

**PERFORMANCE ANALYSIS OF REAL-TIME
PUBLISH-SUBSCRIBE (RTPS) MIDDLEWARE FOR
SENSOR NETWORKS**

BY

ISMAIL MOHAMED H. KESHTA

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

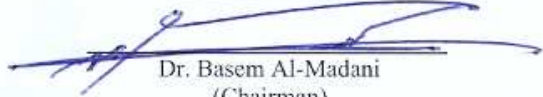
JUNE 2011

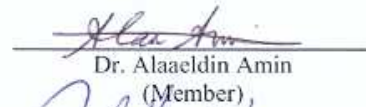
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

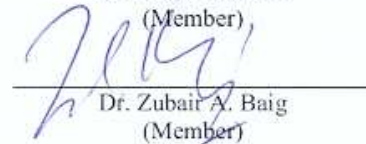
DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Ismail Mohamed Hemdan Keshta** under the supervision of his thesis advisors and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**


Thesis Committee


Dr. Basem Al-Madani
(Chairman)


Dr. Alaaeldin Amin
(Member)


Dr. Zubair A. Baig
(Member)


Dr. Basem Al-Madani
Department Chairman


Dr. Salam A. Zummo
Dean of Graduate Studies



14/6/14

Date

DEDICATION

In the Name of Allah, the Most Gracious, the Most Merciful.

To

My parents, who opened the way for me to success

ACKNOWLEDGEMENT

All praise be to Allah, Subhanahu-wa-Ta'ala, for his limitless blessing and guidance. May Allah bestow peace on his prophet, Muhammad (Peace and blessing of Allah be upon him) and his family.

All my appreciation and thanks to my thesis advisor, Dr. Basem S. Al-Madani, for his guidance and help all the way till the completion of this thesis.

I would like also to thank my thesis committee members, Dr. Alaaeldin Amin and Dr. Zubair Baig for their cooperation and constructive comments.

Last, but not least, thanks to all my colleagues and friends, who encouraged me a lot on my way to achievement of this work.

CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENT	iv
CONTENTS	iv
LIST OF TABLES	ix
LIST OF FIGURES	x
THESIS ABSTRACT	xii
خلاصة الرسالة.....	xiii
CHAPTER 1. INTRODUCTION.....	1
1.1. PROBLEM STATEMENT	3
1.2. THESIS ORGANIZATION.....	4
CHAPTER 2. LITERATURE REVIEW.....	5
2.1. MIDDLEWARE	5
2.2. MIDDLEWARE CHALLENGES FOR WIRELESS SENSOR NETWORKS [WSNs]	9
2.2.1. Limited Power and Resources	9
2.2.2. Scalability	9

2.2.3. Heterogeneity.....	10
2.2.4. Real-World Integration.....	10
2.2.5. Security	10
2.2.6. Data Agregation.....	11
2.2.7. Dynamic network orgnizations.....	11
2.3. MIDDLEWARE APPROACHES FOR WIRELESS SENSOR NETWORKS.....	12
2.3.2. Application Driven	12
2.3.1.1. Milan.....	12
2.3.1.2. AutoSeC.....	13
2.3.2. Distributed databases.....	13
2.3.2.1. Courgar	14
2.3.2.2. DSware	14
2.3.2.3. TinyDB	15
2.3.2.4. SINA	16
2.3.3. Virtual Machine (VM).....	17
3.3.3.1. Maté	17
2.3.3.2. Magnet	19
2.3.3.3. DAVIM.....	20
2.3.4. Mobile Agent.....	20
2.3.4.1. Smart Messages(SM).....	21
2.3.4.2. Agilla	22
2.3.4.3. Impala	22

2.3.5. Macro programming	25
2.3.6. Message-Oriented	26
2.3.6.1. Mires	26
2.3.6.2. SensorBus	29
2.3.7. Other Approaches	31
2.3.7.1. EnviroTrack	31
2.3.7.2. TinyCubus.....	32
2.4. CONCLUDING REMARKS	33
CHAPTER 3. THE RTPS-DDS MIDDLEWARE FOR WSNS	36
3.1. METHODOLOGY	36
3.2. RESEARCH OBJECTIVES	38
3.3. DATA DISTRIBUTION SERVICE (DDS) FOR REAL-TIME SYSTEM.....	40
3.4. DDS QUALITY OF SERVICE (QoS)IN RTPS MIDDLEWARE FOR WSNS	42
CHAPTER 4. THROUGHPUT EXPERIMENTAL SET UP AND RESULTS	45
4.1. One-to-One Throughput Test	51
4.2. One-to-Many Throughput Test (subscribers over different cores).....	56
4.3. One-to-Many high Throughput Reliable Messaging (subscribers over different cores).....	60
4.4. One-to-Many Throughput Scalability Test (subscribers on same core).....	64
CHAPTER 5. RTI ROUTING SERVICE FOR ISOLATION & CLUSTERIN ...	69

5.1 RTI Routing Service for Throughput Scalability Test outcomes	72
5.2 Energy Consumption Estimation in Wireless Sensor Networks	84
CHAPTER 6. LATENCY EXPERIMENTAL SET UP AND RESULTS.....	90
6.1. One-to-One Latency Test	92
6.2. One-to-Many latency Test	97
CHAPTER 7. CONCLUSION AND FUTURE WORK	101
7.1. CONCLUSIONS.....	101
7.2. FUTURE WORK.....	103
REFERENCES	104
VITA	110

LIST OF TABLES

TABLE 4.1. ONE-TO-MANY THROUGHPUT TEST OVER DIFFERENT MACHINES.	59
TABLE 4.2 RESULT ONE-TO-MANY THROUGHPUT: WITH & WITHOUT BATCHING	63
TABLE 5.1 PARAMETERS VALUES USED IN THE SIMULATION	86

LIST OF FIGURES

FIGURE 2.1. RELATIONSHIP OF OPERATING SYSTEM, MIDDLEWARE AND APPLICATIONS..	7
FIGURE 2.2. THE ARCHITECTURE OF MATE	18
FIGURE 2.3. LAYERED SYSTEM ARCHITECTURE FOR IMPALA	23
FIGURE 2.4. MIRES ARCHITECTURE	27
FIGURE 4.1. PUB/SUB ARCHITECTURE	47
FIGURE 4.2. MULTI ONE-TO-ONE THROUGHPUT TEST.....	51
FIGURE 4.3. RESULT OF ONE-TO-ONE THROUGHPUT TEST.....	54
FIGURE 4.4. ONE-TO-MANY THROUGHPUT TEST OVER DIFFERENT MACHINES	57
FIGURE 4.5. RESULT OF ONE-TO-MANY THROUGHPUT TEST OVER DIFFERENT MACHINES	58
FIGURE 4.6. ONE-TO-MANY THROUGHPUT: WITH & WITHOUT BATCHING QoS.....	62
FIGURE 4.7. RESULT ONE-TO-MANY THROUGHPUT: WITH & WITHOUT BATCHING	65
FIGURE 4.8. ONE-TO-MANY THROUGHPUT SCALABILITY TEST.....	67
Figure 5.1. RTI ROUTING SERVICE FOR DDS.....	70
FIGURE 5.2. RTI ROUTING SERVICE IN THE PUBLISHER SIDE	73
FIGURE 5.3 . RTI ROUTING SERVICE IN THE SUBSCRIBER SIDE.....	75
FIGURE 5.4. RTI ROUTING SERVICE IN THE 3 RD NODE.....	77
Figure 5.5. CLUSTERING AND ISOLATION CASES USING RTI ROUTING SERVICE.....	79
FIGURE 5.6. RESULT OF ONE-TO-MANY WITHOUT USING RTI ROUTING SERVICE.....	80
FIGURE 5.7. ONE-TO-MANY WITHOUT USING RTI ROUTING SERVICE	81

FIGURE 5.8. ONE-TO-MANY USING RTI ROUTING SERVICE.....	82
FIGURE 5.9. ENERGY USED IN EACH NON-CLUSTER HEAD NODE VS. SAMPLE SIZE	87
FIGURE 5.10. ENERGY DISSIPATED IN THE CLUSTER HEAD NODE VS. SAMPLE SIZE	88
FIGURE 5.11. TOTAL ENERGY CONSUMED BY THE NETWORK VS. SAMPLE SIZE	88
Figure 6.1. THE ONE-TO-ONE RTT TEST , QoS =BEST EFFORT	93
FIGURE 6.2. ONE-TO-ONE RTT TEST, QoS = RELIABLE.	93
FIGURE 6.3. ONE-TO-ONE RTT ANALYSIS , QoS = BEST EFFORT VS RELIABLE.....	94
Figure 6.4. RESULT OF ONE-TO-ONE RTT LATENCY, QoS =BEST EFFORT VS RELIABLE OVER DIFFERENT SIZES.....	95
FIGURE 6.5. RESULT OF ONE-TO-MANY RTT LATENCY.....	99

THESIS ABSTRACT

Name: **Ismail Mohamed Hemdan Keshta**
Title: **Performance Analysis of Real-Time Publish/Subscribe (RTPS) Middleware for Sensor Networks**
Major Field: **Computer Engineering**
Date of Degree: **June 2011**

Due to the continuing advances in network and application design in Wireless Sensor Networks (WSNs), the development of an appropriate middleware for WSNs is becoming necessary. Also, because WSNs have some limitations compared to traditional networks, such as the lack of structure and resources, middleware solutions are developed to solve some problems related to this issue. But most middleware approaches are not suitable for real-time and mission critical applications. In such applications, middleware should fully satisfy real-time constraints imposed by the network. Therefore, real-time publish/subscribe (RTPS) middleware becomes essential in mission critical applications in many environments where real-time constraints must be met. RTPS is a network middleware for real-time distributed applications. It has the ability to provide the communications service programmers needed to distribute time-critical data between nodes in a given system. Thus, there is a need to use real-time publish/subscribe (RTPS) middleware standard by Object Management Group (OMG), in sensor networks, in order to get improved results and eliminate the disadvantages of the previously proposed middleware approaches. In this thesis, the performance of RTPS middleware is presented to show that it is a better tool that can satisfy the communication requirements for sensor networks. RTPS middleware uses a publish/subscribe communication mechanism, thus this middleware provides flexible and efficient way of communication that is extremely needed in the sensor networks. Furthermore, unlike other proposed middleware approaches, real-time constraints are fully satisfied by the RTPS middleware. Therefore, it is highly suitable and recommended for real-time applications.

MASTER OF SCIENCE DEGREE
King Fahd University of Petroleum & Minerals
Dhahran – Saudi Arabia
June 2011

خلاصة الرسالة

الاسم: **إسماعيل بن محمد بن حمدان قحطه**
عنوان الرسالة: **تحليل الأداء لنظام الاتصال البرمجية الوسطية الوقتية-الحقيقي نشر-اشترك لشبكات المتحسسات اللاسلكية**
التخصص: **هندسة الحاسب الآلي**
تاريخ التخرج: **رجب 1432 هـ**

مما لا شك فيه ان التقدم المستمر في التصميم الشبكي و التطبيقي لشبكة المتحسسات اللاسلكية (و هي شبكة تتألف من عدد كبير من العقد المتحسسة حيث أن هذه العقد تنتشر بشكل مكثف داخل المنطقة المراد تحسسها أو بالقرب منها) كان داعيا ضروريا لتطوير مناسب للنظام الاتصالات البرمجية الوسطية لهذا النوع من الشبكات. وايضا بسبب بعض العيوب التي في شبكة المتحسسات اللاسلكية اذا قورنت بالشبكات الاخرى فمثلا من تلك العيوب الافتقار في البنية الهيكلية و المصادر في هذا النوع من الشبكات ، وبناء على ذلك فإن نظم الاتصالات البرمجية الوسطية تم تطويرها لحل مثل هذه العيوب. ولكن معظم هذه النظم ليست صالحة للتطبيقات التي تتطلب الزمن الحقيقي او تطبيقات الازمنة الحرجة. في مثل هذه التطبيقات نظم الاتصالات البرمجية الوسطية يجب ان تضمن متطلب الزمن الحقيقي المقترض من الشبكة. فمن هنا اصبح (RTPS) واحد من أهم نظم الاتصالات البرمجية الوسطية في التطبيقات الحرجة. فهو نظام يستطيع ان يربط بين الانظمة المختلفة ويتحكم بها. أخذا بعين الإعتبار المتطلبات الزمنية و التطبيقات الحرجة. فهناك حاجة ملحة لاستخدام هذا النظام في مجال شبكة المتحسسات اللاسلكية. علما بأن هذا النظام تم وضع خصائصه ومعاييرته من قبل (OMG). وباستخدامه نستطيع حل المشاكل الموجودة في نظم الاتصالات البرمجية الوسطية المقترحة سابقا لهذا النوع من الشبكات. في هذه الاطروحة سوف نسلط الضوء على مستوى (RTPS) كنظام للاتصالات البرمجية الوسطية في شبكة المتحسسات اللاسلكية. حيث سيتم دراسته بشكل مركز لاثبات انه افضل نظام للاتصال من سابقه. حيث ان هذا النوع يستخدم طريقة (publish/subscribe) كوسيلة فعالة للاتصال بين التطبيقات المختلفة. فهو نظام يتمتع بالمرونة وسهولة التعامل وهذا متطلب ضروري في شبكة المتحسسات اللاسلكية. وعلى العكس من غيره من النظم، فهو نظام صالح للتطبيقات الحرجة و التطبيقات التي تتطلب الزمن الحقيقي التي تحصل في هذا النوع من الشبكات.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران- المملكة العربية السعودية
رجب 1432 هـ

CHAPTER 1

INTRODUCTION

The main concept of WSNs started to appear at the end of the 1990s with the first publication in this field appearing in 1998. WSNs can be defined as a networked collection of sensor nodes that are small-scale devices and have very limited resources such as memory and power supply [21, 24]. In sensor networks, each node has to monitor the environment and some physical conditions such as sensors to detect temperature, sound, humidity, pressure, vibration, motion, light, etc. Depending on the task of the sensor network, sensors can cooperate with other sets of sensors to do a certain task. There are two types of WSNs, namely homogeneous and heterogeneous. If all nodes in a network have the same hardware setup, it is called a homogeneous network; otherwise, it is called heterogeneous [24].

WSNs support a wide area of applications such as environmental monitoring, tracking of vehicles, habitat monitoring, traffic control system, security and defence applications, industrial automation, etc [21].

WSNs have some advantages over traditional networks, such as easy to deploy, wide scalability and ease of use in different complicated environments for some special purposes. However, WSNs have some limitations compared to traditional network.

This is due to the lack of structure and resources such as overhead communications needed between sensor nodes and complicated structure when sensor nodes are added or removed. Middleware solutions are developed to overcome some problems that are faced in WSNs.

Middleware solutions can link between application and low level operating systems to enhance application development. Moreover, middleware can hide the complexity and heterogeneity of the underlying hardware and ease the management of system resources [8, 7].

In general, any middleware solution should support work phases of WSNs, such as development, maintenance and data execution. Also, the middleware solutions should provide additional features that are related to WSNs, for example, ability to save power, scalability, mobility and heterogeneity. Furthermore, there are important challenges which must be provided by a successful middleware, such as ease of use, managing resources, security and quality of service (QoS).

The implementation of a successful middleware for WSNs is not an easy job. It needs to deal with many challenges that are related to WSN characteristics. Many researches are still going on to address the challenges and solutions for WSN middleware [1, 12]. Some of these challenges are discussed in section 2.2 in Chapter2.

1.1 PROBLEM STATEMENT

As will be seen in Chapter 2, none of the proposed middleware approaches mentioned in the literature are suitable for real-time and mission critical applications, where real-time constraints must be met by the middleware. Also, all the previous approaches do not have QoS properties that can be set based on the needs of a given system. Thus, there is a need to use real-time publish/subscribe (RTPS) middleware standard by Object Management Group (OMG), in sensor networks, in order to get improved results and eliminate the disadvantages of the previously proposed middleware approaches. In this thesis, the performance of RTPS middleware is presented to show that it is a better tool that can satisfy the communication requirements for sensor networks. RTPS middleware uses a publish/subscribe communication mechanism, thus this middleware provides flexible and efficient way of communication that is extremely needed in the sensor networks. Furthermore, unlike other proposed middleware approaches, real-time constraints are fully satisfied by the RTPS middleware. Therefore, it is highly suitable and recommended for real-time applications.

This work also will investigate the usage of Quality of Service (QoS) specified in the Data Distribution Service (DDS) middleware standard proposed by Object Management Group (OMG).

1.2 THESIS ORGANIZATION

The organization of this thesis is as follows. In Chapter 2, middleware definitions and some preliminaries are given. Also, a literature review of middleware solutions for wireless sensor networks (WSNs) is presented. In Chapter 3, details of RTPS-DDS middleware are described. The setup of the experiments is explained in Chapter 4, 5 and 6. In these chapters, experimental results are discussed to show the behavior of the RTPS-DDS middleware in a wireless sensor network over different test scenarios. Finally, we conclude and indicate the future work in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

2.1 MIDDLEWARE

As shown in Figure 2.1, a middleware is defined to be an interface layer between the operating system (OS) and the application in a distributed and networking context. In [25], a middleware is defined as a connectivity software that allows nodes in a system to communicate with others across a network. In other words, it is a tool that works to facilitate, manage and control the communication between any two applications that interact across the hardware and networked environments [25, 26].

The details of the underlying computer architecture, operating system and network stack are all hidden by the middleware layer. In addition, the middleware layer works to simplify the development of a distributed system. This is done by making user-applications exchange information with others without the need of an interface with a program that uses low level protocols.

Basically, the major role of middleware is to ease the task of managing and designing distributed systems. A middleware does that by providing a simple and consistent integrated programming environment [25, 26]. There is a certain set of

criteria that is addressed by middleware services including: 1) independence of the chosen platform. These services must provide the portability to various types of system architectures with predictable effort, 2) provide the functionality to meet the real requirement of several kinds of applications.

It is important to highlight that a middleware offers a platform-independent Application Programming Interface (API) which is a set of system calls (functions) used by an application program for providing access to a system's capabilities. These system calls (API) are provided by the middleware to handle an application environment and mask the complexity of distributed processing [23, 26].

In wireless networks, many applications might work fine without having middleware. However, some certain applications might not perform well without them. Middleware is mainly used in applications that involve high transaction volumes and are deployed for many users. In addition, middleware is essentially required in critical applications, and when there are stringent reliability requirements [23].

In general, middleware is of three major types: communications middleware, database middleware and systems middleware. Following are some examples:

The most famous classical middleware systems are the Common Object Request Broker Architecture (CORBA) specifications. CORBA is a standard that is defined by the Object Management Group (OMG) which can enable multiple software components, written in various programming platforms, to run on any computer, to

communicate with each other. The other popular middleware systems are the message-oriented middleware (MOM) specifications.

MOM carries and distributes messages between separate systems in a network in order to connect them in a proper way. An infrastructure for this kind of middleware is based on the queuing system that stores messages, pending delivery. In addition, it monitors when each message has been delivered [25, 26].

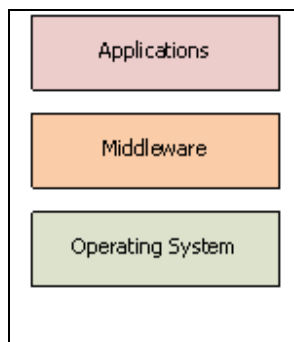


Figure 2.1: Relationship of Operating System, Middleware and Applications

It is important to point out that a range of routines which are usually implemented as part of the operating system (OS) are shifted to middleware to support a certain class of dedicated applications. Also, the limitation in memory that is available has led to implementation middleware that provides the Application Programming Interface (API) rather than establishing a general background of middleware functionalities [23, 25].

There are two different options that are available to characterize middleware in a wireless sensor network (WSN) context. The first one is that the middleware has to provide software that contains minimal set of core routines and functions in order to call it a middleware. Second, it can also be done by looking at the differences between the various approaches in this domain [25, 26].

2.2 MIDDLEWARE CHALLENGES FOR WIRELESS

SENSOR NETWORKS (WSNS)

A middleware approach for wireless sensor networks needs to deal with many challenges that are related to characteristics of this kind of network. This section demonstrates some of these potential challenges [24].

2.2.1 Limited Power and Resources

In WSNs, sensor nodes are small-scale devices. These small devices have very limited storage energy and memory. Moreover, nodes might go down due to environmental influence. In general, the limitation in size and energy means restricted resources. Therefore, the middleware should manage the three basic operations of a WSNs sensing, data processing and communication without consuming additional resources [12, 24].

2.2.2 Scalability

The sensor network should scale from ten to thousand sensor nodes. So, the sensor network should be flexible enough to allow this growth without affecting the performance of the network. For this reason, a middleware should provide mechanisms for self-configuration and self-maintenance for any sensor node in the WSNs [12].

2.2.3 Heterogeneity

In WSNs, cross platform communication is needed to bridge the gaps between the hardware technology's raw potentials, and needed activities such as data execution. Therefore, any proposed middleware should glue the gaps between them by supporting some mechanisms of interfacing systems in various types of hardware and networks [12, 22].

2.2.4 Real-world Integration

For some applications in WSNs, the real-time requirement must be supported. In other words, WSN environments are not constant. They are always changing. This change includes changing in time and space. Therefore, middleware solutions should be developed in such a way that supports real-time requirements to adapt to the changes of such environments like WSNs [12, 22].

2.2.5 Security

The wide deployment of WSNs in complex areas which are difficult to reach might increase the chance for the malicious attackers to access sensitive information. Therefore, maintaining security is a major concern in the WSNs. However, doing such a task is not trivial due to the limited power and resources in this kind of network. Hence, any proposed middleware should concentrate on developing security aspects in the initial phases of software design with initial overhead [12].

2.2.6 Data Aggregation

Most of the WSNs generate lots of redundant data because a sink node combines data from different sources which might make huge data communication between nodes in the networks. So, a middleware should have the capabilities to eliminate redundancy in the data network. In the other words, it should be able to limit the retransmission of similar data over the entire network in order to minimize data communication among sensor nodes. This implies reducing collisions in the network and energy consumption [12, 22, 24].

2.2.7 Dynamic Network Organization

Dealing with resources such as energy and bandwidth that are dynamically changing must be included in WSNs. In addition, such a network has to support long-running applications in order to run it as long as possible. Also, it is important to highlight that knowledge of the network is a major concern in order to operate it in the proper manner. Therefore, a middleware has to provide an ad-hoc wireless network resource discovery. Moreover, adapting to the dynamic changes of the networks must also be supported by the middleware [12, 24].

2.3 MIDDLEWARE APPROACHES FOR WIRELESS

SENSOR NETWORKS

Middleware approaches can be mainly classified into five classes according to middleware architectures and approach mechanisms. Throughout this section, these approaches are listed and reviewed. In addition, for each class, examples and descriptions of their mechanisms and features are provided.

2.3.1 Application Driven

In application driven approaches, middlewares can adjust network configurations according to the requirements stated by the application. This type of middleware has a structure to supply multiple network configurations by choosing suitable protocols in its network protocol stack. Examples of this category are Milan and AutoSeC.

2.3.1.1 Milan

Middleware Linking Applications and Networks (Milan) is being developed at the University of Rochester and has a very good architecture to link the network layer and application layer. The idea behind it is to make the sensor network application control the network operations management. In other words, Milan allows sensor network applications to specify their Quality of Services (QoS). Moreover, this proposed middleware allows adjusting the network characteristics to increase application lifetime while still meeting those needs. To achieve that, Milan can

receive information about the QoS requirements of different sensor network applications over time, the overall system, and the network about available sensors and resources such as energy and channel bandwidth. In addition, Milan collects and combines this information in order to configure the network characteristics to increase application lifetime while still meeting QoS requirements [1, 5, 9].

2.3.1.2 AutoSeC

Automatic Service Composition (AutoSeC) is an application driven middleware, developed at University of California-Irvine. AutoSeC is a dynamic service broker framework for effective utilization of resources within a distributed environment. Based on current system status, AutoSeC is able to dynamically select the best combination of information collection and resource provisioning policies. Moreover, it provides some of the required quality of service (QoS) for sensor applications.

In terms of evaluation, AutoSeC is able to manage resources in a sensor network by providing access control for applications. In addition, power-aware algorithms are provided by this type of middleware. Therefore, energy consumption is reduced.

2.3.2 Distributed Database

All distributed database middleware approaches deal with the whole sensor network as a distributed database. This type of middleware also has a friendly and easy to use interface, using SQL queries to collect data. Examples of this class are Cougar, DSware, SINA and TinyDB [3, 9].

2.3.2.1 Cougar

Cougar is an example of wireless sensor middleware solutions in WSNs that is based on the database approach. This middleware was developed at Cornell University. Cougar middleware is a sensor database system that is composed of sensor database and sensor queries. Sensor data is generated by using signal processing which is executed on each sensor node. Also, this sensor data is stored in a local database system. Abstract Data Type (ADT) in Cougar is used in order to model signal processing functions. In object-relational databases, an ADT can represent all sensor nodes of the same type in the physical world. Cougar also uses SQL-like language to implement WSN management operations in the form of queries.

In terms of evaluation, Cougar middleware can support a large collection of sensors. Also, this kind of middleware can provide a simple scheme for different network operations. However, maintaining the global knowledge of WSNs via centralized optimizer used by Cougar is not suitable for large scale WSNs because of the dynamic nature of these networks. Also, to transfer a large amount of raw data from devices (sensor nodes) to the database server, Cougar consumes more resources compared to other approaches. Furthermore, Cougar does not resolve the problems of hardware heterogeneity and node mobility issues [29, 30].

2.3.2.2 DSWare

Data Service Middleware (DSWare) is another database middleware approach and provides data services for applications. It implements a database-like abstraction that

consists of some service components such as scheduling of all middleware services based on either energy-efficiency or delay performance, data storage for storing data according to the semantics associated with the data, and caching of multiple copies of the data that are requested most often.

In addition, DSWare supports group-based decisions and provides reliable data-centric storage. These features make this middleware more flexible than other database approaches. Also, DSWare uses SQL-like language for event operations such as registration and cancellation of an event. Therefore, it has a friendly-user interface. On the other hand, DSWare does not resolve hardware heterogeneity and mobility issues. Furthermore, the sensor database in each node needs continuous updating for more dynamic applications. Therefore, DSWare middleware does not fully comply with the scalability issue.

2.3.2.3 TinyDB

TinyDB is a query processing system for sensor networks that operates on the TinyOS operating system. It is designed and implemented based on the concept of acquisitional query processing (ACQP) for collecting data in a sensor network. When query processing occurs, the sensor node will directly perform sensing to respond to the requested query. TinyDB is a distributed system with a SQL-interface to execute data from sensor nodes. Compared to traditional technology, TinyDB has features such as low power consumption, which is an important advantage in a resource-limited network environment.

2.3.2.4 SINA

The authors in [30] proposed SINA (Sensor Information Networking Architecture) which is another database middleware approach that uses a query language, developed at the University of Delaware. For querying and monitoring, the SINA architecture was implemented based on a spreadsheet database. Every single database contains cells, where each cell represents an attribute of a sensor node for location and sensor reading. Therefore, applications are able to access a particular data element by using an attribute-based naming for naming a sensor directly. SINA uses hierarchical clustering of sensors where sensor nodes are organized in a specific way to form a hierarchical shape, based on their levels of power. Moreover, it uses a set of protocols to prevent the re-broadcasting of similar information to other nodes. SINA is considered to be more flexible than other database middleware approaches since it supports both Sensor Query and Tasking Languages (SQTL) and SQL-like languages. This language works as the programming interface between sensor applications and the SINA middleware. Also, SINA offers an advantage over other approaches by using hierarchical clustering of sensors for efficient data aggregation. However, like Cougar, SINA does not resolve the problem of hardware heterogeneity. Moreover, it does not fully support scalability because of the fixed global network structure that is maintained by the SINA middleware [30].

2.3.3 Virtual Machine

The system of Virtual Machine (VM) middleware approach consists of virtual machines and translators. In this proposed approach, developers can write applications into small modules. These modules will be distributed throughout the network. Virtual machines translate the modules in order to implement applications. Examples of service middlewares for sensor networks that use the concept of virtual machine are Maté, Magnet, and DAVIM [2].

2.3.3.1 Maté

Maté is an instance of the virtual machine approach, developed at the University of California at Berkeley. It uses the virtual machine as an abstraction layer for implementing its operation. Also, it is a byte code interpreter which is implemented on the TinyOS operating system. As shown in Figure 2.2, Maté has a stack-based architecture that consists of three execution contexts. These states are clock, send, and receive.

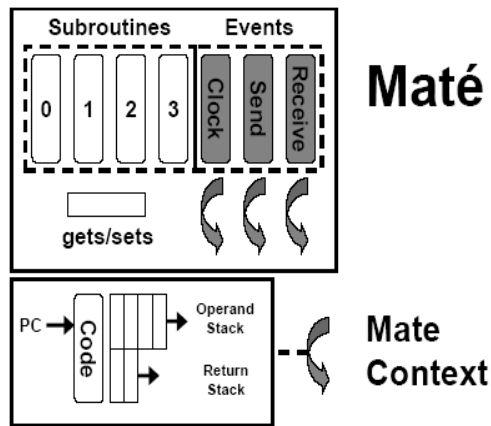


Fig.2.2: The architecture of Maté

This kind of middleware works by breaking down the program into small self-replicating capsules composed of 24 instructions where each instruction is single byte long. This gives the advantage to large programs to be made up of multiple small capsules, thus making it easy to inject them into wireless sensor networks.

Virtual machine (VM), Network, Logger, Hardware and Boot/Scheduler are the five key components of Maté. It uses a synchronous event model which starts execution in reaction to an event such as packet transmission. Therefore, Maté avoids message buffering and does not require large storage. Moreover, using such a model increases simplicity of application level programming.

In terms of level of instructions, low and high level instructions are supported by the Maté program in a stack-based architecture. Maté instructions can be arithmetic operations, loop operations and wireless sensor network specific operations. This set

of instructions helps Maté middleware to provide high level abstraction for an application developer.

The programming model of Maté middleware is considered to be simple and easy to use for the application developer. On the other hand, it does not provide full flexibility. Therefore, it cannot support a wide range of applications.

This kind of middleware (Maté) makes the network easy to reconfigure and also increases the security of the network. However, in terms of energy, it is not suitable for complex applications because of the instruction interpretation overhead [34, 35].

2.3.3.2 Magnet

Magnet is another system-level middleware solution for WSNs. It is also based on the VM approach developed at Cornell University. It consists of a layer known as Single System Image (SSI) which represents the whole network as a single Java Virtual Machine (JVM). Components of the JVM are either dynamic or static. For the dynamic component, each node has the full responsibility to monitor and coordinate applications and also perform application specific tasks. On the other hand, the static component is used to inject java applications into the WSNs.

In terms of performance, Java implementation of Magnet and Single System Image techniques makes the development of an application simple. Also, power-aware algorithms are provided by the Magnet Middleware. Therefore, energy consumption is reduced. In addition, this kind of middleware supports a wide scale of applications, which makes it a general-purpose system. However, heterogeneity is partially

supported by Magnet. Moreover, a lot of overhead on Magnet's instructions are represented because of the use of JVM. To come up with a VM that is more suitable for wireless sensor network applications, a lot of effort and hard work is needed [36].

2.3.3.3 DAVIM

DAVIM (Distrinet Adaptable Virtual Machine) is a new service middleware for sensor networks, implemented as a dynamic management of services on virtual machines. It makes isolation between simultaneous running applications over sensor networks. Regarding the architecture of this middleware, the DAVIM middleware approach is presented based on the architecture of virtual machines. It uses such architecture to run applications and services. The applications are isolated from each other because each virtual machine runs one application.

In terms of evaluation, DAVIM is designed to meet some requirements such as managing available services easily, with multiple applications running on the same sensor network begin kept isolated. It is important to highlight that DAVIM presents similar overhead during installation of new byte-code scripts if it is compared with other approaches [2].

2.3.4 Mobile Agent

The main feature of this approach is that the applications are treated as modules in order to distribute them throughout the network using mobile codes. The sensor networks can implement tasks by transmitting application modules. Transmitting

using modules might consume less power than transmitting full applications. Examples of this category are Smart Messages (SM) and Agilla[6].

2.3.4.1 Smart Messages(SM)

Smart Messages (SM) is a mobile agent middleware. The term SM is a user-defined distributed program which executes on nodes of interest and migrates between nodes to reach other nodes. Its architecture is based on execution migration of executing units. The implementation of this architecture was made on top of an unmodified JVM. Ease of deployment for applications in the network and adaptability to more dynamic network conditions are considered as main benefits provided by SM [37].

In terms of rooting, self-root mechanism can be done by SM, when a SM is required to migrate between two nodes and there are intermediate nodes between them. Each node has a VM for SM execution and a name-based memory called tag space. The SMs use the tag space for content-based naming and persistent shared memory. A VM is assigned to each node for SM execution process. Also, a name-based memory called tag space is allocated for each single node to be used as persistent shared memory.

SM can adapt in a quick way to the changes which might occur in the network topology and the availability of resources at nodes. Moreover, this type of middleware can provide a Networked Embedded Systems (NES). On the other hand, a node in SM can run only a single execution thread. Therefore, SM does not support

multiple applications on a single node. In addition, inter-node communication is not supported by SM [37].

2.3.4.2 Agilla

Agilla is a middleware layer that supports mobile agents for WSNs. It provides mechanisms for better injection of a mobile code into the sensor network to deploy some user applications. In the Agilla system model, each sensor node supports multiple agents and maintains a local shared memory space and a neighbors list. The local shared memory space is shared by the agents residing in the node. The neighbors list contains the addresses of all the one-hop nodes. The agents can move to different locations around the network nodes in an intelligent way based on the changing conditions in the environment, by using move and clone instructions. Regarding the model used by the agents, it is based on the stack architecture and the agent codes are written using assembly language [6, 8].

Agilla has good performance and high reliability. It is more suitable than the flooding mechanisms that are used in the middleware Maté for the same purpose. However, using assembly language programming is mentioned by the authors as a weak point [6, 8].

2.3.4.3 Impala

Impala is a middleware which was specially designed and implemented for the ZetbraBet project. The main goals of this middleware are to ensure reliability and ease of upgrades for long-running sensor network applications. Also, the major

philosophy behind Impala is that mobile environments need continuous fine-tuning.

The methodology used by Impala middleware in its design can be stated as follows:

- 1) Modularity concept which is used for switching the decision process.
- 2) Correctness: this concept is used for making individual program applications instead of having one single big application.
- 3) Ease of updates is an important design issue for using small pieces of software because this makes the update easier.
- 4) Energy efficiency can be achieved by making the transmission of the updates at the granularity of smaller modules.

The architecture of Impala is divided into two main layers, as shown in Figure 2.3.

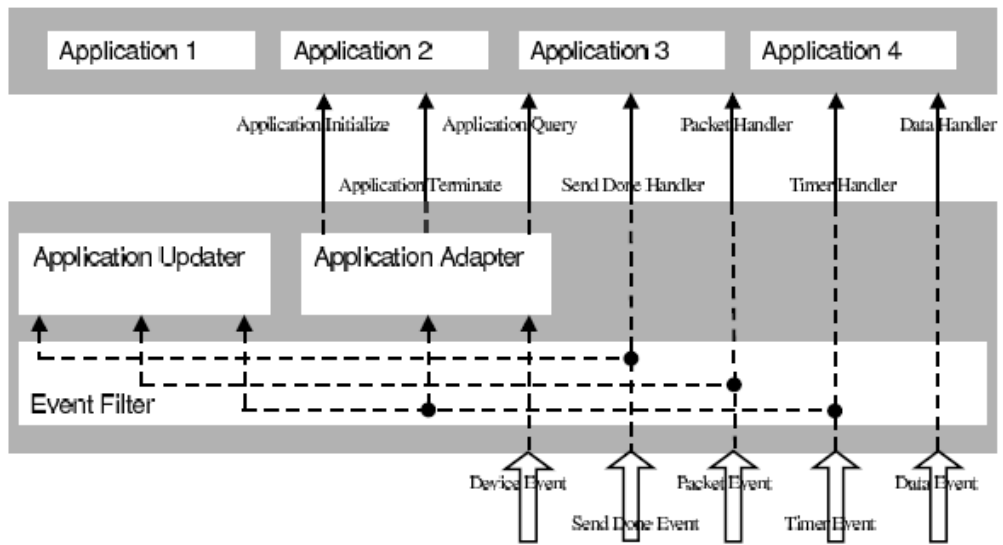


Figure 2.3 –Layered system architecture of Impala

The upper layer contains all the applications and protocols for the ZebraNet project.

Various strategies are used by these applications in order to gather environment

information and route it to a base station. The lower layer contains three agents: the Application Adapter, the Application Updater, and the Event Filter. The Application Adapter has the full responsibility to increase performance and improve robustness by switching between alternative protocols in case of hardware failures. The second agent (the Application Updater) is used to handle some issues such as incomplete updates, propagation protocol, code memory management, and inconsistent updates. The Event Filter (the last agent in the lower layer) is used to capture and dispatch events in the Impala system. There are five different types of events supported by this type of middleware, namely Timer Events, Packet Events, Send Done Events, Data Event and Device Events. These events are processed in sequential order.

In terms of evaluation, it is important to highlight that Impala middleware is a self-organized architecture model. This is because it uses Application Finite State Machine (AFSM) mechanisms to switch between adequate protocols. Also, it ensures the reliability of long-running applications. By its organized architecture, Impala can support application adaptation at runtime. Moreover, little transmission overhead is generated by this kind of middleware. In addition, it provides failure tolerance. On the other hand, the adaptation process in the Impala middleware is limited to the capabilities of the state machine. Also, heterogeneity in hardware platform is not supported by Impala. Therefore, its application domains are limited [28].

2.3.5 Macro programming

Rather than writing low-level software for each single node, the Macro-programming approach introduces different ways on how to program sensor networks by programming the entire network as a whole. High-level specification is used to program wireless sensor global behaviour. Therefore, this view will reduce the load in dealing with low-level concerns at every single node on the network [27].

The famous example on the use of this approach is Kairos middleware. This type of middleware allows the programmer to program the whole sensor network. A centralized program for whole application is written for the overall application. This program will be divided into subprograms and compiled into annotated binary codes (annotated binary codes are node-specific version which contains a code in order to control the behaviour of each node individually), by Kairos middleware. Then, the binary codes are distributed to other nodes on the network to make them communicate with each other. In term of synchronization between the nodes, Kairos provides either a loose synchronization or a tight synchronization based on the programmer's purpose.

In terms of performance, Kairos middleware addresses the mobility issue in a full manner. Also, it supports robust mechanisms for node localization and routing aspects. However, easy to use issues are partially addressed by Kairos. Moreover, resource management constraints are not completely supported [27].

2.3.6 Message Oriented

Message-Oriented Middleware (MOM) implements communication using publish/subscribe mechanisms between sensor nodes and applications. The publish/subscribe service in the middleware is used to exchange messages between the sender who sends the message and the receiver. Examples of this class are Mires and SensorBus middlewares which are explained in the following sections respectively.

2.3.6.1 Mires

Mires follows the characteristics of message-oriented middleware (MOM). This can be done by allowing the applications to communicate in a publish/subscribe fashion. In addition, this middleware proposes an asynchronous communication model, which is suitable for wireless sensors network applications. The main issue regarding this proposed middleware is that Mires can support hardware heterogeneity. In general, the communication throughout the Mires middleware consists of three phases. First of all, each node in the wireless network will make announcements for its available topics. Then, by using a routing algorithm, announcement messages will be routed to a dedicated node called the sink node which is connected to a user application. By a graphical user interface, user applications will be able to select the advertised topics to be monitored. After that, the sink node will broadcast the subscriber's messages to the sensor network nodes. Subsequently, the sensors will publish their collected data throughout the sink node to the network-based applications [1].

An overview of the Mires architecture is shown in Figure 2.4, By using a bottom-up approach, the first block corresponds to the sensor nodes hardware components such as micro-controller unit, sensors, and radio transceivers. This block is directly controlled by the operating system (OS) (second block). The third block is the Mires. This middleware "Mires" has a core component, namely the Publish/Subscribe services and some additional services. Mires implements high-level publish/subscribe by providing services and routing while hiding the complexity of the sensor network [1, 10].

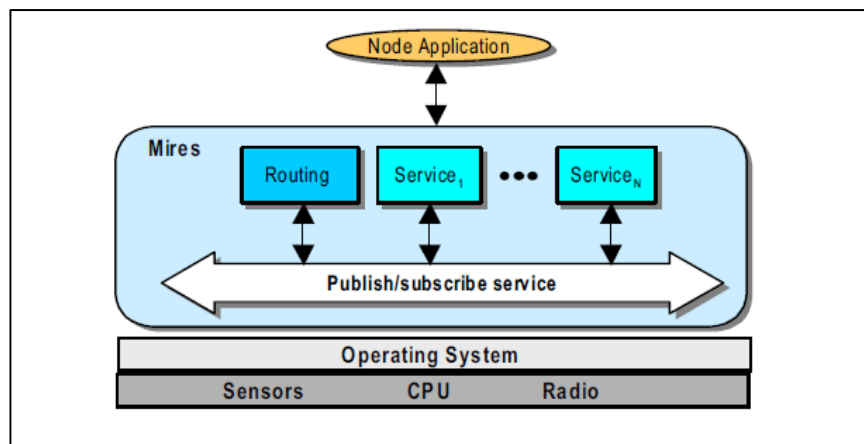


Figure 2.4 Mires Architecture.

In a wireless sensor network, the network consists of multi-nodes that communicate over wireless communication links. A publisher node can publish data that is related to the event of interest for a subscriber node. In other words, the publish/subscribe principle in wireless sensor networks states the nodes, which are interested in

receiving certain information. This process is called subscription. Therefore, the interested node is called a subscriber. Nodes, which intend to produce certain information, can do so by publishing their information. Thus, they are called publishers. Based on this, a subscriber node can state the type of data that it is interested in (e.g., temperature data) by broadcasting the message to all the sensor nodes in the network. After that, the sensor nodes transmit the desired data [1].

The publish/subscribe service mainly goes over two phases (advertise phase and subscription phase). For the advertise phase, a node application advertises to the publish/subscribe service its interest of sensing data which is related to a specific topic. Then, the publish/subscribe service receives and encapsulates this information in a certain message called the advertising message. After that, this message will be sent to the network by using some routing components. In the case of subscription phase, the user application invokes a dedicated node called sink node, which receives the information gathered by the network and delivers it to the final user. This is done by using a send command to broadcast the subscribed topics to the network. There is a dedicated component called broadcast that is used to signal a received event for each node that receives a subscription message. After that, the publish/subscribe will receive this message and extract setup information from it. In the last step of this phase, publish /subscribe service invokes a certain service that is used to publish the processing results, by sending a signal called “Topic Setup Arrival” to notify its components which are attached to it [1].

At the end of the discussion on Mires, it is important to highlight that, the implementation mode for the Mires middleware is still in progress. Also, tests using real sensor nodes (motes) have not yet been done in order to see the real behavior of this kind of middleware in a wireless sensor network. Moreover, the design of Mires needs to support security and resource management issues [10].

2.3.6.2 SensorBus

SensorBus is a middleware model for WSNs. It is based on the publish/subscribe (P/S) paradigm. Using this kind of communication allows free exchange of the communication messages amongst the sensor nodes. As a result, the capability of using more than one communication mechanism is allowed to address the requirements of a large number of applications [15].

In general this approach is similar to what was discussed in section 3.5.1. A sensor node called “publisher” in the SensorBus model tries to generate events. This node publishes types of events which will be available to other nodes called “subscribers”. This proposed approach uses an asynchronous type of communication to send notifications from producers to interested subscribers. In addition, the designers of this model rely on MOM to take care of filtering the messages and routing them to the appropriate subscribers [15].

Regarding the SensorBus Middleware Architecture, SensorBus consists of the following components: an application service, a message service and a context service. For the application service, this element has the full responsibility to provide

the Application Programming Interface (API), which simplifies application development. Also, it is comprised of three components: DataBus is a component that provides a group of operations which are related to bus communication for consumers and producers, Filter, the second component, is used to provide a set of operations that are related to data, and the last component which is language, implements the commands and query language interpreter [15].

The second component in the SensorBus architecture is the message service. This component has the full responsibility for providing communication and coordination for the distributed components. It consists of three main elements: Channel which is used to deal with the specific transport implementations, Transport which will take care of the communication among the nodes and is considered to be like a socket. The last component that belongs to the message service is the sinker which is a dedicated component for routing messages in the network [15].

The third component in the SensorBus architecture is the context service. This element is responsible for managing, monitoring and coordinating the heterogeneous sensors which collect information from various environments [15].

SensorBus is a middleware for WSNs, which can decouple the communication mechanism from the programming interfaces. It also uses more than one communication mechanism to address the requirements of a larger number of applications [15].

2.3.7 Other Approaches

2.3.7.1 EnviroTrack

EnviroTrack is an object-based distributed middleware system. It is considered to be the first programming support for sensor networks which can support tracking mobile objects. This middleware is very well suited for embedded tracking applications. In other words, EnviroTrack WSN middleware supports environmental target tracking [39].

Also, the dynamic behavior of the tracked targets such as mobility is supported by EnviroTrack. This is because of the use of powerful network management mechanisms such as group leader election. Therefore, any moving target can be detected successfully. This feature can be very useful for some military applications. Like other projects, it is also built on top of TinyOS using compiled NesC programs.

The architecture of EnviroTrack consists of two major parts. These are:

- 1) A pre-processor module which is used to interpret user directives in order to produce the appropriate middleware call functions at compile time.
- 2) A run-time group management protocol that is run on the top of the routing service.

Briefly, a context description file goes as an input to the first major part (EnviroTrack preprocessor). Based on the information that is gathered from the context description file to generate appropriate middleware call functions, the pre-

processor can patch a set of NesC program templates. After that, the programs are compiled using TinyOS tools.

In terms of evaluation, EnviroTrack wireless middleware is a very good distributed program for supporting tracking environments. However, it is important to highlight that its performance is based only on a very small-scale implementation. Moreover, it is in the early stage of development. Self-organization and autonomic system approach for EnviroTrack needs a lot of work to be done [39].

2.3.7.2 TinyCubus

TinyCubus [40] is an adaptive cross-layer framework middleware which is implemented on top of TinyOS. The goal of TinyCubus project is to develop a generic reconfigurable system software for sensor networks. The design philosophy of this middleware is its flexibility and adaptation.

In terms of architecture, of TinyCubus middleware is divided into three main parts, namely tiny cross-layer framework, tiny configuration engine, and tiny data management framework.

It is important to highlight that the flexibility of TinyCubus allows it to be used in different environments. Moreover, application optimizations can take place because of the cross-layer approaches used by this type of middleware. However, a lot of overhead is generated due to the cross-layer approach, which may be prohibitive in some sensor network environments. In addition, TinyCubus does not fully support scalability issues [40].

2.4 EVALUATION AND ANALYSIS OF MIDDLEWARE

APPROACHES

Section 2.3 of this thesis shows different existing middleware approaches as middleware solutions for WSNs. These approaches are: application driven, database based, VM, mobile agent based, and message oriented. In this section, these approaches are briefly evaluated based on constraints such as heterogeneity, scalability and power saving, which are quantitatively evaluated in the second half of this thesis.

The application driven approaches such as Milan can provide both application and network QoS by controlling sensor nodes. On the other hand, Milan is found to be a weak approach for mobility. This is because Milan is not able to maintain communications between mobile sensor nodes in WSNs. Also, Milan does not support the heterogeneity constraint, because it does not provide a low level programming paradigm [5] [7].

The database approaches such as SINA and TinyDB, deal with WSNs as a huge virtual database. They are considered as a strong from a usability perspective because they use a database middleware based on query systems and SQL-like interfaces. Moreover, they are suitable for some applications. However, they have some limitations. In other words, the types of data that will be used at every node must be

agreed upon in advance. This is not acceptable in a large size sensor network. Therefore, the scalability issue is not completely supported by this approach [3, 4, 33].

The VM approach provides an efficient programming model that hides the heterogeneity of the hardware resources and supports ease of use for the application developer. On the other hand, the VM approach is not suitable for some complex applications because its instructions introduce a considerable overhead. [34,35].

For mobile agent based middleware for WSN, this approach has a good performance and high reliability. In addition, it strongly supports and fully addresses the power saving and scalability issues. Furthermore, it adapts quickly to changes which might occur in the network topology. On the other hand, mobile agent based approaches do not fully support ease of use. Also, heterogeneity in hardware platform is not supported [6, 8].

The MOM uses a message based communications protocol that is able to store and transform the message as it is being delivered. Moreover, this approach can provide a persistent storage in order to take care of the latecomers that join the network. In other words, MOM does not require for both the sender and receiver to be connected at the same time. However, this approach requires an additional component in the architecture. Therefore, this overhead might lead to reductions in performance and reliability, and can make the system difficult to maintain [1, 10, 15].

As stated in the literature, most of the proposed middleware approaches do not satisfy the communication requirements for sensor networks. Furthermore, none of these proposed middleware approaches are suitable for real-time applications like mission critical applications where real-time constraints must be met by the used middleware. Thus, there is a need to use real-time publish/subscribe (RTPS) middleware and investigate the use of Quality of Service (QoS) aspects specified in the Data Distribution Service (DDS) middleware standard for sensor networks. If we do so, this will provide for solutions to problems which have been identified previously in the literature.

In the coming chapter, we will discuss the Data Distribution Service for real-time System (DDS) in order to address the issue of having a real time publish/subscribe middleware for wireless sensor networks (WSNs).

CHAPTER 3

RTPS-DDS MIDDLEWARE FOR WSNs

3.1 METHODOLOGY

In this thesis, the benefits of the deployment of Data Distribution Service (DDS) standard along with the Quality of Service (QoS) setting for sensor network are pointed out.

Also, some scenarios will be implemented in order to help us test and analyze the performance of the RTI-DDS as a real-time publish/subscribe middleware in sensor networks. This can be achieved by evaluating the performance of RTI-DDS in major aspects that are related to sensor networks. These are:

1. Scalability: The sensor network should scale from tens to hundreds of sensor nodes. So, the network should be flexible enough to allow this growth without affecting the performance of sensor network. Therefore, tests will be performed to show the scalability of RTI-DDS and evaluate how this type of middleware can handle the increasing number of nodes.

2. Reliability: In mission critical applications, the middleware should make sure that every single event will be delivered to the appropriate sensor node correctly. Therefore, RTI-DDS will be used in implementing test scenarios such as having a very large number of requested messages in the network. This will be done to evaluate how RTI-DDS can handle reliability issues.
3. Performance in terms of latency and throughput: The most important performance parameters to be calculated are latency and throughput. Latency is an expression of how much time it takes for a packet of data to get from one designated point to another. Therefore, tests will be conducted to specify the maximum accepted latency from the time the event is published by the publisher sensor nodes until the event is available to the destination subscribers. Also, throughput, i.e., number of received samples per unit of time, is an important performance parameter for any kind of network and distributed system. Therefore, test scenarios will be performed to calculate this.

3.2 RESEARCH OBJECTIVES

Due to the continuing advances in network and application design in Wireless Sensor Networks (WSNs), the development of an appropriate middleware for WSNs is becoming necessary. Also, because WSNs have some limitations compared to traditional networks, such as the lack of structure and resources, middleware solutions are developed to solve some problems related to this issue. But most middleware approaches are not suitable for real-time and mission critical applications. In such applications, middleware should fully satisfy real-time constraints imposed by the network.

Therefore, real-time publish/subscribe (RTPS) middleware becomes essential in mission critical applications in many environments where real-time constraints must be met. RTPS is a network middleware for real-time distributed applications. It has the ability to provide the communications service programmers needed to distribute time-critical data between nodes in a given system.

Real-time publish/subscribe middleware (RTPS) has several advantages over other approaches. Some of these advantages are [23]:

1. RTPS is based on a simple “publish-subscribe” communication model. It is dynamically scalable, and efficient in usage of transmission bandwidth.
2. It can be used with high performance systems because of its low overhead.

3. This type of middleware supports one-to-one, one-to-many, and many-to-many communication paradigms.
4. By using RTPS middleware, optional QoS properties can be set based on the needs of a given system or network.

3.3 DATA DISTRIBUTION SERVICE (DDS) FOR REAL-TIME SYSTEMS

The Data Distribution Service (DDS) is an Object Management Group (OMG) standard for topic-based publish/subscribe middleware. The OMG Data-Distribution Service for Real-Time Systems is considered to be the first open international middleware standard that directly addresses publish/subscribe communications for real-time systems [14].

DDS has many prime advantages such as, it is based on a simple “publish-subscribe” communication model, is dynamically scalable, and efficient in usage of transmission bandwidth. Also, it can be used with high performance systems because of its low overhead. Furthermore, DDS supports one-to-one, one-to-many, many-to-one and many-to-many communication paradigms [13, 14].

Regarding the DDS elements, the specification for DDS can be divided into two important sections. The first section covers Data-Centric Publish Subscribe (DCPS) and the second section covers the Data Local Reconstruction Layer (DLRL). For the first element, DCPS is defined as the lower layer API that can be used to exchange topic data with other DDS-enabled applications. DLRL, the second section, is the upper layer part of the specification that outlines how an application can interface with DCPS data fields [13].

It is important to highlight that DDS has many important entities such as Topic Description that, is the most basic description of the data to be published and subscribed, Publisher, Data Writer which is used to allow the application to set the value of the data to be published under a given Topic, Subscriber. Finally, a Data Reader is associated with one Subscriber and one Topic [13].

Regarding Quality of Service in DDS, Data Distribution Service is able to specify different QoS parameters for each individual Topic, Reader or Writer in order to give a wide range of facilities to the developers to design their system. DDS has many QoS parameters such as user data, ownership, owner ship strength presentation, deadline, durability. Through a combination of DDS QoS parameters a system can satisfy a wide range of needed requirements.

Also, DDS is considered to be “Data-centric” where we have all the QoS parameters, which can be changed on a per message basis. Moreover, Data Distribution Service provides API for sending and receiving data. Therefore, developers will not have problems related to any network programming aspect [13].

In the following section, we will address important QoS parameters in DDS for (RTPS) middleware to optimize data delivery for a specific application of wireless sensor network [WSNs].

3.4 DDS QUALITY OF SERVICE IN RTPS MIDDLEWARE FOR WSNs

DDS QoS controls the flow of the data through any system. In the Publish-Subscribe system, there are QoS policies for Topic, DataReader (DR), Data-Writer (DW), Publisher, and Subscriber. In general, QoS policies of Subscriber, Data-Reader, and Topic control the data on the receiving side. However, QoS policies of Publisher, Data-Writer, and Topic will also control the data on the sending side. Throughout this section, we will go over certain DDS QoS policies.

For QoS policy, any proposed Real-Time Publish/Subscribe (RTPS) middleware for Wireless sensor networks (WSNs) should utilize some important QoS policies of the QoS model of DDS, such as latency budget and reliability [14].

Latency budget QoS policy specifies the maximum accepted latency from the time the event is published by the publisher sensor nodes until the event is available to the destination subscribers. In other words, this QoS policy will determine the maximum acceptable delay from the time the data is written by the Data-Writer in the publisher side until the data is available to a receiving application. For example, if an application creates a latency budget QoS policy to be 200 milliseconds, this policy is applied to any instance of a topic generated by a publisher sensor node in order to

ensure that all the data in this topic is delivered within less than 200 milliseconds [14, 16].

The reliability QoS policy indicates the level of data transmission reliability provided by DDS. In particular, DDS supports two reliability models, RELIABLE and BEST EFFORT. When reliability QoS policy is set to RELIABLE, DDS attempts to deliver all events. The missed events are retransmitted until the number of transmissions is greater than a threshold, or the transmission is successful. On the other hand, when reliability QoS policy is set to BEST EFFORT, DDS sends out each event only once and relies on the MAC layer for successful transmission [14, 16]. Therefore, for any proposed RTPS middleware for WSNs, this should be set to RELIABLE in order to ensure that the necessary data is not lost.

Also under the concept of avoiding flooding the network and in order to clearly illustrate this point, assume that we have many sensor nodes that measure the temperature as example and it is required that subscriber applications need to get temperature readings from the most powerful sensor. If this sensor stops working because of damage or for any other reasons, the applications should automatically use the readings from another temperature sensor. With RTPS for WSNs, we should set the value of OWNERSHIP QoS policy should be set to "exclusive" to ensure that the readers will only receive data from a single sensor node.

Furthermore, DEADLINE QoS policy can be used in RTPS to specify that the subscribers will automatically switch to the sensor with the second highest identifier

number, if it does not receive data within the specified time period. In other words, this setting will satisfy the condition of the real-time term because the data must be provided within a predefined time period, which is the DEADLINE period [14, 16, 17].

CHAPTER 4

THROUGHPUT EXPERIMENTAL SET-UP AND

RESULTS

To study the performance of real-time publish/subscribe (RTPS) middleware standard by Object Management Group (OMG) in wireless sensor networks WSNs, different test scenarios are implemented to address the issues that are related to performance metrics. The main purpose of our experiments is to implement and run throughput, jitter and latency performance tests with fixed and different sample sizes and number of subscribers that is closer to the sensor network environments. The middleware that is used in this work as RTPS is called RTI-DDS. This middleware is implemented by the company Real-Time Innovations (RTI), which is one of the most complete and representative implementations of Data Distribution Service (DDS).

Some of the primary entities used in all tests are briefly defined as follows:

Topic is the basic connection between publishing and subscribing applications. To communicate, the Topic of a given publisher on one node must match the Topic of a subscriber on any other node. Failing to do so, will cause the communication to not take place. A Topic is comprised of a name and a type. In general, a topic can be temperature, sound, humidity, pressure, vibration, motion, light, etc.

DataWriter: this entity is used by the application to publish samples on a topic. Once a DataWriter is defined, a DDS application uses it to do the “write” operation in the publishing phase. It is important to point out that each data writer is bound to a particular topic.

Publisher: a publisher has the full responsibility for taking the published samples and sending them out to the DDS domain. In addition, any publisher works as controller to the data writers. By setting the DDS-QoS behavior for a publisher, all the DataWriters in that publisher’s group will automatically have these settings.

DataReader: this entity is used to take samples from the subscriber and deliver them to the DDS application. Like the *DataWriter*, each data reader is bound to a particular topic.

Subscriber: a subscriber has the responsibility to receive the samples from the publisher and pass them to any relevant data readers that are connected to it. Also, it works as a controller to manage all its DataReaders. Again, setting of QoS parameters of a subscriber will apply to all *DataReaders* in the *Subscriber's* group. The Figure 4.1 shown below illustrates the previous terms and presents the basic components in the implementation of all applications.

In the context of a sensor network application scenario, publisher application will contain a large number of data-writers and each one of them presents a sensor node. In the subscriber side, each subscriber will contain a small number of data-readers and each one of them presents the sink node or the base-station in WSNs.

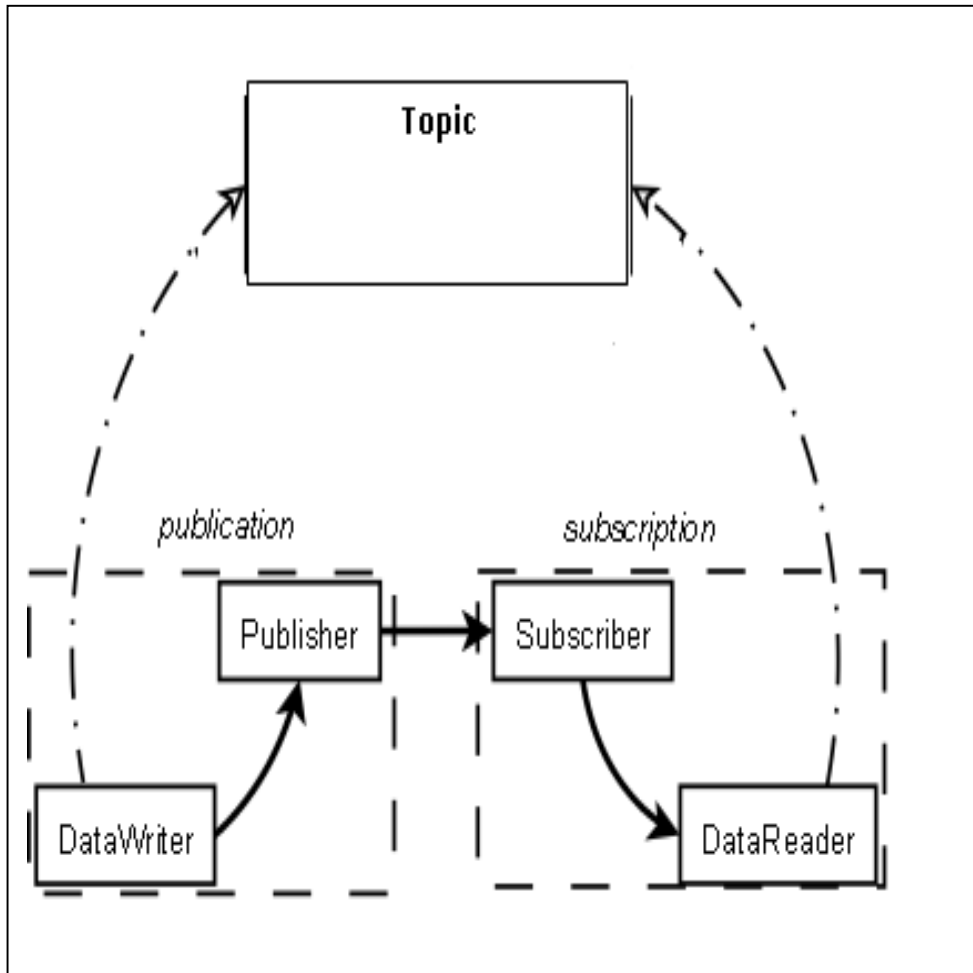


Figure 4.1 this diagram shows the basic and main components that are implemented in all tests. RTI-DDS is defined as publish and subscribe service. In the publication stage, a publisher and a DataWriter send messages to one or more subscribers that include DataReader and a subscriber.

4. Throughput Tests:

Throughput, i.e., number of received samples per unit of time (such as second), is an important performance parameter for any kind of network and distributed system. The primary goals of our throughput tests are to measure how RTI DDS can handle a large number of subscribers and how different communication models (*e.g.*, unicast and multicast protocols) can affect performance. Also, throughout these tests, the investigation of the usage of QoS specified in DDS middleware standard by Object Management Group (OMG) is pointed out.

Scenarios that have been implemented so fare:

4.1 One-to-One Throughput Test

- One publisher application node that has 1000 data-writers as sensor nodes and it runs on Core1.
- One subscriber application node that has 1 data reader and it runs on Core 2.
- Core1 and Core 2 are connected through ad hoc wireless network by having 802.11g/54 Mbps wireless USB adapter on each one of them.
- Size of sample from 8 bytes to 1024 bytes.

- Publisher application node transmits at constant rate of 1000 samples/sec (Frequency = 1000 HZ).

4.2 One-to-Many Throughput Test

- One publisher application node that has 1000 data-writers as sensor nodes and it runs on Core1.
- 3 subscriber application nodes run on Core 2, Core 3 and Core 4, each one has one data-reader.
- Core1, Core 2, Core 3 and Core 4 are connected through ad hoc wireless network by having 802.11g/54 Mbps wireless USB adapter on each one of them.
- Size of sample from 8 bytes to 1024 bytes.
- Publisher application node sends at a constant rate 1000 samples/sec (Frequency = 1000 HZ).

4.3 High Throughput with Reliable Messaging Test

- One publisher application node that has 1000 data-writers as sensor nodes and it runs on core1.
- 3 subscriber application nodes run on Core 2, Core3 and Core4, each has one data-reader.
- Core1, Core 2, Core 3 and Core4 are connected through ad hoc wireless network by having 802.11g/54 Mbps wireless USB adapter on each one of them.

- Size of sample from 8 bytes to 1024 bytes.
- Publisher sensor application node sends at a constant rate of 1000 samples/sec (Frequency = 1000 HZ).
- Using batch and reliable QoS in order to see how the first one can increase the throughput especially for small sample size. In other words, batching can increase throughput when writing small samples at a high rate.

4.4 One-to-Many Throughput Scalability

- One publisher application node that has 1000 data-writers as sensor nodes and it runs on Core 1.
- 15 subscriber applications node run on Core 2, each of these 15 subscribers has one data-reader.
- Core1 and Core 2 are connected through ad hoc wireless network by having 802.11g/54 Mbps wireless USB adapter on each one of them.
- Fixed sample size = 128 bytes because it is the default size in the IEEE 802.15.4 specification.
- Publisher sensor application node sends at a constant rate of 1000 samples/sec (Frequency = 1000 HZ).

4.1 One-to-One Throughput Test Outcomes

A. Experimental set-up

In our network, IP addresses start with 172.16.101, so we refer to nodes by the last part of their IP addresses. The following nodes are used:

120: This core will contain the sensor publisher application. It is important to point out that this application has 1000 datawriters. (2.67 GHz, 3.23G RAM, Windows XP)

123: This core will contain the subscriber application that has one data-reader (2.67GHz, 3.23G RAM, Windows XP)

Nodes 120 and 123 are connected through a wireless communication by having 802.11g/54 Mbps wireless USB adapter on node. Node 120 runs the publisher application to produce the sensor data. The other node (node 123) runs the subscriber application to subscribe for the topic that has been published by the sensor publisher.

See the Figure 4.2 below.

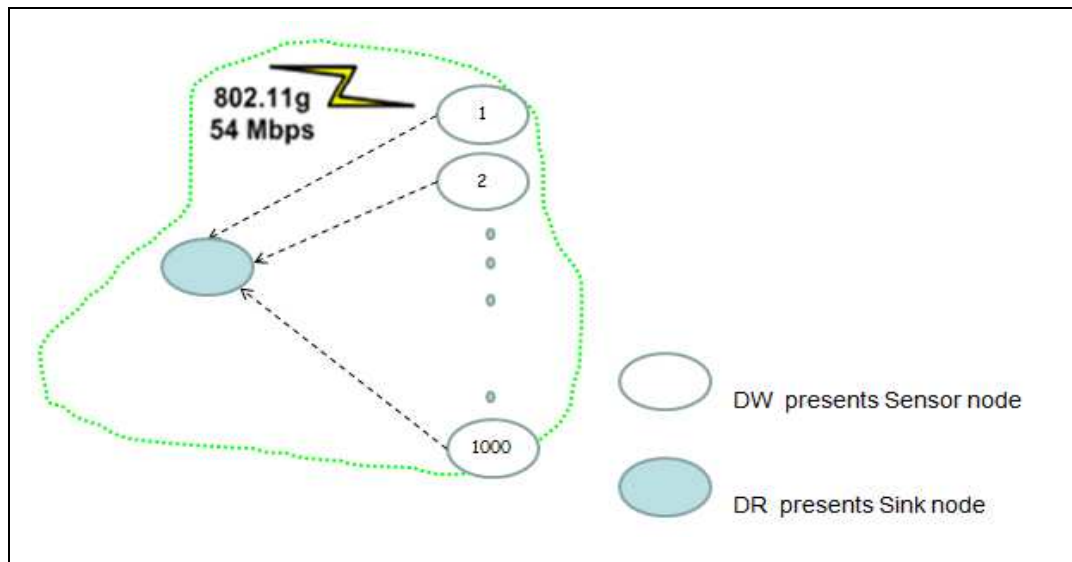


Fig. 4.2: Multi One-to-One Throughput Test

B. Goal

This test was conducted to show the one-to-one (point-to-point) publish/subscribe throughput in terms of received samples in sample per second. In other words, the sensor publisher application sends data where the size varies from 8 bytes to 1024 bytes, and is received by exactly to one subscriber application. The throughput in this test is measured between a sensor producer application and another single consumer application and over a single DDS topic. It is important to highlight that both applications are running on two different machines.

C. Implementation

The implementation of the test is done by running two java applications: one for the publishing node (that has 1000 data-writers as sensor) and the other one for the subscribing node that has one data-reader. Note that the publisher application is sending at the fixed rate of 1000 sample / sec (Frequency = 1000 sample/sec).

In this test, RELIABILITY QoS Policy in the DDS is highlighted in order to see how this kind of QoS can control the communication between the data reader (DR) on the subscriber side and the data-writers (DWs) on the sensor or publisher side.

The connection between the DataWriters (DWs) in the sensor application (node 120) and the DataReaders (DR) in subscriber application (node 123) in terms of reliability can be configured by the user. If RELIABILITY QoS policy is set to Best_Effort, the RTI Data Distribution Service will send samples only once to DR(s). In other words, Best_Effort does not use any resources to monitor the data sent by DWs of

the sensor application to determine whether or not it has been received. Also, it is important to point out that it is the fastest and most efficient way in order to get the latest value of a topic. In other words, since best_effort does not retransmit messages, it is more scalable and timely, therefore it is preferred. However, the delivery is not guaranteed, which means that the data may be lost in the transportation stage over wireless network or even Ethernet.

In the sensor applications that required guaranteed data delivery, RELIABILITY QoS policy is set to RELIABLE. In this mode the RTI Data Distribution Service buffers sent data until all sent samples have been acknowledged by the DataReader. In case of lost samples during the transport stage, RTI will take care by resending them again until they are acknowledged. This kind of connection need extra overhead by using extra packets to monitor and track the status of the sample in the network whether it is acknowledged by the DataReaders or not. Therefore, it needs additional resources to be configured as HISTORY.

The test is done on mainly in two scenarios:

- Setting RELIABILITY QoS Policy in both DataWriters (DWs) in the sensor application (node 120) and DataReader(DR) in the subscriber application (Node123) to RELIABLE.
- Setting RELIABILITY QoS Policy in both DataWriters (DWs) in the sensor application (node 120) and DataReader (DR) in the subscriber application (Node 123) to Best_ Effort.

D. Results and Remarks

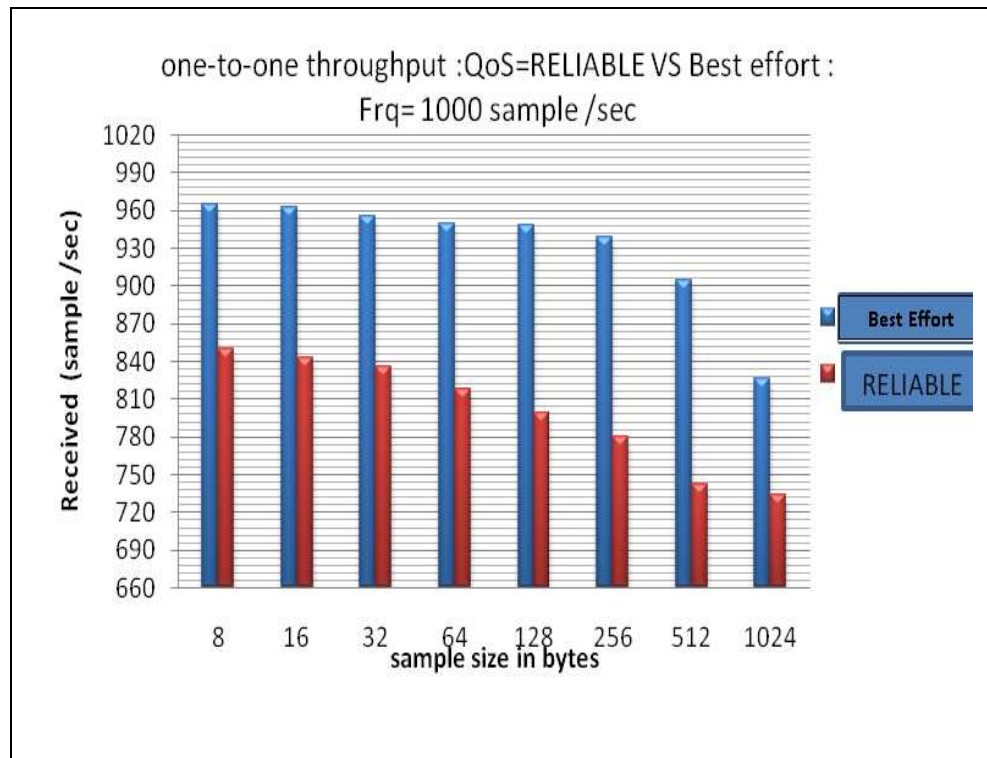


Fig. 4.3: result of One-to-One Throughput

As evident from the above Figure 4.3, the maximum message data is slightly over 960 messages per sec for Best-effort value and around 840 messages per sec for the RELIABLE value. It is an acceptable and expected result to have the RELIABLE mode has small throughput than Best-effort due to the complexity of the RELIABLE operation, i.e. acknowledgment sample must be sent and retransmissions for lost samples are enforced.

As shown above, RTI Data Distribution Service is fully utilizing the available bandwidth. Also, it can be remarked that throughput is limited by the network and

not by the CPU or middleware. Furthermore, the performance of a system depends on the operating system, the networks and how the networks are configured.

4.2 One-to-Many Throughput Test Outcomes (Subscribers on Different Hosts):

A. Experimental set-up

In our network, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the publisher application that has 1000 data-writers. (2.67 GHz, 3.23G RAM, Windows XP).

123: This core will contain the subscriber application 1. It has one data-reader. (2.67 GHz, 3.23G RAM, Windows XP).

121: This core will contain the subscriber application 2. It has one data-reader. (2.67GHz, 3.23G RAM, Windows XP).

173: This core will contain the subscriber application 3. It has one data-reader. (2.67GHz, 3.23G RAM, Windows XP).

Nodes 120, 123, 173 and 121 are connected through a wireless communication by having 802.11g/54 Mbps wireless USB adapter on each node. Node 120 runs the publisher application to produce the sensor data. The other nodes (node 120, 121 and 173) run the subscriber applications to subscribe for the topic that have been published by the sensor publisher application. See the Figure 4.4 below.

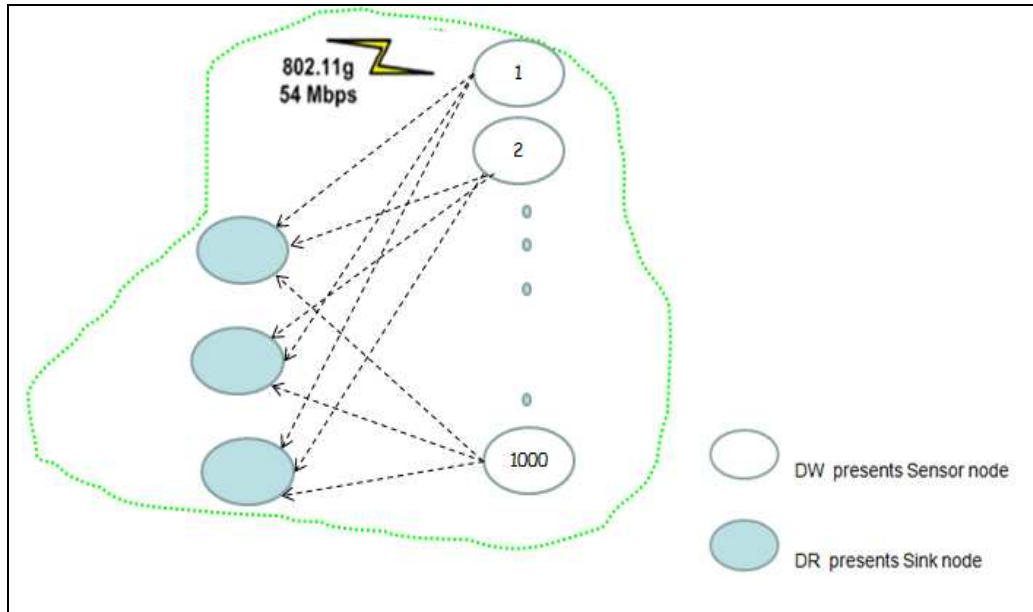


Fig. 4.4: Multi One-to-Many Throughput Test over different

B. Goal

This test is conducted to show the one-to-many (point-to-many point) publish/subscribe throughput in terms of received samples per second (sample/sec). In other words, the sensor publisher application sends data where the size varies from 8 bytes to 1024 bytes and is sent exactly to three subscriber applications. The throughput in this test is measured between a producer application and three consumer applications and over a single DDS topic.

C. Implementation

The test is made by running two java applications: one for the publishing node (that has 1000 data writer as sensor) and the other for the subscribing nodes, each having one data-reader.

The test is done by mainly setting RELIABILITY QoS Policy in both Data DataWriters (DWs) in the sensor application (node 120) and DataReaders(DRs) in the subscriber applications (Node 121, 123 and 173) to RELIABLE.

D. Results and Remarks

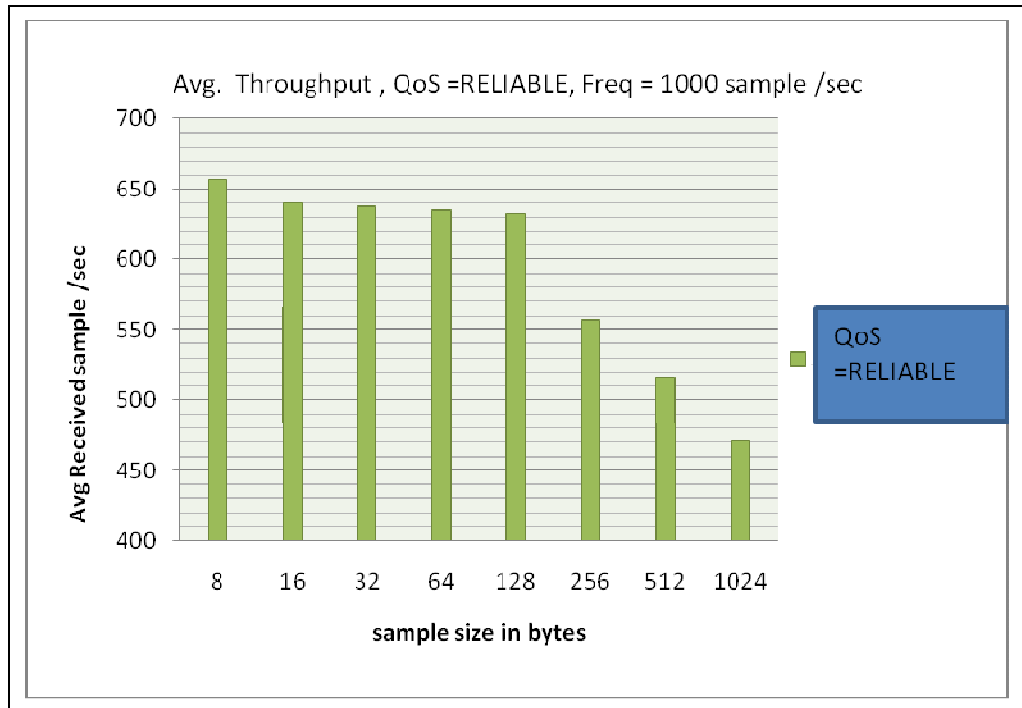


Fig. 4.5: result of One-to-Many Throughput Test over different machines

Sample size	Avg. Received sample/sec (One-to-One)	Avg. Received sample/sec (One-to-Many)
8	846	657
16	839	640
32	836	638
64	824	634
256	780	557
512	742	516
1024	734	471

Table 4.1: One-to-Many Throughput Test over different machines

This table compares this test with one that is done in previous experiments, when we have one-to-one test over QoS= RELIABLE. This is done in order to have an estimation of the difference in the measures caused by increasing the number of subscribers over different machines. For the samples that have sample size, the difference is around 190 samples. However, for large size the difference becomes around 225 samples. It is large for 1024 bytes, it is 263 samples. The reason why there is a difference between this test and the previous test is that, in the second one, traffic is send from one host to another 3 machines. In other words, copy of every sample must be sent to every single subscriber host.

4.3 High Throughput with Reliable Messaging Test Outcomes:

A. Experimental set-up

In our network, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the sensor publisher application. It has 1000 data-writers (2.67 GHz, 3.23G RAM, Windows XP).

121: This core will contain the subscriber application 1 (2.67 GHz, 3.23G RAM, Windows XP).

123: This core will contain the subscriber application 2 (2.67GHz, 3.23G RAM, Windows XP).

173: This core will contain the subscriber application 3 (2.67GHz, 3.23G RAM, Windows XP).

Nodes 120, 123, 127 and 173 are connected through a wireless communication by having 802.11g/54 Mbps wireless USB adapter on each node. Node 120 runs the publisher application to produce the sensor data. The other nodes (node 121, 123 and 173) run the subscriber application to subscribe for the topic that has been published by the sensor publisher.

B. Goal

As we previously mentioned, some sensor applications required a reliable messaging. Therefore, the middleware must keep monitoring the delivery of the data whether or

not it was received by the subscribing applications. Also, in case of data loss, retransmission must be done.

The section (4.3) points out the key QoS settings that are needed in order to achieve strict reliability. Not only that, but it can also be seen how we can set RTI Data Distribution Service QoS Profile in order to get high throughput for reliable data.

C. Implementation

To achieve strict reliability for the critical sensor application we must do the following:

- Setting the RELIABILITY QoS Policy of Data Writer (DWs) of the sensor application and Data Reader (DR) of the subscriber node(s) to RELIABLE.

Some sensor applications produce a large number of small messages at high rate. In such a case, there will be a measurable overhead in transmitting each sample alone in a network especially if the application needs a strict reliable communication. From here, the idea of using the Batch QoS policy in DDS is used. This kind of QoS is helpful to the system in managing many samples together as a group and then sends them as a group to the network. In other words, the batching QoS, with reasonable size because sensor nodes have limited memory sizes, can take advantage of the efficiency of sending larger packets, thus increasing the throughput.

D. Results and Remarks

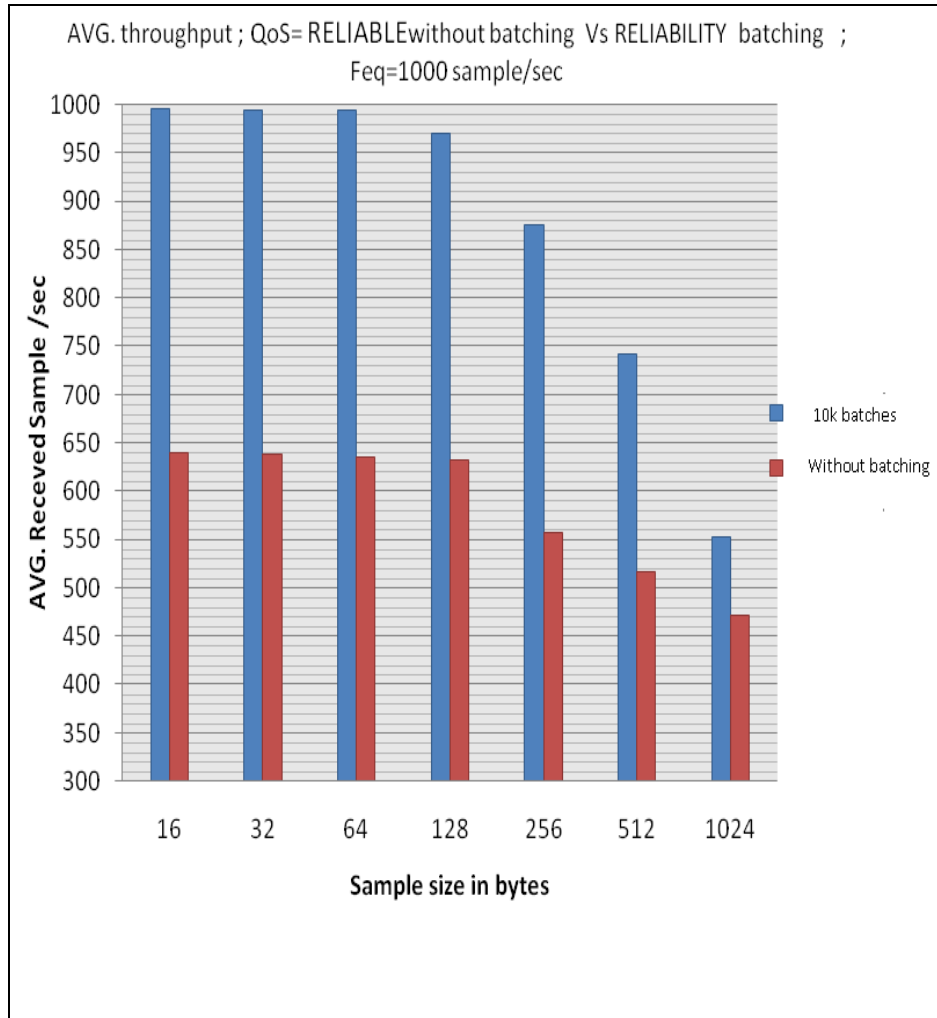


Fig.4.6: result One-to-Many Throughput: with & without batching QoS

As you can see in the result, it is clearly shown that batching service significantly increases the throughput for the small sample size at a higher rate. In general, this kind of QoS policy is used to make communication overhead less in the reliable mode of small size samples.

Batching service collects many smaller samples in a batch to be sent in a single packet. This surely reduces the communication and the acknowledgments flow, thus increasing the throughput in terms of samples per second. See the table below:

Size in bytes	Throughput without batching	Throughput with batching
16	640	996
32	638	994
64	635	994
128	632	970
256	557	876
512	516	742
1024	471	553

Table 4.2: result One-to-Many Throughput: with & without batching QoS

4.4 One-to-Many Throughput Scalability – (Subscribers On Same Host)

A. Experimental set-up :

In our network, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the sensor publisher application. It has 1000 data-writers (2.67 GHz, 3.23G RAM, Windows XP)

123: This core will contain the subscriber applications. (2.67 GHz, 3.23G RAM, Windows XP)

Nodes 120 and 123 are connected through a wireless communication by having 802.11g/54 Mbps wireless USB adapter on each node. Node 120 runs the publisher application to produce the sensor data. The other node (node 123) runs from 1 to 15 subscriber applications to subscribe for the topic that have been published by the sensor publisher. See the Figure 4.7 below.

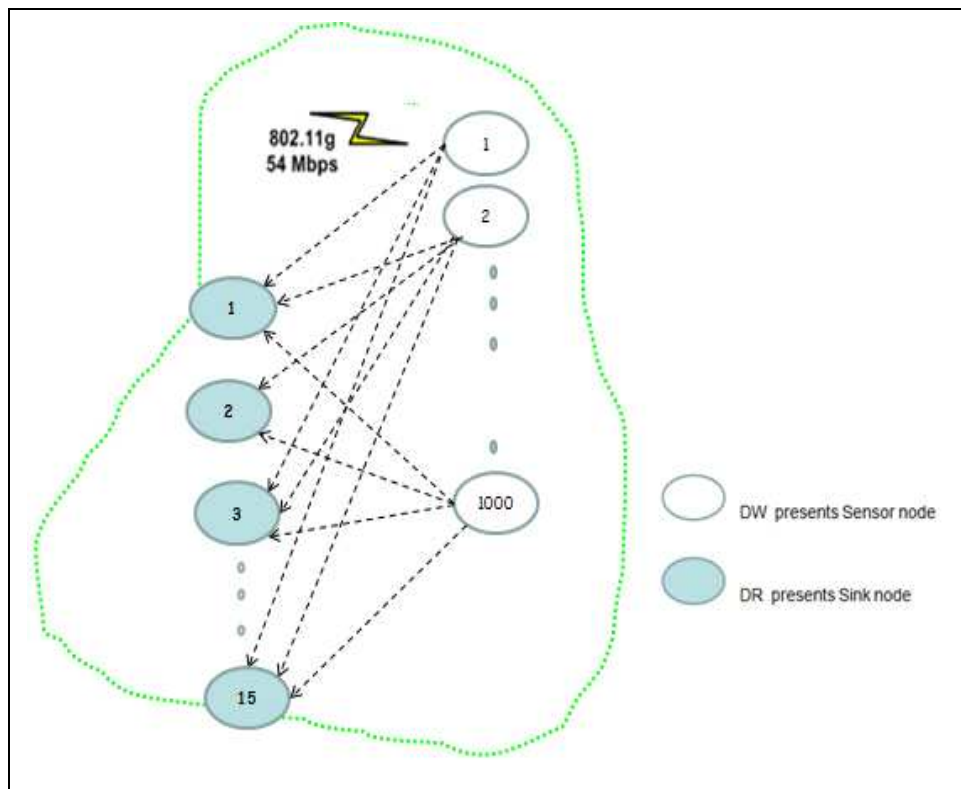


Fig. 4.7: Multi One-to-Many Throughput scalability test

B. Goal

The aim of this scenario is to examine the behavior of transferring data with a fixed rate over a different number of subscriber applications. This test was done to see how the RTI will behave in a similar situation in sensor network. In addition, this scenario test will measure the capability of the RTI in handling multiple of subscribers.

C. Implementation

We used a publisher application and from 1 to 15 subscriber applications over a single topic with fixed sample size of 128 bytes since it is the default size in the

IEEE 802.15.4. It is important to highlight that each sample produced by the sensor application is consumer by 1 to 15 subscriber applications.

The test is done mainly in three scenarios:

- A. Setting RELIABILITY QoS Policy in both DataWriters (DWs) in the sensor application (node 120) and DataReaders (DRs) in the subscriber applications (node 123) to RELIABLE without batching.
 - B. Setting RELIABILITY QoS Policy in both DataWriters (DWs) in the sensor application (node 120) and DataReader (DRs) in the subscriber applications (node123) to Best_Effort.
 - C. Setting RELIABILITY QoS Policy in both DataWriters (DWs) in the sensor application (node 120) and DataReaders (DR) in the subscriber applications (node123) to RELIABLE with 10k batches.
- Number of publisher application = 1, with 1000 DWs
 - Number of subscribers = from 1 to 15
 - Size of the message = 128 bytes
 - Fixed frequency rate =1000 HZ

D. Results and Remarks

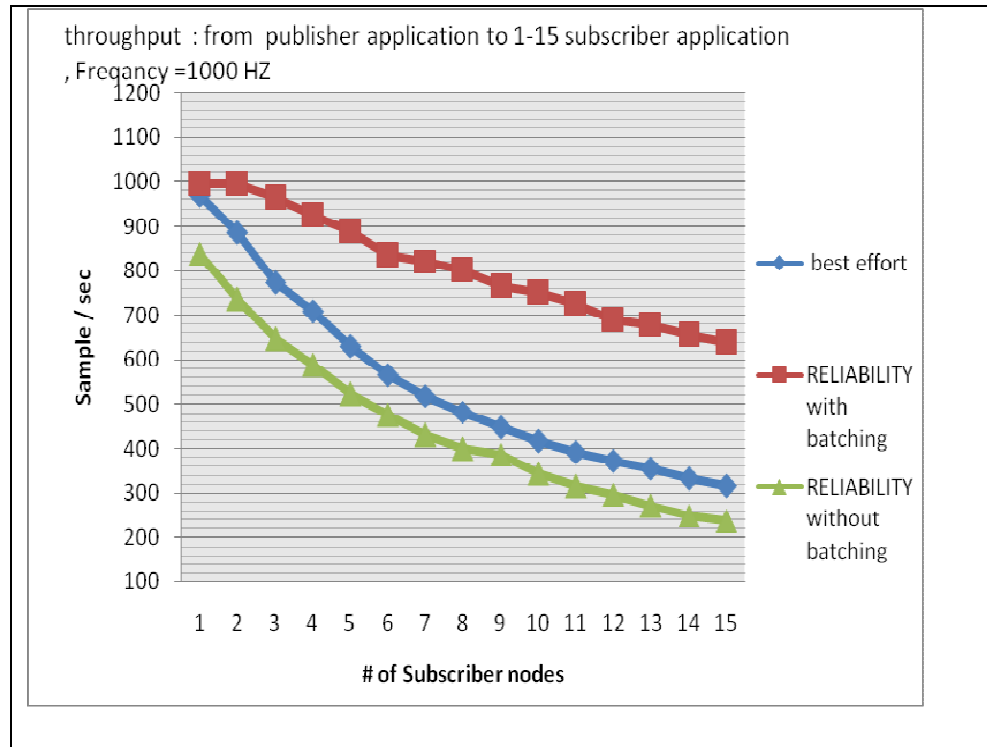


Figure 4.8: result One-to-Many Throughput scalability test

Clearly, Figure 4.8 shows the efficiency of RTI's reliable uni-cast protocol for one-to-many publish/subscribe messaging and real-time data distribution. A producing sensor application was used to send a stream of 128-byte messages at the rate of 1000 HZ to up to 15 consumer nodes, each of these subscribers running on the same core.

This illustrates that RTI Data Distribution Service middleware with batching has the best scalability over the other two (Reliable and best effort). This is because batching

groups many smaller samples in a batch to be sent in a single packet which will reduce the communication overhead.

CHAPTER 5

RTI ROUTING SERVICE FOR ISOLATION AND

CLUSTERING

RTI Routing Service is an important component of RTI Data Distribution Service that is mainly used to integrate separated and isolated systems. This component works to scale DDS applications across domains, Local area Network LANs, WLAN and wide area network WANs. It is important to highlight that RTI Routing Service can work as a bridge between two or more DDS applications. This is done by exchanging the data between DDS systems. Therefore, RTI Routing Service helps in integrating a new DDS application with a legacy one. Furthermore, it can work as an interface between non-DDS and DDS systems.

It is known that Data Distribution Service (DDS) applications can communicate with other applications if they are in the same domain. However, with RTI Routing Service, applications in different domains can communicate by sending and receiving data across domains. In addition, this component of RTI Data Distribution Service is able to transform and filter the transferred data. Also, applications with different data structures are able to communicate across domains by using RTI Routing Service

because this component has the capabilities to change the data's type. Moreover, it is used as a controller to the system by deciding which data is to be sent.

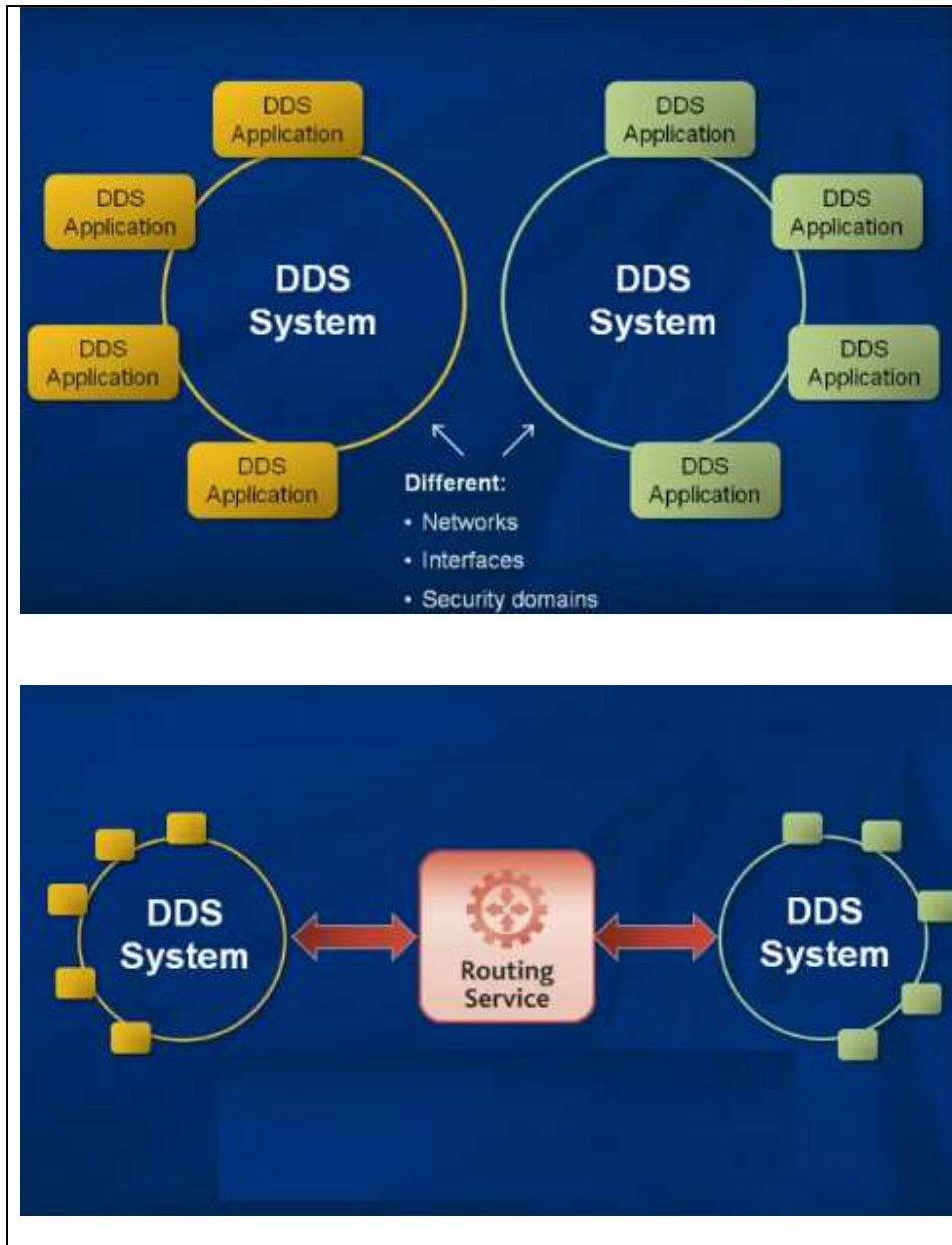


Fig. 5.1: RTI Routing Service for DDS

RTI Routing Service is used to pass data from one domain to another. Also, it is used to specify any desired data filtering and transformations. It is important to point out that no changes are required in the Data Distribution Service DDS applications.

The important benefits of RTI Routing Service are as follows:

1. It reduces the time and effort that are needed to integrate and scale DDS applications across domains. It can scale DDS real-time publish/subscribe (RTPS) data-distribution without making any changes to the existing DDS applications. With this routing service, an existing DDS application can be easily integrated with a new one even if its data structure is different for the old one.
2. It is used to build modular systems out of the existing systems: RTI Routing Service allows dividing the DDS system into public or private domains. Also, it can be used to see certain “global topics” across domains.
3. It supports a secure deployment across multiple DDS applications.
4. It can be used as bridges to integrate DDS and non-DDS systems.
5. It is used to manage and control the evolution of the data at the subsystem level: RTI Routing Service is able to transform data on the fly, changing topic, working as a bridge to link different kinds of DDS applications.

5.1 RTI Routing Service for Throughput Scalability Test

Outcomes:

The goal of these tests are to see the advantages of using the RTI routing service after we correctly program it for transferring data with a fixed rate across domains and a over different number of subscriber applications. This test is done to see how the RTI will behave in a similar situation in sensor networks when the network is divided into clusters and is in an isolated sub networks. These networks might be different in data structure and security domains. In other words, this test scenario will clearly point out how the RTI can act when the sensor application in isolated network from the subscriber application, which usually is the case in wireless sensor networks.

Note that, in this test, we do four different scenarios based on the location of running the RTI routing service, as follows:

Test A: running RTI routing service on the same core (Core 1) that runs the sensor publisher application.

Test B: running RTI routing service on the same core (Core 2) that runs the subscriber application.

Test C: running RTI routing service on different core, say (Core 3).

- **Test A : Experimental Set-up**

In our network, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the sensor publisher application. It also runs RTI Routing Service (2.67 GHz, 3.23G RAM, Windows XP).

123: This core will contain the subscriber applications. (2.67GHz, 3.23G RAM, Windows XP).

Node 120 runs the publisher application to produce the sensor data. The other node (node 123) runs from 1 to 50 subscriber applications to subscribe for the topic that have been published by the sensor publisher .See the Figure 5.2 below.

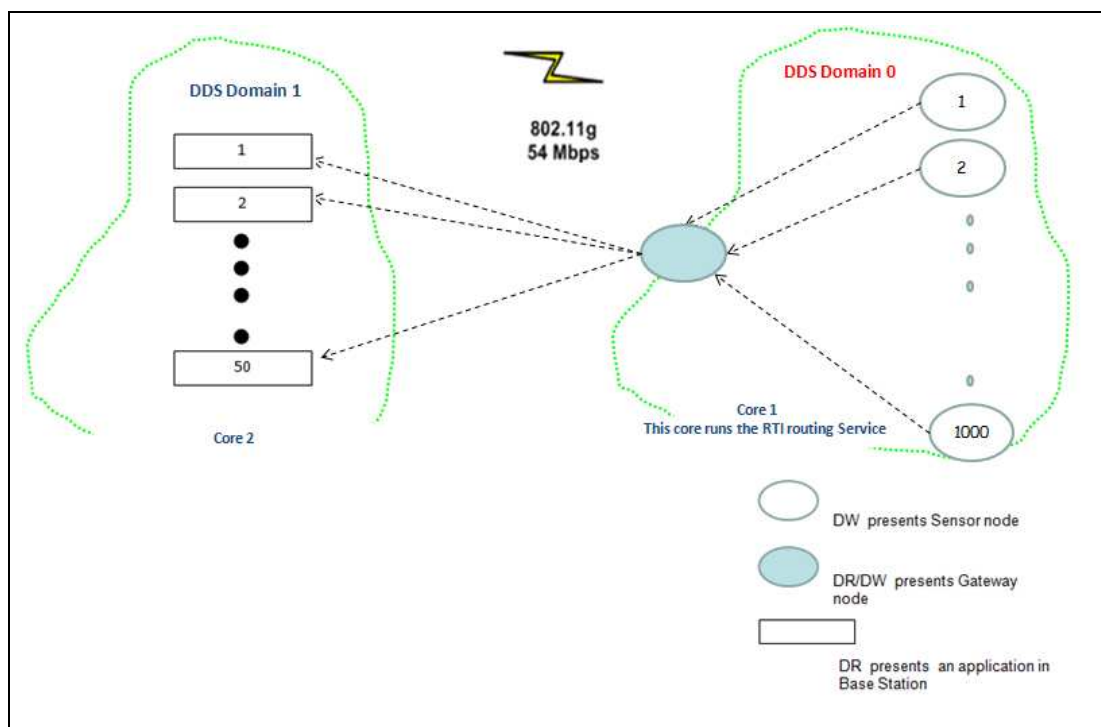


Fig. 5.2: RTI Routing Service in the publisher side

- **Test A : Implementation**

As shown in the Figure above, this implementation is exactly the same as Test A except that the RTI routing service runs on the same core of the sensor publisher application.

- Number of publisher application = 1, it has 1000 DWs ,runs on domain 0
- Number of subscribers = from 1 to 50, run on domain 1
- RTI Routing service runs on the same core as the publisher application
- Size of the sample = 128 bytes
- Fixed frequency rate = 1000 HZ

- **Test B: Experimental set-up :**

In our network, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the sensor publisher application which has 1000 data-writers as sensor generator (2.67 GHz, 3.23G RAM, Windows XP).

123: This core will contain the subscriber applications. It also runs RTI routing service (2.67 GHz, 3.23G RAM, Windows XP).

Node 120 runs the publisher application to produce the sensor data. The other node (node 123) runs from 1 to 50 (not 15 applications, it is 50 to show the scalability issues) subscriber applications to subscribe for the topic that have been published by the sensor publisher. It is important to highlight that both of these applications are on different domains .See the Figure 5.3 below.

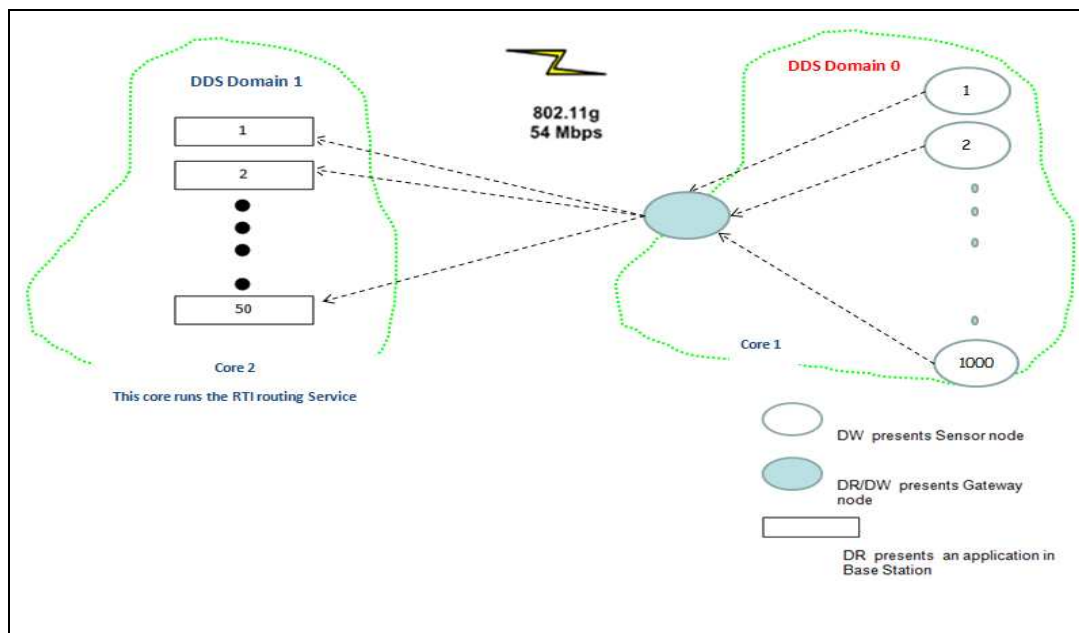


Fig. 5.3: RTI routing service in the subscriber side

- **Test B: Implementation**

As shown in the Figure above, we used a sensor publisher application that runs on Core 1 and domain 0. Also, (from 1 to 50) a subscriber application that runs on Core 2 and it is on domain 1. Subscriber applications subscribe on a single topic with fixed sample size of 128 bytes since it is the default size in IEEE 802.15.4. It is important to highlight that each sample produced by the sensor application is routed by one RTI routing service. 1-to-50 subscriber applications subscribe for the same topic that is published on domain 0 and routed from that domain (domain 0) to domain of the subscribers through the RTI routing service. Note that, the sensor publisher application works on domain 0 and all the subscriber applications run on domain 1. The RTI routing service runs on the same core of the subscriber applications. In this test we set RELIABILITY QoS Policy in both Data DataWriters (DWs) in the sensor application (node 120) and DataReaders (DRs) in the subscriber applications (node 123) to Best_Effort.

- Number of publisher application = 1, it has 1000 DW , runs on domain 0
- Number of subscribers = from 1 to 50, each subscriber has one data reader , run on domain 1
- RTI routing service runs on the same core as the subscribers
- Size of the sample = 128 bytes
- Fixed frequency rate = 1000 HZ

- **Test C : Experimental Set-up**

In our networks, IP addresses start with 172.16.101 so we refer to nodes by the last part of their IP address. The following nodes are used:

120: This core will contain the sensor publisher application. (2.67 GHz, 3.23G RAM, Windows XP).

123: This core will contain the subscriber applications. (2.67 GHz, 3.23G RAM, Windows XP).

173: This core runs RTI routing service. (2.67 GHz, 3.23G RAM, Windows XP).

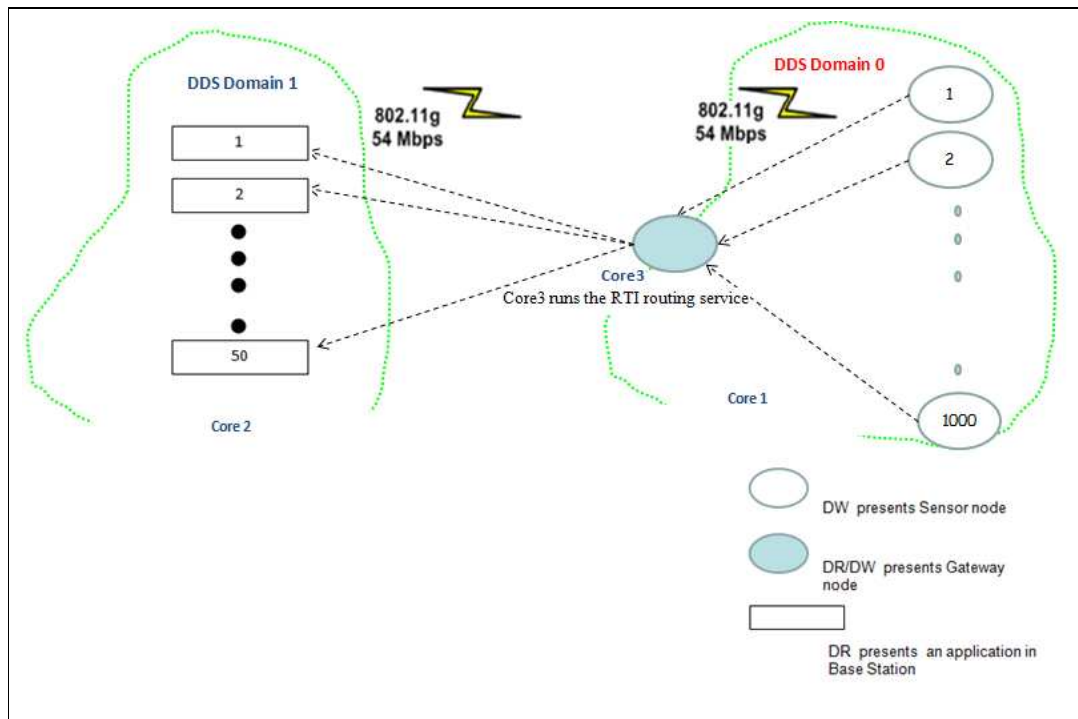


Fig. 5.4: RTI routing service in the 3rd node

- **Test C : Implementation**

As shown in the Figure above, this implementation is exactly the same as Test A except that the RTI routing service runs on the third core (Core 3).

- Number of publisher application= 1, with 1000 DWs , runs on domain 0
- Number of subscribers = from 1 to 50, run on domain 1
- RTI routing service runs on (Core 3).
- Size of the sample = 128 bytes
- Fixed frequency rate = 1000 HZ

- **Remarks of the previous results:**

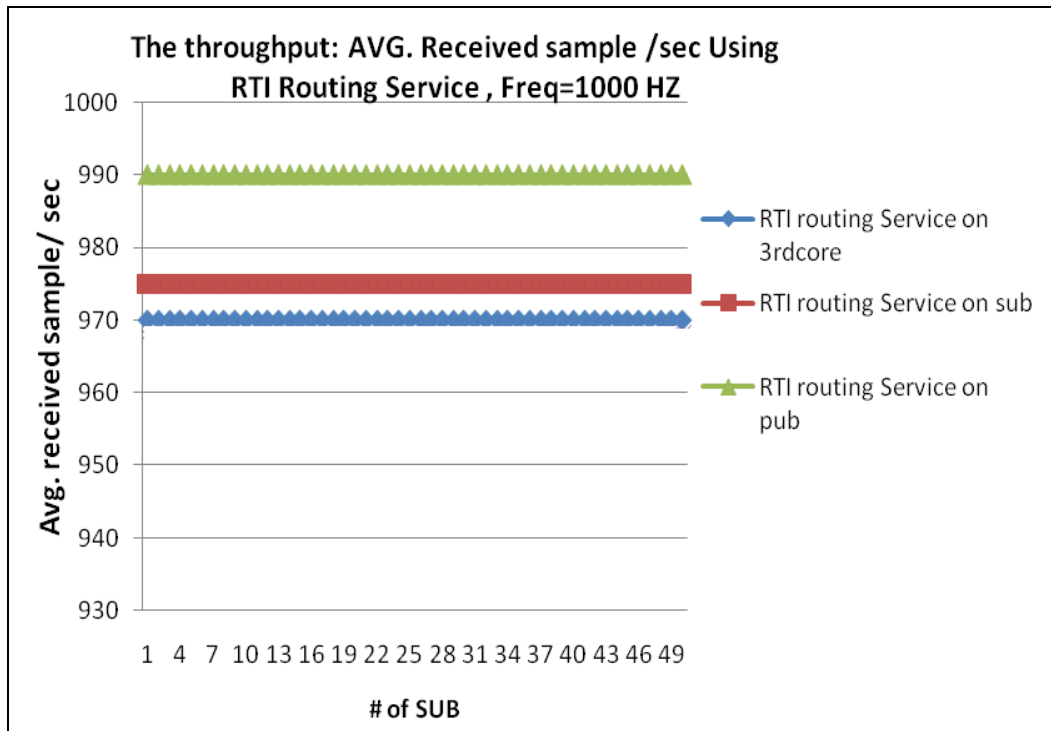


Fig.5.5: Clustering and Isolation cases using RTI routing service

We can notice that throughput (average number of received sample /sec) is invariant to the number of subscribers. This is perfectly normal if you consider the behavior of IP Multicast. The key idea behind that is that RTI routing service uses IP Multicast protocol to distribute samples, this is why we have a throughput (average number of received sample /sec) is invariant to the number of subscribers.

Normally, the publisher sends one datagram per each subscriber, so increasing its number causes an increase in the datagram to be sent. In IP Multicast, the publisher sends just one datagram even if there is more than one subscriber.

In order to clearly point out this important point, we compare what we got in Figure 5.6 with what we got in this test. If we do so, we can understand the behavior of the RTI Data Distribution Service in both cases:

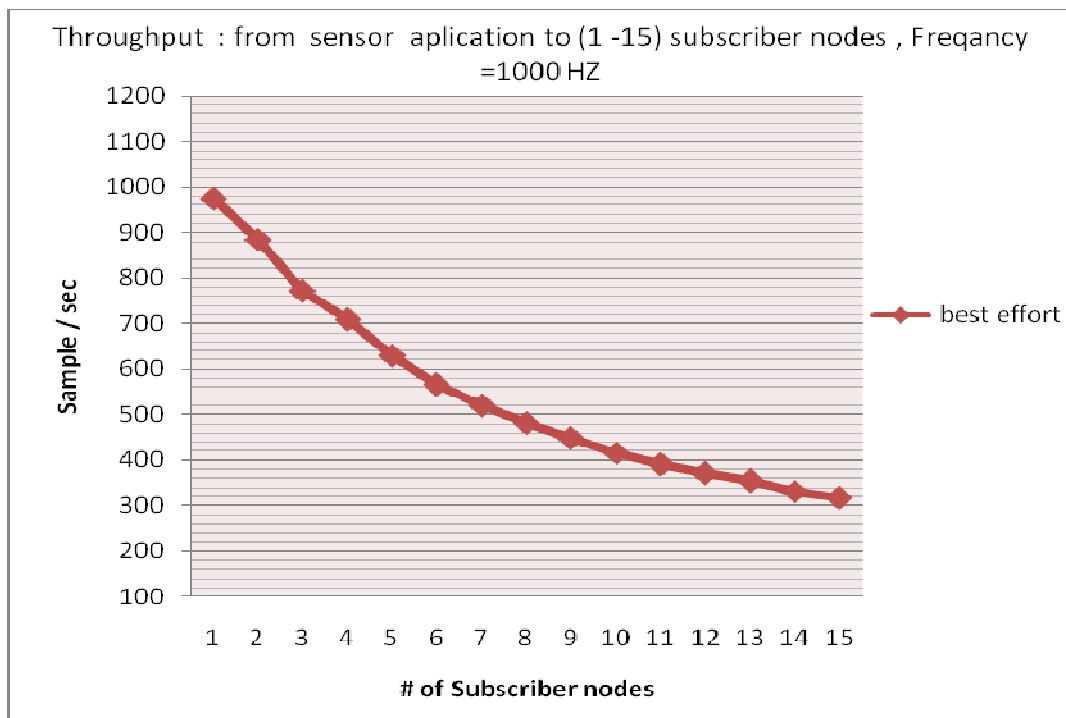


Fig5.6: Result of one-to-many test without using RTI routing

In above Figure 5.6, the RTI Data Distribution Service middleware uses Unicast protocol. In this case, we have seen that as we increase the number of subscribers, the throughput (average number of received sample/sec) significantly decreases. This is clearly because the network bandwidth is wasted since Unicast protocol generates a separate copy of the data to each single subscriber. This obviously means that Unicast does not easily scale to a large number of recipients.

For example: if the number of subscriber applications is 50, then the publisher sensor node will transmit 50 copies of the data and the network forwards them to each subscriber. In other words, the sensor publisher application sends multiple copies of the data, one copy for each subscriber. The following Figure 5.7 shows how the transferring of the data is made by the RTI Data Distribution Service middleware in test of Figure 5.6.

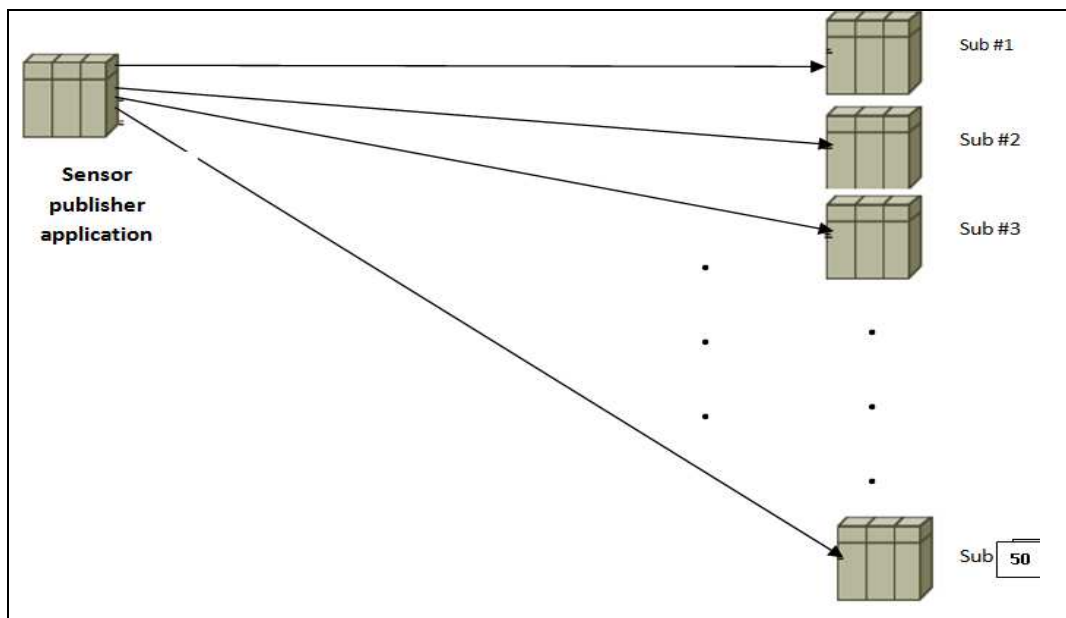


Fig5.7: one-to-many test without using RTI routing service

On the other hand, the RTI routing service uses multicast transmission protocol in order to send a single multicast sample addressed to all subscribers. It provides efficient communication and transmission, optimizes performance, and enables truly distributed applications.

Therefore, if the number of subscriber applications is 50, then the publisher sensor node will transmit only one copy of the data and RTI routing service does the replication for each subscriber.

It is important to point out that the RTI routing service that uses IP multicast protocol allows the sensor publisher application to send to multiple subscribers simultaneously. The following Figure 5.8 shows how the transferring of the data is made by the RTI routing service.

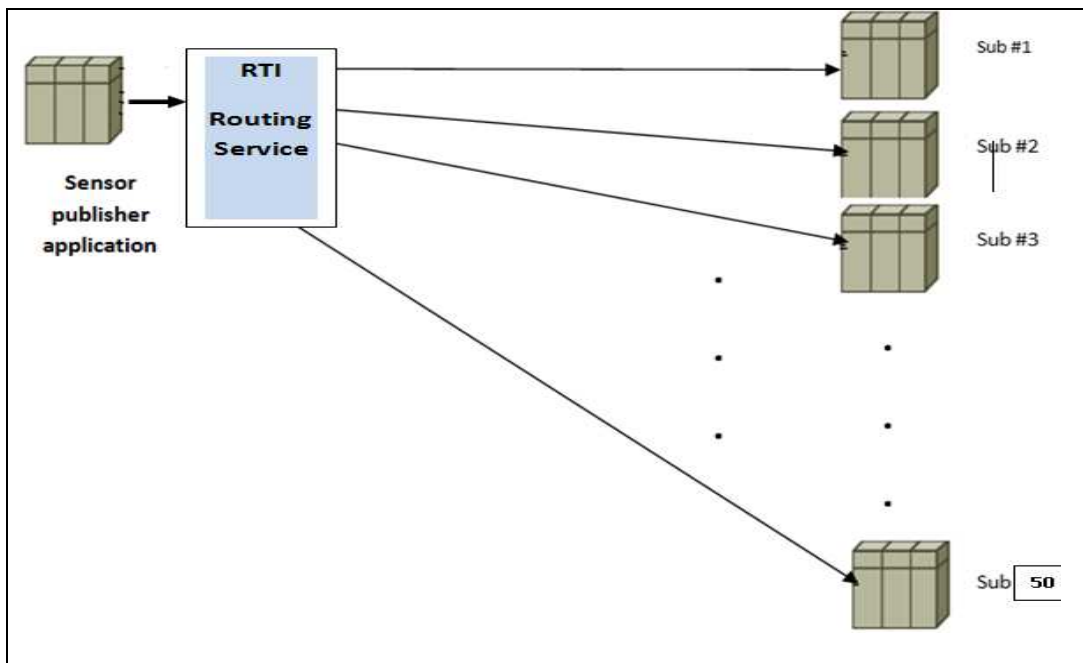


Fig5.8: One-to-Many test using RTI routing service

At the end of this test, it is extremely essential to point out that RTI routing service gives many advantages in a one-to-many environment. This routing service clearly enhances the efficiency because multiple streams of data, which are generated by the

sensor publisher application, are replaced with a single transmission. In other words, the available network bandwidth is utilized more efficiently than the previous test. Furthermore, it optimizes the performance since less number of copies of data requires forwarding and processing. In addition, it highly supports distributed applications.

In this test, the Figures shown above show the one-to-many (point-to-many point) publish/subscribe throughput in terms of sample per second (sample/sec). The sensor publisher application sends samples of fixed size (128-bytes) to up to 50 subscribers, each running on the same core.

As can be seen in Figure 5.5, the number of subscribers has no significant impact on the throughput. Obviously, this means that RTI Data Distribution Service is highly scalable in terms of the number of subscribers supported on a given topic. Moreover, it is important to mention that this figure shows the efficiency of RTI routing service for real-time and mission-critical WSN applications.

5.2 Energy Consumption Estimation in Wireless Sensor

Networks

In WSNs, the energy consumption is one of the important issues to prolong the network lifetime. Unlike wired and wireless networks, WSNs have a main energy issue because wireless sensor nodes are powered by batteries with a limited capacity and they cannot be charged after being deployed. Therefore, they are prone to failures and this will cause the whole network to fail. Many researchers proposed different ways to reduce energy consumption by limiting transmission/reception of data sample as much as possible.

The goal of this section is to estimate the total energy consumed by the network in the previous tests mentioned in section 5.1. In our estimation, it is important to highlight that we assume no node failures.

In the analytical model used to estimate total energy consumed in our network, it is assumed that there are N nodes distributed uniformly. If there are c clusters, there are on average N/c nodes per cluster (one cluster head and $[(N/c)-1]$ non-cluster head nodes). Each cluster-head consumes energy receiving signals from the other nodes, aggregating the signals, and transmitting the aggregate signal to the Base Station. Since the Base Station is far from the nodes, multipath model is used the multipath model (d^4 power loss) [36].

Therefore, the energy dissipated in the cluster head node is

$$E_{CH} = L [n E_{elec} + n E_{DA} + E_{elec} + \epsilon_{mp} (d_{BS}^4)] \quad (5.1)$$

Where, L is the number of bits in each data sample k is the number of bits in each data message, ϵ_{mp} is the coefficient of amplifier energy in multi-path model, d_{BS} is the distance from the cluster head node to the BS, E_{DA} is data aggregation. Because the distance to the cluster head is small, so the energy consumed follows the Friss frees-space model is used to model the power loss (d^2 power loss) [36]. Thus, the energy used in a non-cluster head node is

$$E_{node} = L [E_{elec} + \epsilon_{fs} (d_{CH}^2)] \quad (5.2)$$

Where, d_{CH} is the distance from the node to the cluster head and ϵ_{fs} is coefficient of amplifier energy in free-space model.

Therefore, the total energy consumed in our network is as follows:

$$E_{total} = c. [E_{CH} + (N/c-1) E_{node}] \quad (5.3)$$

Because we have only one single cluster in the network, the energy dissipated in a cluster is given by

$$E_{total} = [E_{CH} + (N-1) E_{node}] \quad (5.4)$$

In our work, we assume a simple model where the values of the constant communication energy parameters were taken as in Table 5.1:

Parameter	Short Description	Value
E_{elec}	Electronics energy	50nJ/bit
E_{DA}	Energy of data aggregation	5nJ/bit
ϵ_{fs}	Amplified transmitting energy using free space	10pJ/bit/ m ²
ϵ_{mp}	Amplified transmitting energy using multipath	0.0013pJ/bit/ m ⁴
N	Number of nodes in the network	1001 nodes

Table 5.1 PARAMETERS VALUES USED IN THE SIMULATION

Also, we assumed Core1 and Core 2 are located on (0, 0) and (50, 0) respectively. Also, we assumed if the cluster head and non-cluster head nodes are on same core, then the cost of the distance is one. Otherwise, d_{CH} will be based on its locations. In other words, for Test A, B and C, $d_{CH}=1, 50, 25$ respectively. Also, $d_{BS}=50$ for Test A, $d_{BS}=1$ for Test B and $d_{BS}=25$ for Test C.

Scenarios that have been implemented:

- **Test A:** Cluster head and non-cluster head nodes are on same core (Core 1)

- **Test B:** Cluster head and non-cluster head nodes are located on Core 1 and Core 2 respectively. Core 2 runs base station application as subscriber.
- **Test C:** Cluster head and non-cluster head nodes are located on Core 1 and Core 3 respectively Core 3 is located on (25, 0) between core 1 and core 2). Core 2 runs base station application.

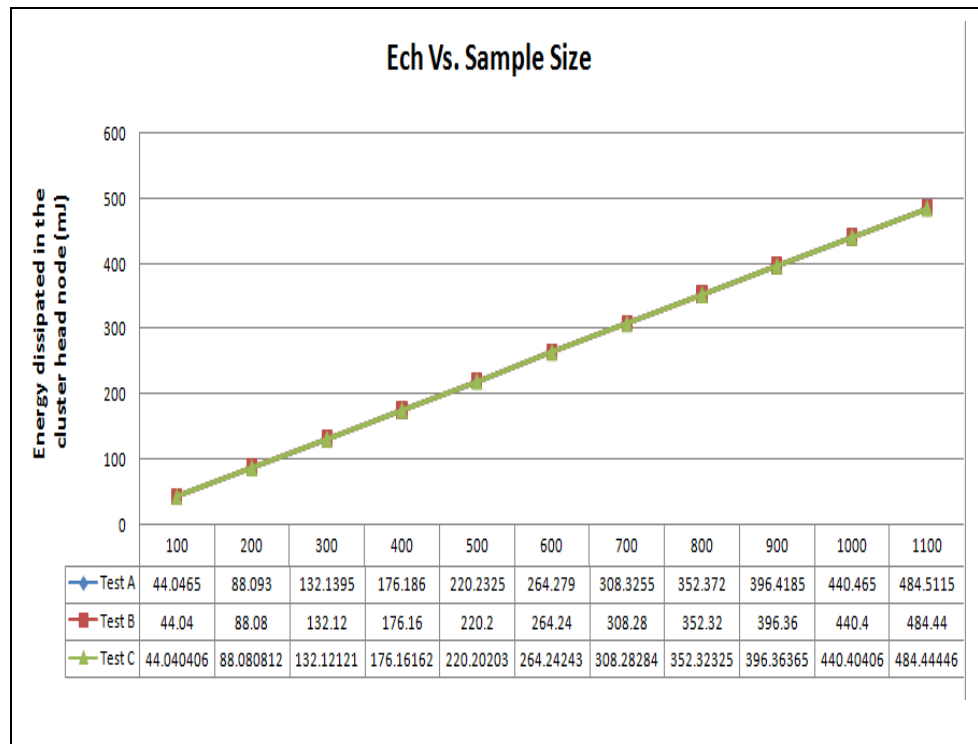


Fig5.9: Energy used in each non-cluster head node vs. Sample size

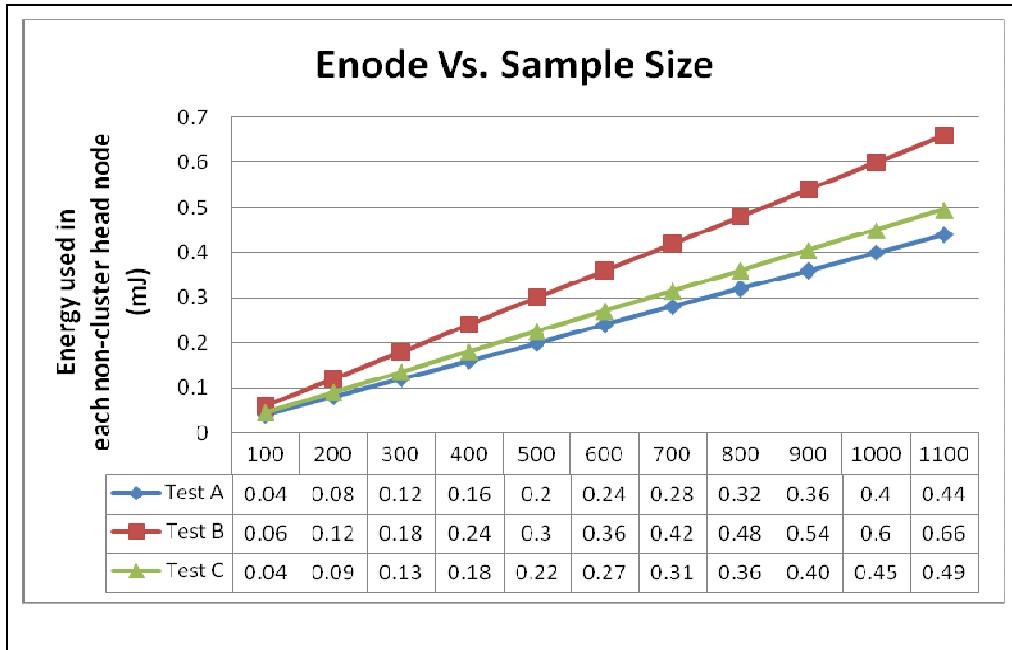


Fig5.10: Energy dissipated in the cluster head node vs. sample size

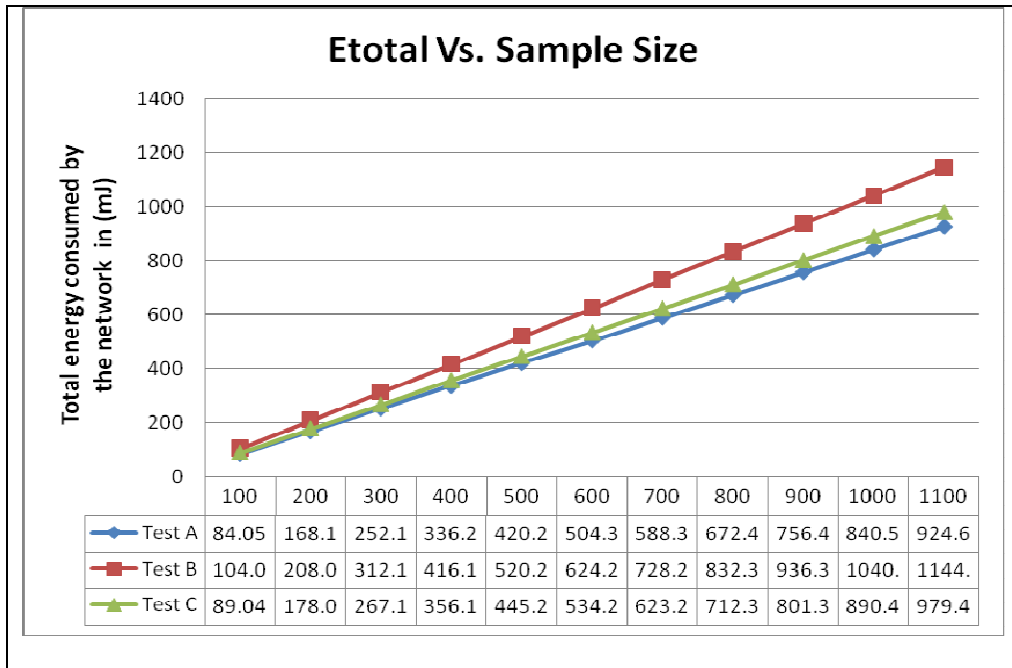


Fig5.11: Total energy consumed by the network vs. sample size

In Figure 5.11, we find that the energy consumptions of all test are proportional to packet size. As can also be seen in this Figure, running the cluster head and non-cluster head nodes on same core has the lowest total energy consumed (E_{total}) by the network. Obviously, this is because if both of them running on the same core, d_{CH} will be small. Therefore, the energy used in a non-cluster head node (E_{node}) will be small. However, for Test B, Cluster head and non-cluster head nodes are located on Core 1 and Core 2 respectively. That means, d_{CH} will have the largest value. Therefore, this test has the largest E_{node} . As result, it has the largest total energy consumed by the network.

CHAPTER 6

LATENCY EXPERIMENTAL SET-UP AND RESULTS

In this test, we calculated the roundtrip time (RTT) between the sending of a message and reception of an acknowledgment from the subscriber. In all latency test scenarios, the roundtrip latency is calculated as the with publisher sensor application send at constant rate 1000 sample/sec (Frequency = 1000 HZ). Similarly to the last throughput test, the publisher sends sample with fixed size to one or more subscriber applications.

During this test, an echo method is used in order to calculate the round trip time (RTT). It is important to highlight and point out that both the sensor publishing node and publisher node are running in identical machines. If we want to estimate “end-to-end” latency, it can be estimated as,

$$T_{\text{end-to-end}} < \text{round trip time (RTT)}$$

For this test (1-to-many latency test scenarios), the echo method is called by the last reader that receive the data from data writer. In other words, RTI-DDS writer will send sample to the readers in a given specific order.

Scenarios that have been done in this phase:

6.1 One-to-One round trip time (RTT) Test

- One publisher sensor application node runs on core1.

- One subscriber application node runs on core2.
- Core1 and Core2 are connected through ad hoc wireless network.
- Size of sample 128 bytes.
- Publisher sensor application node send at constant rate 1000 sample / sec (Frequency = 1000 HZ).

6.2 One-to-Many round trip time (RTT) Test

- One publisher sensor application node runs on core1.
- From 1 to 15 subscriber applications node run on core2.
- Core1 and Core 2 are connected through ad hoc wireless network.
- Fixed sample size = 128 bytes because it is the default size in the IEEE 802.15.4 specification.
- Publisher sensor application node send at constant rate 1000 sample / sec (Frequency = 1000 HZ).

6.1 One-to-One Latency Test Arrangements

A. Experimental set-up

The setup of this experiment is exactly as what we did in test One-to-One Throughput Test.

B. Goal:

This test shows the one-to-one publish/subscribe latency in terms of round trip time RTT. In other words, the sensor publisher application sends data where the size fixed publishing rate 128 bytes and is sent to exactly one subscriber application on another core over a single DDS topic. The applications are running on two different computers.

C. Implementation

The first test will consider one publisher and one subscriber. The message will have a size of 128 bytes and they are sent by the publisher at a constant rate 1000 (Hz) frequency. The test is done on mainly in two scenarios:

- A. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders(DR) in the subscriber application (node 123) to RTI-DDS reliable.
- B. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders (DR) in the subscriber application (Node 123) to Best_Effort.

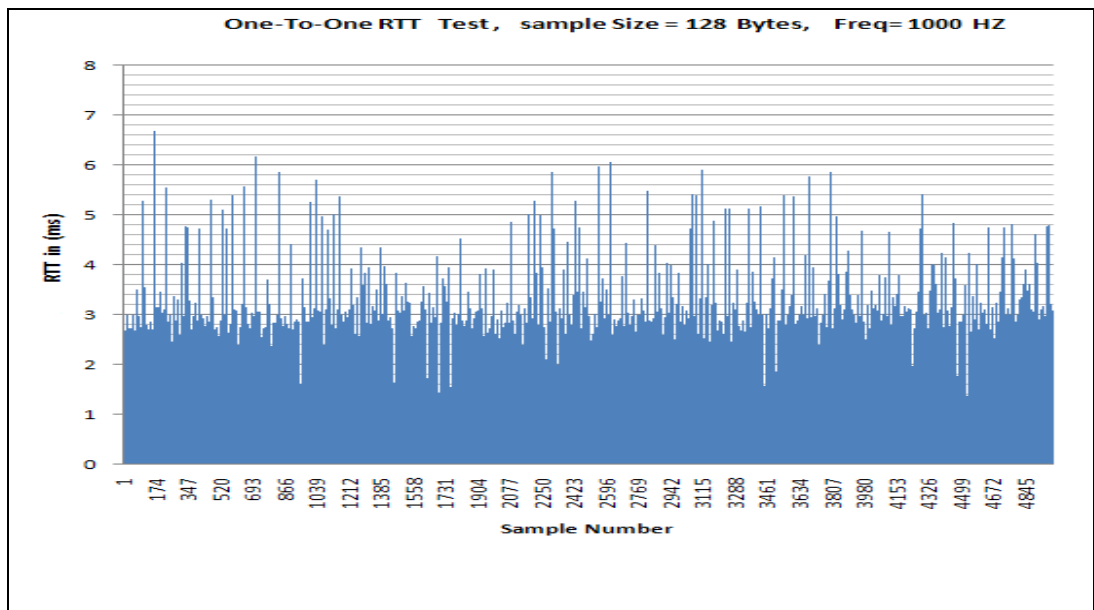


Fig.6.1: one-to-one RTT test, QoS =best effort

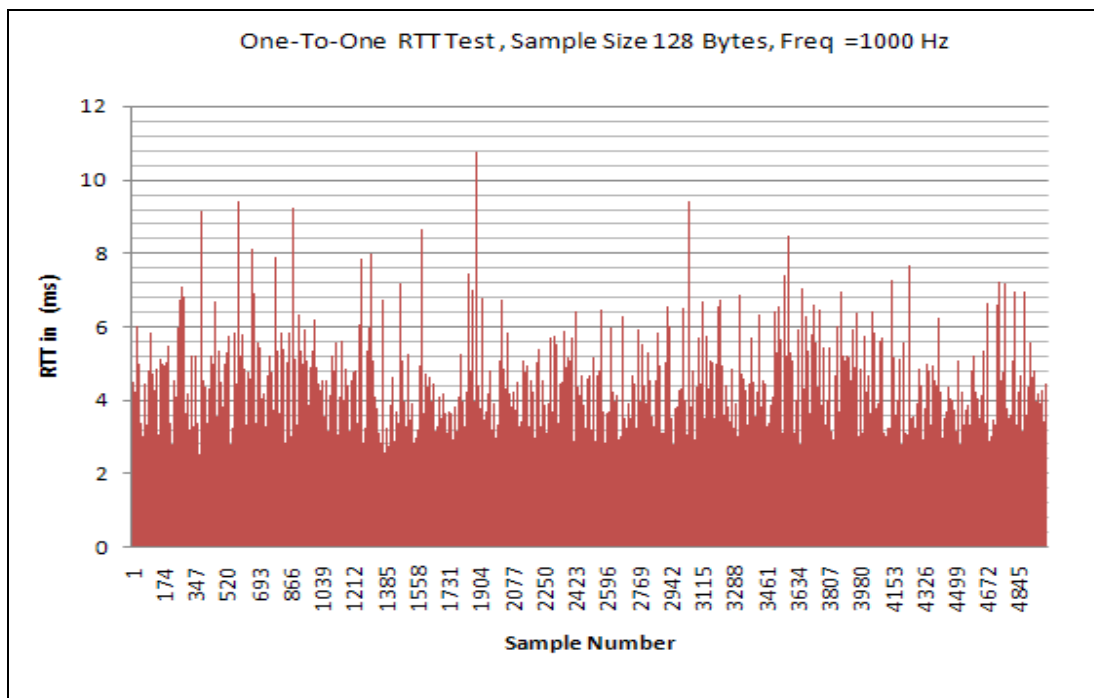


Fig 6.2: one-to-one RTT test, QoS = reliable

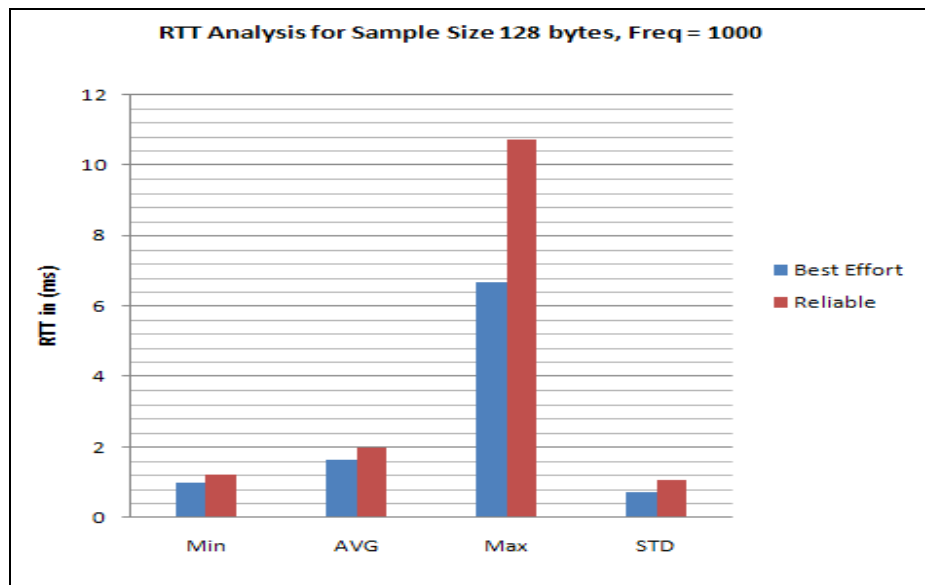


Figure 6.3: one-to-one RTT analysis, QoS = best Effort Vs reliable

The summary of the results is presented in Figure 6.3. This Figure shows the average round trip time of the messages received in the subscriber. As expected, the price of reliability is more overhead than the best-effort case. Therefore; RTT latency for first one is greater than the RTT latency for BEST_EFFORT BE. In other words, when the samples are changed between peers, best effort QoS imposes the least amount of overhead. However, it does not guarantee the delivery of the data. As result, data may be lost due to unreachable peer.

In the case of RELIABLE, it achieves the reliable delivery through two main mechanisms: two-way hand shaking and the negative acknowledgment of lost samples. Each of these mechanisms needs times and this why RTT for this kind of QoS is higher than the one we got in BEST_EFFORT (BE). It is important to

highlight that RELIABLE QoS uses wait_for_acknowledgments method to wait for subscription to acknowledge receipt of all data. Also, in this mode RTI Data Distribution Service automatically sends acknowledgments (ACK/NACKs) as necessary to maintain reliable communications.

Under the same configuration of the previous test, we made the sensor publisher application sends data where the size varies from 8 bytes to 1024 bytes.

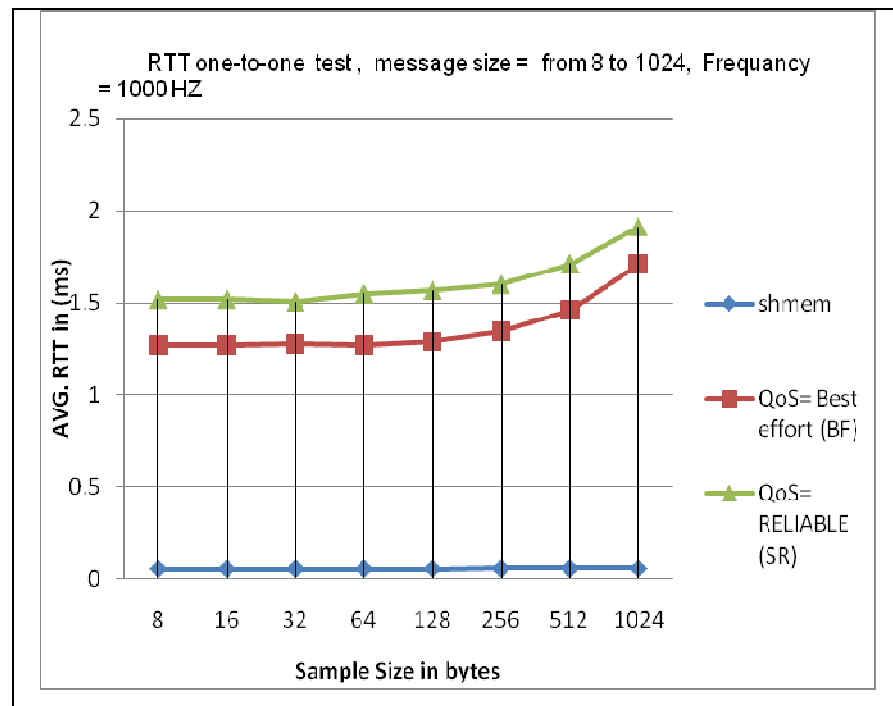


Figure 6.4: result of one-to-one RTT latency, QoS =best Effort Vs reliable over different sizes

By focusing on our interesting part of the chart that is from 8 bytes to 128 (note that in sensor networks the standard sample size is 128 bytes), Fig highlights that, at small message sizes, which is the case in WSNs, RTT remains consistently low. It is a round 1.52 ms for QoS= RELIABLE and 1.23 ms for QoS= BEST_EFFORT. The reason for RTT latency for first one is greater than from the second one is explained in details in the previous test. In addition, this shows that RTI Data Distribution Service exhibits very low jitter, making it suitable for time-time and mission-critical applications in WSNs. At larger messages sizes, which are network-limited, latency is proportional to message size. However, analyzing the data with large sizes is out of our scope.

By Making the subscriber and publisher run on the same host, we got the lowest RTT because the DDS application use shared memory (shmem) concept. It is known that shmem is faster than UDP connection.

6.2 One-to-Many RTT Latency Test Outcomes:

A. Experimental Set-up

The setup of this experiment is exactly as what we did in test One-to-Many Throughput Test. See the fig below.

B. Goal

This test shows the one-to-Many publish/subscribe latency in terms of RTT. In this test, the sensor publisher sends data where the size varies from 8 bytes to 1024bytes. These messages are sent to 15 subscriber applications on another core over a single DDS topic.

C. Implementation

For RTT Latency Test Method, We designed this test to allow only to the last data reader to invoke the echo routine. The other data readers will not invoke the method. This is done to calculate the worst case scenario which is the largest RTT value. For the 1-to-many RTT test in order to work in order to work in the correct way the RTI must send the samples in correct order. RTI-DDS will send samples to the data readers DRs in the provided order.

The test is done on mainly in two scenarios:

- A. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders (DR) in the subscriber application (Node123) to RELIABLE. (using a unicast)

- B. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders(DR) in the subscriber application (Node123) to Best_Effort. (using a unicast protocol)
- C. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders (DR) in the subscriber application (Node123) to RELIABLE. (using a multicast)
- D. Setting RELIABILITY QoS Policy in both Data DataWriters (DW) in the sensor application (node 120) and DataReaders (DR) in the subscriber application (Node123) to Best_Effort. (using a multi cast protocol)

In both tests (this and the above one), it is important to point out that we use request/response way to ensure that the round-trip time is recorded on the publisher node. Also, measuring the RTT latency is done by making times-tamp on the sent messages and subtracting that from the times-tamp value that is received in the ack message from subscriber on the publisher side.

D. Results and Remarks

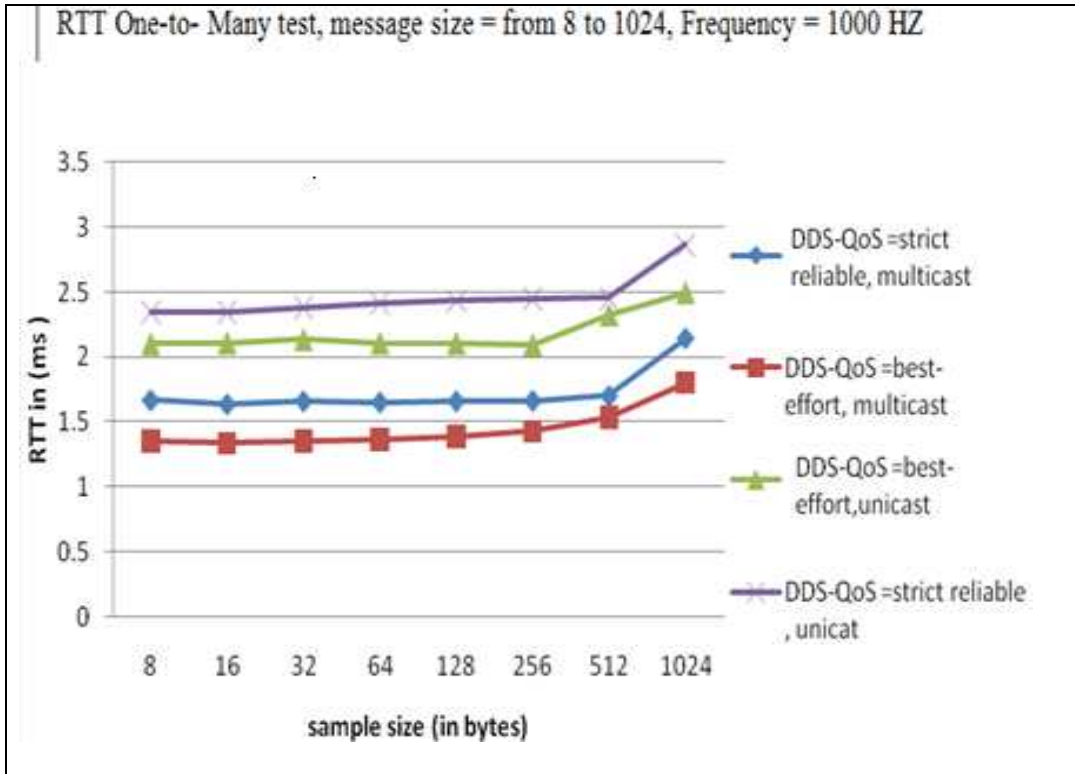


Figure 6.5: result of one-to-Many RTT latency

In the unicast protocol the sensor publisher application will take care of all the data readers. In other words, it sends a message to each single data reader. However, in multicast protocol, it will send a single multicast packet to all data readers who are listening on the same multicast address.

As evident from the above charts, at any message size, the multicast protocol has advantages over unicast. This is happened because RTI-DDS middleware invokes the send process only once, which means RTI-DDS will do less work, therefore, this is

intuitively leads to lower latency. The reason for RTT latency for reliable is greater than from the best effort is explained in details in the previous test

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

In this thesis, some main challenges facing the design of middleware for WSNs have been pointed out. Moreover, a brief description of a list of typical recent existing middleware solutions is provided. Then, focus is made on the middleware that communicates in a publish/subscribe fashion. This is done because publish-subscribe paradigms support asynchronous communications. By using P/S paradigms, the data is sent and received by asynchronous messages. Moreover, this kind of communication provides some properties that are needed in sensor networks. In addition, a P/S paradigm increases the lifetime of the network.

Also, similarities and differences between the approaches that are used to implement different middleware solutions for sensor networks are provided. Furthermore, DDS real-time system is discussed in order to address the issue of a real-time publish/subscribe middleware for WSNs.

The methodology used for evaluating RTPS-DDS middleware implementation is also presented. Moreover, important QoS parameters in DDS for RTPS middleware are addressed to optimize data delivery for a specific application in WSNs.

The tests are programmed in java language using jdk 1.6.0 to see behavior RTI DDS in WSNs. The following performance metrics are used for analysis purpose:

- **Latency**, which is the roundtrip time (RTT) between the sending of a message by the sensor publisher application and reception of an acknowledgment from the subscriber.
- **Jitter time**, which is the variation in RTT latency from sample to message. In other words, it is the standard deviation of the RTT latency.
- **Throughput**, which is defined as the total number of received sample per unit of time (such as m-second).

Finally, it is strongly believed that RTI-DDS is the most suitable middleware for WSNs since it is reliable, flexible and the highest performing implementation of the OMG DDS for real-time systems. It also has QoS properties that can be set based on the needs of a given system. These QoS parameters strongly allow designers to control their applications to get the best combination of performance and resource usage.

7.2 FUTURE WORK

Our work can be extended to cover the following:

- The security issues that are related to RTI routing service in order to make this routing service more secure and prevent anyone who wants to access it from making illegal things. Since security plays a fundamental role in many wireless sensor network applications, the accomplishment of this task will protect these applications from the harm attackers from accessing sensor sensitive information.
- Using the RTI routing Service as administrator to the networks by program it to do some kind of filtration processes. Doing this for sure will protect the WSNs from flooding with too much data.
- Trying to use RTI-DDS middleware in real nodes to see the real behavior of this middleware.

REFERENCES

1. E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, February 2006.
2. S. Michiels et al., “DAVIM: A Dynamically Adaptable Virtual Machine for Sensor Networks,” *Proc. 1st Int’l Workshop Middleware for Sensor Networks (MidSens 06)*, ACM Press, 2006, pp. 7–12
3. Johannes Gehrke, Samuel Madden, "Query Processing in Sensor Networks," *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 46-55, Jan.-Mar. 2004.
4. Madden S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. TinyDB: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.* 30, 1 (2005), 122–173.
5. Wendi, B. Heinzelman, A. L. Murphy, Hervaldo, S. Carvallo, and M. A. Perillo, .Middleware to support Sensor network applications, in *IEEE Network Mag.*, pp. 18(1):6.14., 2004.
6. Fok C, Roman G, Lu C. Mobile agent middleware for sensor networks: An application case study. In *Proc. the 4th Int. Conf. Information Processing in Sensor Networks (IPSN 05)*, UCLA, Los Angeles, California, USA. Apr. 25{27, 2005, pp. 382{387

7. Yang Yu, Bhaskar Krisnamachari and Viktor K. Prasanna. Issues in designing Middleware for wireless Sensor Networks. In COMSWARE, volume 3, pages 1530–1546, April 2006.
8. Christian Hermann, Walteneus Dargie, "Senseive: A Middleware for a Wireless Sensor Network," *AINA*, pp.612-619, 22nd International Conference on Advanced Information Networking and Applications (AINA 2008), 2008.
9. Karen Henricksen and Ricky Robinson, "A survey of Middleware for Sensor Networks: State of the art and future directions", Proceedings of MidSens'06, November 27-December 1, 2006, Melbourne, Australia.
10. Eduardo Souto, Germano Guimaraes, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, and Carlos Ferraz. A message-oriented middleware for sensor networks. In Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pages 127–134, New York, NY, USA, 2004. ACM Press.
11. P. Boonma and J. Suzuki, "Self-Configurable Publish/Subscribe Middleware for Wireless Sensor Networks," In Proc. of IEEE International Workshop on Personalized Networks (PerNets), Las Vegas, NV, January 2009.
12. Salem Hadim, Nader Mohamed. Middleware for wireless sensor networks: A survey. In Proc. the 1st Int. Conf. Comm. System Software and Middleware (Comsware06), New Delhi, India, Jan. 8–12, 2006.

13. J. M. Schlesselmaq Gerard Pardo-Castellote, and Bert Farabaugh ,” OMG Data-Distribution Service (DDS): Architectural Update,” MILCOM 2004 - 2004 IEEE Military Communications Conference.
14. Object Management Group (OMG), "Distribution Data Service for Real-Time Systems Specification", January 2007.
15. Ribeiro, A. R. L., Freitas, L. C., Silva, F. C. S., Francês, C. R. and Costa, J. C. W. “SensorBus: A Middleware Model for Wireless Sensor Networks”, in Proceedings of the 3rd IFIP/ACM Latin America Networking Conference, ACM Press, October 2005.
16. Li, Y.J.; Chen, C.S.; Song, Y.-Q.; Wang, Z. Real-time QoS support in wireless sensor networks: a survey. In Proc of 7th IFAC Int Conf on Fieldbuses & Networks in Industrial & Embedded Systems (FeT'07), Toulouse, France, Nov. 2007
17. P. Boonma and J. Suzuki, "Middleware Support for Pluggable Non-functional Properties in Wireless Sensor Networks," In Proc. of IEEE Workshop on Methodologies for Non-functional Properties in Services Computing, Honolulu, HI, July 2008.
18. Wang, C.; Sohraby, K.; Hu Y.; Li, B. and Tang, W. “Issues of Transport Control Protocols for Wireless Sensor Networks”. Proceedings of International

Conference on Communications, Circuits and Systems (ICCCAS), Hong-Kong, China, March 2005.

19. C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Psfq: a reliable transport protocol for wireless sensor networks," in Proc. of the 1st ACM international workshop on Wireless sensor networks and applications. ACM Press, 2002.
20. Paulo A. C. S. Neves and Joel J. P. C. Rodrigues, "Internet Protocol over Wireless Sensor Networks, from Myth to Reality", in Journal of Communications (JCM), Special Issue on High-performance Routing and Switching in Wireless Networks, Min Song, Yang Yang, and Sheng Fang (Eds.), Academy Publisher, ISSN 1796-2021, Vol.5, No. 3, pp.189-196, March 2010.
21. Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38(4), 393–422.
22. Salem Hadim, Nader Mohamed, "Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 1, Mar. 2006, doi:10.1109/MDSO.2006.19
23. Real-Time Innovations, "RTI Distribution Data Service, User's Manual", Version 4.1, September 2006.
24. Md.Atiqur Rahman."Middleware for wireless sensor Networks: Challenges and Approaches", TKK T-110.5190 Seminar on Internetworking, April 2009.

25. Book Chapter: Kirsten Terfloth, Mesut Günes, Jochen Schiller, “Middleware for Wireless Sensor Networks - The Comfortable Way of Application Development”, In: Guide to Wireless Sensor Networks, (Sudip Misra, Isaac Woungang, Subhas Chandra Ed.), Springer, 2009.
26. Umar, A., “Mobile Computing and Wireless Communications: Applications, Networks, Platforms, Architectures, and Security”, NGE Solutions, July 2004
27. Cougar Project. URL: www.cs.cornell.edu/database/cougar
28. X. Yu, K. Niyogi, S. Mehrotra and N. Venkatasubramanian, “Adaptive Middleware for Distributed Sensor Environments”. IEEE Computer Society, 2003.
29. S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. In Telecommunication Systems, volume 26, pages 351–368, 2004.
30. C. Srisathapornphat, C. Jaikaeo and C. Shen, “Sensor Information Networking Architecture”. International Workshop on Parallel Processing, pp. 23- 30, September 2000.
31. Mohammad M Molla and Sheikh Iqbal Ahamed. “A Survey of Middleware for Sensor Network and Challenges”, 2006 International Conference on Parallel Processing Workshops (ICPPW '06), pg. 223-228

32. Miaomiao Wang, Jiannong Cao, Jing Li, Sajal K. Das, "Middleware for Wireless Sensor Networks: A Survey", *Journal of Computer Science and Technology* (Springer). Vol. 23, No. 3, May 2008, pp. 305–326.
33. R. Barr, J.C. Bicket, D.S Dantas, B.Du, T.W.D. Kim, B. Zhou and E.G. Sirer, "On the Need for System-Level Support for Ad hoc and Sensor Networks" *Operating Systems Review*, ACM, 36(2):1-5, April 2002.
34. Kang, C. Borcea, Gang Xu, et al. "Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems," *Special Issues on Mobile and Pervasive Computing, The Computer Journal*, 2004. URL: <http://discolab.rutgers.edu/sm/papers/sm03.pdf> (accessed in February 2006).
35. Real-Time Innovations RTI URL: <http://www.rti.com> (accessed in February 2011)
36. Zubair A. Baig. *Distributed Denial of Service Attack Detection in Wireless Sensor Networks*, PhD Dissertation, Monash University, Melbourne, Australia, September 2008.

VITAE

• Personal Details

Name: Ismail Mohamed Hemdan Keshta
Date of Birth: October 20 , 1985
Place of Birth: Saudi Arabia
Nationality: Palestinian
Current address: P.O. Box 7802, Dhahran 31261, Saudi Arabia
Permanent address AlQzaan Street, Al Riyadh, Saudi Arabia
E-mail: s237697@hotmail.com
Mobile/Phone: +966506229687 / +96664224574

• Education

- Received B.S. degree in Computer Engineering from KFUPM, Saudi Arabia in February 2009.
- Received Master of Science degree in Computer Engineering from KFUPM, Saudi Arabia in June 2011

- **Work Experience**

- **Research Assistant, Computer Engineering Department, KFUPM, Dhahran, Saudi Arabia (2009-2011):**
 - **Assisting faculty members in research activities**
 - **Carrying out research in various fields of computer engineering**
 - **Grading undergraduate courses in the department**
 - **Advising undergraduate students of the department**
- **COOP Training**

Date : From February 16th, 2008 until August 27th, 2008

Company: Advanced Electronics Company (AEC)

Location: Riyadh ,Saudi Arabia

Work description: Working on Embedded systems such as “Mobile Data Terminal Units “for Automatic Vehicle Location (AVL) System.

- **References**

Available upon request