DDBS: DISTRIBUTED DISCOVERY AND BRIDGING SERVICE FOR DDS OVER INTERNET

BY

Khalid Salem BinAfif

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

September 2015

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Khalid Salem Mohammed BinAfif** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER NETWORKS.**

Thesis Committee

[illegible]

Dr. Basem Al-Madani
(Advisor)

[illegible]

Dr. Ahmad Almulhem
Department Chairman

for [illegible]

Prof. Shuang-Hua Yang
(Member)

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

Marwan [illegible]

Dr. Marwan Abu-Amara
(Member)

Date

13/9/15

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DEDICATION

This work is dedicated to my beloved parents.

ACKNOWLEDGMENTS

All praise and thanks are due to Almighty Allah. He bestowed upon me health, knowledge and patience to complete this work.

I will take this opportunity to acknowledge, with deep gratitude and appreciation, the encouragement, Valuable time and continuous guidance given to me by my thesis advisor, Dr. Basem Al-Madani. I am highly grateful to my thesis committee members: Dr. Prof. Shuang-Hua Yang and Dr. Marwan Abu-Amara for their valuable guidance, suggestions and motivation. Thanks to all KFUPM lab's technicians, engineers for their help and guidance in conducting some of experimental works related the study specially computer engineer department. Acknowledgement is due to KFUPM for the support extended towards my research and for granting me the opportunity to pursue graduate studies. My heartfelt thanks are due to my Family (mother, sisters and brothers) for their prayers, guidance, and moral support throughout my academic life. Finally, special thanks are due to my senior colleagues at KFUPM for their Help and prayers.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURES

LIST OF ABBREVIATIONS

DDS	Data Distribution Service
DDSD	Distributed Discovery Service for DDS
DHT	Distributed Hash Table
NAT	Network Address Translation
IoT	Internet of Things
STUN	Session Traversal Utilities for NAT protocol
TURN	Traversal Using Relay NAT protocol
ICE	Interactive Connectivity Establishment protocol

ABSTRACT

Full Name : Khalid Salem Mohammed BinAfif

Thesis Title :DDBS: DISTRIBUTED DISCOVERY AND BRIDGING SERVICE FOR DDS OVER INTERNET

Major Field : Computer Networks

Date of degree : September 2015

DDS was initially designed for LAN in which data producers entities (*publishers*) and data consumer entities (*subscribers*) are existing in the same physical location. Challenges arise if, for example, a subscriber is looking for a data being originated in a different data-space, especially if these data-spaces are separated by a WAN. The main limitation is the lack of ability of discovery and bridging between isolated DDS domains. The DDS standard, however, does not mention about interconnectivity among isolated DDS domains. The other challenges for implementing DDS over WAN are NAT traversal and firewalls. Real Time Innovations (RTI) has produced a plugin in their DDS package for communication over WAN that involves NATs. Their solution is based on STUN rendezvous server which can be considered as single point of failure. Therefore, DDSD is proposed improvement to their solution by implementing a distributed *Super-peer* structure for resource discovery and NAT traversal between different isolated DDS domains over internet to replace STUN rendezvous server. *Super-peers* constructs a structure DHT overlay and exchanging information. Moreover, the distance between different DDS domains and the distributed Super-peers will minimize the overall delay significantly. Simulation experiment using Omnet++ has been made to test Centralized STUN server and DDSD scenarios. The duration of participants' registration and connection establishment between participants has been measured and gives an advantage of DDSD in term of stable average delay even with large network scale. However, communication overhead between participants and Super-peer is an extra cost for DDSD.

خلاصة الرسالة

الاسم الكامل: خالد سالم محمد بن عفيف

عنوان الرسالة: الإكتشاف والتوصيل اللامركزي لخدمة توزيع البيانات على شبكة الانترنت

التخصص: شبكات الحاسوب

تاريخ الدرجة العلمية: ماجستير

خدمة توزيع البيانات (DDS) صممت لتعمل داخل الشبكات المحلية LAN حيث أن الناشرين والمشتركين موجودون في نفس الموقع. التحديات تنشئ -على سبيل المثال - في حالة وجود مشتركين يتطلعون إلى بيانات موجودة عند ناشرليس في نفس الموقع. القيد الرئيسي على خدمة توزيع البيانات هو عدم القدرة على الاكتشاف والتواصل بين المجالات لوجود الجدران النارية والحاجة إلى تجاوز خدمة ترجمة عنوان الشبكة NAT. شركة RTI سعت إلى توفير جهاز خادم مركزي ليعمل على أكتشاف والتوصيل بين المجالات المعزولة. يعيب على هذه الخدمة المركزية والتي تقلل من الاعتمادية في حالة تعطلها. من هذا المنطلق، قدمت في هذا البحث طريقة توزيع خدمة الاكتشاف لخدمة توزيع البيانات على الانترنت. وتقوم هذه الخدمة على الأعتماد على الاجهزة السوبروالتي تشكل في ما بينها شبكة غطائية (overlay network) وتتبادل المعلومات عن الاجهزة داخل المجالات الداخلية عن طريق جداول الهاش الموزعة (DHT). تمت محاكة الفكرة المقترحة على برنامج OMNET++ واظهرت الفكرة المقترحة ثباتا في المستوى مع اتساع الشبكة وزيادة عدد المستفيدين منها. تأتي هذه الميزة على حساب زيادة البيانات المتداولة على شبكة الانترنت.

CHAPTER 1

INTRODUCTION

1.1 Introduction and Problem Description

The Data Centric Publish/Subscribe (DCPS) model has been widely adopted for coordinating interactions in large-scale distributed applications. It builds on a global data space principle to be a reachable by all interested entities. Object Management Group (OMG) has founded Data Distribution Service (DDS) as an open standard for Real-Time Systems to support scalable, real time, high performance, and interoperable communication between different applications and entities. These entities are either Publisher, which is an application that contributes information to the global data space, or Subscriber, which is an application that is interested in accessing part of the global data space. DDS has been widely used for mission critical applications such as air-traffic control and smart grid management [1].

There are many application for DDS that required its implementation in large networks such as Internet. Internet of things (IoT), video conference, and files sharing are examples of these applications. One of the most trending implementations of IoT is the smart homes. A user can monitor and control most of the facilities and devices in his/her house -such as door-locks, air-conditions, washing machines, and microwave oven- remotely from his/her mobile. Video conference is one of the needed application which required high scalability. It has been implemented through DDS and shows a great advantage over other protocols such as RTP. File sharing through BitTorrent are also a potential application for DDS and it is required to be used over internet. However, its implementation on internet requires solving many obstacles such as Network Address Translator (NAT) traversal, firewall, and multicast banding.

DDS has been developed to be used in Local Area Network (LAN) where IP broadcast is enabled and the set of network joined hosts are located in the same geographical location. Challenges start when subscriber is interested in a set of data that is provided by a Publisher located in a different data space. For example, a company that has interest in data provided by another company which has a different LAN and they are only connected via WAN.

The main challenging issues in this work are: 1) Acquiring peer discovery information, which are located in different domains. 2) Adapting NAT traversal mechanism to allow DDS inter-domain communication. These two problems come from the messages sent from an application on a private LAN appear come from the LAN's gateway address, not from an internal IP address of the host running the application due to the existence of a NAT at the gateway. This does not cause problems for client/server model because only the server needs to be globally addressable; however, it is an obstacle for systems with peer-to-peer (P2P) communication models, such as DDS. Secure WAN Transport is a plug-in provided by Real Time Innovation (RTI) [2] to solve this problem, by allowing communication between peers that are located in separate LANs, using a UDP hole-punching mechanism based on the Session Traversal Utilities for NAT (STUN) protocol for NAT traversal. This requires the use of an additional rendezvous server application. However, in P2P environment, all nodes are distributed and self-organized. It is not suitable to deploy a server such as rendezvous server which is considered as a single point of failure in term of centralization in the server machine or the connections to it.

DDS has been designed to help in data exchanging on real-time distributed systems [3]. Therefore, scalability and reliability are two crucial requirements for implementing DDS over internet. For this reason, Distributed Discovery Service for DDS (DDSD) is distributing discovery and connection establishment function between inter-domains participants to be among multiple Super-peers which is going to improve the fault tolerance, reliability, scalability, and minimizing the delay in large-scale environments such as Internet. This research is targeting to construct a structured P2P overlay between Super-peers so they can exchange and share discovery information of the peers inside multiple domains. These Super-peers are going to tackle the problem of firewall and NAT traversal. An evaluation using a simulation tool Omnet++ for the centralized STUN server and the distributed Super-peers in term of registration and peers connection establishment delay and communication overhead.

1.2 MOTIVATIONAL USE EXAMPLES

In this section, a set of practical cases are presented to show the motivation and need for the proposed solution to solve some real-life problems.

1.2.1 IoT in Smart Homes controlling and monitoring

A typical user data-space contains information about user's vitals, car position, gas remaining in his car, status of his fridge, and status of a user's house doors. All of these information are relevant to a particular user, and should be accessible only by him. The user data-space in this case, a user is able to obtain information of his “user dataspace”, which is the set of data related to himself.

This use case will allow a user to discover and subscribe to any piece of information within his multiple data-spaces. Consequently, it requires the user's application to discover both the publishers and the subscribers associated with a set data-spaces, without restrictions on the types of exchanging data-content.

Industrial IoT has been implemented by RTI. It has been proven DDS successful experience in this area by leading major industrial IoT influencers [4]. It used routing service technology which we are going to discuss in section 3.2. However, the usage of DDS in personal applications like smart homes is still limited. There is a real need for a solution that resolves NAT traversal challenges will simplify its usage for ordinary users rather than configuring their gateway routers.

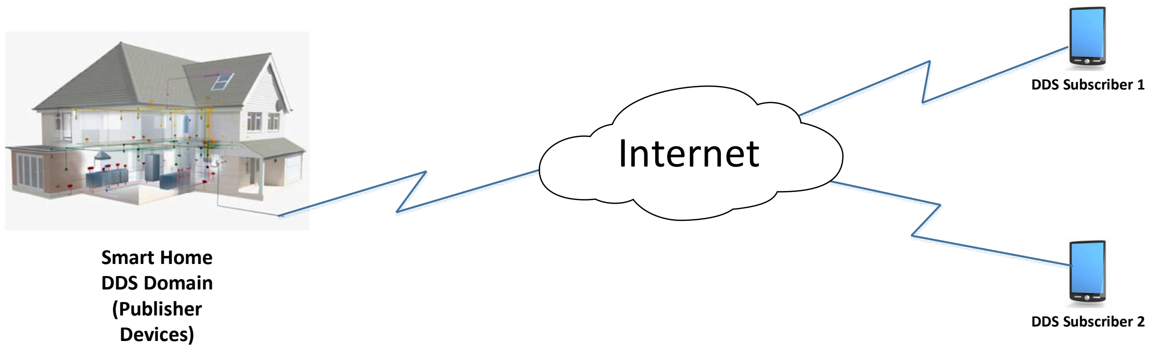


Figure 1-1 Smart-Home monitoring from a Headphone

1.2.2 Video Conference using DDS:

Video conference implementation using DDS has proven its superiority over other protocols such as Real-time Transport Protocol (RTP) especially in case of having multi-users [5]. It is, however, still limited within LAN and not yet extended to be used over WAN. The efficiency of DDS protocol can elevate the usage of video communication over internet, see Figure 1-2.

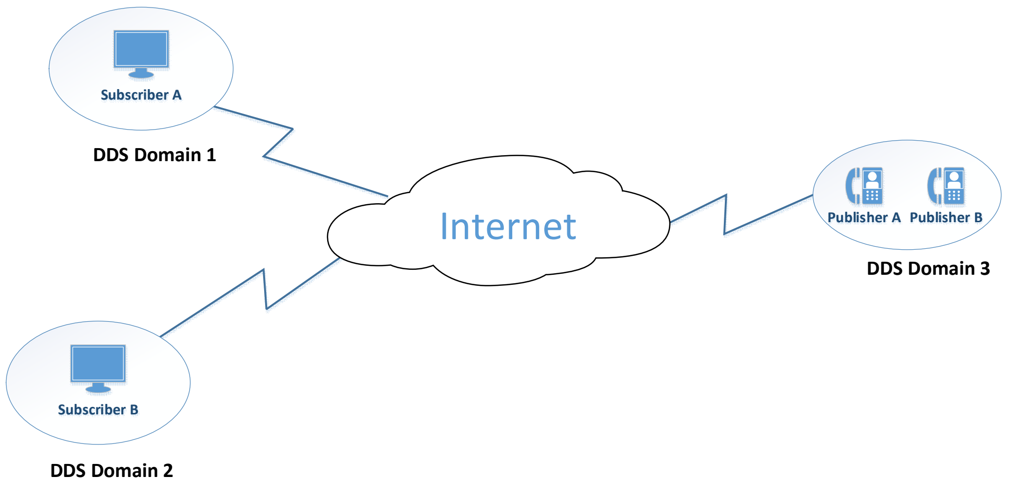


Figure 1-2: Video Streaming over Internet using DDS middleware

1.2.3 File Sharing via DDS:

BitTorrent [6] over DDS has an advantages of trackless and high performance file sharing system. It has been implemented efficiently in LAN and shows the advantages of publish-subscribe paradigm. It will be very helpful if it is implemented over internet using DDS middleware solution as in

Figure 1-3. Solving discovery challenges is the first step to implement Publish-Subscribe BitTorrent over internet.

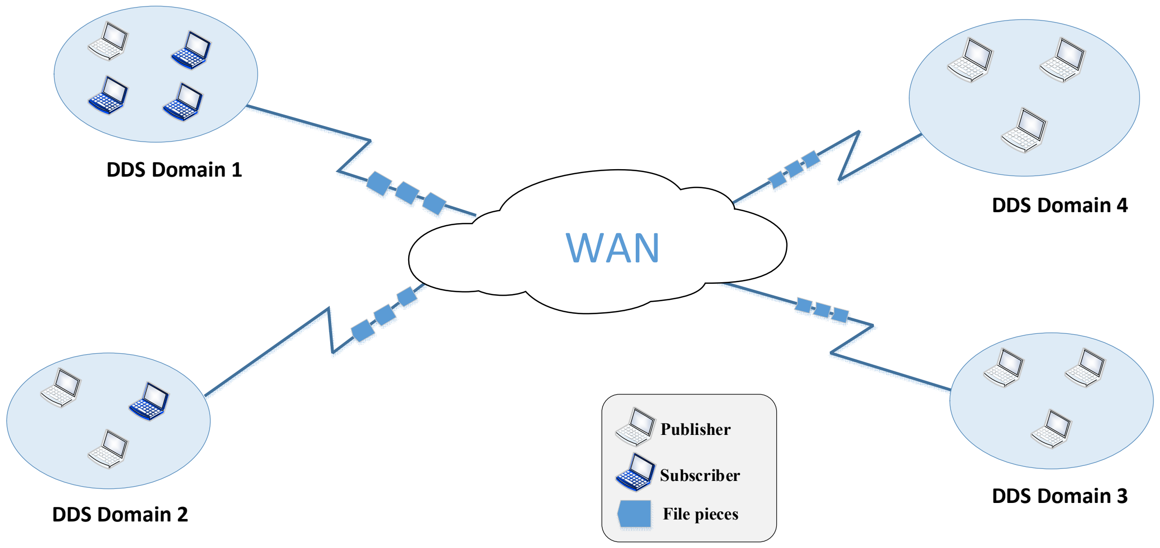


Figure 1-3: BitTorrent File Sharing over Internet using DDS middleware

CHAPTER 2

BACKGROUND

In this chapter, an introduction is presented that discussed and explained the main concepts and techniques related to this thesis work.

2.1 DATA DISTRIBUTION SERVICE (DDS)

DDS is an open standard introduced by the OMG. It is mainly specified in their document which titled Data Distribution Service for Real-time Systems specification [7]. This specification provides a Data-Centric Publish-Subscribe (DCPS) communication standard, programming model, and APIs for a range of distributed real-time and embedded computing environments in different scales. The main advantage of DCPS on heterogeneous environment is allowing applications on different platforms to write/read data to/from the Global Data Space (GDS) in a net-centric system. Each one of those application can exchange its information with other of its peers by announcing its interest to publish certain data that they are interested on it. Likewise, applications can use this particular GDS to access certain data of interest by announcing their interest to become subscribers. The underlying DCPS middleware propagates data samples written by publishers into GDS, where it is disseminated to interested subscribers. DCPS model has flexibility that gives DDS middleware more potential to optimize and support more QoS parameters.

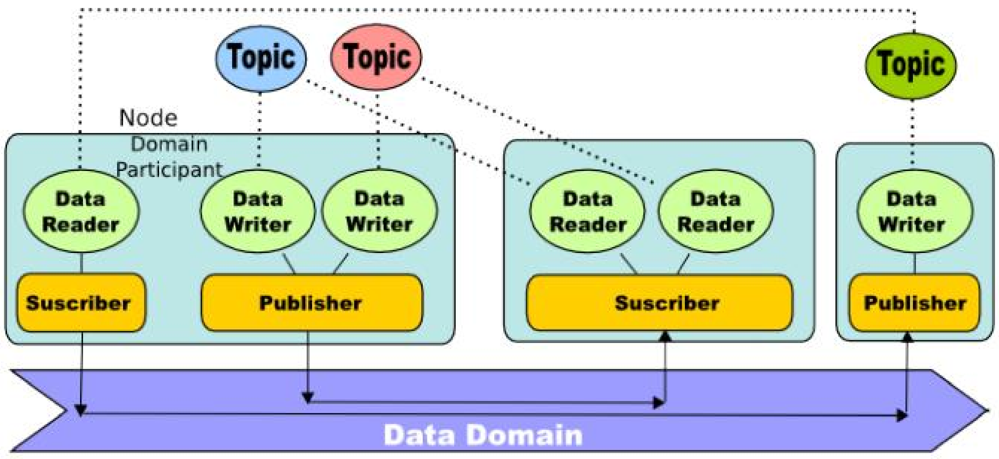


Figure 2-1 DDS Architecture [3]

To have a better understanding of this standard, DDS system components are explained as follow, see Figure **2-1**:

•**Domain**: DDS applications can exchange data inside a Domain, which makes a virtual connection between its participants joining the same Domain ID. It also provides isolation for participants linked with several domains so that participants in the same domain only can communicate to each other. This feature is very crucial for private and optimized communication for communities that have diverse interests where we can give each one of them different domain such as finance, marketing, and human resources departments.

•**Domain Participant**: It is an entry point for DDS application to involve in the domain. It represents each application in the Domain and also gives a container to hold other objects of the entity.

•**Data Writer**: It is used by applications to publish updates and data values to GDS of the domains.

•**Publisher**: It is an object made by the participant and used to initiate and manage a set of DataWriters that publish their data within a GDS. DataWriters and publishers have interrelated QoS requirements that lead their performance as DDS entities.

•**Data Reader**: They are used by applications to receive and collect data that published by DataWriter.

•**Subscriber**: It is an entity in charge of receiving and collecting published data by other applications in the same domain with considering QoS of both Publisher and Subscriber. Subscriber reads topics in the GDS for which a matching subscription exists and informs DataReaders that the data is received.

•**Topic**: A topic connects a data writer with a data reader so that communication can start only if the topic published by a data writer matches a topic subscribed to by a data reader. Communication using topics is anonymous and transparent since the DDS DCPS middleware manages these issues.

A Publisher of a topic is communicating with all Subscribers that joining the same topic. Discovery protocol in DDS gives more flexibility and transparency for all entities in joining and leaving the network. Variety and richness of QoSs that support DDS makes it as the distinguished Publish-Subscribe middleware [2], [8]

Implementation of DDS does not require any minimum level hardware infrastructure or certain platform or transport protocol. Object Management Group (OMG) has addressed a standard called DDS Interoperability Wire Protocol to guarantee the interoperability between different DDS venders [9]. The standard determines the format of exchanged messages, data representation, and entities discovery process in what is called Simple Discovery Protocol (SDP).

2.2 Distributed Hash Table (DHT)

Distributed Hash Tables (DHTs) is a routing service applied in structured P2P network model that uses a hash function to distribute objects (information) over the peers. It is used for routing, lookup, and broadcasting service. Each node in P2P stores key-value pairs so that any peer can look for the value using its key. It could give more scalability, fault recovery, and load balancing. It is going to be used among Super-peers to store participants' discovery information in most efficient way.

DHTs are mostly used for information lookup: a peer inquires the system for some information to which other peer(s) from the system replies. If not found in the original target node, the request travels in the system from peer to peer. In the past decade, several

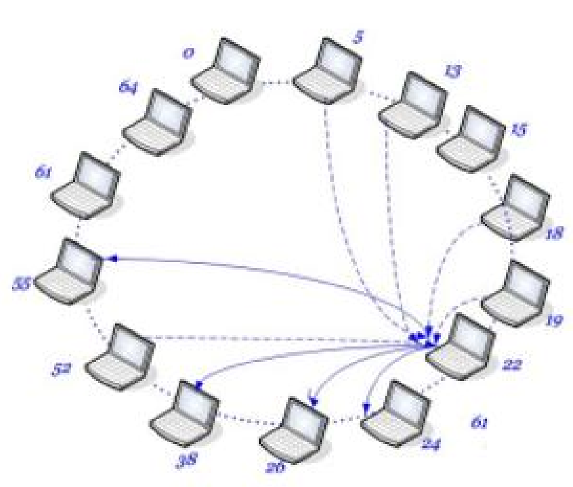
DHTs have been proposed. Basically, these DHT approaches differ in the hash space they consider. The hash space (henceforth called identifier space), can be represented as a unidimensional or multidimensional space (2D, 3D, etc.). It may be a ring, Euclidean space, hypercube, or any other type of graph. In this space, peers are less formally called nodes, while objects are referred to as keys. The objects are assigned to peers based on peer and object identifiers. Usually, a key is mapped to the closest or to the following node in the identifier space (according to a predefined order), but other methods can be employed as well. The node identifiers and their logical links represent the structure of the overlay, thus the identifier space has to be chosen accordingly. Since DHTs deal with node failures, we use the following terminology. A node is alive if it actively participates in the lookup protocol, i.e., it can reply and forward requests. Conversely, a node is dead if it cannot be contacted anymore and, as a consequence, it cannot be used to forward requests. Each peer may issues a request for an object. The process of forwarding the request from peer to peer, from source until destination, is called "routing". Each peer keeps its neighbors in a routing table, which has a fixed number of entries. Thus, when a peer fails, it has to be replaced by a new peer. Usually, there are strict rules for the peers at specific entries. Thus, finding another suitable peer, when it exists, requires additional messages. These operations represent the maintenance costs of the routing tables. The choice of the neighbor to forward a request is given by the routing strategy. DHTs have various routing strategies that depend on the overlay structure.

The DHT structures lies in the node degree, i.e., the number of neighbors with which a node maintains continuous contact for supporting the routing mechanism. Examples of such structures include Chord [10], Pastry [11], Tapestry [12] or Kademlia [13]. While these systems induce high costs for maintaining the multiple entries in the routing tables, they allow for defining routing strategies that exploit alternative paths, using alternative routing table entries for routing a request. Alternative paths have not only the advantage of providing better fault tolerance, but they also offer support for multiple routing strategies. Chord protocol is going to be explained as it is used in the Super-Peer overlay network.

2.2.1 Chord Protocol

Chord is one of the first structured P2P networks that implements Distributed Hash Tables (DHTs). In Chord [10], each node and each key has a $m - bit$ identifier on a 2^{m-1} identifier space designed as a ring, that is derived by respectively hashing the IP address and the name. Each key is mapped to the first node (starting at the key identifier) that follows clockwise on the ring. For routing purposes, each node has a routing table with m entries, each entry i pointing towards the first node on the ring at a distance of at least 2^i, where $i = 0 \ldots m - 1$. The node at entry i is also called "$finger\ i$". The corresponding links are referred to as incoming links on the node they point to.

A graphical representation of a Chord ring is shown in Figure **2-2**, as an example of the identifier space of $2^{m=6} = 64$ addresses with 15 nodes. The figure shows the outgoing and incoming links of node 22, with solid and dashed lines, respectively. Fingers do not point to nodes at a distance exactly equal to a power of 2. The same applies also to the incoming links: they do not come always from distances equal to a power of 2. For example, node 5 comes from a distance of $2^4 + 1$. Chord uses greedy routing, a routing strategy that sends the requests as closely as possible to the destination. The average path length is in the order of $O(log\ N)$. The path is clockwise on the ring, as in the example from Figure **2-2**.B: $61 \rightarrow 15 \rightarrow 19 \rightarrow 22$.



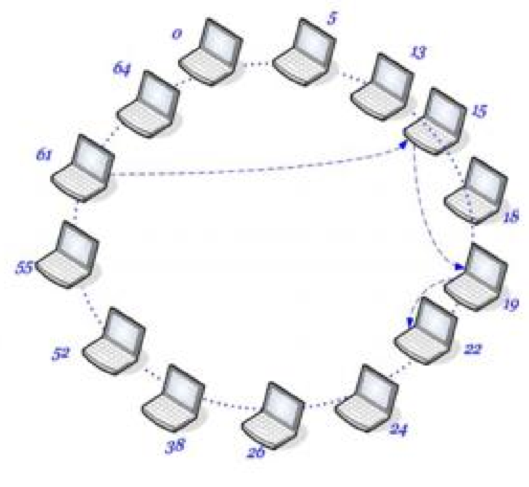


Figure 2-2: Examples of Chord with an identifier space: (A) outgoing (solid) and incoming (dashed) link of node 22. (B) Routing request from node 61 to node 22

In order to assure connectivity and to facilitate the join and departure mechanisms, each node also keeps track of its predecessor. Note that any node knows its successor, which is its first finger. When a new node n_j joins the system, links are created/updated and the responsibility of the keys are reconsidered. In this example, Node n_j creates links towards its predecessor and successor and requests them to consider n_j as their new successor and predecessor, respectively. Node n_j builds its own routing table and the other nodes from the system update their routing tables in order to reflect its arrival. With the links being created, the node n_j obtains from its successor the objects for which it is now responsible for.

A stabilization mechanism is used for the nodes that fail or leave voluntarily: periodically, each node n_s checks its successor, whether it is still alive, it is the right node and if it has n_s as its predecessor. Moreover, and also periodically, each node n_s refreshes its routing table by finding the right nodes for each entry. A node that leaves voluntarily gives the responsibility of its objects to its successor in order to preserve the rule of key mapping to nodes.

To deal with failures, each node maintains a successor-list of r nodes (i.e., its r nearest successors on the ring). Whenever a request needs to be sent to a finger that is not reachable anymore, a lower finger is used instead, and if necessary, the nodes in the successor-list can also be used as alternatives. Objects can also be replicated at several successors. Chord has thus the advantage of simplicity and has been proved to scale well with the number of nodes and to recover from high rates of Churn. [10]

Chord does not waste nodes' capacity but uses them effectively. For efficient routing, each node needs to maintain information for only $O(logN)$ other nodes. The lookups are also resolved in an efficient manner. Each node can carry out a lookup to any given node via $O(logN)$ messages to other nodes.

2.3 A NAT Traversal Technologies

The need for NAT traversal started with the advent of P2P applications, such as file sharing, Voice over IP (VoIP) and online gaming. Some applications rely on a central server to maintain a list of the connected hosts and the resources they have, while the actual data traffic is transmitted directly between the hosts. A direct communication is preferred in order to reduce transmission latency. The problem with NATs is that hosts behind a NAT, without a public address assigned to them, can only be contacted by hosts within the same private network. Even if a host would have a public address assigned to it, it depends on the type of the NAT whether an incoming connection is accepted or not. Consequently, the requirements for P2P communication work badly with NATs. But due to the popularity of P2P applications and the generality of NATs, several NAT traversal techniques have been developed. Almost all connection attempts to a private network are inhibited, unless there are static bindings. The Internet's architecture is designed to work for client-server based communications where all the connections are initiated by the client and servers are located in the public network. However, P2P communications is not as straightforward in the presence of NATs. P2P communication requires direct connections that can be initiated by either of the peers [14].

NAT traversal is a collection of different mechanisms to create connections through different type of NATs. Since the type of NATs is not known by the endpoints, multiple mechanisms may need to be tried out before finding a working one. One simple and practical technique for UDP that many NATs support is known as “UDP Hole Punching”. Hole-punching does not require any changes to be made to the infrastructure settings, instead it tries to work around the security policies of most NATs. UDP hole-punching makes use of a well-known rendezvous server for setting up a direct P2P UDP session. By means of a rendezvous server, it is possible to know if a client is behind a NAT or not by comparing the address that the server sees the client is using with the address the client thinks it is using. If the addresses differ, the client is behind a NAT. STUN is one example of a protocol that uses UDP hole punching for letting a client know its globally valid address and finding out whether it has a NATed address.

Simple Traversal of UDP Through NAT (STUN) is documented in RFC 3489 [15]. It is not a NAT traversal solution but a protocol used as a tool by other protocols to solve the NAT traversal. It can be used by a client to discover if they are behind a NAT, to determine their behavior of this NAT and its external address (IP and port). STUN is based on UDP and provides a keep-alive mechanism to maintain NAT bindings but the mechanism can be extending to TCP. It is also used to perform a connectivity check with a server or a peer. The Hole Punching technique allows two peers to have direct P2P connection through rendezvous server, even if the clients are both behind NATs. The rendezvous server gives the clients the required information about each other -such as address IP and port number- so they can communicate directly. Traversal Using Relay NAT (TURN) is a solution proposed by the Internet Engineering Task Force (IETF) to solve the problem of symmetric NAT [16]. This technique uses a third-party server in public zone or DMZ that plays the role of relay. Each client keeps a connection open with the TURN server. So, when a peer communicates with another one, all the traffic goes through the TURN server. It makes the NAT traversal because it appears for the NAT routers, it is always only the TURN server which communicates with the clients behind the NAT and no direct connection between them. It also solves the symmetric NAT because no other connection is opened. This technique is a quite different from the previous one (STUN) because it is not a specific to a protocol of communication. We do not have to modify our network or router but applications have to be compatible with the TURN protocol. Like all relay solutions, the drawbacks are the high latency, the maintenance of the server. ICE is a standard made by the IETF, it a mix of TURN and STUN [17]. The purpose of ICE is to know the behavior of the NAT with STUN to determine the best method for NAT traversal, relay or direct connection via STUN hole punching.

RTI WAN Transport service uses STUN to establish a connection between domains over WAN. Therefore, STUN is tested in this research as a single server and will be proposed for distributed Super-peers overlay.

2.4 Super-peer Model

Super-peers is a group of peers connected to each other through overlay network and having public addresses and adequate amount of resources (such as CPU, memory and bandwidth) to handle STUN services to other NATed ordinary peers. Each group of NATed peers are connected to one or more Super-peer for discovery and NAT traversal services. Super-peers establish a DHT overlay network so they update each other with all information about the participants that are connected to them, see Figure 2-3. Each group of NATed participants chooses one of the Super-peers to store its information on it and retrieve other participants' information using them. This methodology gives more scalability and reliability than centralized server. [18]–[20]

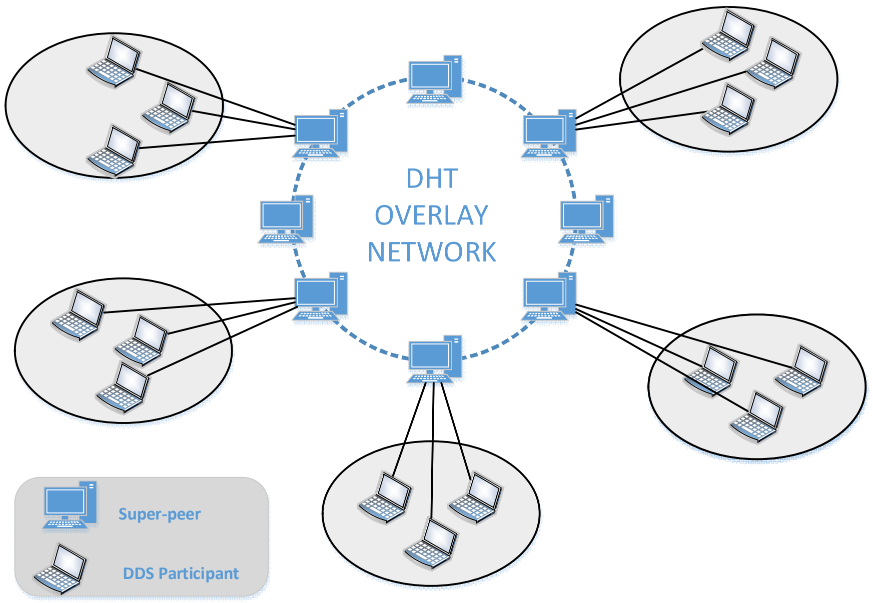


Figure 2-3: Super-Peer Model

CHAPTER 3
LITERATURE REVIEW

There are several problems associated with implementing DDS over WAN. NAT traversal and network broadcast banding are the main challenges need to be resolved. In the next sections, there is a comprehensive review for the recent solutions and technologies to overcome such issues.

3.1 WAN Transport Service:

RTI provides a transport plugin called “Secure WAN Transport” to support inter-domain communication over WAN. This transport plugin allows DDS applications running on different private networks to communicate securely over WAN such as the internet.

The WAN Transport used WAN Server, a rendezvous server that establishes P2P connection, provides the ability to discover public addresses and to register and lookup peer’s addresses based on a unique WAN ID. The WAN Server is based on the STUN protocol [15], with some extensions. The server has only to give information about public addresses for the application and once its peers have obtained this information and connections have been initiated, the server is no longer required to maintain communication with the peers. However, if communication fails, possibly due to changes in dynamically-allocated addresses, the server will be needed to reopen new connection channels.

In order to resolve the problem of communication across NAT boundaries, the WAN Transport implements a UDP hole-punching solution for NAT traversal. This solution uses a rendezvous server (WAN server). STUN protocol, which implemented by WAN server, is a part of the solution used for VoIP applications. A key advantage of STUN is that it is based on UDP and therefore is able to preserve the real-time characteristics of the DDS Interoperability Wire Protocol.

The UDP hole-punching algorithm implemented by RTI WAN transport has two different phases: registration phase and connection phase. This algorithm only works with asymmetric NATs where the same public IPv4 transport addresses (a combination of public IP address and port) is assigned to all the sessions with the same application socket address (a combination of private IP address and port number).

3.1.1 *Registration Phase*:

The RTI WAN Server application runs on a superior machine that resides on the WAN network (i.e., not in a private LAN). It has to be globally accessible from LAN applications. It acts as a rendezvous point for LAN applications. During the registration phase, each Peer is registered with RTI WAN Server using a STUN binding request message as in Figure 3-1. This registration message includes information about the requested peer such as Peer ID and current public IP and port number.

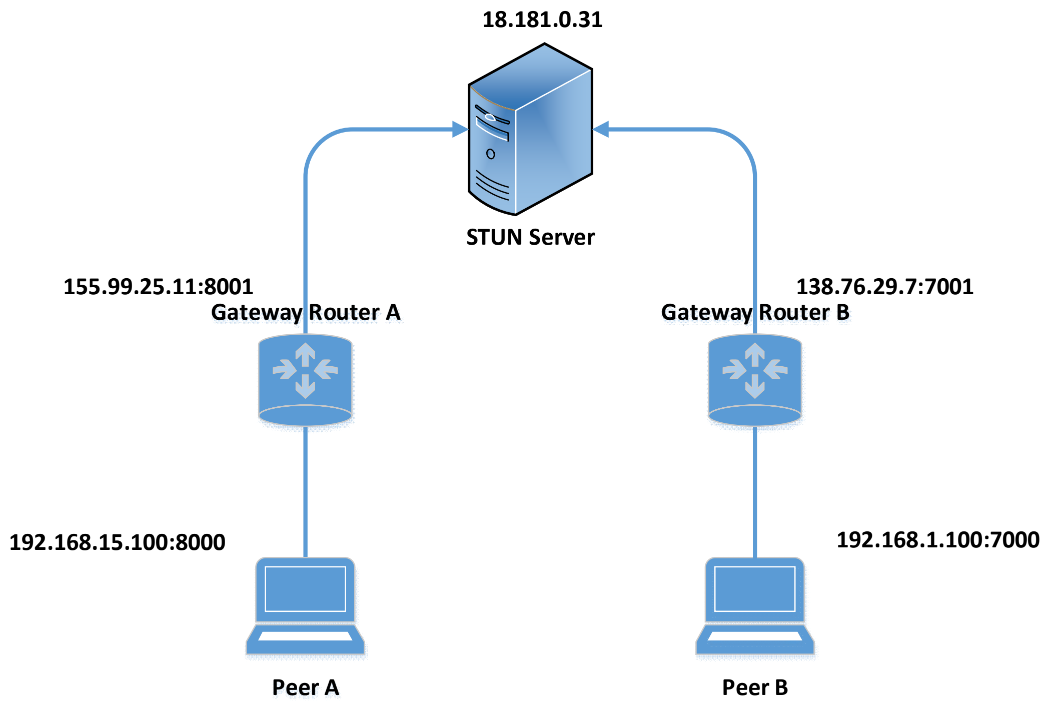


Figure 3-1: Registration phase.

3.1.2 *Connection Phase*:

The connection phase includes two processes: the connect process and NAT hole punching process (Figure 3-2). They are illustrated as follows:

1- The connect process starts when Peer A wants to establish a connection with Peer B by sending a request message to STUN server to obtain information about Peer B through nodes list.

2- STUN server responds to Peer A by sending the needed information about Peer B.

3- STUN server sends a message to Peer B. The message consists an information about Peer A and let it knows that Peer A intends to establish a connection session with it.

4- When Peer A receives the public IP address of Peer B, it will contact B using STUN binding request message. The STUN binding request message sent by Peer A directed to the public transport address of Peer B will open a hole in A's NAT to receive messages from B.

5- Peer B receives the public address of Peer A from the STUN rendezvous server and contacts Peer A by sending a STUN binding request message to that public address. This message will open a hole in B's NAT to receive messages from A.

6- Peer A receives the first STUN binding response from Peer B and it starts sending RTPS traffic.

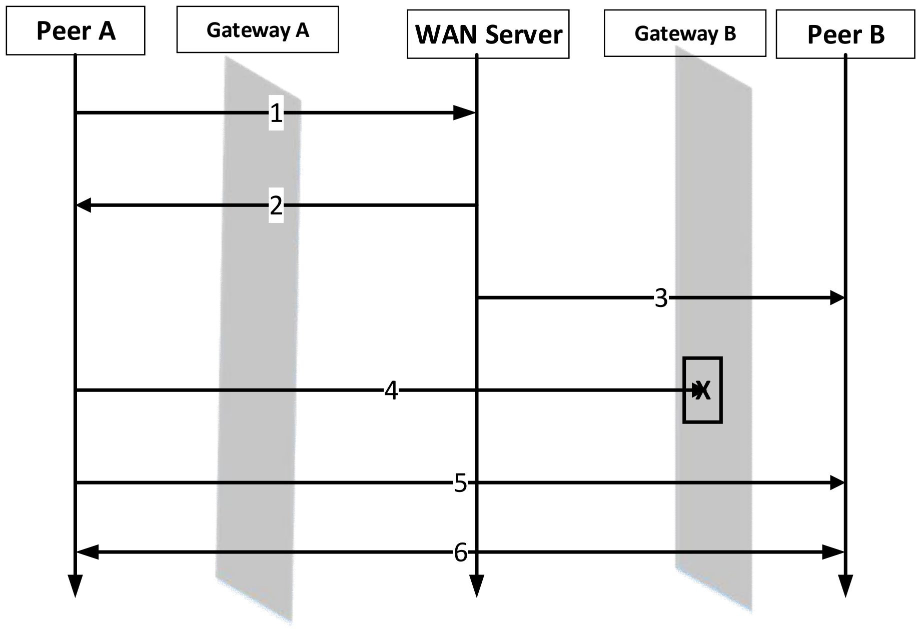


Figure 3-2 Connection phase

3.2 DDS Bridging:

A remarkable effort has been done by [3] in establishing a content-aware interconnection service to enable deploying current DDS application in transparent manner. They have proposed "DDS-to-DDS" bridging between different DDS domains using transparent data transformation mechanisms (i.e., applications can be reused without source code changes). There are also some proposed features (such as domain-bridging and topic-bridging) to allow data flow between different domains by enabling seamless communication between data-spaces. However, they did not resolve NAT traversal because they assumed that all domains are connected to the same switch.

Another recent work has proposed inter-domain DDS gateway communication based on token passing [21]. Their solution assumed that all gateways are part of their DDS domains. Thus, it is required to have an access to network gateway which is not the case in our solution where we can have NAT traversal solution to go through network's gateway without modifying their settings.

Most applications of DDS systems use either LANs or small-scale WANs whose QoS properties are relatively stable. In these environments, DDS middleware uses IP multicast to allow publishers and subscribers to operate in a scalable P2P fashion. In large-scale WANs (such as the Internet or enterprise intranets), however, IP multicast is often disabled for performance and security reasons [22].

3.3 Implementing DHT in DDS Discovery Protocol

Implementing the DHT-like overlay network has been proposed by [23]. They have shown a significant reduction in number of exchanged messages. They have used DHT overlay to implement a look-up discovery where each participant implements an overlay multicast. Even though they have tested their proposed solution for LAN, they did not consider WAN discovery challenges such as NAT traversal and firewalls. Implementing DHT overlay for content discovery and delivery in DCPS WAN-based Internet of Things (IoT) systems has been proposed in [24]. They have implemented an IETF REsource LOcation And Discovery (RELOAD) P2P signaling protocol that offers a generic, self-organized overlay network service. Their proposed solution has been tested in a simulated network that consists of 500 to 10,000 nodes to verify its scalability. However, our solution is not signaling-based protocol and it is mainly designed for DDS. Moreover, it is an enhancement for centralized STUN server to be an improved distributed alternative rather than using ICE as in RELOAD.

3.4 Other DDS discovery process proposed improvement

Content-based Filter Discovery Protocol (CFDP) has been presented in [25]. It aims to conserve system resources and improve scalability of DDS Simple Discovery Protocol (SDP) for large scale system. It develops a content filtering algorithms based on DDS QoS properties to create a special DDS topic called Content Filtered Topic (CFT) that rejects unneeded discovery messages based on matching topic name and endpoint type. However, this paper is limited to DDS implementation over LAN and they did not consider WAN-based systems.

CHAPTER 4

PROPOSED SYSTEM DESIGN AND IMPLEMTATION

This chapter will cover system architecture and design of the proposed technique for DDS over WAN participants' discovery. It also explains the discovery phases in the proposed solution. Our proposed solution is targeting to replace the current STUN protocol currently implemented in Secure WAN Transport in RTI DDS [2] which is based in STUN server. The method includes three phases: super-peers DHT overlay network constructing and maintaining, end-node registration, and discovery and connection.

4.1 System Architecture

The Distributed Discovery Service for DDS (DDSD) is pluggable discovery service used for distributed participants over WAN network such as internet to let them find the needed information to establish a connection and exchange data among them whether they are publishers or subscribers. As seen in Figure 4-1, DDSD has serving both DDS-based application and DDS middleware itself and it is served by the transport layer protocols. So, the policies and properties of DDS QoS do not apply on DDSD plug-in.

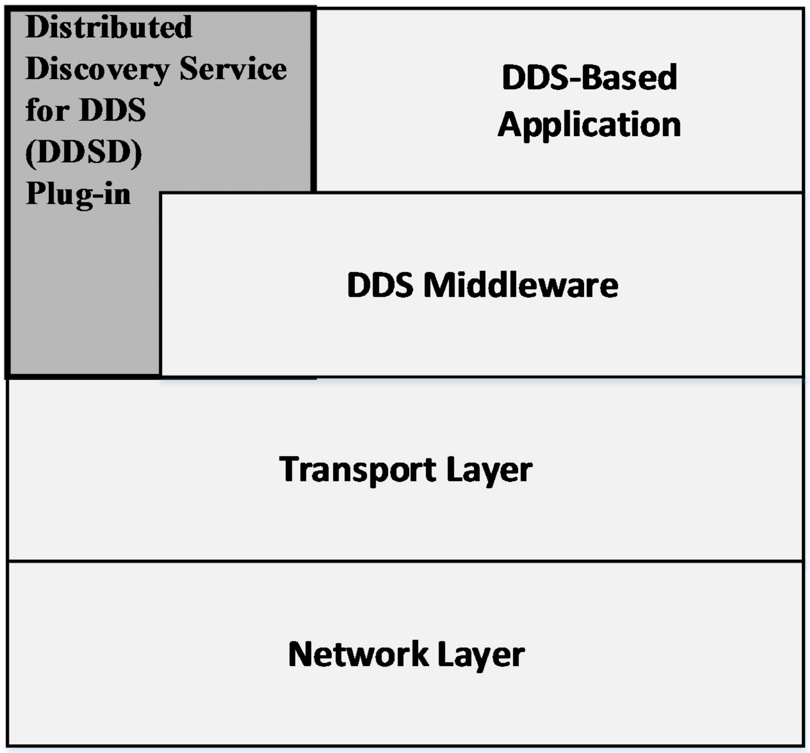


Figure 4-1: DDSD System Architecture

4.2 Super-peers DHT overlay network construction phase

Super-peers are set nodes with public IP and arranged as an overlay DHT network. For each Super-peer to join the overlay, it should go through bootstrapping process. There is a predetermined bootstrap server with a fixed public IP. For every new node to be a Super-peer, it sends a request to the bootstrap server just for the first time joining as a signup process. The bootstrap server is going to give the new joining Super-peer a new unique DHT Node-ID and list of other overlay nodes information (include, node-ID and IP address) so it can cache them and communicate with them. Once a Super-peer is connected to the overlay for the first time, it can start collect a list of public IP addresses which the Super-peer has been reached successfully in its cache for future bootstrapping process. After joining, the Super-peer is a full member in the overlay and can process get/put request as explained earlier in Section 2.2. Chord uses a hash function, Secure Hash Algorithms 1 (SHA-1), to give a unique node-ID and resource identification (such as stored participant discovery information).

4.3 Registration phase

DDS participant who has located should register with one of the available Super-peers and consider it as its Master node for participants registered with it. The participant should have in advance a set of Super-peers public IP addresses by getting them from Bootstrap server for the first signup and cache them. Another operation to have a Super-peer's IP addresses is to contact one of the neighboring participants which located in the same domain and has contacted some Super-peers so they can share their cached Super-peer addresses. This process occurrs in the first time joining. In the successive joining, the cached Super-peers information is going to be used. The participant goes through the following registration steps:

1- The participant selects one of the Super-peers - that has its information in its cache- randomly.
2- It sends a REQUST_SUPER_PEER message to the selected Super-peer to check its existence and ability to handle the participant's discovery request.
3- If the received Super-peer is available and it is handling participants less than its maximum threshold number, it sends MASTER_ACCEPT message to the requested participant. Otherwise, it sends MASTER_REJECT message.
4- If the participant receives MASTER_ACCEPT message, it considers this Super-peer for the coming discovery services. Else, it goes to step 1 again.

The participants keep sending ALIVE_BEAT message to its master Super-peer periodically to make sure that it still exists. In case of Super-peer failure or leaving, DDS participant starts the above mentioned steps again.

4.4 Discovery and Connection phase

The matching and connection establishment process between the participants from different domains are in this phase. If any participant wants to contact another one in different domain, it looks for it by its Master Super-peers using the Participant identifier. Super-peers serve the participants by finding the wanted participants and implementing the

NAT traversal process between the communicated ends. In the following illustrative steps, there are two participants, participant *A* wants to contract participant *B*. There are also two Super-peer, Super-peer *X* and Super-peer *Y* who is mastering participant *A* and participant *B*, respectively. The discovery and connection establishment steps, as shown in **Figure 4-2**, are as follow:

1- Participant A sends a REQ_PTC_MASTER message to its Master Super-peer X. The message contains domain and participant identifier of participant B.

2- Super-peer X starts fetching for the participant B's Master Super-peer (which is Super-peer Y in this scenario) in the DHT overlay. Once it is found, Super-peer X forwards the request message to it.

3- Super-peer Y confirms the request by sending ACK_PTC_MASTER message to Super-peer X. This message ensures that Super-peer Y welling to handle NAT traversal process between the two participants.

4- Super-peer X forwards ACK_PTC_MASTER message to participant A which also includes the address of Super-peer Y to allow participant A to start contacting it.

5- Participant A sends to Super-peer Y a REQ_DDS_PARTICIPANT message to which contains domain and participant identifier of its own and the targeted participant B.

6- Super-peer Y forwards the request message that comes from the source participant A, to the destination participant B. The message includes the needed information about participant A, its IP address and port number, so it can start communicate with it.

7- Super-peer Y sends ACK_DDS_PARTICIPANT to the requested participant A which includes participant B information (IP address and port number).

8- Participant A send HOLE_PUNCH message to participant B. The purpose of this message is to open a hole in participant A's gateway so it can bypass the received messages comes from participant B. However, this message will be discarded by participant B's gateway because there is no initiation from the participant.

9- Participant B send CONN_CONFIRM message to participant A. This message will open a hole in its gateway. Therefore, the following messages from participant A will bypass by the gateway to participant B.

10- By now, the two participants can start exchanging data between them.

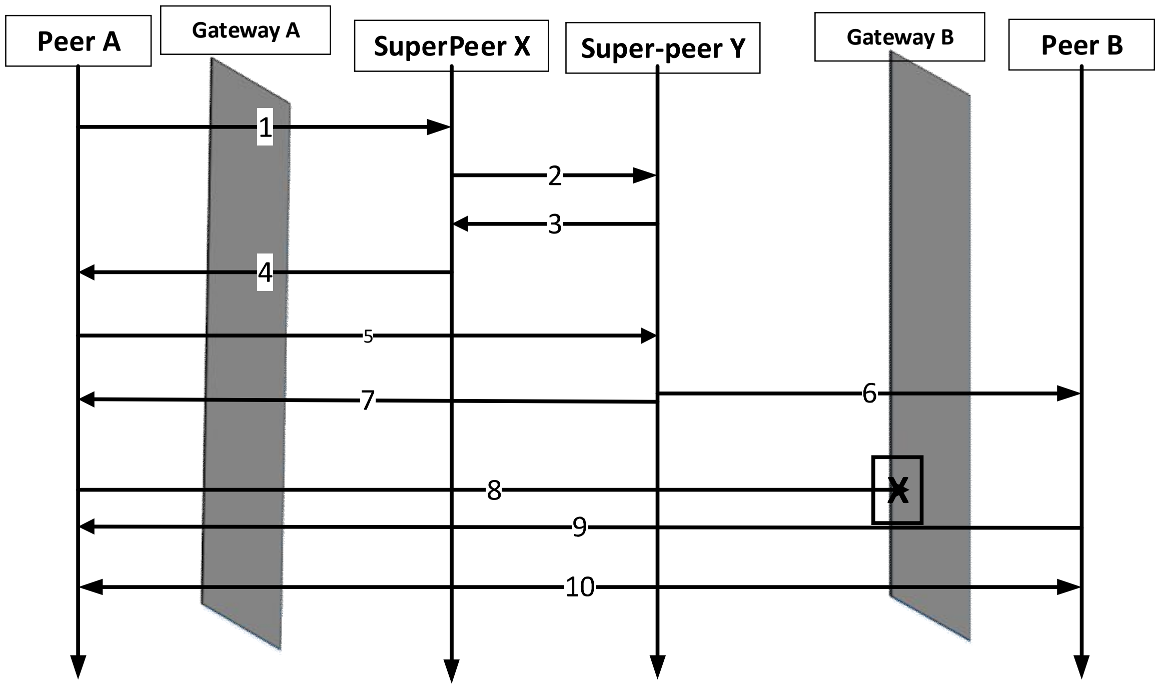


Figure 4-2: Discovery and Connection Process.

CHAPTER 5

PERFORMANCE EVALUATION

In this chapter, performance evaluation of the proposed solution is discussed. Firstly, there is an overview for the simulation tool used in the performance analysis. Then, configuration and setting of the simulation environment is presented. Finally, there is an analysis for the simulation's results.

5.1 Simulation Tool

Implementing a large scale network application in the real-world would requires hundreds or thousands of computers spread across the globe. Such an environment could contain many variables that out of the designer's control. For example, router congestion vary from day to day, which will produce varying latency results.

To allow more control of the environment, as well as having a greater scale, it has been chosen to implement DDBS as a large scale network simulation. A simulation allows for careful selection of the environment parameters and precise control of the parameter values. This helps to keep all but one parameter constant and evaluate the effect of a design decision on the system for varying values of the free parameter. Simulation also allows for greater scalability and simulating on a network with global Internet characteristics. Greater scalability is achieved, because hundreds or thousands of nodes can easily be created, where all the nodes have global scale latency characteristics.

The underlying Omnet++ is a discrete event simulation. It is a powerful network simulator and gives a most realistic environment. It allows for robust message and module definitions and contains many tools to assist with simulation measurement and monitoring such as global statistics gathering tools and built-in plotting tools [26]. Oversim is a P2P and overlay simulation framework for the Omnet++ simulator. Oversim allows for the simulation of many well-known structured or unstructured overlay protocols. It also allows for the development of applications that can use the available implemented overlays in a well-defined architecture [27].

At the base of Oversim framework is the underlay network. The underlay network determines the types of nodes in the simulation. There are three underlay types: the “simple”, “INET” and “SingleHost” underlays. A node type is determined by the protocols executing on every layer of the Oversim architecture. The INET underlay is used in our experiment because it is based on the Omnet++ INET underlay and allows for the simulation of the complete IP level stack. This includes backbone routers and gateways. In the top layer (application layer) of client hosts, we implement a STUN client application that can register with a STUN server app which resides on a centralized server. It can also establish a connection with another peer in different network after acquiring its information from the server.

5.2 Configurations and settings:

There are two scenarios have implemented in the simulation experiment. The first scenario is the implementation of the single centralized WAN server and the second one is the proposed distributed discovery service using Super-peer overlay network. The first scenario is meant to be a bench mark for the proposed solution.

5.2.1 Centralized STUN server Simulation:

It consists of 10 areas where every two areas connected with a backbone router. Each area consists of 2 networks with a gateway router for each one. A network has two types of nodes: DDS participant peer and Background traffic generator. The area is equal in each scenario. We tested STUN server in 10 different scales starts with 100 peers (where there are 5 peers in every network) up to 1000 peers (where there are 50 peers in every networks). It is incremented by 100 peers in every scenario. See Table 1 for more details.

In Figure 5-1, the total WAN network shows the five backbone routers and areas that are connected to them directly. Each area could be a different isolated ISP in a certain geographical region. We have located the STUN server near the middle router to give more balance to the areas connected to both ends. The distance between backbone routers set to

1000 km as if each one of them is in a different cautery. An area is shown in Figure 5-2 which contains two LANs and their gateway routers which are connected to each other and one of them connected to backbone router. A LAN, as in , contains Figure 5-3 A DDS domain and has DDS participants and same amount of background traffic generator hosts. They are uniformly distributed in all LANs. It is started by 5 DDS participants up to 50 hosts to test the scalability of STUN server.

Table 1: CONIFIGURATION PARAMETER

Parameters		**Value**
Simulation Duration		5000 s
Number of Backbone routers		5 routers
Register Timeout		2000 ms
Maximum Retries		5 times
Number of gateways Routers		20 routers
Distance	Between Backbone routers	1000 km
	Area-to- Backbone router	150 m
Delay	Between Backbone routers	5 ms
	Area-to- Backbone router	1 µs
Bitrate	Between Backbone routers	1 Gbps
	Area-to- Backbone router	100 Mbps
Processing Delay	Server	500 µs
	Router	100 µs

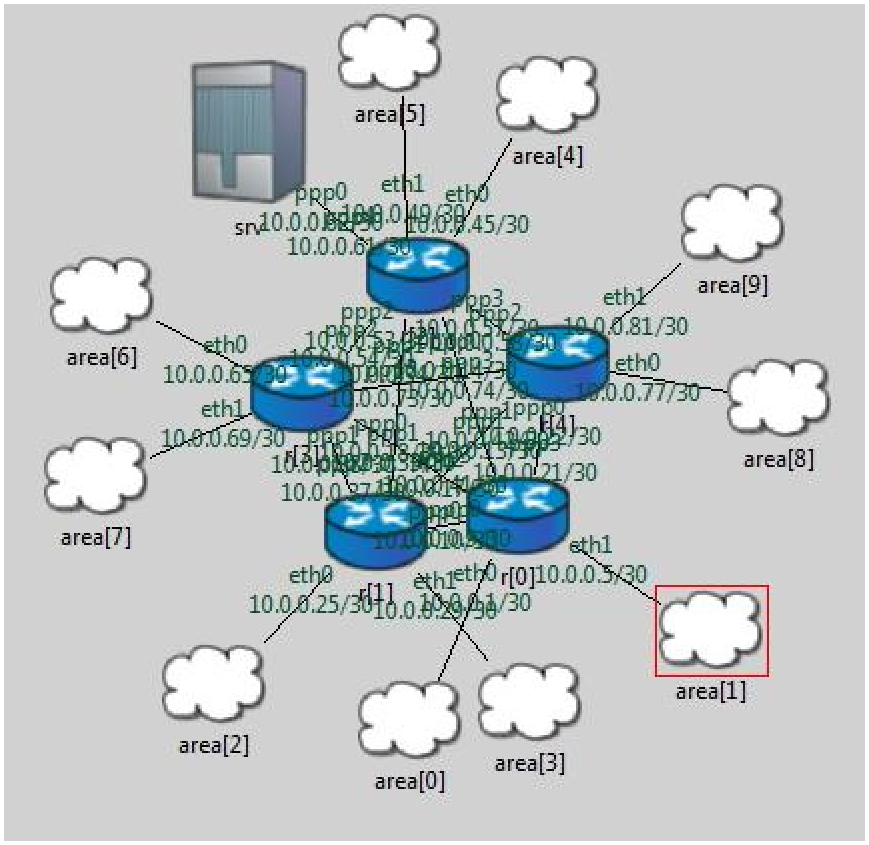


Figure 5-1 The complete WAN network for Centralized STUN Server

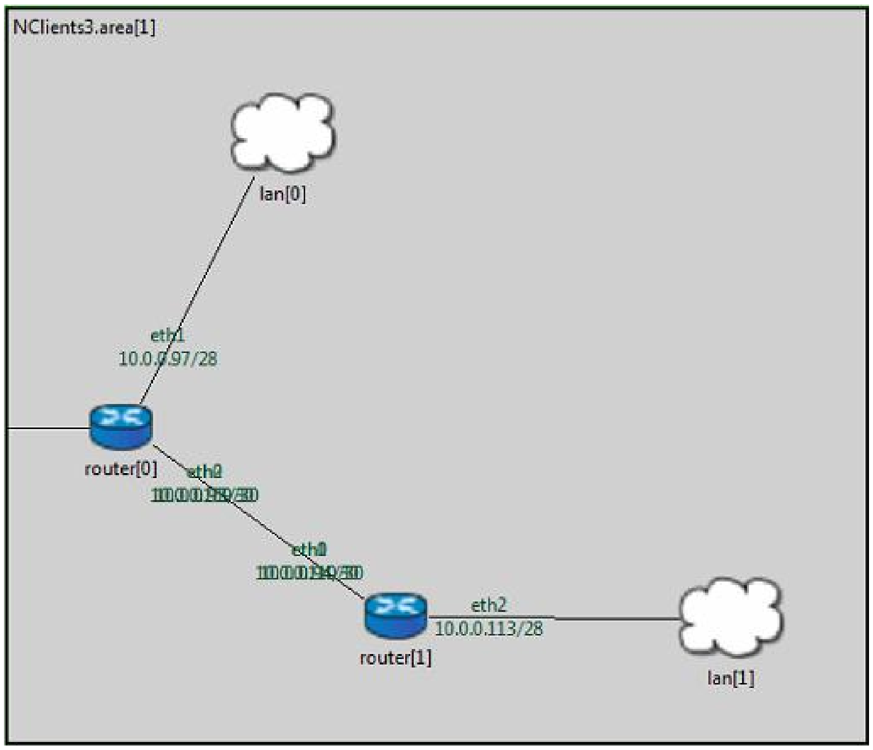


Figure 5-2 An area contains two gateways routers which connected to LANs

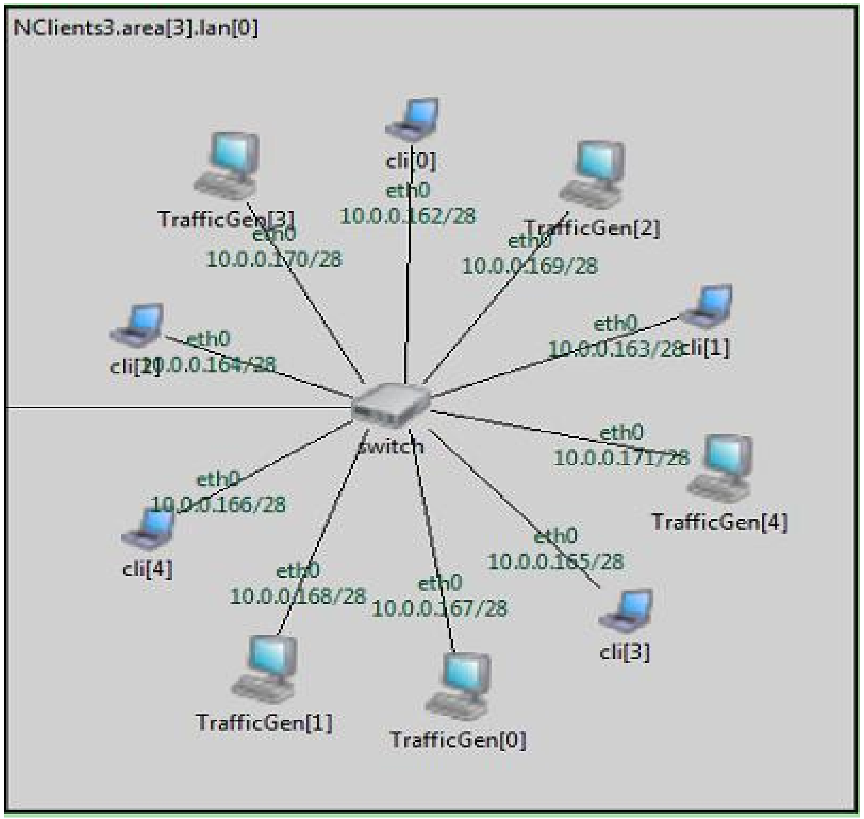


Figure 5-3 A LAN contains DDS host and background traffic generator

5.2.2 DDSD Simulation

5.2.2.1 Super-peer Scalability

In this scenario, effect of the Super-peers ratio to total served participants is studied in term of its communication overhead. Number of participants is fixed to 1000 participants distributed in 10 networks. While number of Super-peers is varying from 1% of the participants up to 20%. Therefore, it is varying from 10 Super-peers and incremented by 10 up to 200 Super-peer.

5.2.2.2 Static Super-peer

The Distributed Super-peer network consists of 5 backbone routers and 10 access routers. The number of super-peers is 10% of the total number of DDS hosts in all domains. The distribution of the DDS hosts in different domains is varies randomly. The super-peers are existing from the beginning of the simulation run until its end. DDS hosts are added to the network gradually during the simulation experiment. Figure 5-4 shows the complete DDSD network before adding Peers to the network. Backbone routers is the core of the WAN network and access routers are connected to them. Super-peers are start joining them at the beginning of the simulation. DDS nodes are also keep joining the network throughout the simulation. Measurements start to be taken after complete joining of both Super-peers and DDS nodes (see Figure 5-5).

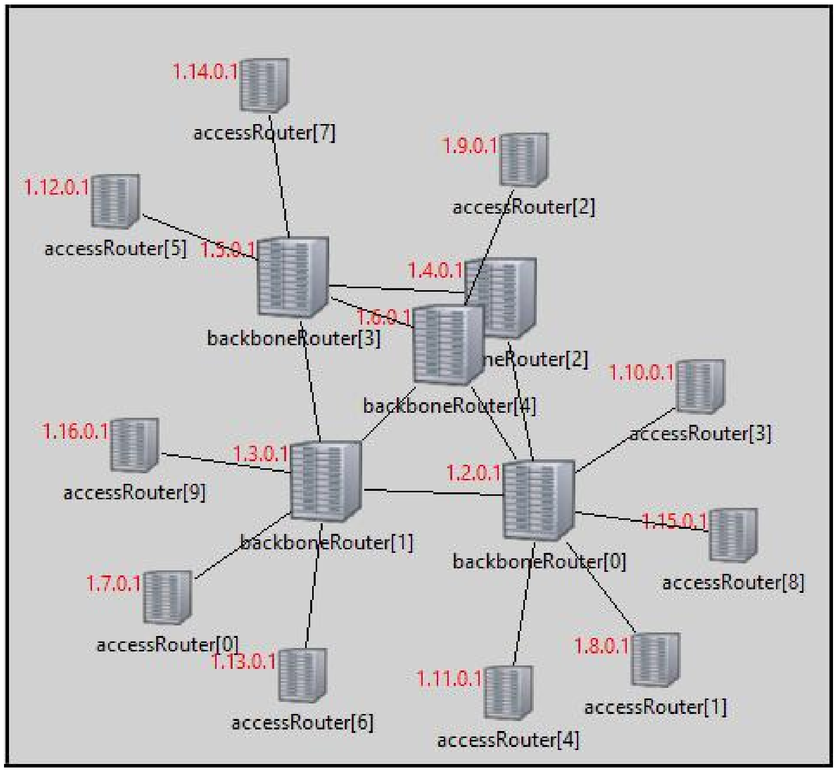


Figure 5-4: Backbone and Access routers of DDSD before Peers joining

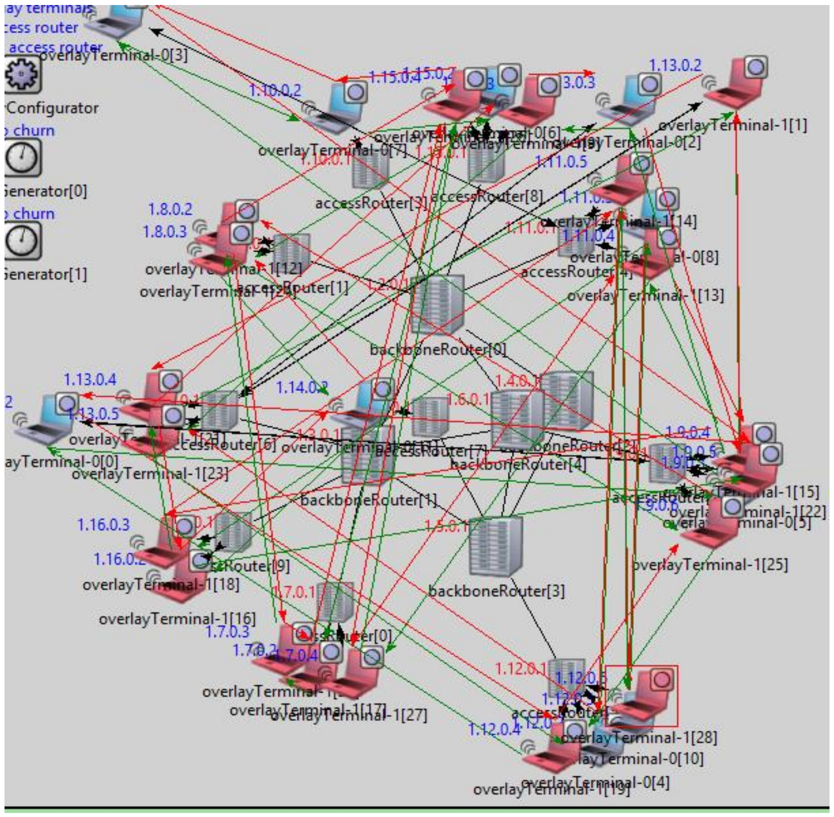


Figure 5-5: Super-peers and DDS nodes starts joining the network.

5.2.2.3 Churn Super-peer

The last scenario has been modified to make the Super-peer joins and leaves periodically. Oversim provides *Lifetime churn* generator model to examine non static network. In this model, the node will leave after the completion of its life time period and new nodes is going to be joining by finishing dead time period. The life time and dead time are specified in the configuration file as 500 seconds. Life time distribution used in this model is based on Weibull function. [28]

5.3 EVALUATION RESULTS AND ANALYSIS

In this section, an analysis for the simulation results of implementing centralized STUN server and DDSD. Number of Super-peer has been studied in term of its effect on communication overhead. There is also a study for the effect of churn of the Super-peer DHT overlay on the DDSD performance. It has been compared with static Super-peer DHT overlay.

5.3.1 Optimum ratio of Super-peers in WAN DDS

The ratio of Super-peers is relative to the total number of participants. In this experiment, the total participant is fixed as 1000 hosts and the number of Super-peers that serves them is ranging between 10-200 Super-peers. The communication overhead is dropping exponentially as the number of Super-peers increased. As noted clearly in Figure 5-6 , the average sent messages drops sharply between 10 and 100 (i.e., 1-10%) Super-peers and starts decreasing slightly after 100 Super-peers (i.e., 10%). Therefore, the optimum number of Super-peers depends on user's requirement. The Super-peer ratio in the coming experiments sets to 10% because the decreasing communication overhead after this percentage is insignificant.

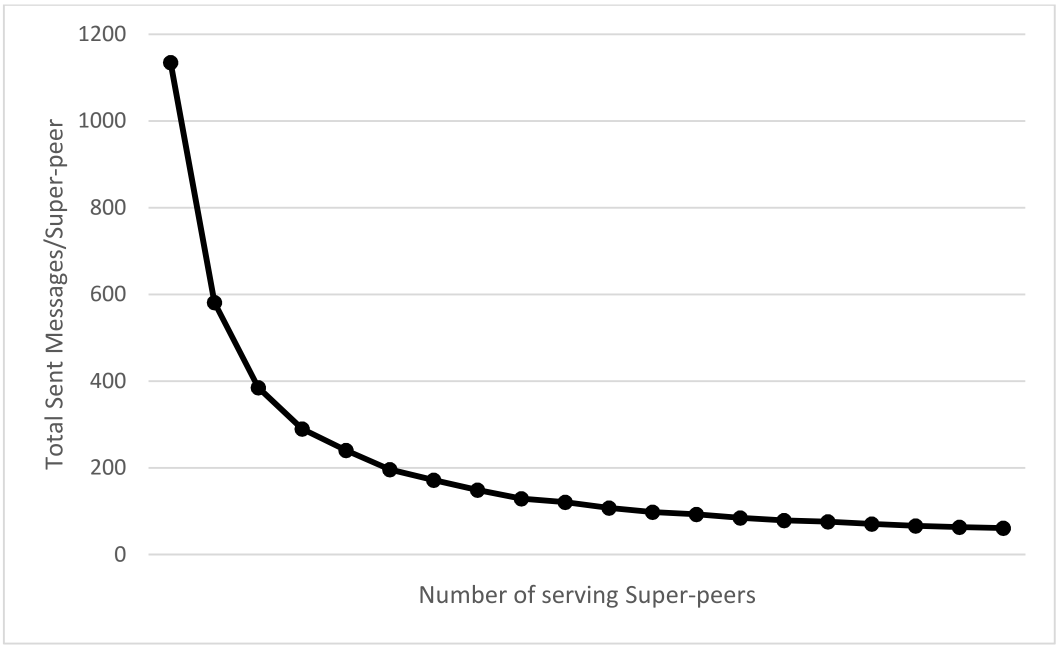


Figure 5-6: Effect of number of serving Super-peer on Communication Overhead

5.3.2 Registration Phase

It is the time required by the client (DDS participant) to register its information (Host ID, IP, and port number) in the centralized STUN server or distributed super-peers. It starts once the client sends the registration message to the server and ends by receiving registering confirmation message from the server.

Measurements taken for registration period in two scenarios: centralized STUN server and DDSD. In each one of them, the average period taken by each host in 10 different network scales starting by 100 hosts up to 1000 hosts. We can easily notice the linear increased in the average registration time for the case of Centralized STUN server. It exceeds DDSD for 400-hosts scale and more. While in case of DDSD, the average remain almost the same as we increase number of hosts that attempt to register in the distrusted Super-peer. This due to network traffic balance allowed by diverse super peer locations. Moreover, Super-peer can be much close in term of distance from DDS participant. However, there are of exchanged DHT messages between Super-peer to get the needed values from each other and that could be the reason for exceeding the delay for DDSD in scale less than 400-hosts.

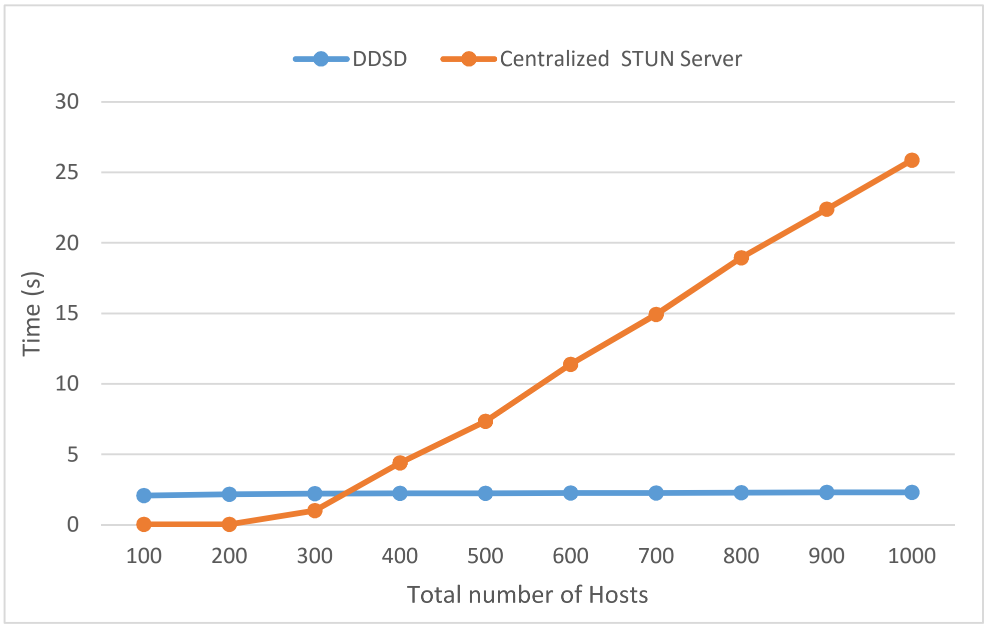


Figure 5-7 Average Registration Time for DDS Participants

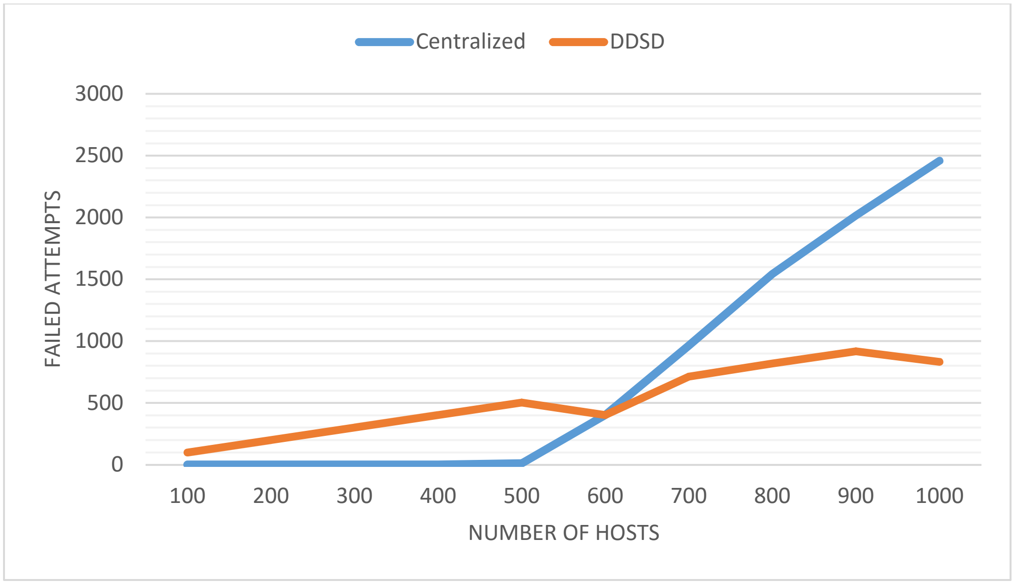


Figure 5-8 Failed registration attempts

5.3.3 Connection Establishment Phase

The time needed for any participant wants to establish a connection with another one in different domain. The process is explained in sectionV. Measurement of this period starts by sending a request message from a participant look for a subscriber in a topic published by a participant in another domain and it ends by receiving acknowledgement message from the targeted participant to the initiated one.

In this phase, communication between the peers and the STUN server is occurred twice; first time is with the initial peer that asks for the needed information about the targeted peer, and the second time is with the other peer to inform it about the request so it shall open a hole in its gateway router. Measurements are also taken in the two cases; Centralized STUN server and DDSD. In Fig. 15, we can see also a gradual increase after 400-hosts case while connection time for DDSD remains stable even with increasing the number of hosts.

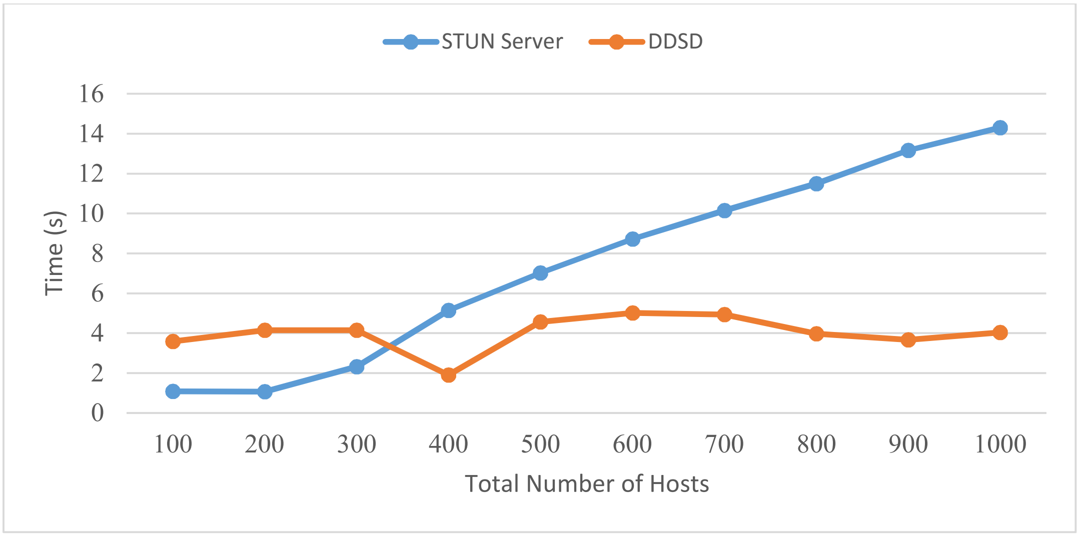


Figure 5-9: Peers connection establishment duration for Centralized STUN server and DDSD.

5.3.4 Effect of the Super-peers Churn:

By enabling the churn in the Super-peer overlay network, it has been figured out its effect on the communication overhead. The absence of a Super-peer effect appears when another Super-peer in the overlay network starts looking for it to retrieve some needed information. It also requires some more overhead to rearrange themselves in the overlay network. Figure 5-10 shows that as the number of Super-peers increases, the average messages exchanges between them decreases. The probability of missing the needed Super-peer in retrieving the required information is increased due to limitation of resources. So, there are more information requests messages. By increasing the number of Super-peers, it is going to be easier to find the needed information even with the absence of the some Super-peers due to information replication.

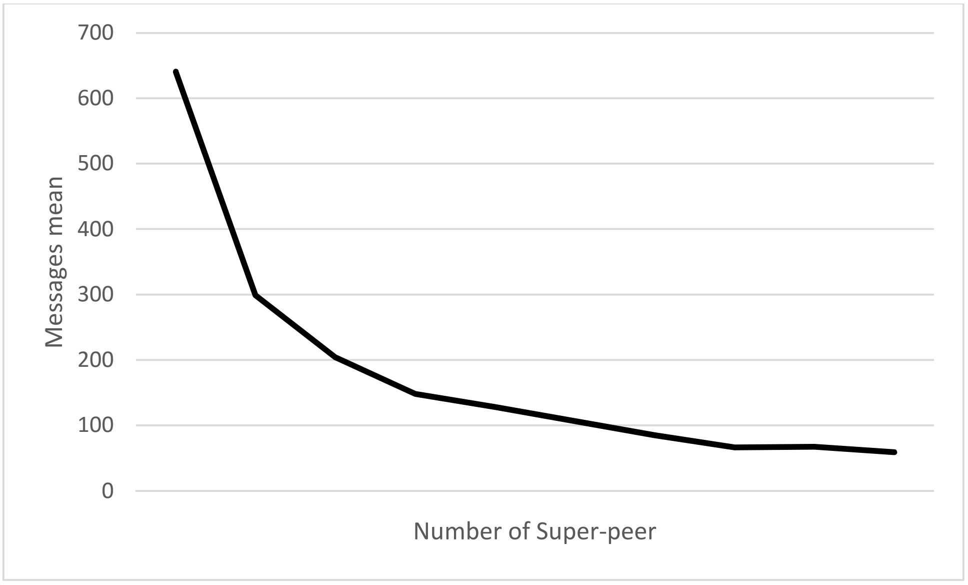


Figure 5-10: Average Sent messages between Super-peers with churn effect.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The current version of OMG DDS is not initially designed to be applied in WAN due to multicast disabling and NAT traversal challenges. The discovery process is the first and most important steps for implementing DDS over WAN. This discovery process should have advantage of reliability and scalability because it is the cornerstone for the Publish-Subscribe connection. RTI has offered an additional plugin to enable DDS participant discovery and data exchange over WAN. However, it depends on a centralized server and has a single point of failure.

In this work, Distributed Discovery Service has been proposed for DDS using DHT in a set of Super-peers. This improves the reliability and fault tolerance by avoiding the single point of failure that exist in the centralized STUN server method. Location distribution and diversity of the Super-peers can minimize the overall average registration and connection requests delay for DDS participants. A simulation has been done using OverSim (a framework in OMNeT++ simulator) for both scenarios and different network scales. The results show that DDSD has an advantage of stable average registration and connection establishment time even with larger network scales comparing with the WAN STUN centralized server. The registration delay and failure connection establishment time in centralized server case starts increasing after network scale reaching 400 hosts. However, the proposed distributed discovery mechanism shows a stable and short registration time delay, even with scalability up to 1000 DDS hosts.

As future work, Practical test in a large-scale environment -such as Planetlab [29]- is needed to test the proposed solution. Minimizing communication overhead for discovery over WAN is required by using one of the compression or hashing mechanism (such as Bloom-Filter). The security aspect of the DDS inter-domain discovery should be studied in the case of distributed discovery implementation.

APPENDIX A
Configuration code for Centralized STUN severe Scenario

```
[General]

network = inet.examples.inet.nclients.NClients3

cmdenv-interactive = false

cmdenv-runs-to-execute = 5

cmdenv-status-frequency = 10s

num-rngs = 10

output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca

parallel-simulation = false

record-eventlog = false

repeat = 10

rng-class = cMersenneTwister

seed-set = ${runnumber}

sim-time-limit = 7000s

tkenv-plugin-path = ../../../etc/plugins

# Command line setting

user-interface=  Tkenv

cmdenv-config-name =  t100hosts

cmdenv-express-mode = true
```

```
cmdenv-message-trace = false


# number of client computers

**.vector-recording = false

**.numHosts = 5

**.numClients =200

# udp apps

**.cli*[*].numUdpApps = 1

**.cli*[*].udpApp[*].typename = "STUNClientApp"

**.cli*[*].udpApp[0].localAddress = ""

**.cli*[*].udpApp[0].localPort = -1

**.cli*[*].udpApp[0].serverAddress = "srv"

**.cli*[*].udpApp[0].serverPort = 80

**.cli*[*].udpApp[0].startTime = exponential(5s)

**.cli*[*].udpApp[0].sendInterval = exponential(1s)

**.cli*[*].udpApp[0].messageLength = 1000B

************************************************************

**.cli*[*].udpApp[0].registerTimeout = 200ms

**.cli*[*].udpApp[0].maxRegisterTry = 5

**************************************************************

**.TrafficGen[*].numUdpApps= 1
```

```
**.TrafficGen[*].udpApp[*].typename = "UDPBasicBurst"

**.TrafficGen[*].udpApp[*].localPort = 100

**.TrafficGen[*].udpApp[*].destPort = 100

**.TrafficGen[*].udpApp[*].messageLength = 1250B

**.TrafficGen[*].udpApp[*].sendInterval = 0.5s

**.TrafficGen[*].udpApp[*].burstDuration = 10s

**.TrafficGen[*].udpApp[*].sleepDuration = 2s

**.TrafficGen[*].udpApp[*].destAddresses=moduleListByNedType("inet.nodes.in
et.StandardHost")

**.TrafficGen[*].udpApp[*].chooseDestAddrMode = "perBurst"

**************************************************

**.srv.numUdpApps = 2

**.srv.udpApp[0].typename = "STUNServerApp"

**.srv.udpApp[0].localPort = 80


*.srv.udpApp[1].typename = "UDPBasicBurst"

*.srv.udpApp[1].destAddresses = ""

*.srv.udpApp[1].burstDuration = 10s

*.srv.udpApp[1].sleepDuration = 1s

*.srv.udpApp[1].chooseDestAddrMode = "perBurst"

*.srv.udpApp[1].localPort = 100
```

```
*.srv.udpApp[1].destPort =100

*.srv.udpApp[1].messageLength = 1250B

*.srv.udpApp[1].sendInterval = 0.05s

**.udpApp[1].startTime = 1ms

**.udpApp[1].stopTime = 5000s

**.udpApp[1].delayLimit = 3ms


**.srv.networkLayer.ip.procDelay = 5000us

**.r[*].networkLayer.ip.procDelay = 100us

*.area[*].router[*].networkLayer.ip.procDelay = 100us

# NIC configuration

**.ppp[*].queueType = "DropTailQueue" # in routers

**.ppp[*].queue.frameCapacity = 5  # in routers


#**.srv.tcpApp[0].replyDelay = 0

************************************************************

 [Config t100hosts]

output-scalar-file = ${resultdir}/a100${configname}-${runnumber}.sca

**.numClients =100

**.numHosts = 5

include ./NS_100.ini
```

```
[Config t200hosts]

output-scalar-file = ${resultdir}/a200${configname}-${runnumber}.sca

seed-set = 5

**.numClients =200

**.numHosts = 10

include ./NS_200.ini


[Config t300hosts]

output-scalar-file = ${resultdir}/a300${configname}-${runnumber}.sca

**.numClients =300

**.numHosts = 15

include ./NS_300.ini


[Config t400hosts]

output-scalar-file = ${resultdir}/a400${configname}-${runnumber}.sca

**.numClients =400

**.numHosts = 20

include ./NS_400.ini


[Config t500hosts]
```

```
output-scalar-file = ${resultdir}/a500${configname}-${runnumber}.sca

**.numClients =500

**.numHosts = 25

include ./NS_500.ini


[Config t600hosts]

output-scalar-file = ${resultdir}/a600${configname}-${runnumber}.sca

**.numClients =600

**.numHosts = 30

include ./NS_600.ini


[Config t700hosts]


output-scalar-file = ${resultdir}/a700${configname}-${runnumber}.sca

**.numClients =700

**.numHosts = 35

include ./NS_700.ini


[Config t800hosts]

output-scalar-file = ${resultdir}/a800${configname}-${runnumber}.sca

**.numClients =800
```

```
**.numHosts = 40

include ./NS_800.ini


[Config t900hosts]

output-scalar-file = ${resultdir}/a900${configname}-${runnumber}.sca

**.numClients =900

**.numHosts = 45

include ./NS_900.ini


[Config t1000hosts]

output-scalar-file = ${resultdir}/$a1000{configname}-${runnumber}.sca

**.numClients =1000

**.numHosts = 50

include ./NS_1000.ini
```

APPENDIX B
Configuration code for Centralized STUN severe Scenario

```
[General]
description = Chord DHT (InetUnderlayNetwork)
network =  oversim.tier2.STUN_over_DHT.InetUnderlayNetwork
num-rngs = 15
record-eventlog = false
tkenv-image-path = ../images
user-interface=  Cmdenv
cmdenv-config-name = S1000
cmdenv-express-mode = true
cmdenv-message-trace = false
**.vector-recording = false
#network patrameters
*.backboneRouterNum = 5
*.overlayAccessRouterNum = 0
*.accessRouterNum = 10
**.terminalTypes = "oversim.tier2.STUN_over_DHT.STUNTerminal"

#underlayConfigurator parameters
```

```
*.underlayConfigurator.churnGeneratorTypes = "oversim.common.NoChurn oversim.common.NoChurn"

**.lifetimeMean    = 10s

**.measurementTime = 2000s

**.transitionTime  = 100s


#churn parameters

**.churnGenerator[0].targetOverlayTerminalNum = 5

**.churnGenerator[1].targetOverlayTerminalNum = 10


**.numSuperpeers = 5   #num. of peers in the network.. it should be equal to churnGenerator[0].targetOverlayTerminalNum

**.numPeers        = 10   #num. of peers in the network.. it should be equal to churnGenerator[1].targetOverlayTerminalNum


#numSuperpeers and numPeers are parameters for oversim.tier2.STUN_over_DHT.STUNSuperPeerAppModules, oversim.tier2.STUN_over_DHT.STUNPeerApp and oversim.tier2.STUN_over_DHT.GlobalDhtMat


*.churnGenerator[0].initPhaseCreationInterval = 0.1s

*.churnGenerator[1].initPhaseCreationInterval = 1s



#overlay parameters

**.overlayType   = "oversim.overlay.chord.ChordModules"

**.tier1*.dht.numReplica = 4
```

```
        **0[*].numTiers  = 2
        **0[*].tier1Type = "oversim.applications.dht.DHTModules"
        **0[*].tier2Type                                                  =
"oversim.tier2.STUN_over_DHT.STUNSuperPeerAppModules"

        **1[*].numTiers  = 2
        **1[*].tier1Type = "oversim.common.TierDummy"
        **1[*].tier2Type = "oversim.common.TierDummy"


        # dhttestapp settings
        **.tier2*.superpeerApp.testInterval = 5s   #the time before sending next self-msg
        **.tier2*.superpeerApp.testTtl = 15
        **.tier2*.superpeerApp.p2pnsTraffic = false

        **.globalObserver.numGlobalFunctions = 2
        **.globalObserver.globalFunctions[0].functionType                 =
"oversim.tier2.STUN_over_DHT.GlobalDhtTestMap"
        **.globalObserver.globalFunctions[1].functionType                 =
"oversim.tier2.STUN_over_DHT.GlobalDhtMat"

        **.superPeerName = "SuperPeerX"

        **.debugOutput = false
```

```
**.drawOverlayTopology = true


##############################################################################################################

**0[*].numUdpApps = 0

#**0[*].udpAppType = "oversim.tier2.STUN_over_DHT.STUNPeerApp"

**0[*].tier2.superpeerApp.localPort = 8088



**1[*].udpApp[0].registerTimeout = 3s

**1[*].udpApp[0].maxRegisterTry = 5

**1[*].numUdpApps = 1

**1[*].udpAppType = "oversim.tier2.STUN_over_DHT.STUNPeerApp"

**1[*].udpApp[0].localAddress = ""

**1[*].udpApp[0].localPort = 5500

**1[*].udpApp[0].superPeerAddress = "overlayTerminal-0[0]"

**1[*].udpApp[0].superPeerPort = 8088


**1[*].udpApp[0].startTime = exponential(3s)

**1[*].udpApp[0].sendInterval = exponential(3s)

**1[*].udpApp[0].messageLength = 1000B


**.rpcUdpTimeout = 1.5s


include ./default.ini
```

```
    ########################################################################
#############################

    [Config S100]

    **.churnGenerator[0].targetOverlayTerminalNum = 10

    **.churnGenerator[1].targetOverlayTerminalNum = 100


    **.numSuperpeers = 10   #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

    **.numPeers      = 100   #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum


    [Config S200]

    **.churnGenerator[0].targetOverlayTerminalNum = 20

    **.churnGenerator[1].targetOverlayTerminalNum = 200


    **.numSuperpeers = 20   #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

    **.numPeers      = 200   #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum



    [Config S300]

    **.churnGenerator[0].targetOverlayTerminalNum = 30

    **.churnGenerator[1].targetOverlayTerminalNum = 300
```

```
**.numSuperpeers = 30  #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

**.numPeers       = 300  #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum


[Config S400]

**.churnGenerator[0].targetOverlayTerminalNum = 40

**.churnGenerator[1].targetOverlayTerminalNum = 400


**.numSuperpeers = 40  #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

**.numPeers       = 400  #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum


[Config S500]

**.churnGenerator[0].targetOverlayTerminalNum = 50

**.churnGenerator[1].targetOverlayTerminalNum = 500


**.numSuperpeers = 50  #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

**.numPeers       = 500  #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum



[Config S600]

**.churnGenerator[0].targetOverlayTerminalNum = 60

**.churnGenerator[1].targetOverlayTerminalNum = 600
```

```
        **.numSuperpeers = 60   #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

        **.numPeers        = 600   #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum



        [Config S700]

        **.churnGenerator[0].targetOverlayTerminalNum = 70

        **.churnGenerator[1].targetOverlayTerminalNum = 700


        **.numSuperpeers = 70   #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

        **.numPeers        = 700   #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum

        [Config S800]

        **.churnGenerator[0].targetOverlayTerminalNum = 80

        **.churnGenerator[1].targetOverlayTerminalNum = 800


        **.numSuperpeers = 80   #num. of peers in the network.. it should be equal to
churnGenerator[0].targetOverlayTerminalNum

        **.numPeers        = 800   #num. of peers in the network.. it should be equal to
churnGenerator[1].targetOverlayTerminalNum


        [Config S900]

        **.churnGenerator[0].targetOverlayTerminalNum = 90

        **.churnGenerator[1].targetOverlayTerminalNum = 900
```

```
**.numSuperpeers = 90  #num. of peers in the network.. it should be equal to churnGenerator[0].targetOverlayTerminalNum

**.numPeers       = 900  #num. of peers in the network.. it should be equal to churnGenerator[1].targetOverlayTerminalNum

[Config S1000]

**.churnGenerator[0].targetOverlayTerminalNum = 100

**.churnGenerator[1].targetOverlayTerminalNum = 1000

**.numSuperpeers = 100  #num. of peers in the network.. it should be equal to churnGenerator[0].targetOverlayTerminalNum

**.numPeers       = 1000  #num. of peers in the network.. it should be equal to churnGenerator[1].targetOverlayTerminalNum
```

REFERENCES

[1] G. Pardo-Castellote, “OMG data distribution service: architectural overviem,” in *IEEE Military Communications Conference, 2003. MILCOM 2003.*, 2003, vol. 1, pp. 242–247.

[2] Real Time Inovation, “Core Libraries and Utilities User ’ s Manual,” no. v5.0, 2012.

[3] J. M. Lopez-Vega, J. Povedano-Molina, G. Pardo-Castellote, and J. M. Lopez-Soler, “A content-aware bridging service for publish/subscribe environments,” *J. Syst. Softw.*, vol. 86, no. 1, pp. 108–124, Jan. 2013.

[4] “Internet of Things Influence Study | Appinions Data Journalism,” 2014. [Online]. Available: http://dj.appinions.com/iot-july-2014/. [Accessed: 22-Apr-2015].

[5] B. Almadani, M. Alsaeedi, and A. Al-Roubaiey, “QoS-Aware Scalable Video Streaming Using Data Distribution Service,” *Multimed. Tools Appl.*, 2015.

[6] “BitTorrent.org.” [Online]. Available: http://www.bittorrent.org/index.html. [Accessed: 11-May-2015].

[7] Object Management Group (OMG), “The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDS-RTPS),” no. September, 2014.

[8] G. Pardo-castellote and D. Ph, “OMG data-distribution service: architectural overview,” in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.

[9] OMG, “The Real-time Publish-Subscribe (RTPS) Wire Protocol DDS Interoperability Wire Protocol Specification v2.1,” vol. 15, no. January, 2009.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[11] A. I. T. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” *Middlew. 2001*, pp. 329–350, Nov. 2001.

[12] B. Zhao, L. Huang, and J. Stribling, “Tapestry: A resilient global-scale overlay for service deployment,” *Sel. Areas …*, 2004.

[13] P. Maymounkov, D. Mazi, and D. Mazieres, “Kademlia: A peer-to-peer information system based on the xor metric,” *Peer-to-Peer Syst.*, pp. 53–65, 2002.

[14] D. Kegel, P. Srisuresh, and B. Ford, “State of Peer-to-Peer(P2P) Communication Across Network Address Translators(NATs),” no. RFC 5128, 2008.

[15] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, “STUN-simple traversal of user datagram protocol (UDP) through network address translators (NATs),” *IETF*, vol. RFC 3489, 2003.

[16] P. Matthews, R. Mahy, and J. Rosenberg, “Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN),” *IETF*, no. RFC 5766, 2010.

[17] J. Rosenberg, “Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” *IETF*, no. RFC 5245, 2010.

[18] Y. Sun, Y. R. Yang, X. Zhang, Y. Guo, J. Li, and K. Salamatian, “THash: A Practical Network Optimization Scheme for DHT-based P2P Applications,” *IEEE J. Sel. Areas Commun.*, vol. 31, no. 9, pp. 379–390, Sep. 2013.

[19] J. Dowling, J. Sacha, and S. Haridi, “Improving ICE Service Selection in a P2P System using the Gradient Topology,” *First Int. Conf. Self-Adaptive Self-Organizing Syst. (SASO 2007)*, pp. 285–288, Jul. 2007.

[20] E. Dutkiewicz, “SMBR: A novel NAT traversal mechanism for structured Peer-to-Peer communications,” in *The IEEE symposium on Computers and Communications*, 2010, pp. 535–539.

[21] W. Lee, S. Chung, M. Choi, S. Cho, I. Joe, J. Park, S. Lee, and W. Kim, “A robust inter-domain DDS gateway based on token passing for large-scale Cyber-Physical Systems.” pp. 868–871, 2014.

[22] A. Hakiri, P. Berthou, and A. Gokhale, “Supporting SIP-based end-to-end data distribution service QoS in WANs,” *J. Syst. Softw.*, vol. 95, pp. 100–121, 2014.

[23] S. Chae, S. Ahn, K. Kang, and J. Kim, “Fast Discovery Scheme Using DHT-Like Overlay Network for a Large-Scale DDS,” *Control Autom. ...*, pp. 128–137, 2011.

[24] J. M. Lopez-Vega, G. Camarillo, J. Povedano-Molina, and J. M. Lopez-Soler, “RELOAD extension for data discovery and transfer in data-centric publish–subscribe environments,” *Comput. Stand. Interfaces*, vol. 36, no. 1, pp. 110–121, 2013.

[25] K. An, A. Gokhale, D. Schmidt, S. Tambe, P. Pazandak, and G. Pardo-castellote, “Content-based Filtering Discovery Protocol (CFDP): Scalable and Efficient OMG DDS Discovery Protocol Categories and Subject Descriptors,” *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst.*, 2014.

[26] “OMNeT++ Discrete Event Simulator.” [Online]. Available: http://omnetpp.org/. [Accessed: 22-Apr-2015].

[27] “The OverSim P2P Simulation Framework.” [Online]. Available: http://www.oversim.org/. [Accessed: 22-Apr-2015].

[28] R. Kaur and K. Kumar, “Analysis of Different Chum Models in Chord Based Overlay Networks,” pp. 6–8, 2014.

[29] “PlanetLab.” [Online]. Available: http://www.planet-lab.org/.

Vitae

Name : Khalid Salem BinAfif

Nationality: Yemen

Date of Birth: 1985, 19 December

Email: binafif.kh@gmail.com

Address: Jeddah- Saudi Arabia

Academic Background:

- Bachelor of Engineering, Communication Engineering (honors).
  - International Islamic University Malaysia (IIUM)
  - Majoring in Wireless Communications.
- Master of Science, Computer Networks
  - King Fahd University of Petroleum and Minerals (KFUPM).
  - Majoring in Distributed Computing and P2P Networking

Working Experience:

- Research Assistance at Computer Engineering Department (COE) (KFUPM - Dhahran – Saudi Arabia).
  - Networking Laboratory Preparation.
  - ABET committee.
- IT Help-Desk at Nestle Saudi Arabia.
  - Experiencing multinational company culture.
  - Experiencing the process of the joint venture establishment.
- Trainee in ICC- Jeddah – Saudi Arabia
  - It is specialized in Satellite Networking (VSAT).
  - Its services cover the whole Middle East.

Professional Courses:

- Participated and completed in a CERTTIFIED PEROJECT MAMAGER (CPM) preparation course.
    - 40 hours course held in Business Leaders institute.
    - -It is based on the content of the International Association of Project and Program Management (IAPPM).
- Participated and completed a CISCO CERTEFIED
    - NETWORK ASOCIETE (CCNA) preparation course.
    - 80 hours course held in Ultimate institute.
    - It covered all the materials required to pass the CCNA exam

Researches and Projects:

- Bachelor Graduate Project: Design and Simulation of Ultra-Wideband (UWB) Synthetic Aperture Radar (SAR).
    - Design and simulate strip-map SAR.
    - Reconstructed and processing images generated by radar echoed signal.
- Other Courses' Projects:
    - Design and Develop a Cloud Computing Platform for a University Environment.
    - Performance analysis of modulation schemes using Monte Carlo tool.
    - Satellite Link Performance analysis at C/Ku/Ka Bands.
    - Priority Based Sensor Assignment for Task-Oriented WSN.
    - Self-Tracker BitTorrent over Publish-Subscribe Middleware.