interest Management for Massively Multiplayer Games,” no. August, 2006.
- [3] D. Lee, “ATLAS : A Scalable Network Framework for Distributed,” vol. 16, no. 2, pp. 125–156, 2007.
- [4] J. Gascon-samson and B. Kemme, “Lamoth : A Message Dissemination Middleware for MMOGs in the Cloud,” no. 978, pp. 1–2, 2013.
- [5] F. Glinka, A. Ploß, J. Müller-iden, and S. Gorchach, “RTF : A Real-Time Framework for Developing Scalable Multiplayer Online Games,” pp. 81–86.
- [6] M. Multiplayer, “Massively Multiplayer,” no. February, 2004.
- [7] A. Denault and J. Kienzle, “Journey : A Massively Multiplayer Online Game Middleware,” pp. 38–44.
- [8] B. Madani, “7-Distributed systems and Middleware.” p. 81, 2014.
- [9] O. M. Group, “The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification,” no. September, 2014.
- [10] C. D. Nguyen, F. Safaei, and P. Boustead, “A DISTRIBUTED SERVER ARCHITECTURE FOR PROVIDING IMMERSIVE AUDIO COMMUNICATION TO MASSIVELY MULTI-PLAYER ONLINE GAMES,” pp. 170–176, 2004.
- [11] B. Anand, K. Thirugnanam, L. T. Long, and D. Pham, “ARIVU : Power-Aware Middleware for Multiplayer Mobile Games.”
- [12] C. Mcknight, “Multi-Server MMO Middleware : Unlocked,” 2012.

- [13] D. I. Terra, “Practical Middleware for Massively Multiplayer,” no. October, pp. 47–54, 2005.
- [14] A. P. Negrão, M. Adaixo, L. Veiga, and P. Ferreira, “On demand Resource Allocation Middleware for Massively Multiplayer Online Games,” pp. 71–74, 2014.
- [15] E. S. Liu, M. K. Yip, and G. Yu, “Lucid Platform : Applying HLA DDM to Multiplayer Online Game Middleware,” vol. 4, no. 4, pp. 1–12, 2006.
- [16] A. Bharambe and J. Pang, “Colyseus : A Distributed Architecture for Online Multiplayer Games.”
- [17] K. Chen, P. Huang, and C. Lei, “Effect of Network Quality on Player Departure Behavior in Online Games,” vol. 20, no. 5, pp. 593–606, 2009.
- [18] D. A. Croucher and H. A. Engelbrecht, “A Peer-to-Peer Middleware Plugin for an Existing MMOG Client-Server Architecture,” 2012.
- [19] X. Tu, L. Wu, L. Zheng, H. Jin, and W. Jiang, “Towards Scalable Middleware for Multiplayer Online Games.”
- [20] Y. Lin, H. Shen, and S. Member, “CloudFog : Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service,” vol. 13, no. 9, 2016.
- [21] C. J. Carter, A. El Rhalibi, and M. Merabti, “A novel scalable hybrid architecture for mmog.”
- [22] E. Carlini and L. Ricci, “Reducing Server Load in MMOG via P2P Gossip,” pp. 12–13, 2012.
- [23] C. Weng and K. Wang, “Dynamic Resource Allocation for MMOGs in Cloud Computing Environments,” pp. 142–146, 2012.
- [24] K. Prasetya, “Artificial Neural Network for Bot Detection System in MMOGs,” pp. 2–3.

- [25] O. B. Gaspareto, D. A. C. Barone, and A. M. Schneider, "Neural Networks Applied to Speed Cheating Detection in Online Computer Games," 2008 Fourth Int. Conf. Nat. Comput., vol. 4, pp. 526–529, 2008.
- [26] V. Nae, A. Iosup, and R. Prodan, "Dynamic Resource Provisioning in Massively Multiplayer Online Games," IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 3, pp. 380–395, 2011.
- [27] V. Nae, R. Prodan, and T. Fahringer, "Neural Network-Based Load Prediction for Highly Dynamic Distributed Online Games," pp. 202–211, 2008.
- [28] S. Gorlatch, F. Glinka, A. Ploss, J. Müller-Iden, R. Prodan, V. Nae, and T. Fahringer, "Enhancing Grids for Massively Multiplayer Online Computer Games," Euro-Par 2008–Parallel ..., pp. 466–477, 2008.
- [29] X.-B. Shi, X. Wang, J. Bi, F. Liu, D. Yang, and X.-Y. Liu, "A DR algorithm based on artificial potential field method," Multimed. Tools Appl., vol. 45, no. 1–3, pp. 247–261, 2009.
- [30] E. Bednar and J. O. Miller, "Player vs . Bot Traffic Analysis Using Artificial Neural Networks," 2010.
- [31] O. M. Group, "Data Distribution Service (DDS)," no. April, 2015.
- [32] C. J. M. John, "The Near Term," Netw. Syst. Support Games, vol. 1, pp. 0–5, 2011.
- [33] Z. Diao, S. Wang, E. Schallehn, and G. Saake, "CloudCraft : Cloud-based Data Management for MMORPGs," vol. 2013, 2014.
- [34] M. Ries, P. Svoboda, and M. Rupp, "Empirical Study of Subjective Quality for Massive Multiplayer Games," Syst. Signals Image Process. 2008. IWSSIP 2008. 15th Int. Conf., vol. 2008, 2008.

Vitae

Name :BASHAR MOHAMMAD MOUSA KHATIB |
Nationality :JORDAN |
Date of Birth :1/3/1988|
Email :ENG.BASHARKH@HOTMAIL.COM|
Address :PALESTINE|
Academic Background :COMPUTER ENGINEERING

An-Najah National University, Nablus- Palestine 2006 – 2011

Bachelor Computer engineering:

Practical Experience:

1. COMPUTER ENGINEER – TEAM LEADER INFINITE TIERS CO.

RESPONSIBILITIES:

Leading a team consisting of 10 computer engineers where I delegate to them the tasks after coordination with the clients and the management.

- ✓ WebSphere Commerce administration and development tasks.
- ✓ Supporting and monitoring web and application servers.
- ✓ Handling the development of websites over J2EE/ WCS.
- ✓ WebSphere commerce frontend support GWT/jQuery/HTML/CSS
- ✓ Managing, developing and supporting for WebSphere commerce sites such as: www.lee.com , www.vans.com , www.thenorthface.com .
- ✓ Developing WCS websites over tablets and smartphones.
- ✓ Developing web applications based on the Model View Controller (MVC) struts framework.

- ✓ Web supporter and handling clients technical issues (Bug fixes) using Jira and remedy systems.
- ✓ Meeting with clients and provide them with professional consultancy and suggestions.
- ✓ Provide an estimation of the time period will be taken to deliver the projects requested.
- ✓ Daily scrum calls with the clients and provide reports on the work progress plus discussions.
- ✓ On-call duties during weekends and holidays. (PagerDuty)

2. Software Engineer – Buytech Co. 2009 - 2011

- ✓ Working as Software Engineer and Database structure.
- ✓ Frontend and backend
- ✓ jQuery ,HTML5 and CSS3 (front end)
- ✓ Developing and supporting php web sites.
- ✓ Receive client's requirements.
- ✓ Coordinator among the teams on the technical tasks.

Other Activities:

- ✓ Giving Security + course for Saudi Electrical engineers.
- ✓ Giving courses and Program for one month for talented students at SABIC in Saudi arabia
- ✓ Giving PHP and MySql, Java courses at the Gate IT, SHARP centers. **(trainer)**
- ✓ **vice-chairman** of Palestinian Engineering Friends Association. |